EDISOFT
DEFENCE & AEROSPACE TECHNOLOGIES

embedded brains

CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

jenaoptronik

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Lero THE IRISH SOFTWARE RESEARCH CENTRE

# RTEMS SMP
## Ready to Fly

# QT-109 Technical Note
# RTEMS SMP
# Qualification Target
## Release 6

# CONTENTS



**Identification, Copyrights and License**



| Release | Git Hash | Date | Status | Changes |
|---|---|---|---|---|
| 01 | e487910c | 2019-10-18 | Replaced | Initial version |
| 02 | acc1b518 | 2019-11-11 | Replaced | Resolved the following *RID* (actions):<br>• PDR-MF-01<br>• PDR-MF-02 (#223)<br>• PDR-MF-05 (#224)<br>• PDR-MV-13 (#81, #85, #126, #127, #131, #132, #133, #193)<br>• PDR-MV-16 (#198)<br>• PDR-MV-17 (#200)<br>• PDR-MV-18 (#201)<br>• PDR-TT-03 (#205) |
| 03 | 5101c7ad | 2020-06-29 | Replaced | • The document was synchronized with the RTEMS Software Engineering manual. The specification item section was automatically generated by the Qualification Toolchain using the specification of RTEMS. Mention removal of Doorstop as the requirements management tool. Update SDD proposal to address review comments. |
| 04 | 64677d1d | 2020-10-14 | Replaced | • Complete IEC 61508 analysis<br>• SVR and SPAMR for RTEMS are open issues again, see #89, #128, and #557.<br>• Synchronize with RTEMS Software Engineering manual. |

continues on next page

Table 1 – continued from previous page

| Release | Git Hash | Date | Status | Changes |
|---|---|---|---|---|
| 05 | 7ca702df | 2021-01-07 | Replaced | • Add SVR and SPAMR proposals for RTEMS document, according with the above issues and: #225<br>• Update license according with issue #572 |
| 06 | 5b07f71d | 2021-06-01 | Approved | • Add SCF proposal according to: #67<br>• Address review comments according to: #701<br>• Update, according RID PDR-MF-10: #229<br>• Do not mention use of Doorstop: #477<br>• Synchronize with RTEMS Software Engineering manual.<br>• Remove MIL-STD-1553 Driver from work package list (RID CDR-MV-05): #650 |

| Action | Name | Organization | Signature |
|---|---|---|---|
| Written by | Cláudio Maia | CISTER Research Centre | Assinado por: CLÁUDIO ROBERTO RIBEIRO MAIA<br>Num. de Identificação Civil: BI116110171<br>Data: 2021.06.15 12:15:31 +0100 |
| | Joel Pinto | CISTER Research Centre | |
| | José Valdez | EDISOFT | VALDEZ Jose Digitally signed by VALDEZ Jose Date: 2021.06.14 11:40:17 +01'00' |
| | Sebastian Huber | embedded brains GmbH | Digitally signed by Sebastian Huber Date: 2021.06.15 10:13:55 +02'00' |
| | Ting Peng | embedded brains GmbH | |
| Verified by | Rute Mateus | EDISOFT | MATEUS Rute Digitally signed by MATEUS Rute Date: 2021.06.15 12:49:28 +01'00' |
| Approved by | Nuno Ramos | EDISOFT | RAMOS Nuno-Vicente Digitally signed by RAMOS Nuno-Vicente Date: 2021.06.15 13:13:55 +01'00' |

CHAPTER

ONE

# INTRODUCTION

This technical note contains

- a list of RTEMS SMP items to be qualified,
- an identification of all work items to execute and artefacts to produce,
- a sanity check against other standards,
- an identification of potential re-use items from the EDISOFT RTEMS activities, and
- two compliance matrices of RTEMS to ECSS standards,
- the guideline for requirements management tools.

CHAPTER

TWO

# APPLICABLE AND REFERENCE DOCUMENTS

## 2.1 Applicable Documents

The following is an *applicable document*:

- Technical Note: Space Profile [eb19]

## 2.2 Reference Documents

For reference documents see the *bibliography*.

CHAPTER

# THREE

# TERMS, DEFINITIONS AND ABBREVIATED TERMS

**ABI** Application Binary Interface

**ADP** Acceptance Data Package

**AI** Action Item

**analysis** Verification method which consists of interpretation (or interpolation/extrapolation) of data under defined conditions, or reasoning to show theoretical compliance with specification

**API** Application Programming Interface

**applicable document** This term is defined by ECSS-S-ST-00-01C as a "document that contains provisions which, through reference in the source document, constitute additional provisions of the source document".

**AR** Acceptance Review (for the organization and conduct of reviews see [ECS08c])

**assembler language** A programming language which can be translated very easily into machine code and data. For this project assembler languages are restricted to languages accepted by the *GNU* assembler program for the target architectures.

**BSP** Board Support Package

**C language** See *C11*.

**C11** The standard ISO/IEC 9899:2011.

**CCB** Change Control Board

**CDR** Critical Design Review (for the organization and conduct of reviews see [ECS08c])

**CI** Configuration Item

**CM** Configuration Management

**configurable code** This term is defined by ECSS-E-ST-40C 3.2.5 as "code (source code or executable code) that can be tailored by setting values of parameters". The same section contains also a note which lists some examples for configurable code:

> This definition covers in particular classes of configurable code obtained by the following configuration means:
>
> - configuration based on the use of a compilation directive;
> - configuration based on the use of a link directive;

- configuration performed through a parameter defined in a configuration file;
- configuration performed through data defined in a database with impact on the actually executable parts of the software (e.g. parameters defining branch structures that result in the non-execution of existing parts of the code).

**DDR** Detailed Design Review (for the organization and conduct of reviews see [ECS08c])

**deactivated code** This term is defined by ECSS-E-ST-40C 3.2.8 as "code that, although incorporated through correct design and coding, is intended to execute in certain software product configurations only, or in none of them [adapted from RTCA/DO-178B]".

**dead code** This term is defined by Wikipedia_deadcode as "a section in the source code of a program which is executed but whose result is never used in any other computation.".

**Doorstop** Doorstop is a requirements management tool.

**EARS** Easy Approach to Requirements Syntax

**ECSS** European Cooperation for Space Standardization

**ELF** Executable and Linkable Format

**FCV** Functional Configuration Verification

**feature** This term is used by ECSS-E-ST-40C, however, it is undefined by ECSS. For this project, a feature is defined as a functional requirement.

**FMEA** Failure Modes and Effects Analysis [ECS09e]

**FMECA** Failure Modes, Effects and Criticality Analysis [ECS09e]

**FR** Final Review (for the organization and conduct of reviews see [ECS08c])

**GCC** GNU Compiler Collection

**GNAT** *GNAT* is the *GNU* compiler for Ada, integrated into the *GCC*.

**GNU** GNU's Not Unix!

**GTR** Generic Test Report

**HMI** Human Machine Interface

**HSIA** Hardware/Software Interaction Analysis

**HW** Hardware

**ICD** Interface Control Document

**inspection** this verification method consists in examining the item against the applicable documentation/source code to verify the compliance with requirement(s)

**interrupt service** An *interrupt service* consists of an interrupt service routine which is called with a user provided argument upon reception of an interrupt service request. The routine is invoked in interrupt context. Interrupt service requests may have a priority and an affinity to a set of processors. An *interrupt service* is a *software component*.

**IRD** Software Interface Requirements Document

**ISVV** Independent Software Verification and Validation

**ITP** Interface Test Plan

**ITR** Interface Test Report

**ITT** Invitation To Tender

**JDD** Joint Design Direction

**KO** Kick-Off meeting (for the organization and conduct of reviews see [ECS08c])

**NA**

**N/A** Not Applicable

**NCR** Non-Conformity Report

**NRB** Non-Conformance Review Board

**OAR** Online Applications Research Corporation

**PA** Product Assurance

**PAM** Product Assurance Manager

**PCV** Physical Configuration Verification

**PDR** Preliminary Design Review (for the organization and conduct of reviews see [ECS08c])

**property annotation** A formal annotation in C code defining a logical property required of that code. It can be part of a functional specification or requirement.

**QA** Quality Assurance

**QDP** Qualification Data Package

**QR** Qualification Review (for the organization and conduct of reviews see [ECS08c])

**QT** Qualification Toolchain

**RAMS** Reliability, Availability, Maintainability and Safety

**RB** Requirements Baseline

**ReqIF** Requirements Interchange Format

**RFC** Request For Comments

**RFD** Request For Deviation

**RFW** Request For Waiver

**RID** Review Item Discrepancy

**RSB** RTEMS Source Builder

**RTEMS** Real-Time Executive for Multiprocessor Systems

**SCAR** Software Criticality Analysis Report

**SCC** Software Criticality Category

**SCF** Software Configuration File

**SCM** Software Configuration Management

**SCS** Software Coding Standard

**SDD** Software Design Document

**SDP** Software Development Plan

**SDS** Software Design Standard

**SFMECA** See FMECA

**SFTA** Software Fault Tolerance Analysis

**SFW** Software

**SIS** SPARC/RISCV instruction simulator

**SoC** System on Chip

**SOC** Statement Of Compliance

**software component** This term is defined by ECSS-E-ST-40C 3.2.28 as a "part of a software system". For this project a *software component* shall be any of the following items and nothing else:

- *software unit*
- explicitly defined *ELF* symbol in a *source code* file
- *assembler language* data in a source code file
- *C language* object with static storage duration
- C language object with thread-local storage duration
- *thread*
- *interrupt service*
- collection of *software components* (this is a software architecture element)

Please note that explicitly defined ELF symbols and assembler language data are considered a software component only if they are defined in a *source code* file. For example, this rules out symbols and data generated as side-effects by the toolchain (compiler, assembler, linker) such as jump tables, linker trampolines, exception frame information, etc.

**software item** See *software product*.

**software product** The *software product* is the *RTEMS* real-time operating system.

**software unit** This term is defined by ECSS-E-ST-40C 3.2.24 as a "separately compilable piece of source code". For this project a *software unit* shall be any of the following items and nothing else:

- *assembler language* function in a *source code* file
- *C language* function (external and internal linkage)

A *software unit* is a *software component*.

**source code** This project uses the *source code* definition of the Linux Information Project: "Source code (also referred to as source or code) is the version of software as it is originally written (i.e., typed into a computer) by a human in plain text (i.e., human readable alphanumeric characters)."

**SOW** Statement Of Work

**SPAMR** Software Product Assurance Milestone Report

**SPAP** Software Product Assurance Plan

**SPAR** Software Product Assurance Report

**SPR** Software Problem Report

**SRD** Software Requirements Document

**SRR** System Requirements Review (for the organization and conduct of reviews see [ECS08c])

**SRS** Software Requirements Specification

**SSS** Software System Specification

**SVR** Software Verification Report

**SW**

**S/W** Software

**SW&D** Software Waiver and Deviation

**SwRR** Software Requirements Review (for the organization and conduct of reviews see [ECS08c])

**TBC** To Be Confirmed

**TBD** To Be Defined

**technical specification** The technical specification contains a set of technical requirements. In ECSS-E-ST-40C it consists of the Software Requirements Specification (SRS) defined by Annex D and the Interface Control Document (ICD) defined by Annex E. General requirements for technical requirements specifications are defined by ECSS-E-ST-10-06C.

**test** Verification method which consists in an action by which the operational, functional and performance capabilities of the item are verified when subject to controlled environment conditions (real or simulated) and stimuli.

**thread** This project uses the *thread* definition of Wikipedia_thread: "a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system." A *thread* is a *software component*.

**UID** Unique IDentifier (in contrast to the *UUID* it may be only unique within a certain scope, e.g. a project)

**unreachable code** This term is defined by ECSS-E-ST-40C 3.2.42 as a "code that cannot be executed due to design or coding error".

**UTP** Unit Test Plan

**UTR** Unit Test Report

**UUID** Universally Unique IDentifier

**VR** Verification Report

**VTP** Validation Test Plan

**VTR** Validation Test Report

**YAML** YAML Ain't Markup Language

CHAPTER

FOUR

# ARTEFACTS

## 4.1 Requirements Baseline

### 4.1.1 Software System Specification (SSS)

Not included in the QDP, see *No Requirements Baseline (RB)*.

### 4.1.2 Interface Requirements Document (IRD)

Not included in the QDP, see *No Requirements Baseline (RB)*.

### 4.1.3 Safety and Dependability Analysis Results for Lower Level Suppliers

Not included in the QDP, see *No Requirements Baseline (RB)*.

## 4.2 Technical Specification (TS)

General requirements on a technical requirements specification are defined in ECSS-E-ST-10-06C [ECS09a].

### 4.2.1 Software Requirements Specification (SRS)

The Software Requirements Specification (SRS) is a part of the Technical Specification (TS) and its content is defined by ECSS-E-ST-40C Annex D [ECS09b]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

```
D.2     Expected response
D.2.1   Scope and content
<1>     Introduction
a.      The SRS shall contain a description of the purpose, objective, content and
        the reason prompting its preparation.
```

This will be provided as hand written content in Sphinx format.

```
<2>     Applicable and reference documents
a.      The SRS shall list the applicable and reference documents to support the
        generation of the document.
```

See *Applicable and Reference Documents*.

```
<3>     Terms, definitions and abbreviated terms
a.      The SRS shall include any additional terms, definition or abbreviated
        terms used.
```

See *Terms, Definitions and Abbreviated Terms*.

```
<4>     Software overview
<4.1>   Function and purpose
a.      The SRS shall describe the purpose of the product.
```

This will be provided as hand written content in Sphinx format.

```
<4.2>   Environmental considerations
a.      The SRS shall summarize:
        1.      the physical environment of the target system;
        2.      the hardware environment in the target system;
        3.      the operating environment in the target system;
```

Content will be provided by specialized *Specification Items*.

```
<4.3>   Relation to other systems
a.      The SRS shall describe in detail the product's relationship to other
        systems.
b.      If the product is a component of an integrated HWSW product, then the
        SRS shall:
        1.      summarize the essential characteristics of this larger product;
        2.      list the other HW or SW component the software interfaces with,
                and summarize the computer hardware and peripheral equipment
                to be used.
c.      A block diagram may be presented showing the major components of the
        larger system or project, interconnections, and external interfaces.
```

This will be provided as hand written content in Sphinx format. The content of this section will be abstract. In particular, it will not depend on the QDP configuration.

```
<4.4>   Constraints
a.      The SRS shall describe any items that limit the developer's options for
        building the software.
b.      The SRS should provide background information and seek to justify the
        constraints.
```

Constraints will be maintained as *Specification Items* with a dedicated type. For background information and justification, see also *Justification of Requirements*.

```
<5>     Requirements
<5.1>   General
        NOTE    The following provisions apply to the software
```

(continues on next page)

(continued from previous page)

```
                requirements listed in the SRS, as specified in
                <5.2> to <5.17> below.
c.      Each requirement shall be uniquely identified.
```

See *Identification*.

```
d.      When requirements are expressed as models, the supplier shall establish
        a way to assign identifiers within the model for sake of traceability.
```

See *No Logical and Computational Model*.

```
e.      The traceability information of each requirement derived from higher
        level documentation, to the applicable higher level requirement, shall be
        stated.
        NOTE    The documented trace can be provided
                automatically by tools when models are used to
                express requirements.
```

See *Backward Traceability of Specification Items*.

```
f.      Requirements may be characterized, for example as essential or not, with
        a priority level to prepare incremental delivery, stable or not.
```

This option is not used.

```
<5.2>   Functional requirements
a.      The SRS shall describe the capabilities to be provided by the software
        item under definition.
```

Capabilities will be maintained as *Specification Items* with a dedicated type.

```
b.      The SRS shall provide where applicable the link between the
        requirements and the system states and modes.
```

The goal is to ensure this with EARS, see *Syntax*.

```
c.      Functional requirement shall be grouped by subject, in accordance with
        the logical model organization (e.g. per controlled subsystem).
```

See *Identification*.

```
d.      Each requirement definition should be organized according to the
        following:
        1.      General
        2.      Inputs
        3.      Outputs
        4.      Processing
```

It is proposed to use the EARS, see *Syntax*.

```
e.      The SRS shall describe the functional requirements related to software
        safety and dependability.
```

Safety- and dependability-functions will be maintained as *Specification Items* with a dedicated type.

```
<5.3>   Performance requirements
a.      The SRS shall list any specific requirement to the specified performance
        of software item under definition.
```

See *Resources and Performance*.

```
<5.4>   Interface requirements
a.      The SRS shall list and describe (or reference in the ICD) the software item
        external interfaces.
b.      The following interfaces shall be fully described either in the SRS itself or
        by reference to the ICD:
        1.      interfaces between the software item and other software items;
        2.      interfaces between the software item and hardware products;
        3.      interfaces requirements relating to the manmachine interaction.
```

A reference to the ICD will be made, see *Software Interface Control Document (ICD)*.

```
c.      Naming convention applicable to the data and command interface shall
        be also described.
```

There is no data and command interface.

```
d.      The definition of each interface shall include at least the provided service,
        the description (name, type, dimension), the range and the initial value.
```

See *Software Interface Control Document (ICD)*.

```
<5.5>   Operational requirements
a.      The SRS shall list any specific requirement related to the operation of the
        software in its intended environment.
b.      The information specified in <5.5>a. should include, at least, any specified
        operational mode and mode transition for the software, and, in case of
        manmachine interaction, the intended use scenarios.
```

Operational requirements will be maintained as *Specification Items* with a dedicated type.

```
c.      Diagrams may be used to show the intended operations and related
        modestransitions.
```

Diagrams will not be included in the SRS, they may be included in the SDD.

```
<5.6>   Resources requirements
a.      The SRS shall describe all the resource requirements related to the
        software and the hardware requirements (target hardware on which the
        software is specified to operate), as follows:
        1.      List of the requirements relevant to hardware environment in
                which the software is specified to operate.
        2.      List of the sizing and timing requirements applicable to the
                software item under specification.
        3.      Description of the computer software to be used with the software
                under specification or incorporated into the software item (e.g.
```

(continues on next page)

(continued from previous page)

```
                operating system and software items to be reused).
        4.      Description of the real time constraints to respect (e.g. time
                management with respect to the handling of input data before its
                loss of validity).
```

See *Resources and Performance*.

```
<5.7>   Design requirements and implementation constraints
a.      The SRS shall list any requirements driving the design of the software
        item under specification and any identified implementation constraint.
b.      Requirements applicable to the following items shall be included:
        1.      software standards (e.g. applicable coding standards, and
                development standards);
        2.      design requirements;
        3.      specific design methods to be applied to minimize the number of
                critical software components (see ECSS-Q-ST-80 6.2.2.4);
        4.      requirements relevant to numerical accuracy management;
        5.      design requirements relevant to the “inflight modification” of the
                software item;
        6.      specific design requirements to be applied if the software is
                specified to be designed for intended reuse;
        7.      specific constraints induced by reused software (e.g. COTS, free
                software and open source).
```

Design requirements will be maintained as *Specification Items* with a dedicated type.

```
<5.8>   Security and privacy requirements
a.      The SRS shall describe any security and privacy requirement applicable
        to the software item.
```

There will be no security and privacy requirements, see *Tailoring of ECSS Standards for the QDP*.

```
<5.9>   Portability requirements
a.      The SRS shall list any portability requirement applicable to the software
        item.
```

Portability requirements will be maintained as *Specification Items* with a dedicated type.

```
<5.10>  Software quality requirements
a.      The SRS shall list any quality requirement applicable to the software
        item.
```

Quality requirements will be maintained as *Specification Items* with a dedicated type.

```
<5.11>  Software reliability requirements
a.      The SRS shall list any reliability requirement applicable to the software
        item.
```

Reliability requirements will be maintained as *Specification Items* with a dedicated type.

```
<5.12>  Software maintainability requirements
a.      The SRS shall list any maintainability requirement applicable to the
        software item.
```

Maintainability requirements will be maintained as *Specification Items* with a dedicated type.

```
<5.13>  Software safety requirements
a.      The SRS shall list any safety requirement applicable to the software item.
```

Safety requirements will be maintained as *Specification Items* with a dedicated type.

```
<5.14>  Software configuration and delivery requirements
a.      The SRS shall list any requirement applicable to the selected delivery
        medium and any software configuration applicable to the software item.
```

Application and build configuration requirements will be maintained as *Specification Items* with a dedicated type. The application configuration is a use-case in the software development life cycle. Application configuration requirements are not considered functional requirements.

```
<5.15>  Data definition and database requirements
a.      The SRS shall list any requirement related to specific data format or
        structure to be exchanged with other systems or any database
        requirements allowing to take into account e.g. for a flight software, the
        mission and product specific constraints.
```

There will be no data definition and database requirements, see *Tailoring of ECSS Standards for the QDP*.

```
<5.16>  Human factors related requirements
a.      The SRS shall list any requirement applicable to:
        1.      the personnel and to the specific software product under
                definition;
        2.      manual operations, humanequipment interactions, constraints on
                personnel, concentrated human attention areas and that are
                sensitive to human errors and training, and human factors
                engineering.
```

There will be no human factors related requirements, see *Tailoring of ECSS Standards for the QDP*.

```
<5.17>  Adaptation and installation requirements
a.      This SRS shall list any requirement applicable to adaptation data and to
        specific installation.
```

There will be adaptation and installation requirements, see *Tailoring of ECSS Standards for the QDP*.

```
<6>     Validation requirements
a.      The SRS shall describe, per each uniquely identified requirement in <5>,
        the validation approach.
```

Each validation approach is stored in validation specification items, see *Requirement Validation*. The validation method is indicated by the specification item type. The validation specification items are linked to the corresponding requirement. A report can be generated automatically, showing the validation method for each specification item.

```
b.      A validation matrix (requirements to validation approach correlation
        table) shall be utilized to describe the validation approach applicable to
        each requirement.
```

The validation matrix can be automatically generated from the *Specification Items*.

```
<7>     Traceability
a.      The SRS shall report the traceability matrices
        1.      from the upper level specification requirements to the
                requirements contained in <5> (forward traceability table), and
        2.      from the requirements contained in <5> to the upper level
                applicable specification (backward traceability table).
```

See *Traceability of Specification Items*.

```
b.      In case the information in <7>a. is separately provided in the DJF,
        reference to this documentation shall be clearly stated.
```

The information is included in the SRS.

```
<8>     Logical model description
a.      The SRS shall include a topdown description of the logical model of the
        software.
        NOTE 1  The logical model can be the result of an
                iterative verification process with the customer.
                It also supports the requirements capture,
                documents and formalizes the software
                requirements.
        NOTE 2  A logical model is a representation of the
                technical specification, independent of the
                implementation, describing the functional
                behaviour of the software product. The logical
                model is written with a formalized language
                and it can be possibly executable. Formal
                methods can be used to prove properties of the
                logical model itself and therefore of the
                technical specification. The logical model
                allows in particular to verify that a technical
                specification is complete (i.e. by checking a
                software requirement exists for each logical
                model element), and consistent (because of the
                model checking).
                The logical model can be completed by specific
                feasibility analyses such as benchmarks, in
                order to check the technical budgets (e.g.
                memory size and computer throughput). In
                case the modelling technique allows for it,
                preliminary automatic code generation can be
                used to define the contents of the software
                validation test specification.
        NOTE 3  If software system co-engineering activities are
                considered, the logical model is a refinement of
                the following system models: data, application
                function, event and failure
```

(continues on next page)

(continued from previous page)

```
b.      The method used to express the logical model shall be described
c.      Diagrams, tables, data flows and explanatory text may be included.
d.      The functionality at each level should be described, to enable the reader
        to 'walkthrough' e.g. the model levelbylevel, functionbyfunction,
        and flowbyflow.
e.      The behavioural view of the software logical model shall be also
        described in the SRS.
        NOTE    This is particularly relevant for flight software
                applications.
```

This is not included, see *No Logical and Computational Model*.

### 4.2.2 Software Interface Control Document (ICD)

The Software Interface Control Document (ICD) is a part of the Technical Specification (TS) and its content is defined by ECSS-E-ST-40C Annex E [ECS09b]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

```
E.2     Expected response
E.2.1   Scope and content
<1>     Introduction
a.      The ICD shall contain a description of the purpose, objective, content and
        the reason prompting its preparation.
```

This will be provided as hand written content in Sphinx format.

```
<2>     Applicable and reference documents
a.      The ICD shall list the applicable and reference documents to support the
        generation of the document.
```

See *Applicable and Reference Documents*.

```
<3>     Terms, definitions and abbreviated terms
a.      The ICD shall include any additional terms, definition or abbreviated
        terms used.
```

See *Terms, Definitions and Abbreviated Terms*.

```
<4>     Software overview
a.      The ICD may reference the software overview done in the SRS.
```

A reference to SRS will be made, see *Software Requirements Specification (SRS)*.

```
<5>     Requirements and design
<5.1>   General provisions to the requirements in the IRD
a.      Each requirement shall be uniquely identified.
```

See *Identification*.

```
b.      When requirements are expressed as models, the supplier shall establish
        a way to assign identifiers within the model for sake of traceability.
```

See *No Logical and Computational Model*.

```
c.      The traceability information of each requirement derived from higher
        level documentation, to the applicable higher level requirement, shall be
        stated.
        NOTE    The documented trace can be provided
                automatically by tools when models are used to
                express requirements.
```

See *Backward Traceability of Specification Items*.

```
<5.2>   Interface requirements
a.      In case the requirements of the IRD need to be further detailed, the ICD
        shall list and describe the software item external interfaces.
b.      The following interfaces shall be fully described:
        1.      interfaces between the software item and other software items;
        2.      interfaces between the software item and hardware products;
        3.      interfaces requirements relating to the man-machine interaction.
        4.      This can be also information about e.g. :
                o       detailed requirements on database structure
                o       logical interface architecture
                o       requirements on signal
                o       communication protocols
                o       timing requirements
                o       required behaviour in case of error
                o       telecommands (e.g. PUS selection, words contents)
                o       observable data
                o       telemetry
```

The interface requirements will be provided through *Non-Functional Requirement Item Type* specification items. It is not clear what interface requirements will be and how they differ from the interface design in RTEMS.

```
<5.3>   Interface design
a.      The ICD shall describe the external interfaces design of the software item
b.      The external interface may be expressed by models.
c.      The following interfaces shall be fully described:
        1.      interfaces between the software item and other software items;
        2.      interfaces between the software item and hardware products;
        3.      interfaces requirements relating to the man-machine interaction.
        4.      This can be also information about e.g.:
                o       Physical interface architecture
                o       Complete TM/TC plan
                o       Design of all commands and telemetry stream
                o       Protocol detailed implementation
                o       specific design requirements to be applied if the software is
                        specified to be designed for intended reuse
d.      The definition of each interface shall include at least the provided service,
        the description (name, type, dimension), the range and the initial value.
e.      For each interface external to the software , this can be e.g. organized as
        follows:
```

(continues on next page)

(continued from previous page)

```
        -       Data item (Name , description, unique identifier, description,
                source, destination, unit of measure, limit/range, accuracy,
                precision, frequency, rate, legality checks, data type, data
                representation)
        -       Message item
        -       Communication protocol (by reference to the applicable
                documents)
```

The interface design will be provided through *Interface Item Type* specification items.

```
<6>     Validation requirements
a.      The ICD shall describe, per each uniquely identified requirement in <5>,
        the validation approach.
```

Each validation approach is stored in validation specification items, see *Requirement Validation*. The validation method is indicated by the specification item type. The validation specification items are linked to the corresponding requirement. A report can be generated automatically, showing the validation method for each specification item.

```
b.      A validation matrix (requirements to validation approach correlation
        table) shall be utilized to describe the validation approach applicable to
        each requirement.
```

The validation matrix can be automatically generated from the *Specification Items*.

```
<7>     Traceability
a.      The ICD shall report the traceability matrices
        1.      from the upper level specification requirements to the
                requirements contained in <5> (forward traceability table), and
        2.      from the requirements contained in <5> to the upper level
                applicable specification (backward traceability table).
```

See *Traceability of Specification Items*.

```
b.      In case the information in <7>a.1. is separately provided in the DJF,
        reference to this documentation shall be clearly stated.
```

The information is included in the ICD.

## 4.3 Design Definition File (DDF)

### 4.3.1 Software Design Document (SDD)

The Software Design Document (SDD) is a part of the Design Definition File (DDF) and its content is defined by ECSS-E-ST-40C Annex F [ECS09b]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

The entire SDD will be produced by Doxygen. This means that the section numbering will not be in line with the standard. For each scope and content demand of the standard a mapping to Doxygen features will be presented in this section.

```
F.2     Expected response
F.2.1   Scope and content
<1>     Introduction
a.      The SDD shall contain a description of the purpose, objective, content
        and the reason prompting its preparation.
```

The introduction will be provided by the main page (`@mainpage`).

```
<2>     Applicable and reference documents
a.      The SDD shall list the applicable and reference documents to support the
        generation of the document.
```

The applicable and reference documents are generated by `@cite` special commands using the RTEMS Project bibliography.

```
<3>     Terms, definitions and abbreviated terms
a.      The SDD shall include any additional terms, definition or abbreviated
        terms used.
```

There is no out of the box support for a glossary in Doxygen. The workaround proposed in Doxygen issue #1808 will be used. See also Doxygen ALIASES configuration option.

The glossary of terms is contained in the specification through *Glossary Term Item Type* items. From this we could generate the glossary sections described in the Doxygen issue. In the normal Doxygen markup we could use an `@glossary{abc}` to reference glossary terms. Alternatively, we can generate links to the project-wide glossary in the *RTEMS Classic API Guide*.

```
<4>     Software design overview
        NOTE    The SDD briefly introduces the system context
                and design and discuss the background to the
                project detailed as follows.
<4.1>   Software static architecture
a.      The SDD shall describe the architecture of the software item, as well as
        the main relationship with the major components identified.
```

The software architecture will be described via Doxygen groups which contain software components.

```
b.      The SDD shall also describe any system state or mode in which the
        software operates.
```

The system states will be described in the System State Handler.

```
c.      The SDD shall describe the separated mission and configuration data.
        NOTE    Data can be classified in the following
                categories:
                - data resulting from the mission analysis and
                  which thus vary from one mission to
                  another;
                - reference data which are specific to a family
                  of software product;
                - reference data which never change;
                - data depending only on the specific mission
```

(continues on next page)

(continued from previous page)

```
                requirements (e.g. calibration of sensors);
              - data required for the software operation
                which only vary the higher level system
                design (in which is embedded the software)
                is changed;
```

There is no mission data. The configuration data will be described in the RTEMS Classic API Guide [RTEa].

```
<4.2>   Software dynamic architecture
a.      The SDD shall describe the design choices to cope with the real time
        constraints (e.g. selection and description of the computational model).
```

See *No Logical and Computational Model*. There are no plans to provide content for this section.

```
<4.3>   Software behaviour
```

The software behaviour is described for each software unit in Doxygen markup.

```
<4.4>   Interfaces context
a.      The SDD shall identify all the external interfaces or refer to the ICD.
```

The application programming interface will be contained in an API top level Doxygen group. The goal is to generate the corresponding header files from interface design items, see *Specification Items*. Other interface will be contained in Doxygen groups, e.g. CPU port interface.

```
b.      The description in <4.4>a. should be based on system block diagram or
        context diagram to illustrate the relationship between this system and
        other systems.
```

This information will be already provided by the ICD.

```
<4.5>   Long lifetime software
a.      The SDD shall describe the design choices to cope with the long planned
        lifetime of the software, in particular minimum dependency on the
        operating system and the hardware to improve portability.
```

Content provided by @page special command. Referenced by main page. Alternatively, reference to RTEMS Software Engineering manual [RTEb].

```
<4.6>   Memory and CPU budget
a.      The SDD shall document and summarize the allocation of memory and
        processing time to the software components.
```

There is one SDD for all RTEMS target architectures, platforms, and build configurations. The data structures and algorithmic complexity will be documented at software component level if necessary. However, the size of particular data structures is platform-dependent. So, this information will be included in the user manual. It will present the information so that application designers can create this section in their SDD. The goal is to get the resource usage and performance characteristics through the test reports. Using the test reports we can generate the documentation for a specific platform. See also *Resources and Performance*.

```
<4.7>   Design standards, conventions and procedures
a.      The SDD shall summarize (or reference in the SDP) the software methods
        adopted for the architectural and the detailed design.
        NOTE    A design method offers often the following
                characteristics:
                - decomposition of the software architecture
                  in design objects having integral parts that
                  communicate with each other and with the
                  outside environment
                - explicit recognition of typical activities of
                  real-time systems (i.e. cyclic and sporadic
                  threads, protected resources)
                - integration of appropriate scheduling
                  paradigms with the design process
                - explicit definition of the application timing
                  requirements for each activity
                - static verification of processor allocation,
                  schedulability and timing analysis
                - consistent code generation
```

A reference to RTEMS Software Engineering manual [RTEb] will be made. In a normal ECSS project, there would be a reference to the SDP, however, the SDP of this activity is not integrated in the RTEMS Project documentation set. Everything which should be integrated in the RTEMS Project (this includes the Doxygen markup) should only reference documents in the RTEMS Project eco-system. The RTEMS Software Engineering manual is basically a container for our activity which can hold content which is normally present in the SDP. The SDP should then reference the RTEMS Software Engineering manual. For an RTEMS Project integration and the long-term maintenance of our work it is important that everything gets documented in the standard documentation set of the RTEMS Project.

```
b.      The following information shall be summarized:
        1.      software architectural design method;
        2.      software detailed design method;
        3.      code documentation standards;
        4.      naming conventions;
        5.      programming standards;
        6.      intended list of reuse components
        7.      main design trade-off.
```

No summary will be provided, there will be references to the RTEMS Software Engineering manual provided by `@page` special command [RTEb]. Referenced by main page.

```
<5>     Software design
<5.1>   General
a.      The SDD shall describe the software architectural design.
```

The software architecture will be described via Doxygen groups.

```
b.      The architecture structure of the software item shall be described,
        identifying the software components, their hierarchical relationships, any
        dependency and interfaces between them.
```

This is implemented via the use of standard Doxygen features.

```
c.      For flight software, the design shall reflect in flight modification
        requirements.
```

There will be no in flight modifications.

```
d.      The structure in <5.2> to <5.5> should be used.
```

A modified structure will be used, see below.

```
<5.2>   Overall architecture
a.      The SDD shall describe the software architectural design, from a static
        point of view and also, when the software to be developed has real time
        constraints, from a dynamic point of view, and from a behaviour point of
        view.
```

The static point of view is documented via Doxygen groups. No dynamic and behaviour point of view will be documented, see *No Logical and Computational Model*.

```
b.      The software static architecture shall be summarized describing its
        components.
```

Each Doxygen group will have an `@brief` and a detailed description. The software components of a group are presented by Doxygen automatically.

```
c.      For real-time software, the software dynamic architecture shall be
        summarized describing its selected computational model.
        NOTE    An analysable computational model generally
                consists in defining:
                - the types of components (objects)
                  participating to the real-time behaviour,
                  from which the system is constructed (e.g.
                  active-periodic, active-sporadic, protected,
                  passive, actors, process, blocks, drivers)
                - the scheduling type (e.g. sequential or
                  multithreaded), the scheduling model (e.g.
                  cyclic or pre-emptive, fixed or dynamic
                  priority based), and the analytical model
                  (e.g. Rate Monotonic Scheduling, Deadline
                  Monotonic Scheduling, Earliest Deadline
                  First), under which the system is executed
                  and its associated mechanisms
                - the means of communication between
                  components/objects (e.g. mailboxes, entry
                  parameters)
                - the means of synchronization between
                  components or objects (e.g. mutual
                  exclusion, protected object entries, basic
                  semaphores)
                - If applicable , the means of distribution and
                  internode communication (e.g. virtual
                  nodes, Remote Procedure Call)
                  and (optional for non flight software):
                - the means of providing timing facilities (e.g.
```

(continues on next page)

(continued from previous page)

```
                  real clock, with or without interrupt,
                  multiple interrupting count-down, relative
                  or absolute delays, timers time-out)
                - the means of providing asynchronous
                  transfer of control (e.g. watchdog to transfer
                  control from anywhere to the reset
                  sequence, software service of the underlying
                  run-time system to cause transfer of control
                  within the local scope of the thread)
d.      The description in <5.2>c. should consist in the following information:
        1.      type of components participating to the real time behaviour,
        2.      scheduling type (e.g. single or multi-threads),
        3.      scheduling model (e.g. pre-emptive or not, fixed or dynamic
                priority based),
        4.      analytical model (e.g. rate monotonic scheduling, deadline
                monotonic scheduling),
        5.      Tasks identification and priorities,
        6.      Means of communication and synchronization,
        7.      Time management.
```

No dynamic architecture and computational model will be documented, see *No Logical and Computational Model*.

```
e.      The software behaviour shall be described e.g. with automata or
        scenarios.
```

The software behaviour will only be described at software component level if necessary. Automata and scenarios may be visualized with `@dot` special commands, `@msc` special commands, or PlantUML diagrams.

```
f.      The software static, dynamic and behavioural architecture shall be
        described in accordance with the selected design method.
```

Doxygen is thoroughly used by the RTEMS Project for more than ten years to describe the software. It is evident that Doxygen is a proven in use tool and in line with the design method of the RTEMS Project.

```
g.      The SDD shall describe the error handling and fault tolerance principles
        (e.g. error detection, reporting, logging, and fault containment regions.)
```

Content provided by `@page` special command. Referenced by main page. Since this information is also relevent for the RTEMS users, it should be included in the RTEMS Classic API Guide [RTEa]. If we add the information to this guide, then a reference will be made.

```
<5.3>   Software components design - General
a.      The SDD shall describe:
        1.      The software components, constituting the software item.
        2.      The relationship between the software components.
        3.      The purpose of each software component.
        4.      For each software component, the development type (e.g. new
                development, software to be reused).
        5.      If the software is written for the reuse,
```

(continues on next page)

(continued from previous page)

```
                - its provided functionality from an external point of view,
                  and
                - its external interfaces.
        6.      Handling of existing reused components.
        NOTE    See Annex N.
```

Implemented via the use of standard Doxygen features.

```
b.      The following shall apply to the software components specified in
        <5.3>a.1.:
        1.      Each software component is uniquely identified.
```

Software components are identified by domain-specific designators, e.g. Doxygen group name, function name, symbol name, variable name. No separate design-specific identifier is provided. The identifiers are unique if they follow the naming convention.

```
2.      When components are expressed as models, the supplier
        establishes a way to assign identifiers within the model for sake of
        traceability.
```

Models are not used, see *No Logical and Computational Model*.

```
3.      The software requirements allocation provides for each software
        component;
NOTE    The documented trace can be provided
        automatically by tools when models are used to
        express components.
```

See *Traceability between Software Requirements, Architecture and Design*.

```
c.      The description of the components should be laid out hierarchically, in
        accordance with the following aspects for each component, further
        described in <5.4>:
        - <Component identifier>
        - <Type>
        - <Purpose>
        - <Function>
        - <Subordinates>
        - <Dependencies>
        - <Interfaces>
        - <Resources>
        - <References>
        - <Data>
        NOTE    Detailed description of the aspects for each
                component are describe in <5.4>.
```

Implemented via the use of standard Doxygen features.

```
<5.4>   Software components design - Aspects of each component
<5.4.1> General
a.      This part of the DRD, as well as <5.5>, may be produced as the detailed
        design model of a tool, if agreed with the customer.
```

Doxygen is the proposed tool.

```
<5.4.2> <Component identifier>
a.      Each component should have a unique identifier.
```

See *<5.3>b.1*.

```
b.      The component should be named according to the rules of the
        programming language or operating system to be used.
```

Yes, this is obvious.

```
c.      A hierarchical naming scheme should be used that identifies the parent
        of the component (e.g. ParentName_ChildName).
```

Yes, the RTEMS coding style guide has a naming scheme.

```
<5.4.3> <Type>
a.      Component type should be described by stating its logical and physical
        characteristics.
b.      The logical characteristics should be described by stating the package,
        library or class that the component belongs to.
c.      The physical characteristics should be described by stating the type of
        component, using the implementation terminology (e.g. task, subroutine,
        subprogram, package and file).
        NOTE    The contents of some components description
                clauses depend on the component type. For the
                purpose of this guide, the following categories
                are used: executable (i.e. contains computer
                instructions) or non-executable (i.e. contains
                only data).
```

Implemented via the use of standard Doxygen features.

```
<5.4.4> <Purpose>
a.      The purpose of a component should describe its trace to the software
        requirements that it implements.
        NOTE    Backward traceability depends upon each
                component description explicitly referencing
                the requirements that justify its existence.
```

See *Traceability between Software Requirements, Architecture and Design*.

```
<5.4.5> <Function>
a.      The function of a component shall be described in the software
        architectural design.
b.      The description specified in <5.4.5>a. should be done by stating what the
        component does.
        NOTE 1  The function description depends upon the
                component type. Therefore, it can be a
                description of the process.
        NOTE 2  Process descriptions can use such techniques as
                structured English, precondition-postcondition
                specifications and state-transition diagrams.
```

Yes.

```
<5.4.6> <Subordinates>
a.      The subordinates of a component should be described by listing the
        immediate children.
        NOTE 1  The subordinates of a unit are the units that are
                "called by" it. The subordinates of a database
                can be the files that "compose" it.
        NOTE 2  The subordinates of an object are the objects
                that are "used by" it.
```

Implemented via the use of standard Doxygen features. Call and caller graphs can be enabled via the CALL_GRAPH and CALLER_GRAPH Doxygen configuration options. Please note that the call and caller graphs generated by Doxygen do not cover static functions.

```
<5.4.7> <Dependencies>
a.      The dependencies of a component should be described by listing the
        constraints upon its use by other components.
        NOTE    Examples are:
                - Operations to take place before this
                  component is called,
                - Operations that are excluded when this
                  operation takes place.
```

Yes, but only if necessary and not for all components.

```
<5.4.8> <Interfaces>
a.      Both control flow and data flow aspects of an interface shall be described
        for each "executable" component.
b.      Data aspects of "non executable" components should be described.
c.      The control flow to and from a component should be described in terms
        of how to start (e.g. subroutine call) and terminate (e.g. return) the
        execution of the component.
d.      If the information in <5.4.8>c. is implicit in the definition of the type of
        component, a description need not be done.
e.      If control flows take place during execution (e.g. interrupt), they should
        be described.
f.      The data flow input to and output from each component shall be
        described.
g.      It should be ensured that data structures:
        1.      are associated with the control flow (e.g. call argument list);
        2.      interface components through common data areas and files.
```

Yes.

```
<5.4.9> <Resources>
a.      The resources' needs of a component should be described by itemising
        what the component needs from its environment to perform its function.
        NOTE 1  Items that are part of the component interface
                are excluded.
        NOTE 2  Examples of resources' needs of a component
                are displays, printers and buffers.
```

Yes, but only if necessary and not for all components.

```
<5.4.10> <References>
a.      Explicit references should be inserted where a component description
        uses or implies material from another document.
```

Yes.

```
<5.4.11> <Data>
a.      The data internal to a component should be described.
        NOTE    The amount of details to be provided depends
                strongly on the type of the component.
b.      The data structures internal to a program or subroutine should also be
        described.
c.      Data structure definitions shall include the:
        1.      description of each element (e.g. name, type, dimension);
        2.      relationships between the elements (i.e. the structure);
        3.      range of possible values of each element;
        4.      initial values of each element.
```

Yes.

```
<5.5>   Internal interface design
a.      The SDD shall describe the internal interfaces among the identified
        software components.
b.      The interface data specified in a., by component, shall be organized
        showing the complete interfaces map, using as appropriate diagrams or
        matrices supporting their cross-checking.
c.      For each identified internal interface, all the defined data elements shall
        be included.
        NOTE    The amount of detail to be provided depends
                strongly on the type of component.
d.      The logical and physical data structure of files that interface major
        component should be postponed to the detailed design.
e.      Data structure definitions shall include:
        1.      the description of each element (e.g. name, type, dimension);
        2.      the relationships between the elements (i.e. the structure);
        3.      the initial values of each element.
```

Implemented via the use of standard Doxygen features.

```
<6>     Requirements to design components traceability
a.      The SDD shall provide traceability matrices
        1.      from the software requirements to component down to the lower
                identified component in the software hierarchy (forward
                traceability) and
        2.      from the software components to its upper level component up to
                the software requirements (backward traceability).
b.      In case the information in <6>a. is provided as separate documentation in
        the DJF, a reference to it shall be stated.
```

See *Traceability between Software Requirements, Architecture and Design*.

```
c.      The SDD shall define the potential specific measures taken for critical
        software in the design documentation.
```

There will be no specific measures taken for critical software.

### 4.3.2 Software Configuration File (SCF)

The Software Configuration File (SCF) is a part of the Design Definition File (DDF) and its content is defined by ECSS-M-ST-40C Rev.1 Annex E [ECS09d]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

Since this document contains information about the configuration of the QDP and contains hash values of the QDP and its content, it shall be placed outside the QDP archive. It is the first document a user of the QDP is supposed to read and can be used to verify the integrity of the QDP archive.

```
E.2     Expected response
E.2.1   Scope and content
<1>     Introduction
a.      The introduction shall describe the purpose and
        objective of the SCF.
```

This will be provided as hand written content in Sphinx format.

```
<2>     Applicable and reference documents
a.      The SCF shall list the applicable and reference documents to
        support the generation of the document.
```

See *Applicable and Reference Documents*.

```
<3>     Terms, definitions and abbreviated terms
a.      The SCF shall include any additional terms,
        definition or abbreviated terms used.
```

See *Terms, Definitions and Abbreviated Terms*.

```
<4>     Software configuration item overview
a.      The SCF shall contain a brief description of
        the software configuration item.
```

This will be provided as hand written content in Sphinx format.

```
b.      For the software configuration item, the following
        information shall be provided:
        1.      how to get information about the software
                configuration item;
        2.      composition of the software configuration item:
                code, documents;
        3.      means to develop, modify, install, run the software
                configuration item;
        4.      differences from the reference or previous version;
        5.      status of software problem reports, software change
                requests, and software waivers and deviations related
                to the software configuration item.
```

This will be provided as hand written content in Sphinx format.

```
<5>     Inventory of materials
a.      The SCF shall list all physical media and associated
        documentation released with the software configuration item.
NOTE    Example of physical media are listings, tapes, cards and disks.
b.      The SCF shall define the instructions necessary to get information
        included in physical media.
NOTE    For example, to get files.
```

This will be provided as hand written content in Sphinx format. Although there is no physical media for the QDP, this section should contain the reference to the ESA website which will contain the QDPs and the Qualification Toolchain, which will allow generation of customized QDPs.

```
<6>     Baseline documents
a.      The SCF shall identify all the documents applicable to the
        delivered software configuration item version.
```

This will be provided as hand written content in Sphinx format. Since the QDP contains all project-specific documents, references to sections 4.1, 4.2, and 4.3 will be made.

```
<7>     Inventory of software configuration item
a.      The SCF shall describe the content of the software
        configuration item.
b.      The SCF shall list all files constituting the software
        configuration item:
        1.      source codes with name, version, description;
        2.      binary codes with name, version, description;
        3.      associated data files necessary to run the software;
        4.      media labelling references;
        5.      checksum values;
        6.      identification and protection method and tool description.
```

This will be provided as hand written content in Sphinx format. The QDP archive will contain an automatically generated Python script (for example `verify_package.py`) which can be used to verify the integrity of the unpacked QDP files. The script may be used to get a list of the files of the QDP.

```
<8>     Means necessary for the software configuration item
a.      The SCF shall describe all items (i.e. hardware and software)
        that are not part of the software configuration item, and
        which are necessary to develop, modify, generate and run the
        software configuration item, including:
        1.      items related to software development;
        NOTE    For example, compiler name and version,
        linker, and libraries.
        2.      build files and software generation process;
        3.      other software configuration items.
```

This will be provided as hand written content in Sphinx format. This section should reference to the QDP UM *Software User Manual (SUM)*, which should contain a section with the necessary software to use the QDP.

```
<9>     Installation instructions
a.      The SCF shall describe how to install the software
        configuration item version, its means and procedures
        necessary to install the product and to verify its installation.
```

This will be provided as hand written content in Sphinx format. This section should reference to the QDP UM *Software User Manual (SUM)*, which should contain a section with the instructions on how to install and use the QDP.

```
<10>    Change list
a.      This SCF shall contain the list of all changes incorporated into
        the software configuration item version, with a cross reference
        to the affected software configuration item document, if any.
b.      Changes not incorporated yet but affecting the S/W CI shall
        also be listed and include.
        1.      software problem reports;
        2.      software change requests and proposals;
        3.      contractual change notices;
        4.      software waivers and deviations.
```

This will be provided as hand written content in Sphinx format. There will be one subsection for each QDP version. For item b., simply a reference to section 12 will be made.

```
<11>    Auxiliary information
a.      The SCF shall include any auxiliary information to describe
        the software configuration.
```

This will be provided as hand written content in Sphinx format. This section will contain auxiliary information about the QDP configuration (if applicable).

```
<12>    Possible problems and known errors
a.      The SCF shall identify any possible problems or known errors
        with the software configuration item version and any steps
        being taken to resolve the problems or errors.
```

This will be provided as hand written content in Sphinx format. This section should reference to the SRelD *Software Release Document (SRelD)*, which will contain the list of SPRs and NCRs.

### 4.3.3 Software Release Document (SRelD)

The Software Release Document (SRelD) is a part of the Design Definition File (DDF) and its content is defined by ECSS-E-ST-40C Annex G [ECS09b]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

```
G.2     Expected response
G.2.1   Scope and content
<1>     Introduction
a.      The SRelD shall contain a description of the purpose, objective, content
        and the reason prompting its preparation.
```

This will be provided as hand written content in Sphinx format.

```
<2>     Applicable and reference documents
a.      The SRelD shall list the applicable and reference documents to support
        the generation of the document.
```

See *Applicable and Reference Documents*.

```
<3>     Terms, definitions and abbreviated terms
a.      The SRelD shall include any additional terms, definition or abbreviated
        terms used.
```

See *Terms, Definitions and Abbreviated Terms*.

```
<4>     Software release overview
a.      The SRelD shall contain a brief description of the information to be
        associated with a software release, including:
        1.      reference of the corresponding SCF,
        2.      version of the delivered software configuration item,
        3.      status of SPRs, SCRs and SW&D related to the software
                configuration item, and
        4.      advice for use of the software configuration item.
        NOTE    The software release document is a subset of
                the software configuration file that describes a
                new version by comparison with "reference" or
                the previous one. It is used for the delivery of a
                new version of a software configuration item to
                a customer.
```

This will be provided as hand written content in Sphinx format.

```
<5>     Status of the software configuration item
<5.1>   Evolution since previous version
a.      The SRelD shall
        1.      summarize the main information on the software configuration
                item, and
        2.      describe the changes implemented since previous version.
```

This will be provided as hand written content in Sphinx format.

```
<5.2>   Known problems or limitations
a.      The SRelD shall list all the unsolved SPR and approved SW&D related to
        the version of the software configuration item.
```

The list of unsolved SPRs and approved SW&D should be automatically generated.

```
<6>     Advice for use of the software configuration item
a.      The SRelD shall provide advice for the use of this version of the software
        configuration item.
        NOTE    For example: Potential problems, and
                compatibility with other configuration items).
```

This will be provided as hand written content in Sphinx format.

```
<7>     On-going changes
a.      The SRelD shall provide information on planned evolution of the
        software configuration item.
```

This will be provided as hand written content in Sphinx format.

### 4.3.4 Software User Manual (SUM)

The Software User Manual (SUM) is a part of the Design Definition File (DDF) and its content is defined by ECSS-E-ST-40C Annex H [ECS09b]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP. It is proposed to use the RTEMS User Manual as is for the SUM of the QDP. The *Software Configuration File (SCF)* will address QDP-specific use cases.

```
H.2     Expected response
H.2.1   Scope and content
<1>     Introduction
a.      The SUM shall contain a description of the purpose, objective, content
        and the reason prompting its preparation.
```

The RTEMS User Manual has an *Introduction* chapter.

```
<2>     Applicable and reference documents
a.      The SUM shall list the applicable and reference documents to support the
        generation of the document.
```

The RTEMS User Manual has no applicable and reference documents.

```
<3>     Terms, definitions and abbreviated terms
a.      The SUM shall include any additional terms, definition or abbreviated
        terms used.
```

The RTEMS User Manual has a *Glossary* chapter.

```
<4>     Conventions
a.      The SUM shall summarise symbols, stylistics conventions, and command
        syntax conventions used in the document.
        NOTE    An example of stylistic conventions is using
                boldface and courier font to distinguish user
                input. Examples of syntax conventions are the
                rules for combining commands, keywords and
                parameters.
```

The RTEMS Project documentation has currently no section to summarise symbols, stylistics conventions, and command syntax conventions used in the documents.

```
<5>     Purpose of the Software
a.      The SUM shall include a description of the intended uses of the software,
        in terms of capabilities, operating improvements, and benefits expected
        from its use.
```

The purpose of the software is described in the *Features* section of the *Introduction* chapter of the RTEMS User Manual.

```
<6>     External view of the software
a.      The SUM shall identify the software files, including databases and data
        files, which are necessary for the software to operate, including security
        and privacy considerations for each file and identification of the software
        necessary to continue or resume operation in case of an emergency.
[...]
<11.3>  Using the software on a typical task
a.      The SUM shall describe a typical use case of the software, using graphical
        pictures and diagrams to demonstrate the actions performed by the user.
```

The RTEMS User Manual does not follow the Annex H scope and content descriptions for sections *<6>* to *<11>* since they are inappropriate for an operating system library which should be used to develop applications. They target full software products which can be used in missions, e.g. with an operational phase. The RTEMS User Manual focuses on the installation and use of RTEMS and its ecosystem in general.

```
<12>    Analytical Index
a.      The SUM, if more than 40 pages, shall include an index containing a
        systematic list of topics from the user's point of view, the major
        synonyms and variants (especially if these are well known to users but
        are not employed in the operational manual for technical reasons),
        pointing to topics in the body of the manual by:
        - page number,
        - section number,
        - illustration number,
        - primary index entry (one level of reference only).
        NOTE    Index entries usefully contain auxiliary
                information, especially cross-references to
                contrasting or related terms. For example, the
                entry for INSERT says 'see also DELETE'.
                Indexes are made particularly helpful if
                attention is drawn primarily to important
                keywords, and to important locations in the
                body of the manual. This is achieved by
                highlighting such entries, and by grouping
                minor entries under major headings. Indexes
                do not contain more than two levels of entry. If
                a single index points to different kinds of
                location, such as pages and illustration
                numbers,        these    are     unambiguously
                distinguished, (e.g. Page 35, Figure 7), since the
                use of highlighting (35, 7) is not for instance
                sufficient to prevent confusion in this case.
```

The RTEMS User Manual has a document index.

### 4.3.5 Software Source Code and Media Labels

The source code in the QDP will consist of:

- Git clones of RTEMS Project repositories (e.g. rtems, rtems-docs, rtems-source-builder) maybe with additional project-specific commits
- tool chain sources downloaded by the *RSB*
- build and configuration files

No media labels will be provided. Due to the inclusion of full Git clones (Git repositories) the user of the QDP has full access to the project history. This allows an efficient integration of source code into the Git work flow or any other version control system of the user.

### 4.3.6 Software Product and Media Labels

The software product in the QDP will consist of:

- an installation of the RTEMS *BSP* in the configuration defined by the QDP variant
- an installed tool chain built by the *RSB* of the QDP on a 64-bit Debian 10 machine with only standard packages installed

No media labels will be provided.

### 4.3.7 Training Material

No training material will be provided by the QDP. There are training options available for users of the QDP from embedded brains.

## 4.4 Design Justification File (DJF)

### 4.4.1 Software Verification Plan (SVerP)

The SVerP will be included in the SDP, see [EDI19c].

### 4.4.2 Software Validation Plan (SValP)

The Software Validation Plan is included in the SDP, see [EDI19c].

### 4.4.3 Independent Software Verification & Validation Plan

Not included in the QDP, see *No Independent Software Verification and Validation*.

### 4.4.4 Software Unit and Integration Test Plan (SUITP)

The Software Unit and Integration Test Plan is a part of the Design Justification File (DJF) and its content is defined by ECSS-E-ST-40C Annex K [ECS09b]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

There will be a combined plan for unit and integration tests, see *Combined Unit and Integration Testing*. This is a deviation from ECSS-E-ST-40C.

```
K.2     Expected response
K.2.1   Scope and content
<1>     Introduction
a.      The SUITP shall contain a description of the purpose, objective, content
        and the reason prompting its preparation.
```

This will be provided as hand written content in Sphinx format.

```
<2>     Applicable and reference documents
a.      The SUITP shall list the applicable and reference documents to support
        the generation of the document.
```

See *Applicable and Reference Documents*.

```
<3>     Terms, definitions and abbreviated terms
a.      The SUITP shall include any additional terms, definition or abbreviated
        terms used.
```

See *Terms, Definitions and Abbreviated Terms*.

```
<4>     Software overview
a.      The SUITP shall contain a brief description of the software under test and
        its context: a summary of its functionality, its configuration, its
        operational environment and its external interfaces.
        NOTE    Reference to technical documentation can be
                done.
```

References to the *Software Requirements Specification (SRS)* and *Software Interface Control Document (ICD)*.

```
<5>     Software unit testing and software integration testing
        NOTE    The SUITP describes the responsibility and
                schedule information for the software unit
                testing and integration testing, detailed as
                follows.
<5.1>   Organization
a.      The SUITP shall describe the organization of software unit testing and
        integration testing activities.
```

(continues on next page)

(continued from previous page)

```
b.      The following topics should be included:
        1.      roles,
        2.      reporting channels,
        3.      levels of authority for resolving problems,
        4.      relationships to the other activities such as project management,
                development, configuration management and product assurance.
```

This will be provided as hand written content in Sphinx format.

```
<5.2>   Master schedule
a.      The SUITP shall describe the schedule for the software unit testing and
        integration testing activities, in particular, test milestones identified in
        the software project schedule and all item delivery events.
b.      The SUITP should include:
        1.      a reference to the master schedule given in the software
                development plan,
        2.      any additional test milestones and state the time required for each
                testing task,
        3.      the schedule for each testing task and test milestone,
        4.      the period of use for the test facilities.
```

This section will be empty.

```
<5.3>   Resource summary
a.      The SUITP shall summarize the resources needed to perform the
        software unit testing / integration testing activities such as staff,
        hardware and software tools.
```

This will be provided as hand written content in Sphinx format.

```
<5.4>   Responsibilities
a.      The SUITP shall describe the specific responsibilities associated with the
        roles described in a.
b.      The responsibilities specified in <5.4>a. should be described by
        identifying the groups responsible for managing, designing, preparing,
        executing the tests.
        NOTE    Groups can include developers, technical
                support staff, and product assurance staff.
```

This will be provided as hand written content in Sphinx format.

```
<5.5>   Tools, techniques and methods
a.      The SUITP shall describe the hardware platforms, software tools,
        techniques and methods used for software unit testing and integration
        testing activities.
```

This will be provided as hand written content in Sphinx format.

```
<5.6>   Personnel and personnel training requirements
a.      The SUITP shall list any requirement for software unit testing and
        integration testing personnel and their training needs.
```

This will be provided as hand written content in Sphinx format.

```
<5.7>   Risks and contingencies
a.      The SUITP shall describe (or refer to the SDP) risks to the software unit
        testing and integration testing campaign.
b.      Contingency plans should be included.
```

This will be provided as hand written content in Sphinx format. Reference to *Software Development Plan (SDP)*.

```
<6>     Control procedures for software unit testing / integration
        testing
a.      The SUITP shall contain information (or reference to) about applicable
        management procedures concerning the following aspects:
        1.      problem reporting and resolution;
        2.      deviation and waiver policy;
        3.      control procedures.
```

This will be provided as hand written content in Sphinx format. Reference to *Software Development Plan (SDP)*.

```
<7>     Software unit testing and integration testing approach
        NOTE    The SUITP describes the approach to be utilized
                for the software unit testing and integration
                testing, detailed as follows.
<7.1>   Unit/integration testing strategy
a.      The SUITP shall describe the software integration strategy
```

This will be provided as hand written content in Sphinx format.

```
<7.2>   Tasks and items under test
a.      The SUITP shall describe which are the tasks and the items under tests, as
        well as criteria to be utilized.
```

This will be provided as hand written content in Sphinx format.

```
<7.3>   Features to be tested
a.      The SUITP shall describe all the features to be tested, making references
        to the applicable documentation.
```

This will be provided as hand written content in Sphinx format.

```
<7.4>   Features not to be tested
a.      The SUITP shall describe all the features and significant combinations not
        to be tested.
```

This will be provided as hand written content in Sphinx format.

```
<7.5>   Test pass - fail criteria
a.      The SUITP shall describe the general criteria to be used to determine
        whether or not test are passed.
```

This will be provided as hand written content in Sphinx format.

```
<7.6>   Manually and automatically generated code
a.      The SUITP shall address separately the activities to be performed for
        manually and automatically generated code, although they have the
        same objective (ECSS-Q-ST-80 clause 6.2.8.2 and 6.2.8.7).
```

This will be provided as hand written content in Sphinx format.

```
<8>     Software unit test / integration test design
<8.1>   General
a.      The SUITP shall provide the definition of unit and integration test design.
```

This will be provided as hand written content in Sphinx format.

```
b.      For each identified test design, the SUITP shall provide the information
        given in <8.2>
        NOTE    This can be simplified in the software unit test
                plan.
<8.2>   Organization of each identified test design
        NOTE    The SUITP provides the definition of each unit
                test and integration test design, detailed as
                follows.
<8.2.1> Test design identifier
a.      The SUITP shall identify each test design uniquely.
```

See *Identification*.

```
b.      The SUITP shall briefly describe the test design.
<8.2.2> Features to be tested
a.      The SUITP shall list the test items and describe the features to be tested.
b.      Reference to appropriate documentation shall be made and traceability
        information shall be provided.
```

For each test design (*Test Suite Item Type*), this information can be generated from the *Test Case Item Type* specification items referencing the test design. The term feature in the SUITP is defined as the *software unit* referenced by a unit test case, in contrast to the project definition of *feature*.

```
<8.2.3> Approach refinements
a.      The SUITP shall describe the test approach implemented for the specific
        test design and the specific test class.
b.      The description specified in a. shall provide the rationale for the test case
        selection and grouping into test procedures.
c.      The method for analysing test results shall be identified (e.g. compare
        with expected output).
d.      Configuration of the facility (both hardware and software) to be used to
        execute the identified test shall be described.
<8.2.4> Test case identifier
a.      The SUITP shall list the test cases associated with the test design and
        provide a summary description of each ones.
```

The test designs will be provided through *Test Suite Item Type* specification items.

```
<9>     Software unit and integration test case specification
<9.1>   General
a.      The SUITP shall provide an identification of software unit test and
        integration test cases.
```

See *Identification*.

```
b.      For each identified test case, the SUITP shall provide the information
        given in <9.2>.
        NOTE    Each test case can be described through one or
                several description sheets.
<9.2>   Organization of each identified test case
        NOTE    The SUITP provides the definition of each unit
                and integration test case, detailed as follows.
<9.2.1> Test case identifier
a.      The SUITP shall identify the test case uniquely.
b.      A short description of the test case purpose shall be provided.
<9.2.2> Test items
a.      The SUITP shall list the test items.
b.      Reference to appropriate documentation shall be performed and
        traceability information shall be provided.
<9.2.3> Inputs specification
a.      The SUITP shall describe the inputs to execute the test case.
<9.2.4> Outputs specification
a.      This SUITP shall describe the expected outputs.
<9.2.5> Test pass - fail criteria
a.      The SUITP shall list the criteria to decide whether the test has passed or
        failed.
<9.2.6> Environmental needs
a.      The SUITP shall describe:
        1.      the exact configuration and the set up of the facility used to
                execute the test case as well as the utilization of any special test
                equipment (e.g. bus analyser);
        2.      the configuration of the software utilized to support the test
                conduction (e.g. identification of the simulation configuration);
<9.2.7> Special procedural constraints (ECSS-Q-ST-80 clause 6.3.5.25)
a.      The SUITP shall describe any special constraints on the used test
        procedures.
<9.2.8> Interfaces dependencies
a.      The SUITP shall describe all the test cases to be executed before this test
        case.
```

The test cases will be provided through *Test Case Item Type* specification items.

```
<9.2.9> Test script
a.      The SUITP shall describe all the test script used to execute the test case.
        NOTE    The test scripts can be collected in an appendix.
```

Reference to the Qualification Toolchain documentation.

```
<10>    Software unit and integration test procedures
<10.1>  General
a.      The SUITP shall provide a identification of software unit and integration
        test procedures.
```

See *Identification*.

```
b.      For each identified test procedure, the SUITP shall provide the
        information given in <10.2>.
<10.2>  Organization of each identified test procedure
        NOTE    The SUITP provides the definition of each unit
                and integration test procedure, detailed as
                follows.
<10.2.1> Test procedures identifier
a.      The SUITP shall include a statement specifying the test procedure
        uniquely.
<10.2.2> Purpose
a.      The SUITP shall describe the purpose of this procedure.
b.      A reference to each test case implemented by the test procedure shall be
        given.
<10.2.3> Procedure steps
a.      The SUITP shall describe every step of the procedure execution:
        1.      log: describe any special methods or format for logging the results
                of test execution, the incidents observed, and any other event
                pertinent to this test;
        2.      set up: describe the sequence of actions to set up the procedure
                execution;
        3.      start: describe the actions to begin the procedure execution;
        4.      proceed: describe the actions during the procedure execution;
        5.      test result acquisition: describe how the test measurements is
                made;
        6.      shut down: describe the action to suspend testing when
                interruption is forced by unscheduled events;
        7.      restart: identify any procedural restart points and describe the
                actions to restart the procedure at each of these points;
        8.      wrap up: describe the actions to terminate testing.
```

The test procedures will be provided through *Test Procedure Item Type* specification items.

```
<11>    Software test plan additional information
a.      The following additional information shall be provided:
        1.      test procedures to test cases traceability matrix;
        2.      test cases to test procedures traceability matrix;
        3.      test scripts;
        4.      detailed test procedures.
        NOTE 1  This information can be given in separate
                appendices.
        NOTE 2  One test design uses one or more test cases.
        NOTE 3  One test procedure execute one or more test
                cases.
```

The traceability matrices will be generated from the *Specification Items*. Test script information will be provided by reference to the Qualification Toolchain documentation. No extra detailed test procedures will be provided in this section.

### 4.4.5 Software Validation Specification (SVS) with Respect to TS

The Software Validation Specification with Respect to TS is a part of the Design Justification File (DJF) and its content is defined by ECSS-E-ST-40C Annex L [ECS09b]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

```
L.2     Expected response
L.2.1   Scope and content
<1>     Introduction
a.      The SVS w.r.t. TS or RB shall contain a description of the purpose,
        objective, content and the reason prompting its preparation.
```

This will be provided as hand written content in Sphinx format.

```
<2>     Applicable and reference documents
a.      The SVS w.r.t. TS or RB shall list the applicable and reference documents
        to support the generation of the document.
```

See *Applicable and Reference Documents*.

```
<3>     Terms, definitions and abbreviated terms
a.      The SVS w.r.t. TS or RB shall include any additional terms, definition or
        abbreviated terms used.
```

See *Terms, Definitions and Abbreviated Terms*.

```
<4>     Software overview
a.      The SVS w.r.t. TS or RB shall contain a brief description of the software
        under test and its context: a summary of its functionality, its
        configuration, its operational environment and its external interfaces.
        NOTE    Reference to technical documentation can be
                done.
```

References to the *Software Requirements Specification (SRS)*, *Software Interface Control Document (ICD)*, and *Software Design Document (SDD)*.

```
<5>     Software validation specification task identification
        NOTE    The SVS w.r.t. TS or RB describes the approach
                to be utilized for the software validation
                specification, detailed as follows.
<5.1>   Task and criteria
a.      The SVS w.r.t. TS or RB shall describe which are the tasks and the items
        under tests, as well as criteria to be utilized.
```

This will be provided as hand written content in Sphinx format.

```
<5.2>   Features to be tested
a.      The SVS w.r.t. TS or RB shall describe all the features to be tested, making
        references to the applicable documentation.
```

This will be provided as hand written content in Sphinx format.

```
<5.3>   Features not to be tested
a.      The SVS w.r.t. TS or RB shall describe all the features and significant
        combinations not to be tested.
```

This will be provided as hand written content in Sphinx format.

```
<5.4>   Test pass - fail criteria
a.      The SVS w.r.t. TS or RB shall describe the general criteria to be used to
        determine whether or not tests are passed.
```

This will be provided as hand written content in Sphinx format.

```
<5.5>   Items that cannot be validated by test
a.      The SVS w.r.t. TS or RB shall list the tasks and items under tests that
        cannot be validated by test.
b.      Each of them shall be properly justified
c.      For each of them, an analysis, inspection or review of design shall be
        proposed.
```

The validation by analysis, inspection, or review of design will be provided by *Requirement Validation*.

```
<5.6>   Manually and automatically generated code
a.      The SVS shall address separately the activities to be performed for
        manually and automatically generated code, although they have the
        same objective (ECSS-Q-ST-80 clause 6.2.8.2 and 6.2.8.7).
```

This will be provided as hand written content in Sphinx format.

```
<6>     Software validation testing specification design
<6.1>   General
a.      The SVS w.r.t. TS or RB shall provide the definition of software
        validation testing specification design, giving the design grouping
        criteria such as function, component or equipment management.
```

This will be provided as hand written content in Sphinx format.

```
b.      For each identified test design, the SVS w.r.t. TS or RB shall provide the
        information given in <6.2>.
<6.2>   Organization of each identified test design
        NOTE    The SVS w.r.t. TS or RB provides the definition
                of each validation test design, detailed as
                follows
<6.2.1> General
a.      The SVS w.r.t. TS or RB shall briefly describe the test design.
<6.2.2> Features to be tested
a.      The SVS w.r.t. TS or RB shall describe the test items and the features to be
        tested.
b.      Reference to appropriate documentation shall be performed and
        traceability information shall be provided.
```

For each test design (*Test Suite Item Type*), this information can be generated from the *Test Case Item Type* specification items referencing the test design.

```
<6.2.3> Approach refinements
a.      The SVS w.r.t. TS or RB shall describe the test approach implemented for
        the specific test design and the specific test class implemented.
b.      The description specified in a. shall provide the rationale for the test case
        selection and grouping into test procedures.
c.      The method for analysing test results shall be identified (e.g. compare
        with expected output, and compare with old results).
d.      Configuration of the facility (both hardware and software) to be used to
        execute the identified test shall be described.
```

The test designs will be provided through *Test Suite Item Type* specification items.

```
<7>     Software validation test case specification
<7.1>   General
a.      The SVS w.r.t. TS or RB shall provide the identification of software
        validation test cases.
```

See *Identification*.

```
b.      For each identified test case, the SVS w.r.t. TS or RB shall provide the
        information given in 7.2
<7.2>   Organization of each identified test case
        NOTE    The SVS w.r.t. TS or RB provides the definition
                of each validation test case, detailed as follows.
<7.2.1> Test case identifier
a.      The SVS w.r.t. TS or RB shall describe the test case uniquely.
b.      A short description of the test case purpose shall be provided.
<7.2.2> Inputs specification
a.      The SVS w.r.t. TS or RB shall describe, for each test case, the inputs to
        execute the test case.
<7.2.3> Outputs specification
a.      The SVS w.r.t. TS or RB shall describe, for each test case, the expected
        outputs.
<7.2.4> Test pass - fail criteria
a.      The SVS w.r.t. TS or RB shall describe, for each test case, the criteria to
        decide whether the test has passed or failed.
<7.2.5> Environmental needs
a.      The SVS w.r.t. TS or RB shall describe:
        1.      the exact configuration and the set up of the facility used to
                execute the test case as well as the utilization of any special test
                equipment (e.g. bus analyser);
        2.      the configuration of the software utilized to support the test
                conduction (e.g. identification of the simulation configuration);
<7.2.6> Special procedural constraints(ECSS-Q-ST-80 clause 6.3.5.25)
a.      The SVS w.r.t. TS or RB shall describe any special constraints on the used
        test procedures.
<7.2.7> Interfaces dependencies
a.      The SVS w.r.t. TS or RB shall list all the test cases to be executed before
        this test case.
```

The test cases will be provided through *Test Case Item Type* specification items.

```
<8>     Software validation test procedures
<8.1>   General
```

(continues on next page)

(continued from previous page)

```
a.      This part of the DRD may be placed in a different document, if agreed
        with the customer.
        NOTE    Procedures are not always attached to each test
                case
b.      The SVS w.r.t. TS or RB shall provide the identification of software
        validation test procedures.
```

See *Identification*.

```
c.      For each identified validation test procedure, the SVS w.r.t. TS or RB shall
        provide the information presented in 8.2
<8.2>   Organization of each identified test procedure
        NOTE    The SVS w.r.t. TS or RB provides the
                description of each identified validation test
                procedure, detailed as follows.
<8.2.1> Test procedure identifier
a.      The SVS w.r.t. TS or RB shall identify each test procedure uniquely.
<8.2.2> Purpose
a.      The SVS w.r.t. TS or RB shall describe the purpose of each test procedure.
b.      A reference to each test case used by the test procedure shall be given.
<8.2.3> Procedure steps
a.      The SVS w.r.t. TS or RB shall describe every step of each procedure
        execution:
        1.      log: describe any special methods or format for logging the results
                of test execution, the incidents observed, and any other event
                pertinent to this test;
        2.      set up: describe the sequence of actions necessary to set up the
                procedure execution;
        3.      start: describe the actions necessary to begin the procedure
                execution;
        4.      proceed: describe the actions necessary during the procedure
                execution;
        5.      test result acquisition: describe how the test measurements is
                made;
        6.      shut down: describe the action necessary to suspend testing when
                interruption is forced by unscheduled events;
        7.      restart: identify any procedural restart points and describe the
                actions necessary to restart the procedure at each of these points;
        8.      wrap up: describe the actions necessary to terminate testing.
```

The test procedures will be provided through *Test Procedure Item Type* specification items.

```
<8.2.4> Test script
a.      The SVS w.r.t. TS or RB shall list all the test script used to execute the test
        case.
        NOTE    The test scripts can be collected in an appendix.
```

Reference to the Qualification Toolchain documentation.

```
<9>     Software validation analysis, inspection, review of
        design
a.      The SVS w.r.t. TS or RB shall include, for each items where it can be
        justified that a test is not possible, another validation method based on
```

(continues on next page)

(continued from previous page)

```
        analysis, inspection, review of design.
```

The validation by analysis, inspection, or review of design will be provided by *Requirement Validation*.

```
<10>    Validation test platform requirements
a.      The SVS w.r.t. TS or RB shall list the validation requirements related to
        the validation test platform to be used (for example, benches or
        simulators capabilities and their representativity with respect to e.g. real
        time constraints, target or real hardware equipments on which the
        software is specified to operate).
```

The *Test Procedure Item Type* specification items may reference validation test platform requirements.

```
<11>    Software validation specification additional information
a.      The following additional information shall be included in the SVS w.r.t.
        TS or RB:
        1.      Test/analysis/inspection/review of design to requirement
                traceability matrix,
        2.      Requirement to test/analysis/inspection/review of design
                traceability matrix,
        3.      Test procedures to test cases traceability matrix,
        4.      Test cases to test procedures traceability matrix,
        5.      Test scripts,
        6.      Detailed test procedures.
        NOTE 1  This information can be given in separate
                appendices.
        NOTE 2  One test design uses one or more test cases.
        NOTE 3  One test procedure execute one or more test
                cases.
        NOTE 4  Traceability matrices include the title of the
                requirement or test in addition to its number
                for readability purpose.
```

The traceability matrices will be generated from the *Specification Items*. Test script information will be provided by reference to the Qualification Toolchain documentation. No extra detailed test procedures will be provided in this section.

### 4.4.6 Software Validation Specification (SVS) with Respect to RB

Not included in the QDP, see *No Requirements Baseline (RB)*.

### 4.4.7 Acceptance Test Plan

Not included in the QDP, see *No Installation and Acceptance*.

### 4.4.8 Software Unit and Integration Test Report

The report shall be generated by the Qualification Toolchain from the test output of the RTEMS Test Framework. The generator should support histograms for performance tests. The generator should support timing diagrams for interrupts and thread switches if the test cases produces tracing output. See proof of concept.

There will be a combined report for unit and integration tests, see *Combined Unit and Integration Testing*. This is a deviation from ECSS-E-ST-40C.

**Note:** This report will be included in the *Software Verification Report (SVR)*.

### 4.4.9 Software Validation Report with Respect to TS

The report shall be generated by the Qualification Toolchain from the test output of the RTEMS Test Framework and validation by analysis, inspection, and review of design specification items (*Requirement Validation*). The generator should support histograms for performance tests. The generator should support timing diagrams for interrupts and thread switches if the test cases produces tracing output. See proof of concept.

**Note:** This report will be included in the *Software Verification Report (SVR)*.

### 4.4.10 Software Validation Report with Respect to RB

Not included in the QDP, see *No Requirements Baseline (RB)*.

### 4.4.11 Acceptance Test Report

Not included in the QDP, see *No Installation and Acceptance*.

### 4.4.12 Installation Report

Not included in the QDP, see *No Installation and Acceptance*.

### 4.4.13 Software Verification Report (SVR)

The Software Verification Report (SVR) is a part of the Design Justification File (DJF) and its content is defined by ECSS-E-ST-40C Annex M [ECS09b]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

```
M.2     Expected response
M.2.1   Scope and content
<1>     Introduction
a.      The SVR shall contain a description of the purpose, objective, content and
        the reason prompting its preparation.
```

This will be provided as hand written content in Sphinx format.

```
<2>     Applicable and reference documents
a.      The SVR shall list the applicable and reference documents to support the
        generation of the document.
```

See *Applicable and Reference Documents*.

```
<3>     Terms, definitions and abbreviated terms
a.      The SVR shall include any additional terms, definition or abbreviated
        terms used.
```

See *Terms, Definitions and Abbreviated Terms*.

```
<4>     Verification activities reporting and monitoring
<4.1>   General
a.      The SVR shall address separately the activities to be performed for
        manually and automatically generated code.
```

`<4.1>.a.` essentially requests that all sections below should report on manual and generated code distinctively. Yet, this document will not make a difference between hand written and automatically generated code. The following justification will be provided:

> Automatically generated code has its origin in some specification items. These items contain pieces of human written code. We use tools to assemble these pieces of manual conceived code into entire files of code. These files contain the collected pieces of hand written code, some generated header and trailer code as well as tables which are also derived from manually created specification items. All other code is fully hand written.
>
> A automatically generated code files are treated as if there were hand written:
>
> - The tool to generate the code is run by a programmer.
> - The generated files are inspected and tested by that programmer.

- The generated files are sent to the developer mailing list as if that programmer had created them by hand.
- The generated files are reviewed on the mailing list like any other files and any findings must be fixed.
- Finally, the generated code is committed to the source code repository like all hand written code.
- This means, the automatically produced code is not generated a new every time the operating system is build. Instead, the code is generated once, reviewed, committed to the other source code. The operating system is then build again and again from that once checked-in source code without generating it every time from scratch.

As automatically generated code in RTEMS is handled like hand written code, the verification activities performed on generated code are exactly the same as those for hand written code.

```
<4.2>   Software related system requirements process verification
        (for the SRR)
a.      The SVR shall include the report of the verification of the requirement
        baseline and the interface requirements specification as specified in
        5.8.3.1.
```

There is no requirement baseline, see *No Requirements Baseline (RB)* and *ECSS-E-ST-40C 5.8.3.1*.

```
b.      If system models are available, a model checking report (e.g. data, event,
        failure) shall be included in the SVR.
```

There is no system model, see *No Logical and Computational Model*.

```
<4.3>   Software requirements and architecture engineering process
        verification (for the PDR)
<4.3.1> Traceability (when not already included in related software
        requirements, interface and design documents)
a.      The SVR shall present the following traceability matrices:
        -       software requirements to system requirements
        -       Software architectural design to requirements
```

There is no requirement baseline. Therefore, traceability from software requirements to system requirements is not included (see *No Requirements Baseline (RB)*). The software architectural design to requirements traceability is included in the *Software Design Document (SDD)*. See also *Traceability between Software Requirements, Architecture and Design*.

```
<4.3.2> Feasibility
a.      The SVR shall present in gathering all the specific verification
        reports that have been planned to be provided w.r.t. the SVerP,
        including e.g.:
```

In case there are several items which need to be verified (such as different software documents), there may be a report on each individual item or one report on all items. The reports may be included in this SVR or this SVR will reference the documents containing the reports.

```
-      Software requirements verification as per 5.8.3.2.
```

Each point of *ECSS-E-ST-40C section 5.8.3.2* will be covered by hand written content with the exception of these:

2. Not applicable because of *No Requirements Baseline (RB)*
3. Not applicable because of *No Requirements Baseline (RB)*
10. Point 10 asks to check whether the verification method defined for each requirement because of *ECSS-Q-ST-80C 7.2.1.3* is feasible. Whereby such a method is to be defined for:
    a. **Requirement baseline** This is not applicable because of *No Requirements Baseline (RB)*.
    b. **Technical specification** These requirements are those stated in the *Software Requirements Specification (SRS)* and arise from specification items. Whether their validation method is feasible will be provided in hand written form.
11. Not applicable because of *No Logical and Computational Model*

```
      -      HMI evaluation by e.g. mock-up as per ECSS-E-ST-10-11
```

There is no HMI.

```
      -      Behavioural verification of the logical model and architectural
             design verification as per 5.8.3.13a. and b.
```

There is no logical model, see *No Logical and Computational Model*.

```
      -      Verification of the software architectural and interface design as
             per 5.8.3.3.
```

Each point of *ECSS-E-ST-40C section 5.8.3.3* will be covered by hand written content with the exception of point 3 which will be automatically checked and the resulting report will be included.

```
      -      Architectural design behavioural model checking
```

There is no behavioural model, see *No Logical and Computational Model*.

```
      -      Verification of software documentation as per 5.8.3.10.
```

This sub-point repeats in the SVR in the following sections:

- `<4.3.2>.a` for PDR
- `<4.4>.a.2` for CDR
- `<4.5>.a.2` for QR and AR
- `<4.6>.a.2` for CDR, QR and AR

Since this document has not been presented at PDR, all software documentation ready for CDR will be covered in section <4.4>.a.2. Verifications of later changes to software documentation and new software documentation shall be reported in section <4.5>.a.2.

Consequently, this section here (<4.3.2>.a) will be empty and point to <4.4>.a.2 in this SVR.

```
        -       Other specific inspections, analyses or review of design report (e.g.
                numerical accuracy , technical risks analysis, evaluation of reuse
                potential)
        -       Others specific verification related to RAMS requirements (e.g.
                analysis reports using HSIA, SFTA, SFMECA)
```

The above sub-point repeats in the SVR in the following sections:

- <4.3.2>.a for PDR
- <4.4>.a.2 for CDR
- <4.5>.a.2 for QR and AR (only partial repetition)

Since this document has not been presented at PDR and the content will be the same for PDR and CDR, only section <4.4>.a.2 will contain the report and this sub-section here (<4.3.2>.a) will only contain a pointer there.

```
<4.4>   Software design and implementation engineering process
        verification (for CDR)
a.      The SVR shall present in gathering all the specific verification reports that
        have been planned to be provided w.r.t. the SVerP, including e.g.:
        1.      Traceability (when not already included in related software design
                documents or software code), presenting the following traceability
                matrices:
                o       software detailed design to software architectural design
                o       Software code to software detailed design
                o       Software unit test to requirements, design
```

The traceability information will be provided by the SDD (see *Software Design Document (SDD)* and *Traceability between Software Requirements, Architecture and Design*). The relevant sections of that document will be referenced.

The third bullet is covered by *validation tests*; not by *unit tests*. RTEMS has existing test suites with about 600 to 700 tests. Most of these tests exist since long before the start of this qualification project and have naturally no relation to the requirements.

In contrast, the specification items define tests specifically designed to test a particular requirement. These tests are called *verification tests*. These test are linked to the requirement they test. Therefore, these verification tests are the ones which must be used in the above mentioned traceability matrix.

See also *On Demand Unit and Integration Testing* and *Combined Unit and Integration Testing*.

```
        2.      Feasibility, presenting in gathering all the specific verification
                reports that have been planned to be provided w.r.t. the SVerP,
                including e.g.:
                o       Software detailed design verification as per 5.8.3.4
```

Each point of *ECSS-E-ST-40C 5.8.3.4* will be covered by hand written content with the exception of

- point 3 which will be automatically checked and the resulting report will be included and
- point 9 which is not applicable (see *No Logical and Computational Model*)

```
        o   Design model checking (including behavioural verification
            as per 5.8.3.13b.)
```

There is no behavioural model, see *No Logical and Computational Model* and *ECSS-E-ST-40C 5.8.3.13b*.

```
        o   Software code verification as per 5.8.3.5a.
```

Each point of *ECSS-E-ST-40C 5.8.3.5a* will be covered by hand written content with the exception of these:

3. partly hand written and partly automatically generated
9. not applicable because RTEMS has no floating point operations (see *No Numerical Accuracy Analysis*)

```
        o   Structural code coverage achievement.
```

This will be automatically generated content obtained from code coverage tests.

```
        o   Deactivated code verification as per ECSS-Q-ST-80 6.2.6.5
```

There is no deactivated code in RTEMS, since the pre-qualified RTEMS will achieve 100% code coverage.

```
        o   Configurable code verification as per ECSS-Q-ST-80 6.2.6.6
```

This sub-section will contain the following text:

> This requirement is met by RTEMS design:
>
> - There are checks of the C pre-processer, which detects invalid/illegal configuration.
> - RTEMS has configuration checks at link-time as well as at start-up
> - The user can check certain configuration parameters by diagnostic functions

```
        o   Source code robustness verification
```

This will be provided as hand written content which references to the results of the static analyzers which will be provided in the SPAMR (*Software Product Assurance Milestone Report (SPAMR)*) and explains relevance of the output of the tools. See *ECSS-E-ST-40C 5.8.3.5f*

```
        o   Verification of software unit testing as per 5.8.3.6
```

RTEMS calls these kind of test *validation tests* instead of *unit tests*. Each point of *ECSS-E-ST-40C 5.8.3.6* will be covered by hand written content.

```
                o     Verification of software integration as per 5.8.3.7
```

RTEMS calls these kind of test *validation tests* instead of *integration tests*. *ECSS-E-ST-40C 5.8.3.7* will be covered by hand written text.

```
                o     Verification of software documentation as per 5.8.3.10
```

There will be a report for each software document. Each point of *ECSS-E-ST-40C 5.8.3.10* will be covered by hand written content.

See *repeating document verification in SVR*.

```
                o     Others specific inspections, analyses or review of design
                      report (e.g. technical risks analysis, evaluation of reuse
                      potential)
```

The reused software is referred in the SRF [EDI20]. No more specific inspections, analysis or review of design are necessary.

```
o     Others specific verification related to RAMS requirements
      (e.g. analysis reports using HSIA, SFTA, SFMECA).
```

This sub-section will contain the following text:

> The HSIA and SFTA are not applicable to RTEMS. A SFMECA was performed to RTEMS (see SPAP [EDI19e], chapter 6) and derived into recommendations but as a final conclusion none of them is applicable to Pre-Qualified RTEMS (space profile). Hence, there are no RAMS requirements.

See *repeating other verifications in SVR*

```
<4.5>   Software delivery and acceptance process verification (for QR
        and AR)
a.      The SVR shall present in gathering all the specific verification reports that
        have been planned to be provided w.r.t. the SVerP, including e.g.:
        1.      Traceability (when not already included in related software
                acceptance documents), presenting the following traceability
                matrices:
                o       Software acceptance testing to requirement baseline
```

There is no requirement baseline, see *No Requirements Baseline (RB)*.

```
        2.      Feasibility, presenting in gathering all the specific verification
                reports that have been planned to be provided w.r.t. the SVerP,
                including:
                o       Structural code coverage achievement (update for QR and
                        AR)
```

This will be automatically generated content obtained from code coverage tests.

```
o       Verification of software documentation as per 5.8.3.10
```

There will be a report for each new software document or document change. Each point of *ECSS-E-ST-40C 5.8.3.10* will be covered by hand written content.

See *repeating document verification in SVR*.

```
o       Others specific verification related to RAMS design (e.g. unit
        and integration testing coverage ratio)
```

There are no RAMS requirements/design. See above.

See also *repeating other verifications in SVR*

```
<4.6>   Software validation process verification (for CDR, QR, AR)
a.      The SVR shall present in gathering all the specific verification reports that
        have been planned to be provided w.r.t. the SVerP, including e.g.:
        1.      Traceability (when not already included in related software
                validation specification), presenting the following traceability
                matrices:
                o       software validation specifications to TS
```

The traceability from software validation to technical specification will be generated automatically and be included in the *Software Validation Specification (SVS) with Respect to TS*.

```
o       software validation specifications to RB
```

Not applicable as there is no requirement baseline (see *No Requirements Baseline (RB)*).

```
        2.      Feasibility, presenting in gathering all the specific verification
                reports that have been planned to be provided w.r.t. the SVerP,
                including e.g.:
                o       Verification of software validation w.r.t. TS as per 5.8.3.8a.
```

The verification is done automatically and the resulting report will be provided.

```
                o       Verification of software validation w.r.t. RB as per 5.8.3.8b.
```

Not applicable as there is no requirement baseline (see *No Requirements Baseline (RB)*).

```
                o       Verification of software documentation as per 5.8.3.10
```

Empty here but will be done for QR and AR in `<4.5>.a.2`. See *repeating document verification in SVR*.

```
                o       Verification of testing as per ECSS-Q-ST-80 clause 7.2.3.6
```

The content will be generated automatically.

```
<4.7>   Software quality requirements verification
a.      The SVR shall present in gathering all the specific verification reports
        related to software quality that have been planned to be provided in the
        SVerP. This include in particular the verification of the software quality
        requirements according to ECSS-Q-ST-80 clause 6.2.6.1, and the
        verification of the application of the chosen measures to handle
        automatically generated code.

        NOTE    Deactivated and configurable code verification
                reports are in section <4.4>a.2. Numerical
                accuracy report is in section <6> of this DRD.
```

The verifications will be done according to the Software Verification Plan (part of SDP-000 [EDI19c]) section 10.3.3.1:

- For *collection and analysis of metrics defined in SPAP-002 deliverable*: This will only reference the metrics already reported and justified in the *Software Product Assurance Milestone Report (SPAMR)* section <7>.
- For *quality review of the deliverables on each formal review*: This will only refrence the reviews and findings already reported in the *Software Product Assurance Milestone Report (SPAMR)* section <4>.a.5.

Moreover, a hand written content about the measures to handle automatically generated code will be provided:

- Generated code is reviewed on the mailing list like manual written code.
- Generated code is subject to testing (in order to achieve the code coverage threshold)
- Generated code undergoes the same static analyzing and metering as the manual generated code

See *ECSS-Q-ST-80C Rev.1 6.2.6.1*.

```
<5>     Margin and technical budget status
        NOTE    This section is often placed in a separate
                document named STSB (Software Timing and
                Sizing Budget).
<5.1>   Technical budgets and margins computation
a.      The SVR shall include the way to compute the technical budgets and
        margins.
```

This will contain a reference to [RTEb] sections:

- 8.1.5 *Test Case Resource Accounting*
- 8.1.10 *Time Services*
- 8.1.11 *Code Runtime Measurements*
- 8.1.14 *Test Reporting*

In addition, there will be a hand written description on how the memory sizes are measured.

```
<5.2>   Software budget (sizing and timing)
a.      The status of margins regarding the technical budgets shall be presented
        in the SVR at each milestone, describing the utilized analytical
        hypothesis.
b.      The margins shall be established by estimation for PDR, by analysis of
        design after detailed design, and consolidated by performance
        measurements commensurate with the software implementation for
        CDR, QR and AR.
c.      The SVR shall include at PDR:
        1.      the memory size for static code size, static data size and stack size;
        2.      the CPU utilization;
        3.      the deadline fulfilment, the margin available for every deadline in
        the worst case and, if feasible, the jitter in the nominal case;
d.      The SVR shall include after detailed design:
```

(continues on next page)

(continued from previous page)

```
        1.      the memory size refined for static code size, static data size and
                stack size expressed on a thread basis, measuring them per lowest
                level design component;
        2.      the CPU utilization, refined, considering the worst case execution
                time of each lowest level design component having its own control
                flow (therefore including the call to the protected objects)
                (expressed in time and in percentage of a reference period);
        3.      the deadline fulfilment.
        NOTE    The worst case execution time of each lowest
                level design component having its own control
                flow is multiplied by the number of times this
                component is executed per second. The
                resulting quantity is summed over all other
                design components. The result is the estimated
                percentage processor utilization.
```

The content of this section will be mostly automatically generated from

- performance test results
- inspection of binary (i.e. ELF) files

Since we do not deliver an application but an operating system certain of the above demands are not applicable – for example, there is no deadline defined; CPU utilization is meaningless for a single function call. Moreover, the worst case execution time cannot be determined in practice. Instead values meaningful for the practitioner like the maximum, middle and minimum execution time per function will be reported. See also *Resources and Performance*.

There will a hand written explanation of each kind of value appearing in the above report.

```
<5.3>   Schedulability simulation and analyses
a.      The SVR shall include the result of the schedulability analysis or the
        schedulability simulation, based on:
        1.      estimated values at PDR,
        2.      refined values after detailed design,
        3.      measured values at CDR.
        NOTE    An example of schedulability analysis report is
                a table with the following columns:
                *       process name
                *       P: process priority
                *       C: process worst case execution time
                *       T: process period
                *       D: process deadline
                *       I: process interference time, the time that the
                        process can be interrupted by processes of
                        higher priority
                *       B: process blocking time, the time that the
                        process can be blocked on a protected object
                        access by a process of lower priority
                *       S: process schedulability factor in
                        percentage, computed as the sum of C, I and
                        B, this sum divided by D
```

Not applicable. See *No Schedulability Analysis*.

```
<6>     Numerical accuracy analysis
a.      The SVR shall include the estimation and the verification of the
        numerical accuracy.
```

Not applicable. See *No Numerical Accuracy Analysis*.

### 4.4.14 Independent Software Verification & Validation Report

Not included in the QDP, see *No Independent Software Verification and Validation*.

### 4.4.15 Software Reuse File (SRF)

The Software Reuse File (SRF) is a part of the Design Justification File (DJF) and its content is defined by ECSS-E-ST-40C Annex N [ECS09b]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

```
N.2     Expected response
N.2.1   Scope and content
<1>     Introduction
a.      The SRF shall contain a description of the purpose, objective, content and
        the reason prompting its preparation.
```

This will be provided as hand written content in Sphinx format.

```
<2>     Applicable and reference documents
a.      The SRF shall list the applicable and reference documents to support the
        generation of the document.
```

See *Applicable and Reference Documents*.

```
<3>     Terms, definitions and abbreviated terms
a.      The SRF shall include any additional terms, definition or abbreviated
        terms used.
```

See *Terms, Definitions and Abbreviated Terms*.

```
<4>     Presentation of the software intended to be reused
a.      The SRF shall describe the technical and management information
        available on the software intended for reuse.
b.      For each software item, the SRF shall provide (or state the absence of) the
        following information:
        1.      software item name and main features;
        2.      developer name;
        3.      considered version and list of components;
        4.      licensing conditions;
        5.      industrial property and exportability constraints, if any;
        6.      implementation language;
        7.      development and execution environment (e.g. platform, and
                operating system);
        8.      applicable dispositions for warranty, maintenance, installation and
```

(continues on next page)

(continued from previous page)

```
                training;
        9.      commercial software necessary for software execution, if any;
        10.     1size of the software (e.g. number of source code lines, and size of
                the executable code).
<5>     Compatibility of existing software with project
        requirements
a.      The SRF shall describe which part of the project requirements (RB) are
        intended to be implemented through software reuse
b.      For each software item, the SRF shall provide the availability and quality
        status (completeness, correctness, etc.) of the following information:
        1.      software requirements documentation;
        2.      software architectural and detailed design documentation;
        3.      forward and backward traceability between system requirements;
        4.      software requirements, design and code;
        5.      unit tests documentation and coverage;
        6.      integration tests documentation and coverage;
        7.      validation documentation and coverage;
        8.      verification reports;
        9.      performance (e.g. memory occupation, CPU load);
        10.     1operational performances;
        11.     1residual non conformance and waivers;
        12.     1user operational documentation (e.g. user manual);
        13.     1code quality (adherence to coding standards, metrics).
c.      For each of the points in <5>b, the SRF shall document the quality level of
        the existing software with respect to the applicable project requirements,
        according to the criticality of the system function implemented.
<6>     Software reuse analysis conclusion
a.      The SRF shall document the results of the software reuse analysis.
b.      For each software item, the SRF shall provide the following information:
        1.      decision to reuse or not reuse, based on the information provided
                in previous chapters;
        2.      estimated level of reuse;
        3.      assumptions and methods applied when estimating the level of
                reuse.
<7>     Detailed results of evaluation
a.      The SRF shall include the detailed results of the evaluation.
        NOTE    The detailed results of the evaluation can be
                presented in an appendix.
<8>     Corrective actions
a.      The SRF shall document any corrective actions identified to ensure that
        the software intended for reuse meets the applicable project
        requirements.
b.      The SRF shall document the detailed results of the implementation of the
        identified corrective actions.
<9>     Configuration status
a.      The SRF shall include the detailed configuration status of the reused
        software baseline.
```

This will be provided as hand written content in Sphinx format.

Reused software running on the host computer falls under the tool selection scope, e.g. GNU Binutils, GCC. The assumption is that only software running on the target system will be covered by the SRF of the QDP. All software reused by RTEMS is declared as an integral part of RTEMS. It will be specified in the SRS and ICD. So, there will be no reused software in the QDP. This

makes the SRF quite trivial.

### 4.4.16 Software Problems Reports and Nonconformance Reports

See [EDI19b] and [EDI19a].

### 4.4.17 Joint Review Reports

Not included in the QDP.

### 4.4.18 Justification of Selection of Operational Ground Equipment and Support Services

Not included in the QDP. There is are no operational ground equipment and support services in the QDP.

## 4.5 Management File (MGT)

### 4.5.1 Software Development Plan (SDP)

This document will be provided as a copy of the latest SDP of the overall activity, see [EDI19d].

### 4.5.2 Software Review Plan (SRevP)

This document will be provided as a copy of the latest SRevP of the overall activity, see [EDI19g].

### 4.5.3 Software Configuration Management Plan (SCMP)

This document will be provided as a copy of the latest SCMP of the overall activity, see [EDI19b].

### 4.5.4 Training Plan

Not included in the QDP. No trainings planned in this activity.

### 4.5.5 Interface Management Procedures

The Interface Management Procedures will be included in the SCMP, see [EDI19b].

### 4.5.6 Identification of NRB SW and Members

The Identification of NRB SW and Members will be included in the SCMP, see [EDI19b].

### 4.5.7 Procurement Data

Not included in the QDP. No procurement planned in this activity.

## 4.6 Maintenance File (MF)

### 4.6.1 Maintenance Plan

Not included in the QDP, see *No Maintenance (MF)*.

### 4.6.2 Maintenance Records

Not included in the QDP, see *No Maintenance (MF)*.

### 4.6.3 SPR and NCR

Not included in the QDP, see *No Maintenance (MF)*.

### 4.6.4 Modification Analysis Report

Not included in the QDP, see *No Maintenance (MF)*.

### 4.6.5 Problem Analysis Report

Not included in the QDP, see *No Maintenance (MF)*.

### 4.6.6 Modification Documentation

Not included in the QDP, see *No Maintenance (MF)*.

### 4.6.7 Baseline for Change

Not included in the QDP, see *No Maintenance (MF)*.

### 4.6.8 Joint Review Reports

Not included in the QDP, see *No Maintenance (MF)*.

### 4.6.9 Migration Plan and Notification

Not included in the QDP, see *No Maintenance (MF)*.

### 4.6.10 Retirement Plan and Notification

Not included in the QDP, see *No Maintenance (MF)*.

## 4.7 Operational (OP)

### 4.7.1 Software Operation Support Plan

Not included in the QDP, see *No Operational Phase (OP)*.

### 4.7.2 Operational Testing Results

Not included in the QDP, see *No Operational Phase (OP)*.

### 4.7.3 SPR and NCR

Not included in the QDP, see *No Operational Phase (OP)*.

### 4.7.4 User's Request Record

Not included in the QDP, see *No Operational Phase (OP)*.

### 4.7.5 Post Operation Review Report

Not included in the QDP, see *No Operational Phase (OP)*.

## 4.8 Product Assurance File (PAF)

### 4.8.1 Software Product Assurance Plan (SPAP)

This document will be provided as a copy of the latest SPAP of the overall activity, see [EDI19e].

### 4.8.2 Software Product Assurance Requirements For Suppliers

The Software Product Assurance Requirements For Suppliers will be included in the SPAP, see [EDI19e].

### 4.8.3 Audit Plan and Schedule

The Audit Plan and Schedule be included in the SPAP, see [EDI19e].

### 4.8.4 Review and Inspection Plans or Procedures

The Review and Inspection Plans or Procedures will be included in the SPAP, see [EDI19e].

### 4.8.5 Procedures and Standards

The Procedures and Standards will be defined in the SPAP, see [EDI19e].

### 4.8.6 Modelling and Design Standards

The Modelling and Design Standards will be defined in the SPAP and the adherence will be reported in the SPAMR, see [EDI19e] and [EDI19d].

### 4.8.7 Coding Standards and Description of Tools

The Coding Standards and Description of Tools will be included in the SDP, see [EDI19c]. The SPAP defines the related product assurance activities, see [EDI19e].

### 4.8.8 Software Problem Reporting Procedure

The Software Problem Reporting Procedure is defined in the SCMP, see [EDI19b].

### 4.8.9 Software Dependability and Safety Analysis Report

The Software Dependability and Safety Analysis Report will be included in the SPAP, see [EDI19e].

### 4.8.10 Criticality Classification of Software Components

Not included in the QDP. Criticality defined by contract.

### 4.8.11 Software Product Assurance Report

Not included in the QDP. Only a SPAMR will be provided, see *Software Product Assurance Milestone Report (SPAMR)*.

### 4.8.12 Software Product Assurance Milestone Report (SPAMR)

The Software Product Assurance Milestone Report (SPAMR) is a part of the Product Assurance File (PAF) and its content is defined by ECSS-Q-ST-80C Rev.1 Annex C [ECS17d]. This section presents verbatim copies of the expected response from the standard highlighted as blocks followed by a content proposal for the QDP.

```
C.2     Expected response
C.2.1   Scope and content
<1>     Introduction
a.      The SPAMR shall contain a description of the purpose, objective, content
        and the reason prompting its preparation.
```

This will be provided as hand written content in Sphinx format.

```
<2>     Applicable and reference documents
a.      The SPAMR shall list the applicable and reference documents to support
        the generation of the document.
```

See *Applicable and Reference Documents*.

```
<3>     Terms, definitions and abbreviated terms
a.      The SPAMR shall include any additional terms, definition or abbreviated
        terms used.
```

See *Terms, Definitions and Abbreviated Terms*.

```
<4>     Verification activities performed
a.      The SPAMR shall contain reporting on verification activities performed
        by the product assurance function, including:
        1.      reviews;
```

This will be provided as hand written content in Sphinx format. The following content will be placed into this section:

> The review processes followed the planed procedures detailed in the Software Verification Plan (part of SDP-000 [EDI19c]) and in the Software Product Assurance Plan (SPAP-002 [EDI19e]). The reports for these reviews were produced according with SCF *Software Configuration File (SCF)*, SPAMR *Software Product Assurance Milestone Report (SPAMR)* and SVR *Software Verification Report (SVR)* deliverables proposals, as described in this document.

```
2.      inspections;
```

This will be provided as hand written content in Sphinx format and provide the report on the verification of the quality requirements inspection.

```
3.      walk-throughs;
```

Walk-throughs are not planed.

```
4.      review of traceability matrices;
```

The traceability matrices will be automatically checked and the results will be reported.

```
5.      documents reviewed.
```

The quality review for each QDP document will be provided as hand written content in Sphinx format. For each QDP document the following information will be presented:

- list of findings
- resolution status of each finding

```
b.      The SPAMR shall contain reporting on the verification of the measures
        applied for the handling of critical software.
```

This will be provided as hand written content in Sphinx format with the verification as specified in [ECS17d] section 6.2.3.

```
<5>     Methods and tools
a.      The SPAMR shall include or reference a justification of the suitability of
        the methods and tools applied in all the activities of the development
        cycle, including requirements analysis, software specification, design,
        coding, validation, testing, configuration management, verification and
        product assurance.
```

This will be provided as hand written content in Sphinx format. The tools listed in section 5.4 of [EDI19c] will be justified according to [ECS17d] section 5.6.1.2.

```
b.      The SPAMR shall include reporting on the correct use of methods and
        tools.
```

This will be provided as hand written content in Sphinx format with the verification as specified in [ECS17d] section 5.6.1.3.

```
<6>     Adherence to design and coding standards
a.      The SPAMR shall include reporting on the adherence of software
        products to the applicable modelling, design and coding standards,
        including:
        1.      reporting on the application of measures meant to ensure that the
                design complexity and modularity meet the quality requirements;
```

The Static Analyzer results will be presented in this section. Since this document will be available to RTEMS community, they may decide whether or not they apply the findings presented

here. Also, for the violations found a justification will be provided, explaining why the source code is still correct

```
2.      reporting on design documentation w.r.t. suitability for
        maintenance.
```

This section will contain the following text:

> RTEMS has been maintained by the RTEMS community for several decades. Consequently, the available design documentation has proven itself to be suitable for maintenance.
>
> RTEMS code was developed since the late 1980s under ever changing coding standards. The current code contains all these different coding styles. Changing these styles now would have grave negative effects on the maintainability. For example, changing the coding style will make it hard to figure out who has written a certain code in the past. Only new RTEMS source files adhere to the current coding standards as defined in [RTEb] section 6.3
>
> The project does not use models (see *No Logical and Computational Model*).

```
<7>     Product and process metrics
a.      The SPAMR shall include reporting on the collected product and process
        metrics, the relevant analyses performed, the corrective actions
        undertaken and the status of these actions.
b.      The results of the software maturity analysis shall also be reported.
```

This content will be provided mixed hand written and automatically generated for metrics which can be calculate using a tool. The list below stems from the table 5.1 *Quality model* of the Software Product Assurance Plan (SPAP-002 [EDI19e]):

- Requirements Allocation – Not applicable (see *No Requirements Baseline (RB)*).
- Requirement Implementation Coverage – Produced automatically
- Requirements Completeness – Produced automatically
- **V&V Coverage:**
  - Requirement Level – Produced automatically
  - Unit/Integration Level – Produced automatically
- Requirement Clarity – Hand written
- Suitability of Development Documentation – Hand written
- Adherence to Coding Standards – Not applicable (see discussion about *coding standard* above)
- SPR/NCR Status – Produced automatically
- Structural Coverage – Reference to the automatically generated coverage report in the SVR
- Requirement Testability - Produced automatically
- Cyclomatic Number – Produced automatically

- Nesting Level – Produced automatically
- Module Lines of Code – Produced automatically
- Comment Frequency – Produced automatically; Notes:
    1. The comment frequency will be calculated with the headers stripped from the source files (the header comments, which include authors, licenses, etc are not code comments)
    2. There are RTEMS files that violate the comment frequency threshold. The source code of these files is self-explaining. This means that these source files will not be corrected to cope with this metric.
- User Documentation Clarity – Hand written
- User Documentation Completeness – Produced automatically
- User Manual Adequacy/Suitability – Hand written
- Reuse Modification Rate – Hand written
- Safety activities adequacy – Hand written
- Milestone Tracking – Hand written
- Action Status and Code Size – Hand written as far as automatic production is impossible. Code size will be calculated automatically.

Each metric is described in and will be calculated in accordance with [ECS11].

```
<8>     Testing and validation
a.      The SPAMR shall include reporting on adequacy of the testing and
        validation documentation (including feasibility, traceability
        repeatability), and on the achieved test coverage w.r.t. stated goals.
```

This will be provided as hand written content in Sphinx format with the verifications as specified in [ECS17d] sections 6.3.5.3, 6.3.5.5 and 6.3.5.12

```
<9>     SPRs and SW NCRs
a.      The SPAMR shall include reporting on the status of software problem
        reports and nonconformances relevant to software.
```

This will be provided as hand written content in Sphinx format. This section should reference to the SRelD *Software Release Document (SRelD)*, which will contain the list of SPRs and NCRs.

```
<10>    References to progress reports
a.      Whenever relevant and up-to-date information has been already
        delivered as part of the regular PA progress reporting, a representative
        summary shall be provided, together with a detailed reference to the
        progress report(s) containing that information.
```

This will be provided as hand written content in Sphinx format.

### 4.8.13 Statement of Compliance With Test Plans and Procedures

The Statement of Compliance With Test Plans and Procedures will be included in the SPAMR, see [EDI19d].

### 4.8.14 Records of Training and Experience

Anonymized Records of Training and Experience may be annexed to the SPAMR [EDI19d]. Personal data should not be included in the QDP.

### 4.8.15 (Preliminary) Alert Information

Not included in the QDP.

### 4.8.16 Results of Pre-Award Audits and Assessments, and of Procurement Sources

Not included in the QDP. No procurement planned in this activity.

### 4.8.17 Software Process Assessment Plan

Process assessment will be performed internally by EDISOFT and reported in the SPAMR, see [EDI19d].

### 4.8.18 Software Process Assessment Report

The Software Process Assessment Report will be provided as an annex to the SPAMR, see [EDI19d].

### 4.8.19 Review and Inspection Reports

The proposal is to maintain review and inspection reports with *Requirement Validation Item Type* specification items.

### 4.8.20 Receiving Inspection Report

The Receiving Inspection Reports will be included in the SPAMR, see [EDI19d].

### 4.8.21 Input to Product Assurance Plan for Systems Operation

Not included in the QDP, see *No Operational Phase (OP)*.

CHAPTER

FIVE

# QUALIFICATION DATA PACKAGE

## 5.1 Variants

The *QDP* will be delivered in four variants (two *BSP*, two build configurations):

- BSP `sparc/gr712rc` configured with `--disable-smp`
- BSP `sparc/gr712rc` configured with `--enable-smp`
- BSP `sparc/gr740` configured with `--disable-smp`
- BSP `sparc/gr740` configured with `--enable-smp`

## 5.2 Content

This section proposes the QDP content. It will be included in a GNU tar archive compressed with xz. The proposed file name is `rtems-qdp-<arch>-<bsp>-<config>-<version>.tar.xz` with

- `<arch>` being `sparc`,
- `<bsp>` being one of `gr712rc` and `gr740`,
- `<config>` being one of `uni` and `smp`, and
- `<version>` being an integer starting at zero which is incremented with each generation of the QDP variant.

The directory and file structure should follow this incomplete proposal:

- doc
  - ts
    - srs: *Software Requirements Specification (SRS)*
      - srs.pdf
      - html/index.html
    - icd: *Software Interface Control Document (ICD)*
      - icd.pdf
      - html/index.html

- ddf
  - sdd: *Software Design Document (SDD)*
    - sdd.pdf: Doxygen generated PDF output
    - html/index.html: Doxygen generated HTML output
  - sreld: *Software Release Document (SRelD)*
    - sreld.pdf
    - html/index.html
- djf
  - suitp: *Software Unit and Integration Test Plan (SUITP)*
    - suitp.pdf
    - html/index.html
  - svsts: *Software Validation Specification (SVS) with Respect to TS*
    - svsts.pdf
    - html/index.html
  - svr: *Software Verification Report (SVR)*
    - svr.pdf
    - html/index.html
  - srf: *Software Reuse File (SRF)*
    - srf.pdf
    - html/index.html
- mgt
  - sdp: *Software Development Plan (SDP)*
    - sdp.pdf
    - html/index.html
  - srevp: *Software Review Plan (SRevP)*
    - srevp.pdf
    - html/index.html
  - scmp: *Software Configuration Management Plan (SCMP)*
    - scmp.pdf
    - html/index.html
- paf
  - spap: *Software Product Assurance Plan (SPAP)*
    - spap.pdf

        - html/index.html
      * spamr: *Software Product Assurance Milestone Report (SPAMR)*
        - spamr.pdf
        - html/index.html
  - technical-notes
    * tn-qt.pdf: this document
    * tn-ti.pdf: [EDI19i]
  - rtems: standard RTEMS Project documentation
    * user : adopted as the RTEMS QDP User Manual
      - user.pdf
      - html/index.html
    * c-user
      - c-user.pdf
      - html/index.html
    * eng
      - eng.pdf
      - html/index.html
    * cpu-supplement
      - cpu-supplement.pdf
      - html/index.html
  - fm: Formal Methods documentation
    * fvp
      - FV1-200.pdf
      - html/index.html
    * fva
      - FV2-201.pdf
      - html/index.html
    * fvr
      - FV3-202.pdf
      - html/index.html
- src: source code (*Software Source Code and Media Labels*)
  - example: a project with an example RTEMS application and build files
  - rtems: software product (*Software Product and Media Labels*)

CHAPTER

SIX

# WORK ITEMS

## 6.1 Traceability

In the relevant ECSS standards for this activity, traceability from a source item to a destination item is mentioned in several clauses and normative document scope and content sections. This section collects all the items subject to traceability from the ECSS-E-ST-40C [ECS09b] and ECSS-Q-ST-80C Rev.1 [ECS17d] standards. For each link from a source item to a destination item a technical solution is outlined if it is applicable to the QDP. The items need a clear definition for this activity. They need a unique, human and machine readable identifier.

The wording of items in the standard is context sensitive. In the table the full item names are given. For example in ECSS-E-ST-40C 5.8.3.4.a.3 we have:

> "the traceability between the architecture and the detailed design is complete;"

From the context it is clear that software architecture and software detailed design is meant. This may not be that clear in every clause of the standards.

In ECSS-Q-ST-80C Rev.1 traceability is mentioned in *Reuse of existing software (6.2.7.4a)* and *Testing and validation (6.3.5.3a)*. These two clauses define goals of product assurance activities. Several ECSS-E-ST-40C clauses ensure that a software product engineered according to this standard meets these goals.

The following table presents all items subject to traceability relevant for the QDP. Definitions in quotes ("...") are verbatim copies from the referenced standard.

Table 1: Traceability Item Definitions

| Item | Reference | Definition | Realization in QDP |
|---|---|---|---|
| acceptance test | ECSS-S-ST-00-01C Annex A | WARNING: Undefined in this standard. | *No Requirements Baseline (RB)* |
| analysis | ECSS-S-ST-00-01C 2.3.8 | "verification method utilizing techniques and tools to confirm that verification requirements have been satisfied" | *Requirement Validation* |
| feature | | | *feature* or *software unit* |

continues on next page

Table 1 – continued from previous page

| Item | Reference | Definition | Realization in QDP |
|---|---|---|---|
| inspection | ECSS-S-ST-00-01C 2.3.110 | “conformance evaluation by observation and judgement accompanied as appropriate by measurement, testing or gauging [ISO 9000:2005]” | *Requirement Validation* |
| integration | ECSS-S-ST-00-01C 2.3.111 | “functionally combining lower-level functional entities (hardware or software) so they operate together to constitute a higher-level functional entity” | |
| interface | ECSS-S-ST-00-01C 2.3.113 | “interface boundary where two or more products meet and interact” | *Interface Item Type* |
| interface requirement | | See items *interface* and *requirement*. | *Non-Functional Requirement Item Type* |
| object code | | | SPARC machine code in ELF file |
| requirement | ECSS-S-ST-00-01C 2.3.173 | “documented demand to be complied with” | *Specification Items* |
| require-ments baseline | ECSS-E-ST-40C Annex B (SSS); ECSS-E-ST-40C Annex C (IRD) | See references. | *No Requirements Baseline (RB)* |
| review | ECSS-S-ST-00-01C 2.3.175 | “activity undertaken to determine the suitability, adequacy and effectiveness of the subject matter to achieve established objectives” | |
| review of software design | | See items *review* and *software design*. | *Requirement Validation* |
| software | ECSS-E-ST-40C 3.2.27 | “software product” | |
| software architecture | ECSS-E-ST-40C 5.4.3 | | Doxygen markup throughout the header, source and assembler files; see *Software Design Document (SDD)* |

continues on next page

Table 1 – continued from previous page

| Item | Reference | Definition | Realization in QDP |
|---|---|---|---|
| software component | ECSS-E-ST-40C 3.2.28 | “part of a software system” | *software component* |
| software design | ECSS-E-ST-40C 5.5.2 | | Doxygen markup throughout the header, source and assembler files; see *Software Design Document (SDD)* |
| software integration | | See items *software* and *integration*. | |
| software item | ECSS-E-ST-40C 3.2.30 | “software product” | |
| software product | ECSS-S-ST-00-01C Annex A | WARNING: Undefined in this standard. | subset of RTEMS (space profile); see *software product* |
| software requirement | | See items *software* and *requirement*. | *Specification Items* |
| software unit | ECSS-E-ST-40C 3.2.34 | “separately compilable piece of source code” | *software unit* |
| source code | | | *source code* |
| system | ECSS-S-ST-00-01C 2.3.212 | “set of interrelated or interacting functions constituted to achieve a specified objective | |
| system requirement | | See items *system* and *requirement*. | *No Requirements Baseline (RB)* |
| test | ECSS-E-ST-40C 2.3.215 | “measurement of product characteristics, performance or functions under representative environments” | |
| test case | ECSS-E-ST-40C 3.2.37 | “set of test inputs, execution conditions and expected results developed for a particular objective such as to exercise a particular program path or to verify compliance with a specified requirement” | *Test Case Item Type* |
| test code | | See items *test* and *source code*. | RTEMS Test Framework test case code |

continues on next page

Table 1 – continued from previous page

| Item | Reference | Definition | Realization in QDP |
|---|---|---|---|
| test procedure | ECSS-E-ST-40C 3.2.39 | "detailed instructions for the set up, operation and evaluation of the results for a given test" | *Test Procedure Item Type* |
| validation result | | | *Requirement Validation* |

The following table presents a complete list of all clauses and normative scope and content sections which mention the traceability from a source item to a destination item in ECSS-E-ST-40C. The reference column specifies the corresponding clause or normative document scope and content section. For readers not knowing all the clauses off by heart a subject is given. The source item and destination item columns define the traceability relationship (link from source to destination). The item names refer to the table above. The last column gives an outline of the solution provided by the QDP to satisfy the ECSS requirement.

Table 2: Traceability from Item to Item in ECSS-E-ST-40C

| Reference | Subject | Source | Destination | Solution in QDP |
|---|---|---|---|---|
| *5.7.3.5a* | Evaluation of acceptance testing | acceptance test | require-ments baseline | *No Requirements Baseline (RB)* |
| *5.8.3.2a* 2 | Verification of the technical specification | software requirement | system requirement | *No Requirements Baseline (RB)* |
| *5.8.3.3a* 3 | Verification of the software architectural design | software component | software requirement | *Traceability between Software Requirements, Architecture and Design* |
| *5.8.3.4a* 3 | Verification of the software detailed design | software design | software architecture | *Traceability between Software Requirements, Architecture and Design* |
| *5.8.3.5a* 3 | Verification of code | source code | software design | *Traceability between Software Requirements, Architecture and Design* |
| *5.8.3.5a* 3 | Verification of code | source code | software requirement | *Traceability between Software Requirements, Architecture and Design* |
| *5.8.3.5e* | Verification of code | object code | source code | N/A (only required for criticality A software) |
| *5.8.3.6a* 2 | Verification of software unit testing (plan and results) | test case | software re-quirements | each software unit references a software requirement; see *Software Design Document (SDD)* |

continues on next page

Table 2 – continued from previous page

| Reference | Subject | Source | Destination | Solution in QDP |
|---|---|---|---|---|
| *5.8.3.6a* 2 | Verification of software unit testing (plan and results) | test case | software design | each software unit is contained in a Doxygen group representing the software design component; see *Software Design Document (SDD)* |
| *5.8.3.6a* 2 | Verification of software unit testing (plan and results) | test case | source code | each test case references a *software unit*; see *Test Case Item Type* |
| *5.8.3.7a* 1 | Verification of software integration | software integration | software architecture | traceability is provided by the *Software Design Document (SDD)* through standard Doxygen means; a software integration is documented through Doxygen groups which are contained in software architecture Doxygen groups |
| *5.8.3.8a* eo.a | Verification of software validation with respect to the technical specifications and the baseline requirements | validation result | software requirement | *No Requirements Baseline (RB)* |
| *5.8.3.8a* eo.b | Verification of software validation with respect to the technical specifications and the baseline requirements | validation result | software requirement | *Requirement Validation* |
| *Annex D (SRS)* <5.1>.e | Requirements - General | requirement | system requirement | *No Requirements Baseline (RB)* |
| *Annex D (SRS)* <7>.a.1 | Traceability | requirement | requirement | *Forward Traceability of Specification Items* |
| *Annex D (SRS)* <7>.a.2 | Traceability | requirement | requirement | *Backward Traceability of Specification Items* |

continues on next page

Table 2 – continued from previous page

| Reference | Subject | Source | Destination | Solution in QDP |
|---|---|---|---|---|
| *Annex E (ICD)* <5.1>.c | General provisions to the requirements in the IRD | requirement | system requirement | *No Requirements Baseline (RB)* |
| *Annex E (ICD)* <7>.a.1 | Traceability | requirement | requirement | *Forward Traceability of Specification Items* |
| *Annex E (ICD)* <7>.a.2 | Traceability | requirement | requirement | *Backward Traceability of Specification Items* |
| *Annex F (SDD)* <5.3>.b.3 | Software components design - General | software component | software requirement | *Traceability between Software Requirements, Architecture and Design* |
| *Annex F (SDD)* <5.4.4>.a | Software components design - Aspects of each component - Purpose | software component | software requirement | *Traceability between Software Requirements, Architecture and Design* |
| *Annex F (SDD)* <6>.a.1 | Requirements to design components traceability | software requirement | software component | *Traceability between Software Requirements, Architecture and Design* |
| *Annex F (SDD)* <6>.a.1 | Requirements to design components traceability | software component | software component | *Traceability between Software Requirements, Architecture and Design* |
| *Annex F (SDD)* <6>.a.2 | Requirements to design components traceability | software component | software component | *Traceability between Software Requirements, Architecture and Design* |
| *Annex F (SDD)* <6>.a.2 | Requirements to design components traceability | software component | software requirement | *Traceability between Software Requirements, Architecture and Design* |
| *Annex K (SUITP)* <8.2.2>.b | Organization of each identified test design | test design | feature | the feature is a *software unit* in the SUITP; obtained through the references in the *Test Case Item Type* specification items of the test design (*Test Suite Item Type*) |
| *Annex K (SUITP)* <9.2.2>.a | Organization of each identified test case | test case | test item | reference in the *Test Case Item Type* |

continues on next page

Table 2 – continued from previous page

| Reference | Subject | Source | Destination | Solution in QDP |
|---|---|---|---|---|
| *Annex K (SUITP)* <11>.a.1 | Software test plan additional information | test procedure | test case | *Forward Traceability of Specification Items* |
| *Annex K (SUITP)* <11>.a.2 | Software test plan additional information | test case | test procedure | *Backward Traceability of Specification Items* |
| *Annex L (SVS)* <6.2.2>.b.2 | Organization of each identified test design | test design | feature | obtained through the references in the *Test Case Item Type* specification items of the test design (*Test Suite Item Type*); see *feature* |
| *Annex L (SVS)* <11>.a.1 | Software validation specification additional information | analysis | software requirement | *Requirement Validation* and *Backward Traceability of Specification Items* |
| *Annex L (SVS)* <11>.a.1 | Software validation specification additional information | analysis | system requirement | *No Requirements Baseline (RB)* |
| *Annex L (SVS)* <11>.a.1 | Software validation specification additional information | inspection | software requirement | *Requirement Validation* and *Backward Traceability of Specification Items* |
| *Annex L (SVS)* <11>.a.1 | Software validation specification additional information | inspection | system requirement | *No Requirements Baseline (RB)* |
| *Annex L (SVS)* <11>.a.1 | Software validation specification additional information | review of software design | software requirement | *Requirement Validation* and *Backward Traceability of Specification Items* |
| *Annex L (SVS)* <11>.a.1 | Software validation specification additional information | review of software design | system requirement | *No Requirements Baseline (RB)* |

continues on next page

Table 2 – continued from previous page

| Reference | Subject | Source | Destination | Solution in QDP |
|---|---|---|---|---|
| *Annex L (SVS)* <11>.a.1 | Software validation specification additional information | test case | software requirement | *Test Case Item Type* and *Backward Traceability of Specification Items* |
| *Annex L (SVS)* <11>.a.1 | Software validation specification additional information | test case | system requirement | *No Requirements Baseline (RB)* |
| *Annex L (SVS)* <11>.a.2 | Software validation specification additional information | software requirement | analysis | *Requirement Validation* and *Forward Traceability of Specification Items* |
| *Annex L (SVS)* <11>.a.2 | Software validation specification additional information | system requirement | analysis | *No Requirements Baseline (RB)* |
| *Annex L (SVS)* <11>.a.2 | Software validation specification additional information | software requirement | inspection | *Requirement Validation* and *Forward Traceability of Specification Items* |
| *Annex L (SVS)* <11>.a.2 | Software validation specification additional information | system requirement | inspection | *No Requirements Baseline (RB)* |
| *Annex L (SVS)* <11>.a.2 | Software validation specification additional information | software requirement | review of software design | *Requirement Validation* and *Forward Traceability of Specification Items* |
| *Annex L (SVS)* <11>.a.2 | Software validation specification additional information | system requirement | review of software design | *No Requirements Baseline (RB)* |
| *Annex L (SVS)* <11>.a.2 | Software validation specification additional information | software requirement | test case | *Test Case Item Type* and *Forward Traceability of Specification Items* |

continues on next page

Table 2 – continued from previous page

| Reference | Subject | Source | Destination | Solution in QDP |
|---|---|---|---|---|
| *Annex L (SVS)* <11>.a.2 | Software validation specification additional information | system requirement | test case | *No Requirements Baseline (RB)* |
| *Annex L (SVS)* <11>.a.3 | Software validation specification additional information | test procedure | test case | *Test Procedure Item Type*, *Test Case Item Type*, and *Forward Traceability of Specification Items* |
| *Annex L (SVS)* <11>.a.4 | Software validation specification additional information | test case | test procedure | *Test Case Item Type*, *Test Procedure Item Type*, and *Backward Traceability of Specification Items* |
| *Annex M (SVR)* <4.3.1>.a | Software requirements and architecture engineering process verification (for the PDR) | software requirement | system requirement | *No Requirements Baseline (RB)* |
| *Annex M (SVR)* <4.3.1>.a | Software requirements and architecture engineering process verification (for the PDR) | software architecture | software requirement | provided in *Software Design Document (SDD)* |

## 6.2 Software Requirements Engineering

**Warning:** This section is supposed to be in synchronization with the corresponding chapter in the RTEMS Software Engineering manual. The content here may be not up to date.

Software engineering standards for critical software such as ECSS-E-ST-40C demand that software requirements for a software product are collected in a software requirements specification (technical specification in ECSS-E-ST-40C terms). They are usually derived from system requirements (requirements baseline in ECSS-E-ST-40C terms). RTEMS is designed as a reusable software product which can be utilized by application designers to ease the development of their applications. The requirements of the end system (system requirements) using RTEMS are only known to the application designer. RTEMS itself is developed by the RTEMS maintainers and they do not know the requirements of a particular end system in general. RTEMS is designed as a real-time operating system to meet typical system requirements for a wide range of applica-

tions. Its suitability for a particular application must be determined by the application designer based on the technical specification provided by RTEMS accompanied with performance data for a particular target platform.

Currently, no technical specification of RTEMS exists in the form of a dedicated document. Since the beginning of the RTEMS evolution in the late 1980s it was developed iteratively. It was never developed in a waterfall model. During initial development the RTEID [Mot88] and later the ORKID [VIT90] draft specifications were used as requirements. These were evolving during the development and an iterative approach was followed often using simple algorithms and coming back to optimise. In 1993 and 1994 a subset of pthreads sufficient to support *GNAT* was added as requirements. At this time the Ada tasking was defined, however, not implemented in GNAT, so this involved guessing during the development. Later some adjustments were made when Ada tasking was actually implemented. So, it was consciously iterative with the specifications evolving and feedback from performance analysis. Benchmarks published from other real time operating systems were used for comparison. Optimizations were carried out until the results were comparable. Development was done with distinct contractual phases and tasks for development, optimization, and the addition of priority inheritance and rate monotonic scheduling. The pthreads requirement has grown to be as much POSIX as possible.

Portability from FreeBSD to use its network stack, USB stack, SD/MMC card stack and device drivers resulted in another set of requirements. The addition of support for symmetric multiprocessing (SMP) was a huge driver for change. It was developed step by step and sponsored by several independent users with completely different applications and target platforms in mind. The high performance OpenMP support introduced the Futex as a new synchronization primitive.

**Guidance**

A key success element of RTEMS is the ability to accept changes driven by user needs and still keep the operating system stable enough for production systems. Procedures that place a high burden on changes are doomed to be discarded by the RTEMS Project. We have to keep this in mind when we introduce a requirements management work flow which should be followed by RTEMS community members and new contributors.

We have to put in some effort first into the reconstruction of software requirements through reverse engineering using the RTEMS documentation, test cases, sources, standard references, mailing list archives, etc. as input. Writing a technical specification for the complete RTEMS code base is probably a job of several person-years. We have to get started with a moderate feature set (e.g. subset of the Classic API) and extend it based on user demands step by step.

The development of the technical specification will take place in two phases. The first phase tries to establish an initial technical specification for an initial feature set. This technical specification will be integrated into RTEMS as a big chunk. In the second phase the technical specification is modified through arranged procedures. There will be procedures

- to modify existing requirements,
- add new requirements, and
- mark requirements as obsolete.

All procedures should be based on a peer review principles.

### 6.2.1 Requirements for Requirements

#### 6.2.1.1 Identification

Each requirement shall have a unique identifier (UID). The question is in which scope should it be unique? Ideally, it should be universally unique. Therefore all UIDs used to link one specification item to another should use relative UIDs. This ensures that the RTEMS requirements can be referenced easily in larger systems though a system-specific prefix. The standard ECSS-E-ST-10-06C recommends in section 8.2.6 that the identifier should reflect the type of the requirement and the life profile situation. Other standards may have other recommendations. To avoid a bias of RTEMS in the direction of ECSS, this recommendation will not be followed.

The *absolute UID* of a specification item (for example a requirement) is defined by a leading `/` and the path of directories from the specification base directory to the file of the item separated by `/` characters and the file name without the `.yml` extension. For example, a specification item contained in the file `build/cpukit/librtemscpu.yml` inside a `spec` directory has the absolute UID of `/build/cpukit/librtemscpu`.

The *relative UID* to a specification item is defined by the path of directories from the file containing the source specification item to the file of the destination item separated by `/` characters and the file name of the destination item without the `.yml` extension. For example the relative UID from `/build/bsps/sparc/leon3/grp` to `/build/bsps/bspopts` is `../../bspopts`.

Basically, the valid characters of an UID are determined by the file system storing the item files. By convention, UID characters shall be restricted to the following set defined by the regular expression `[a-zA-Z0-9_-]+`. Use `-` as a separator inside an UID part.

In documents the URL-like prefix `spec:` shall be used to indicated specification item UIDs.

The UID scheme for RTEMS requirements shall be component based. For example, the UID `spec:/classic/task/create-err-invaddr` may specify that the `rtems_task_create()` directive shall return a status of `RTEMS_INVALID_ADDRESS` if the `id` parameter is `NULL`.

A initial requirement item hierarchy could be this:

- build (building RTEMS BSPs and libraries)
- acfg (application configuration groups)
  - opt (application configuration options)
- classic
  - task
    - create-* (requirements for `rtems_task_create()`)
    - delete-* (requirements for `rtems_task_delete()`)
    - exit-* (requirements for `rtems_task_exit()`)
    - getaff-* (requirements for `rtems_task_get_affinity()`)
    - getpri-* (requirements for `rtems_task_get_priority()`)
    - getsched-* (requirements for `rtems_task_get_scheduler()`)
    - ident-* (requirements for `rtems_task_ident()`)

        * issusp-* (requirements for `rtems_task_is_suspended()`)
        * iter-* (requirements for `rtems_task_iterate()`)
        * mode-* (requirements for `rtems_task_mode()`)
        * restart-* (requirements for `rtems_task_restart()`)
        * resume* (requirements for `rtems_task_resume()`)
        * self* (requirements for `rtems_task_self()`)
        * setaff-* (requirements for `rtems_task_set_affinity()`)
        * setpri-* (requirements for `rtems_task_set_priority()`)
        * setsched* (requirements for `rtems_task_set_scheduler()`)
        * start-* (requirements for `rtems_task_start()`)
        * susp-* (requirements for `rtems_task_suspend()`)
        * wkafter-* (requirements for `rtems_task_wake_after()`)
        * wkwhen-* (requirements for `rtems_task_wake_when()`)
    - sema
        * ...
- posix
- ...

A more detailed naming scheme and guidelines should be established. We have to find the right balance between the length of UIDs and self-descriptive UIDs. A clear scheme for all Classic API managers may help to keep the UIDs short and descriptive.

The specification of the validation of requirements should be maintained also by specification items. For each requirement directory there should be a validation subdirectory named *test*, e.g. `spec/classic/task/test`. A test specification directory may contain also validations by analysis, by inspection, and by design, see *Requirement Validation*.

#### 6.2.1.2 Level of Requirements

The level of a requirement shall be expressed with one of the verbal forms listed below and nothing else. The level of requirements are derived from RFC 2119 [Bra97] and ECSS-E-ST-10-06C [ECS09a].

#### 6.2.1.2.1 Absolute Requirements

Absolute requirements shall be expressed with the verbal form *shall* and no other terms.

#### 6.2.1.2.2 Absolute Prohibitions

Absolute prohibitions shall be expressed with the verbal form *shall not* and no other terms.

> **Warning:** Absolute prohibitions may be difficult to validate. They should not be used.

#### 6.2.1.2.3 Recommendations

Recommendations shall be expressed with the verbal forms *should* and *should not* and no other terms with guidance from RFC 2119:

> SHOULD This word, or the adjective “RECOMMENDED”, mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
>
> SHOULD NOT This phrase, or the phrase “NOT RECOMMENDED” mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

#### 6.2.1.2.4 Permissions

Permissions shall be expressed with the verbal form *may* and no other terms with guidance from RFC 2119:

> MAY This word, or the adjective “OPTIONAL”, mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

#### 6.2.1.2.5 Possibilities and Capabilities

Possibilities and capabilities shall be expressed with the verbal form *can* and no other terms.

### 6.2.1.3 Syntax

Use the Easy Approach to Requirements Syntax (*EARS*) to formulate requirements. A recommended reading list to get familiar with this approach is [MWHN09], [MW10], [MWGU16], and Alisair Mavin's web site. The patterns are:

- Ubiquitous

  The <system name> shall <system response>.

- Event-driven

  **When** <trigger>, the <system name> shall <system response>.

- State-driven

  **While** <pre-condition>, the <system name> shall <system response>.

- Unwanted behaviour

  **If** <trigger>, **then** the <system name> shall <system response>.

- Optional

  **Where** <feature is included>, the <system name> shall <system response>.

- Complex

  **Where** <feature 0 is included>, **where** <feature 1 is included>, ..., **where** <feature *n* is included>, **while** <pre-condition 0>, **while** <pre-condition 1>, ..., **while** <pre-condition *m*>, **when** <trigger>, the <system name> shall <system response>.

  **Where** <feature 0 is included>, **where** <feature 1 is included>, ..., **where** <feature *n* is included>, **while** <pre-condition 0>, **while** <pre-condition 1>, ..., **while** <pre-condition *m*>, **if** <trigger>, **then** the <system name> shall <system response>.

The optional pattern should be only used for application configuration options. The goal is to use the `enabled-by` attribute to enable or disable requirements based on configuration parameters that define the RTEMS artefacts used to build an application executable (header files, libraries, linker command files). Such configuration parameters are for example the architecture, the platform, CPU port options, and build configuration options (e.g. uniprocessor vs. SMP).

#### 6.2.1.4 Wording Restrictions

To prevent the expression of imprecise requirements, the following terms shall not be used in requirement formulations:

- “acceptable”
- “adequate”
- “almost always”
- “and/or”
- “appropriate”
- “approximately”
- “as far as possible”
- “as much as practicable”
- “best”
- “best possible”
- “easy”
- “efficient”
- “e.g.”
- “enable”
- “enough”
- “etc.”
- “few”
- “first rate”
- “flexible”
- “generally”
- “goal”
- “graceful”
- “great”
- “greatest”
- “ideally”
- “i.e.”
- “if possible”
- “in most cases”
- “large”
- “many”

- “maximize”
- “minimize”
- “most”
- “multiple”
- “necessary”
- “numerous”
- “optimize”
- “ought to”
- “probably”
- “quick”
- “rapid”
- “reasonably”
- “relevant”
- “robust”
- “satisfactory”
- “several”
- “shall be included but not limited to”
- “simple”
- “small”
- “some”
- “state-of-the-art”.
- “sufficient”
- “suitable”
- “support”
- “systematically”
- “transparent”
- “typical”
- “user-friendly”
- “usually”
- “versatile”
- “when necessary”

For guidelines to avoid these terms see Table 11-2, “Some ambiguous terms to avoid in requirements” in [WB13]. There should be some means to enforce that these terms are not used, e.g.

through a client-side pre-commit Git hook, a server-side pre-receive Git hook, or some scripts run by special build commands.

#### 6.2.1.5 Separate Requirements

Requirements shall be stated separately. A bad example is:

**spec:/classic/task/create** The task create directive shall evaluate the parameters, allocate a task object and initialize it.

To make this a better example, it should be split into separate requirements:

**spec:/classic/task/create** When the task create directive is called with valid parameters and a free task object exists, the task create directive shall assign the identifier of an initialized task object to the `id` parameter and return the `RTEMS_SUCCESSFUL` status.

**spec:/classic/task/create-err-toomany** If no free task objects exists, the task create directive shall return the `RTEMS_TOO_MANY` status.

**spec:/classic/task/create-err-invaddr** If the `id` parameter is `NULL`, the task create directive shall return the `RTEMS_INVALID_ADDRESS` status.

**spec:/classic/task/create-err-invname** If the `name` parameter is invalid, the task create directive shall return the `RTEMS_INVALID_NAME` status.

...

#### 6.2.1.6 Conflict Free Requirements

Requirements shall not be in conflict with each other inside a specification. A bad example is:

**spec:/classic/sema/mtx-obtain-wait** When a mutex is not available, the mutex obtain directive shall enqueue the calling thread on the wait queue of the mutex.

**spec:/classic/sema/mtx-obtain-err-unsat** If a mutex is not available, the mutex obtain directive shall return the RTEMS_UNSATISFIED status.

To resolve this conflict, a condition may be added:

**spec:/classic/sema/mtx-obtain-wait** When a mutex is not available and the RTEMS_WAIT option is set, the mutex obtain directive shall enqueue the calling thread on the wait queue of the mutex.

**spec:/classic/sema/mtx-obtain-err-unsat** If a mutex is not available, when the RTEMS_WAIT option is not set, the mutex obtain directive shall return the RTEMS_UNSATISFIED status.

#### 6.2.1.7 Use of Project-Specific Terms and Abbreviations

All project-specific terms and abbreviations used to formulate requirements shall be defined in the project glossary.

#### 6.2.1.8 Justification of Requirements

Each requirement shall have a rationale or justification recorded in a dedicated section of the requirement file. See `rationale` attribute for *Specification Items*.

#### 6.2.1.9 Requirement Validation

The validation of each *Requirement Item Type* item shall be accomplished by one or more specification items of the types *Test Case Item Type* or *Requirement Validation Item Type* through a link from the validation item to the requirement item with the *Requirement Validation Link Role*.

Validation by test is strongly recommended. The choice of any other validation method shall be strongly justified. The requirements author is obligated to provide the means to validate the requirement with detailed instructions.

#### 6.2.1.10 Resources and Performance

Normally, resource and performance requirements are formulated like this:

- The resource U shall need less than V storage units.
- The operation Y shall complete within X time units.

Such statements are difficult to make for a software product like RTEMS which runs on many different target platforms in various configurations. So, the performance requirements of RTEMS shall be stated in terms of benchmarks. The benchmarks are run on the project-specific target platform and configuration. The results obtained by the benchmark runs are reported in a human readable presentation. The application designer can then use the benchmark results to determine if its system performance requirements are met. The benchmarks shall be executed under different environment conditions, e.g. varying cache states (dirty, empty, valid) and system bus load generated by other processors. The application designer shall have the ability to add additional environment conditions, e.g. system bus load by DMA engines or different system bus arbitration schemes.

To catch resource and performance regressions via test suite runs there shall be a means to specify threshold values for the measured quantities. The threshold values should be provided for each validation platform. How this can be done and if the threshold values are maintained by the RTEMS Project is subject to discussion.

### 6.2.2 Specification Items

#### 6.2.2.1 Specification Item Hierarchy

The specification item types have the following hierarchy:

- *Root Item Type*
  - *Build Item Type*
    - *Build Ada Test Program Item Type*
    - *Build BSP Item Type*
    - *Build Configuration File Item Type*
    - *Build Configuration Header Item Type*
    - *Build Group Item Type*
    - *Build Library Item Type*
    - *Build Objects Item Type*
    - *Build Option Item Type*
    - *Build Script Item Type*
    - *Build Start File Item Type*
    - *Build Test Program Item Type*
  - *Constraint Item Type*
  - *Glossary Item Type*
    - *Glossary Group Item Type*
    - *Glossary Term Item Type*
  - *Interface Item Type*
    - *Application Configuration Group Item Type*
    - *Application Configuration Option Item Type*
      - *Application Configuration Feature Enable Option Item Type*
      - *Application Configuration Feature Option Item Type*
      - *Application Configuration Value Option Item Type*
    - *Interface Compound Item Type*
    - *Interface Container Item Type*
    - *Interface Define Item Type*
    - *Interface Domain Item Type*
    - *Interface Enum Item Type*
    - *Interface Enumerator Item Type*

    - *Interface Forward Declaration Item Type*
    - *Interface Function Item Type*
    - *Interface Group Item Type*
    - *Interface Header File Item Type*
    - *Interface Macro Item Type*
    - *Interface Typedef Item Type*
    - *Interface Unspecified Item Type*
    - *Interface Variable Item Type*
  - *Requirement Item Type*
    - *Functional Requirement Item Type*
      - *Action Requirement Item Type*
      - *Generic Functional Requirement Item Type*
    - *Non-Functional Requirement Item Type*
      - *Design Group Requirement Item Type*
      - *Generic Non-Functional Requirement Item Type*
      - *Runtime Performance Requirement Item Type*
  - *Requirement Validation Item Type*
  - *Runtime Measurement Test Item Type*
  - *Specification Item Type*
  - *Test Case Item Type*
  - *Test Platform Item Type*
  - *Test Procedure Item Type*
  - *Test Suite Item Type*

#### 6.2.2.2 Specification Item Types

##### 6.2.2.2.1 Root Item Type

The technical specification of RTEMS will contain for example requirements, specializations of requirements, interface specifications, test suites, test cases, and requirement validations. These things will be called *specification items* or just *items* if it is clear from the context.

The specification items are stored in files in *YAML* format with a defined set of key-value pairs called attributes. Each attribute key name shall be a *Name*. In particular, key names which begin with an underscore (_) are reserved for internal use in tools.

This is the root specification item type. All explicit attributes shall be specified. The explicit attributes for this type are:

**SPDX-License-Identifier** The attribute value shall be a *SPDX License Identifier*. It shall be the license of the item.

**copyrights** The attribute value shall be a list. Each list element shall be a *Copyright*. It shall be the list of copyright statements of the item.

**enabled-by** The attribute value shall be an *Enabled-By Expression*. It shall define the conditions under which the item is enabled.

**links** The attribute value shall be a list. Each list element shall be a *Link*.

**type** The attribute value shall be a *Name*. It shall be the item type. The selection of types and the level of detail depends on a particular standard and product model. We need enough flexibility to be in line with ECSS-E-ST-10-06 and possible future applications of other standards. The item type may be refined further with additional type-specific subtypes.

This type is refined by the following types:

- *Build Item Type*
- *Constraint Item Type*
- *Glossary Item Type*
- *Interface Item Type*
- *Requirement Item Type*
- *Requirement Validation Item Type*
- *Runtime Measurement Test Item Type*
- *Specification Item Type*
- *Test Case Item Type*
- *Test Platform Item Type*
- *Test Procedure Item Type*
- *Test Suite Item Type*

#### 6.2.2.2.2 Build Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `build`. This set of attributes specifies a build item. All explicit attributes shall be specified. The explicit attributes for this type are:

**build-type** The attribute value shall be a *Name*. It shall be the build item type.

This type is refined by the following types:

- *Build Ada Test Program Item Type*
- *Build BSP Item Type*
- *Build Configuration File Item Type*
- *Build Configuration Header Item Type*

- *Build Group Item Type*
- *Build Library Item Type*
- *Build Objects Item Type*
- *Build Option Item Type*
- *Build Script Item Type*
- *Build Start File Item Type*
- *Build Test Program Item Type*

#### 6.2.2.2.3 Build Ada Test Program Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `ada-test-program`. This set of attributes specifies an Ada test program executable to build. Test programs may use additional objects provided by *Build Objects Item Type* items. Test programs have an implicit `enabled-by` attribute value which is controlled by the option action *set-test-state*. If the test state is set to `exclude`, then the test program is not built. All explicit attributes shall be specified. The explicit attributes for this type are:

**ada-main** The attribute value shall be a string. It shall be the path to the Ada main body file.

**ada-object-directory** The attribute value shall be a string. It shall be the path to the Ada object directory (`-D` option value for `gnatmake`).

**adaflags** The attribute value shall be a list of strings. It shall be a list of options for the Ada compiler.

**adaincludes** The attribute value shall be a list of strings. It shall be a list of Ada include paths.

**cflags** The attribute value shall be a list. Each list element shall be a *Build C Compiler Option*.

**cppflags** The attribute value shall be a list. Each list element shall be a *Build C Preprocessor Option*.

**includes** The attribute value shall be a list. Each list element shall be a *Build Include Path*.

**ldflags** The attribute value shall be a list. Each list element shall be a *Build Linker Option*.

**source** The attribute value shall be a list. Each list element shall be a *Build Source*.

**stlib** The attribute value shall be a list. Each list element shall be a *Build Link Static Library Directive*.

**target** The attribute value shall be a *Build Target*.

**use-after** The attribute value shall be a list. Each list element shall be a *Build Use After Directive*.

**use-before** The attribute value shall be a list. Each list element shall be a *Build Use Before Directive*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
ada-main: testsuites/ada/samples/hello/hello.adb
ada-object-directory: testsuites/ada/samples/hello
adaflags: []
adaincludes:
- cpukit/include/adainclude
- testsuites/ada/support
build-type: ada-test-program
cflags: []
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
cppflags: []
enabled-by: true
includes: []
ldflags: []
links: []
source:
- testsuites/ada/samples/hello/init.c
stlib: []
target: testsuites/ada/ada_hello.exe
type: build
use-after: []
use-before: []
```

#### 6.2.2.2.4 Build BSP Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `bsp`. This set of attributes specifies a base BSP variant to build. All explicit attributes shall be specified. The explicit attributes for this type are:

**arch** The attribute value shall be a string. It shall be the target architecture of the BSP.

**bsp** The attribute value shall be a string. It shall be the base BSP variant name.

**cflags** The attribute value shall be a list. Each list element shall be a *Build C Compiler Option*.

**cppflags** The attribute value shall be a list. Each list element shall be a *Build C Preprocessor Option*.

**family** The attribute value shall be a string. It shall be the BSP family name. The name shall be the last directory of the path to the BSP sources.

**includes** The attribute value shall be a list. Each list element shall be a *Build Include Path*.

**install** The attribute value shall be a list. Each list element shall be a *Build Install Directive*.

**source** The attribute value shall be a list. Each list element shall be a *Build Source*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
arch: myarch
bsp: mybsp
build-type: bsp
```

(continues on next page)

(continued from previous page)

```
cflags: []
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
cppflags: []
enabled-by: true
family: mybsp
includes: []
install:
- destination: ${BSP_INCLUDEDIR}
  source:
  - bsps/myarch/mybsp/include/bsp.h
  - bsps/myarch/mybsp/include/tm27.h
- destination: ${BSP_INCLUDEDIR}/bsp
  source:
  - bsps/myarch/mybsp/include/bsp/irq.h
- destination: ${BSP_LIBDIR}
  source:
  - bsps/myarch/mybsp/start/linkcmds
links:
- role: build-dependency
  uid: ../../obj
- role: build-dependency
  uid: ../../opto2
- role: build-dependency
  uid: abi
- role: build-dependency
  uid: obj
- role: build-dependency
  uid: ../start
- role: build-dependency
  uid: ../../bspopts
source:
- bsps/myarch/mybsp/start/bspstart.c
type: build
```

#### 6.2.2.2.5 Build Configuration File Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `config-file`. This set of attributes specifies a configuration file placed in the build tree. The configuration file is generated during the configure command execution and is placed in the build tree. All explicit attributes shall be specified. The explicit attributes for this type are:

**content** The attribute value shall be a string. It shall be the content of the configuration file. A ${VARIABLE} substitution is performed during the configure command execution using the variables of the configuration set. Use $$ for a plain $ character. To have all variables from sibling items available for substitution it is recommended to link them in the proper order.

**install-path** The attribute value shall be a *Build Install Path*.

**target** The attribute value shall be a *Build Target*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
build-type: config-file
content: |
  # ...
  Name: ${ARCH}-rtems${__RTEMS_MAJOR__}-${BSP_NAME}
  # ...
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
enabled-by: true
install-path: ${PREFIX}/lib/pkgconfig
links: []
target: ${ARCH}-rtems${__RTEMS_MAJOR__}-${BSP_NAME}.pc
type: build
```

#### 6.2.2.2.6 Build Configuration Header Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `config-header`. This set of attributes specifies configuration header file. The configuration header file is generated during configure command execution and is placed in the build tree. All collected configuration defines are written to the configuration header file during the configure command execution. To have all configuration defines from sibling items available it is recommended to link them in the proper order. All explicit attributes shall be specified. The explicit attributes for this type are:

**guard** The attribute value shall be a string. It shall be the header guard define.

**include-headers** The attribute value shall be a list of strings. It shall be a list of header files to include via `#include <...>`.

**install-path** The attribute value shall be a *Build Install Path*.

**target** The attribute value shall be a *Build Target*.

#### 6.2.2.2.7 Build Group Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `group`. This set of attributes provides a means to aggregate other build items and modify the build item context which is used by referenced build items. The `includes`, `ldflags`, `objects`, and `use` variables of the build item context are updated by the corresponding attributes of the build group. All explicit attributes shall be specified. The explicit attributes for this type are:

**includes** The attribute value shall be a list. Each list element shall be a *Build Include Path*.

**install** The attribute value shall be a list. Each list element shall be a *Build Install Directive*.

**ldflags** The attribute value shall be a list of strings. It shall be a list of options for the linker. They are used to link executables referenced by this item.

**use-after** The attribute value shall be a list. Each list element shall be a *Build Use After Directive*.

**use-before** The attribute value shall be a list. Each list element shall be a *Build Use Before Directive*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
build-type: group
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
enabled-by:
- BUILD_TESTS
- BUILD_SAMPLES
includes:
- testsuites/support/include
install: []
ldflags:
- -Wl,--wrap=printf
- -Wl,--wrap=puts
links:
- role: build-dependency
  uid: ticker
type: build
use-after: []
use-before:
- rtemstest
```

#### 6.2.2.2.8 Build Library Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `library`. This set of attributes specifies a static library. Library items may use additional objects provided by *Build Objects Item Type* items through the build dependency links of the item. All explicit attributes shall be specified. The explicit attributes for this type are:

**cflags** The attribute value shall be a list. Each list element shall be a *Build C Compiler Option*.

**cppflags** The attribute value shall be a list. Each list element shall be a *Build C Preprocessor Option*.

**cxxflags** The attribute value shall be a list. Each list element shall be a *Build C++ Compiler Option*.

**includes** The attribute value shall be a list. Each list element shall be a *Build Include Path*.

**install** The attribute value shall be a list. Each list element shall be a *Build Install Directive*.

**install-path** The attribute value shall be a *Build Install Path*.

**source** The attribute value shall be a list. Each list element shall be a *Build Source*.

**target** The attribute value shall be a *Build Target*. It shall be the name of the static library, e.g. `z` for `libz.a`.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
build-type: library
cflags:
- -Wno-pointer-sign
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
cppflags: []
cxxflags: []
enabled-by: true
includes:
- cpukit/libfs/src/jffs2/include
install:
- destination: ${BSP_INCLUDEDIR}/rtems
  source:
  - cpukit/include/rtems/jffs2.h
install-path: ${BSP_LIBDIR}
links: []
source:
- cpukit/libfs/src/jffs2/src/build.c
target: jffs2
type: build
```

#### 6.2.2.2.9 Build Objects Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `objects`. This set of attributes specifies a set of object files used to build static libraries or test programs. All explicit attributes shall be specified. The explicit attributes for this type are:

**cflags** The attribute value shall be a list. Each list element shall be a *Build C Compiler Option*.

**cppflags** The attribute value shall be a list. Each list element shall be a *Build C Preprocessor Option*.

**cxxflags** The attribute value shall be a list. Each list element shall be a *Build C++ Compiler Option*.

**includes** The attribute value shall be a list. Each list element shall be a *Build Include Path*.

**install** The attribute value shall be a list. Each list element shall be a *Build Install Directive*.

**source** The attribute value shall be a list. Each list element shall be a *Build Source*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
build-type: objects
cflags: []
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
cppflags: []
cxxflags: []
enabled-by: true
includes: []
install:
```

(continues on next page)

(continued from previous page)

```
- destination: ${BSP_INCLUDEDIR}/bsp
  source:
  - bsps/include/bsp/bootcard.h
  - bsps/include/bsp/default-initial-extension.h
  - bsps/include/bsp/fatal.h
links: []
source:
- bsps/shared/start/bootcard.c
- bsps/shared/rtems-version.c
type: build
```

#### 6.2.2.2.10 Build Option Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `option`. This set of attributes specifies a build option. The following explicit attributes are mandatory:

- `actions`
- `default`
- `default-by-variant`
- `description`

The explicit attributes for this type are:

**actions** The attribute value shall be a list. Each list element shall be a *Build Option Action*. Each action operates on the *action value* handed over by a previous action and action-specific attribute values. The actions pass the processed action value to the next action in the list. The first action starts with an action value of `None`. The actions are carried out during the configure command execution.

**default** The attribute value shall be a *Build Option Value*. It shall be the default value of the option if no variant-specific default value is specified. Use `null` to specify that no default value exits. The variant-specific default values may be specified by the `default-by-variant` attribute.

**default-by-variant** The attribute value shall be a list. Each list element shall be a *Build Option Default by Variant*. The list is processed from top to bottom. If a matching variant is found, then the processing stops.

**description** The attribute value shall be an optional string. It shall be the description of the option.

**format** The attribute value shall be an optional string. It shall be a Python format string, for example `'{}'` or `'{:#010x}'`.

**name** The attribute value shall be a *Build Option Name*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
actions:
```

(continues on next page)

(continued from previous page)

```
- get-integer: null
- define: null
build-type: option
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
default: 115200
default-by-variant:
- value: 9600
  variants:
  - m68k/m5484FireEngine
  - powerpc/hsc_cm01
- value: 19200
  variants:
  - m68k/COBRA5475
description: |
  Default baud for console and other serial devices.
enabled-by: true
format: '{}'
links: []
name: BSP_CONSOLE_BAUD
type: build
```

#### 6.2.2.2.11 Build Script Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `script`. This set of attributes specifies a build script. The optional attributes may be required by commands executed through the scripts. The following explicit attributes are mandatory:

- `do-build`
- `do-configure`
- `prepare-build`
- `prepare-configure`

The explicit attributes for this type are:

**asflags** The attribute value shall be a list. Each list element shall be a *Build Assembler Option*.

**cflags** The attribute value shall be a list. Each list element shall be a *Build C Compiler Option*.

**cppflags** The attribute value shall be a list. Each list element shall be a *Build C Preprocessor Option*.

**cxxflags** The attribute value shall be a list. Each list element shall be a *Build C++ Compiler Option*.

**do-build** The attribute value shall be an optional string. If this script shall execute, then it shall be Python code which is executed via `exec()` in the context of the `do_build()` method of the `wscript`. A local variable `bld` is available with the `waf` build context. A local variable `bic` is available with the build item context.

**do-configure** The attribute value shall be an optional string. If this script shall execute, then it shall be Python code which is executed via `exec()` in the context of the `do_configure()` method of the `wscript`. A local variable `conf` is available with the `waf` configuration context. A local variable `cic` is available with the configuration item context.

**includes** The attribute value shall be a list. Each list element shall be a *Build Include Path*.

**ldflags** The attribute value shall be a list. Each list element shall be a *Build Linker Option*.

**prepare-build** The attribute value shall be an optional string. If this script shall execute, then it shall be Python code which is executed via `exec()` in the context of the `prepare_build()` method of the `wscript`. A local variable `bld` is available with the `waf` build context. A local variable `bic` is available with the build item context.

**prepare-configure** The attribute value shall be an optional string. If this script shall execute, then it shall be Python code which is executed via `exec()` in the context of the `prepare_configure()` method of the `wscript`. A local variable `conf` is available with the `waf` configuration context. A local variable `cic` is available with the configuration item context.

**stlib** The attribute value shall be a list. Each list element shall be a *Build Link Static Library Directive*.

**use-after** The attribute value shall be a list. Each list element shall be a *Build Use After Directive*.

**use-before** The attribute value shall be a list. Each list element shall be a *Build Use Before Directive*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
build-type: script
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
default: null
default-by-variant: []
do-build: |
  bld.install_as(
      "${BSP_LIBDIR}/linkcmds",
      "bsps/" + bld.env.ARCH + "/" + bld.env.BSP_FAMILY +
      "/start/linkcmds." + bld.env.BSP_BASE
  )
do-configure: |
  conf.env.append_value(
      "LINKFLAGS",
      ["-qnolinkcmds", "-T", "linkcmds." + conf.env.BSP_BASE]
  )
enabled-by: true
links: []
prepare-build: null
prepare-configure: null
type: build
```

#### 6.2.2.2.12 Build Start File Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `start-file`. This set of attributes specifies a start file to build. A start file is used to link an executable. All explicit attributes shall be specified. The explicit attributes for this type are:

**asflags** The attribute value shall be a list. Each list element shall be a *Build Assembler Option*.

**cppflags** The attribute value shall be a list. Each list element shall be a *Build C Preprocessor Option*.

**includes** The attribute value shall be a list. Each list element shall be a *Build Include Path*.

**install-path** The attribute value shall be a *Build Install Path*.

**source** The attribute value shall be a list. Each list element shall be a *Build Source*.

**target** The attribute value shall be a *Build Target*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
asflags: []
build-type: start-file
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
cppflags: []
enabled-by: true
includes: []
install-path: ${BSP_LIBDIR}
links: []
source:
- bsps/sparc/shared/start/start.S
target: start.o
type: build
```

#### 6.2.2.2.13 Build Test Program Item Type

This type refines the *Build Item Type* through the `build-type` attribute if the value is `test-program`. This set of attributes specifies a test program executable to build. Test programs may use additional objects provided by *Build Objects Item Type* items. Test programs have an implicit `enabled-by` attribute value which is controlled by the option action *set-test-state*. If the test state is set to `exclude`, then the test program is not built. All explicit attributes shall be specified. The explicit attributes for this type are:

**cflags** The attribute value shall be a list. Each list element shall be a *Build C Compiler Option*.

**cppflags** The attribute value shall be a list. Each list element shall be a *Build C Preprocessor Option*.

**cxxflags** The attribute value shall be a list. Each list element shall be a *Build C++ Compiler Option*.

**features** The attribute value shall be a string. It shall be the `waf` build features for this test program.

**includes** The attribute value shall be a list. Each list element shall be a *Build Include Path*.

**ldflags** The attribute value shall be a list. Each list element shall be a *Build Linker Option*.

**source** The attribute value shall be a list. Each list element shall be a *Build Source*.

**stlib** The attribute value shall be a list. Each list element shall be a *Build Link Static Library Directive*.

**target** The attribute value shall be a *Build Target*.

**use-after** The attribute value shall be a list. Each list element shall be a *Build Use After Directive*.

**use-before** The attribute value shall be a list. Each list element shall be a *Build Use Before Directive*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
build-type: test-program
cflags: []
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
cppflags: []
cxxflags: []
enabled-by: true
features: c cprogram
includes: []
ldflags: []
links: []
source:
- testsuites/samples/ticker/init.c
- testsuites/samples/ticker/tasks.c
stlib: []
target: testsuites/samples/ticker.exe
type: build
use-after: []
use-before: []
```

#### 6.2.2.2.14 Constraint Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `constraint`. This set of attributes specifies a constraint. All explicit attributes shall be specified. The explicit attributes for this type are:

**rationale** The attribute value shall be an optional string. If the value is present, then it shall state the rationale or justification of the constraint.

**text** The attribute value shall be a *Requirement Text*. It shall state the constraint.

#### 6.2.2.2.15 Glossary Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `glossary`. This set of attributes specifies a glossary item. All explicit attributes shall be specified. The explicit attributes for this type are:

**glossary-type** The attribute value shall be a *Name*. It shall be the glossary item type.

This type is refined by the following types:

- *Glossary Group Item Type*
- *Glossary Term Item Type*

#### 6.2.2.2.16 Glossary Group Item Type

This type refines the *Glossary Item Type* through the `glossary-type` attribute if the value is `group`. This set of attributes specifies a glossary group. All explicit attributes shall be specified. The explicit attributes for this type are:

**name** The attribute value shall be a string. It shall be the human readable name of the glossary group.

**text** The attribute value shall be a string. It shall state the requirement for the glossary group.

#### 6.2.2.2.17 Glossary Term Item Type

This type refines the *Glossary Item Type* through the `glossary-type` attribute if the value is `term`. This set of attributes specifies a glossary term. All explicit attributes shall be specified. The explicit attributes for this type are:

**term** The attribute value shall be a string. It shall be the glossary term.

**text** The attribute value shall be a string. It shall be the definition of the glossary term.

#### 6.2.2.2.18 Interface Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `interface`. This set of attributes specifies an interface specification item. Interface items shall specify the interface of the software product to other software products and the hardware. Use *Interface Domain Item Type* items to specify interface domains, for example the *API*, C language, compiler, interfaces to the implementation, and the hardware. All explicit attributes shall be specified. The explicit attributes for this type are:

**index-entries** The attribute value shall be a list of strings. It shall be a list of additional document index entries. A document index entry derived from the interface name is added automatically.

**interface-type** The attribute value shall be a *Name*. It shall be the interface item type.

This type is refined by the following types:

- *Application Configuration Group Item Type*
- *Application Configuration Option Item Type*
- *Interface Compound Item Type*
- *Interface Container Item Type*
- *Interface Define Item Type*
- *Interface Domain Item Type*
- *Interface Enum Item Type*
- *Interface Enumerator Item Type*
- *Interface Forward Declaration Item Type*
- *Interface Function Item Type*
- *Interface Group Item Type*
- *Interface Header File Item Type*
- *Interface Macro Item Type*
- *Interface Typedef Item Type*
- *Interface Unspecified Item Type*
- *Interface Variable Item Type*

#### 6.2.2.2.19 Application Configuration Group Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `appl-config-group`. This set of attributes specifies an application configuration group. All explicit attributes shall be specified. The explicit attributes for this type are:

**description** The attribute value shall be a string. It shall be the description of the application configuration group.

**name** The attribute value shall be a string. It shall be human readable name of the application configuration group.

**text** The attribute value shall be a *Requirement Text*. It shall state the requirement for the application configuration group.

#### 6.2.2.2.20 Application Configuration Option Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `appl-config-option`. This set of attributes specifies an application configuration option. All explicit attributes shall be specified. The explicit attributes for this type are:

**appl-config-option-type** The attribute value shall be a *Name*. It shall be the application configuration option type.

**description** The attribute value shall be an *Interface Description*.

**name** The attribute value shall be an *Application Configuration Option Name*.

**notes** The attribute value shall be an *Interface Notes*.

This type is refined by the following types:

- *Application Configuration Feature Enable Option Item Type*
- *Application Configuration Feature Option Item Type*
- *Application Configuration Value Option Item Type*

#### 6.2.2.2.21 Application Configuration Feature Enable Option Item Type

This type refines the *Application Configuration Option Item Type* through the `appl-config-option-type` attribute if the value is `feature-enable`. This set of attributes specifies an application configuration feature enable option.

#### 6.2.2.2.22 Application Configuration Feature Option Item Type

This type refines the *Application Configuration Option Item Type* through the `appl-config-option-type` attribute if the value is `feature`. This set of attributes specifies an application configuration feature option. All explicit attributes shall be specified. The explicit attributes for this type are:

**default** The attribute value shall be a string. It shall describe what happens if the configuration option is undefined.

#### 6.2.2.2.23 Application Configuration Value Option Item Type

This type refines the following types:

- *Application Configuration Option Item Type* through the `appl-config-option-type` attribute if the value is `initializer`
- *Application Configuration Option Item Type* through the `appl-config-option-type` attribute if the value is `integer`

This set of attributes specifies application configuration initializer or integer option. All explicit attributes shall be specified. The explicit attributes for this type are:

**default-value** The attribute value shall be an *Integer or String*. It shall describe the default value of the application configuration option.

#### 6.2.2.2.24 Interface Compound Item Type

This type refines the following types:

- *Interface Item Type* through the `interface-type` attribute if the value is `struct`
- *Interface Item Type* through the `interface-type` attribute if the value is `union`

This set of attributes specifies a compound (struct or union). All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**definition** The attribute value shall be a list. Each list element shall be an *Interface Compound Member Definition Directive*.

**definition-kind** The attribute value shall be an *Interface Compound Definition Kind*.

**description** The attribute value shall be an *Interface Description*.

**name** The attribute value shall be a string. It shall be the name of the compound (struct or union).

**notes** The attribute value shall be an *Interface Notes*.

#### 6.2.2.2.25 Interface Container Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `container`. Items of this type specify an interface container. The item shall have exactly one link with the *Interface Placement Link Role* to an *Interface Domain Item Type* item. This link defines the interface domain of the container.

#### 6.2.2.2.26 Interface Define Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `define`. This set of attributes specifies a define. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**definition** The attribute value shall be an *Interface Definition Directive*.

**description** The attribute value shall be an *Interface Description*.

**name** The attribute value shall be a string. It shall be the name of the define.

**notes** The attribute value shall be an *Interface Notes*.

#### 6.2.2.2.27 Interface Domain Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `domain`. This set of attributes specifies an interface domain. Items of the types *Interface Container Item Type* and *Interface Header File Item Type* are placed into domains through links with the *Interface Placement Link Role*. All explicit attributes shall be specified. The explicit attributes for this type are:

**description** The attribute value shall be a string. It shall be the description of the domain

**name** The attribute value shall be a string. It shall be the human readable name of the domain.

#### 6.2.2.2.28 Interface Enum Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `enum`. This set of attributes specifies an enum. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**definition-kind** The attribute value shall be an *Interface Enum Definition Kind*.

**description** The attribute value shall be an *Interface Description*.

**name** The attribute value shall be a string. It shall be the name of the enum.

**notes** The attribute value shall be an *Interface Description*.

#### 6.2.2.2.29 Interface Enumerator Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `enumerator`. This set of attributes specifies an enumerator. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**definition** The attribute value shall be an *Interface Definition Directive*.

**description** The attribute value shall be an *Interface Description*.

**name** The attribute value shall be a string. It shall be the name of the enumerator.

**notes** The attribute value shall be an *Interface Notes*.

#### 6.2.2.2.30 Interface Forward Declaration Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `forward-declaration`. Items of this type specify a forward declaration. The item shall have exactly one link with the *Interface Target Link Role* to an *Interface Compound Item Type* item. This link defines the type declared by the forward declaration.

#### 6.2.2.2.31 Interface Function Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `function`. This set of attributes specifies a function. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**definition** The attribute value shall be an *Interface Function Definition Directive*.

**description** The attribute value shall be an *Interface Description*.

**name** The attribute value shall be a string. It shall be the name of the function.

**notes** The attribute value shall be an *Interface Notes*.

**params** The attribute value shall be a list. Each list element shall be an *Interface Parameter*.

**return** The attribute value shall be an *Interface Return Directive*.

#### 6.2.2.2.32 Interface Group Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `group`. This set of attributes specifies an interface group. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**description** The attribute value shall be an *Interface Description*.

**identifier** The attribute value shall be an *Interface Group Identifier*.

**name** The attribute value shall be a string. It shall be the human readable name of the interface group.

**text** The attribute value shall be a *Requirement Text*. It shall state the requirement for the interface group.

#### 6.2.2.2.33 Interface Header File Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `header-file`. This set of attributes specifies a header file. The item shall have exactly one link with the *Interface Placement Link Role* to an *Interface Domain Item Type* item. This link defines the interface domain of the header file. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**path** The attribute value shall be a string. It shall be the path used to include the header file. For example `rtems/confdefs.h`.

**prefix** The attribute value shall be a string. It shall be the prefix directory path to the header file in the interface domain. For example `cpukit/include`.

#### 6.2.2.2.34 Interface Macro Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `macro`. This set of attributes specifies a macro. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**definition** The attribute value shall be an *Interface Definition Directive*.

**description** The attribute value shall be an *Interface Description*.

**name** The attribute value shall be a string. It shall be the name of the macro.

**notes** The attribute value shall be an *Interface Notes*.

**params** The attribute value shall be a list. Each list element shall be an *Interface Parameter*.

**return** The attribute value shall be an *Interface Return Directive*.

#### 6.2.2.2.35 Interface Typedef Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `typedef`. This set of attributes specifies a typedef. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**definition** The attribute value shall be an *Interface Definition Directive*.

**description** The attribute value shall be an *Interface Description*.

**name** The attribute value shall be a string. It shall be the name of the typedef.

**notes** The attribute value shall be an *Interface Notes*.

#### 6.2.2.2.36 Interface Unspecified Item Type

This type refines the following types:

- *Interface Item Type* through the `interface-type` attribute if the value is `unspecified`
- *Interface Item Type* through the `interface-type` attribute if the value is `unspecified-define`
- *Interface Item Type* through the `interface-type` attribute if the value is `unspecified-function`
- *Interface Item Type* through the `interface-type` attribute if the value is `unspecified-group`
- *Interface Item Type* through the `interface-type` attribute if the value is `unspecified-type`

This set of attributes specifies an unspecified interface. All explicit attributes shall be specified. The explicit attributes for this type are:

**name** The attribute value shall be a string. It shall be the name of the unspecified interface.

**references** The attribute value shall be an *Interface References Set*.

#### 6.2.2.2.37 Interface Variable Item Type

This type refines the *Interface Item Type* through the `interface-type` attribute if the value is `variable`. This set of attributes specifies a variable. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**definition** The attribute value shall be an *Interface Definition Directive*.

**description** The attribute value shall be an *Interface Description*.

**name** The attribute value shall be a string. It shall be the name of the variable.

**notes** The attribute value shall be an *Interface Notes*.

#### 6.2.2.2.38 Requirement Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `requirement`. This set of attributes specifies a requirement. All explicit attributes shall be specified. The explicit attributes for this type are:

**rationale** The attribute value shall be an optional string. If the value is present, then it shall state the rationale or justification of the requirement.

**references** The attribute value shall be a list. Each list element shall be a *Requirement Reference*.

**requirement-type** The attribute value shall be a *Name*. It shall be the requirement item type.

**text** The attribute value shall be a *Requirement Text*. It shall state the requirement.

This type is refined by the following types:

- *Functional Requirement Item Type*
- *Non-Functional Requirement Item Type*

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de
enabled-by: true
functional-type: capability
links: []
rationale: |
  It keeps you busy.
requirement-type: functional
text: |
  The system shall do crazy things.
type: requirement
```

#### 6.2.2.2.39 Functional Requirement Item Type

This type refines the *Requirement Item Type* through the `requirement-type` attribute if the value is `functional`. This set of attributes specifies a functional requirement. All explicit attributes shall be specified. The explicit attributes for this type are:

**functional-type** The attribute value shall be a *Name*. It shall be the functional type of the requirement.

This type is refined by the following types:

- *Action Requirement Item Type*
- *Generic Functional Requirement Item Type*

#### 6.2.2.2.40 Action Requirement Item Type

This type refines the *Functional Requirement Item Type* through the `functional-type` attribute if the value is `action`. This set of attributes specifies functional requirements and corresponding validation test code. The functional requirements of an action are specified. An action performs a step in a finite state machine. An action is implemented through a function or a macro. The action is performed through a call of the function or an execution of the code of a macro expansion by an actor. The actor is for example a task or an interrupt service routine.

For action requirements which specify the function of an interface, there shall be exactly one link with the *Interface Function Link Role* to the interface of the action.

The action requirements are specified by

- a list of pre-conditions, each with a set of states,
- a list of post-conditions, each with a set of states,
- the transition of pre-condition states to post-condition states through the action.

Along with the requirements, the test code to generate a validation test is specified. For an action requirement it is verified that all variations of pre-condition states have a set of post-condition states specified in the transition map. All transitions are covered by the generated test code. All explicit attributes shall be specified. The explicit attributes for this type are:

**post-conditions** The attribute value shall be a list. Each list element shall be an *Action Requirement Condition*.

**pre-conditions** The attribute value shall be a list. Each list element shall be an *Action Requirement Condition*.

**skip-reasons** The attribute value shall be an *Action Requirement Skip Reasons*.

**test-action** The attribute value shall be a string. It shall be the test action code.

**test-brief** The attribute value shall be an optional string. If the value is present, then it shall be the test case brief description.

**test-cleanup** The attribute value shall be an optional string. If the value is present, then it shall be the test cleanup code. The code is placed in the test action loop body after the test post-condition checks.

**test-context** The attribute value shall be a list. Each list element shall be a *Test Context Member*.

**test-context-support** The attribute value shall be an optional string. If the value is present, then it shall be the test context support code. The context support code is placed at file scope before the test context definition.

**test-description** The attribute value shall be an optional string. If the value is present, then it shall be the test case description.

**test-header** The attribute value shall be a *Test Header*.

**test-includes** The attribute value shall be a list of strings. It shall be a list of header files included via `#include <...>`.

**test-local-includes** The attribute value shall be a list of strings. It shall be a list of header files included via `#include "..."`.

**test-prepare** The attribute value shall be an optional string. If the value is present, then it shall be the early test preparation code. The code is placed in the test action loop body before the test pre-condition preparations.

**test-setup** The attribute value shall be a *Test Support Method*.

**test-stop** The attribute value shall be a *Test Support Method*.

**test-support** The attribute value shall be an optional string. If the value is present, then it shall be the test case support code. The support code is placed at file scope before the test case code.

**test-target** The attribute value shall be a string. It shall be the path to the generated test case source file.

**test-teardown** The attribute value shall be a *Test Support Method*.

**transition-map** The attribute value shall be a list. Each list element shall be an *Action Requirement Transition*.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
enabled-by: true
functional-type: action
links: []
post-conditions:
- name: Status
  states:
  - name: Success
    test-code: |
      /* Check that the status is SUCCESS */
    text: |
      The status shall be SUCCESS.
  - name: Error
    test-code: |
      /* Check that the status is ERROR */
    text: |
      The status shall be ERROR.
  test-epilogue: null
  test-prologue: null
- name: Data
  states:
  - name: Unchanged
    test-code: |
      /* Check that the data is unchanged */
    text: |
      The data shall be unchanged by the action.
  - name: Red
    test-code: |
      /* Check that the data is red */
    text: |
      The data shall be red.
  - name: Green
    test-code: |
      /* Check that the data is green */
    text: |
      The data shall be green.
  test-epilogue: null
  test-prologue: null
pre-conditions:
- name: Data
  states:
  - name: NullPtr
    test-code: |
      /* Set data pointer to NULL */
    text: |
      The data pointer shall be NULL.
  - name: Valid
    test-code: |
      /* Set data pointer to reference a valid data buffer */
    text: |
      The data pointer shall reference a valid data buffer.
```

(continues on next page)

(continued from previous page)

```
  test-epilogue: null
  test-prologue: null
- name: Option
  states:
  - name: Red
    test-code: |
      /* Set option to RED */
    text: |
      The option shall be RED.
  - name: Green
    test-code: |
      /* Set option to GREEN */
    text: |
      The option shall be GREEN.
  test-epilogue: null
  test-prologue: null
requirement-type: functional
skip-reasons: {}
test-action: |
  /* Call the function of the action */
test-brief: null
test-cleanup: null
test-context:
- brief: null
  description: null
  member: void *data
- brief: null
  description: null
  member: option_type option
test-context-support: null
test-description: null
test-header: null
test-includes: []
test-local-includes: []
test-prepare: null
test-setup: null
test-stop: null
test-support: null
test-target: tc-red-green-data.c
test-teardown: null
transition-map:
- enabled-by: true
  post-conditions:
    Status: Error
    Data: Unchanged
  pre-conditions:
    Data: NullPtr
    Option: all
- enabled-by: true
  post-conditions:
    Status: Success
    Data: Red
  pre-conditions:
```

(continues on next page)

(continued from previous page)

```
    Data: Valid
    Option: Red
- enabled-by: true
  post-conditions:
    Status: Success
    Data: Green
  pre-conditions:
    Data: Valid
    Option: Green
rationale: null
references: []
text: |
  ${.:/text-template}
type: requirement
```

#### 6.2.2.2.41 Generic Functional Requirement Item Type

This type refines the following types:

- *Functional Requirement Item Type* through the `functional-type` attribute if the value is `capability`
- *Functional Requirement Item Type* through the `functional-type` attribute if the value is `dependability-function`
- *Functional Requirement Item Type* through the `functional-type` attribute if the value is `function`
- *Functional Requirement Item Type* through the `functional-type` attribute if the value is `operational`
- *Functional Requirement Item Type* through the `functional-type` attribute if the value is `safety-function`

Items of this type state a functional requirement with the functional type defined by the specification type refinement.

#### 6.2.2.2.42 Non-Functional Requirement Item Type

This type refines the *Requirement Item Type* through the `requirement-type` attribute if the value is `non-functional`. This set of attributes specifies a non-functional requirement. All explicit attributes shall be specified. The explicit attributes for this type are:

**non-functional-type** The attribute value shall be a *Name*. It shall be the non-functional type of the requirement.

This type is refined by the following types:

- *Design Group Requirement Item Type*
- *Generic Non-Functional Requirement Item Type*
- *Runtime Performance Requirement Item Type*

#### 6.2.2.2.43 Design Group Requirement Item Type

This type refines the *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `design-group`. This set of attributes specifies a design group requirement. Design group requirements have an explicit reference to the associated Doxygen group specified by the `identifier` attribute. Design group requirements have an implicit validation by inspection method. The qualification toolchain shall perform the inspection and check that the specified Doxygen group exists in the software source code. All explicit attributes shall be specified. The explicit attributes for this type are:

**identifier** The attribute value shall be an *Interface Group Identifier*.

#### 6.2.2.2.44 Generic Non-Functional Requirement Item Type

This type refines the following types:

- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `build-configuration`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `constraint`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `design`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `documentation`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `interface`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `interface-requirement`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `maintainability`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `performance`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `portability`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `quality`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `reliability`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `resource`
- *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `safety`

Items of this type state a non-functional requirement with the non-functional type defined by the specification type refinement.

#### 6.2.2.2.45 Runtime Performance Requirement Item Type

This type refines the *Non-Functional Requirement Item Type* through the `non-functional-type` attribute if the value is `performance-runtime`. The item shall have exactly one link with the *Runtime Measurement Request Link Role*. A requirement text processor shall support a substitution of ${.:/limit-kind}:

- For a *Runtime Measurement Value Kind* of `min-lower-bound` or `min-upper-bound`, the substitution of ${.:/limit-kind} shall be `"minimum"`.
- For a *Runtime Measurement Value Kind* of `mean-lower-bound` or `mean-upper-bound`, the substitution of ${.:/limit-kind} shall be `"mean"`.
- For a *Runtime Measurement Value Kind* of `max-lower-bound` or `max-upper-bound`, the substitution of ${.:/limit-kind} shall be `"maximum"`.

A requirement text processor shall support a substitution of ${.:/limit-condition}:

- For a *Runtime Measurement Value Kind* of `min-lower-bound`, `mean-lower-bound`, or `max-lower-bound`, the substitution of ${.:/limit-condition} shall be `"greater than or equal to <value>"` with <value> being the value of the corresponding entry in the *Runtime Measurement Value Table*.
- For a *Runtime Measurement Value Kind* of `min-upper-bound`, `mean-upper-bound`, or `max-upper-bound`, the substitution of ${.:/limit-condition} shall be `"less than or equal to <value>"` with <value> being the value of the corresponding entry in the *Runtime Measurement Value Table*.

A requirement text processor shall support a substitution of ${.:/environment}. The value of the substitution shall be `"<environment> environment"` with <environment> being the environment of the corresponding entry in the *Runtime Measurement Environment Table*.

This set of attributes specifies a runtime performance requirement. Along with the requirement, the validation test code to execute a measure runtime request is specified. All explicit attributes shall be specified. The explicit attributes for this type are:

**limits** The attribute value shall be a *Runtime Performance Limit Table*.

**params** The attribute value shall be a *Runtime Performance Parameter Set*.

**test-body** The attribute value shall be a *Test Support Method*. It shall provide the code of the measure runtime body handler. In contrast to other methods, this method is mandatory.

**test-cleanup** The attribute value shall be a *Test Support Method*. It may provide the code to clean up the measure runtime request. This method is called before the cleanup method of the corresponding *Runtime Measurement Test Item Type* item and after the request.

**test-prepare** The attribute value shall be a *Test Support Method*. It may provide the code to prepare the measure runtime request. This method is called after the prepare method of the corresponding *Runtime Measurement Test Item Type* item and before the request.

**test-setup** The attribute value shall be a *Test Support Method*. It may provide the code of the measure runtime setup handler.

**test-teardown** The attribute value shall be a *Test Support Method*. It may provide the code of the measure runtime teardown handler.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
enabled-by: true
links:
- role: runtime-measurement-request
  uid: ../val/performance
limits:
  sparc/leon3:
    DirtyCache:
      max-upper-bound: 0.000005
      mean-upper-bound: 0.000005
    FullCache:
      max-upper-bound: 0.000005
      mean-upper-bound: 0.000005
    HotCache:
      max-upper-bound: 0.000005
      mean-upper-bound: 0.000005
    Load/1:
      max-upper-bound: 0.00001
      mean-upper-bound: 0.00001
    Load/2:
      max-upper-bound: 0.00001
      mean-upper-bound: 0.00001
    Load/3:
      max-upper-bound: 0.00001
      mean-upper-bound: 0.00001
    Load/4:
      max-upper-bound: 0.00001
      mean-upper-bound: 0.00001
params: {}
rationale: null
references: []
test-body:
  brief: |
    Get a buffer.
  code: |
    ctx->status = rtems_partition_get_buffer( ctx->part_many, &ctx->buffer );
  description: null
test-cleanup: null
test-prepare: null
test-setup: null
test-teardown:
  brief: |
    Return the buffer.
  code: |
    rtems_status_code sc;
```

(continues on next page)

(continued from previous page)

```
    T_quiet_rsc_success( ctx->status );

    sc = rtems_partition_return_buffer( ctx->part_many, ctx->buffer );
    T_quiet_rsc_success( sc );

    return tic == toc;
  description: null
text: |
  When a partition has exactly ${../val/performance:/params/buffer-count} free
  buffers, the ${.:limit-kind} runtime of exactly
  ${../val/performance:/params/sample-count} successful calls to
  ${../if/get-buffer:/name} in the ${.:/environment} shall be
  ${.:limit-condition}.
non-functional-type: performance-runtime
requirement-type: non-functional
type: requirement
```

#### 6.2.2.2.46 Requirement Validation Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `validation`. This set of attributes provides a requirement validation evidence. The item shall have exactly one link to the validated requirement with the *Requirement Validation Link Role*. All explicit attributes shall be specified. The explicit attributes for this type are:

**method** The attribute value shall be a *Requirement Validation Method*. Validation by test is done through *Test Case Item Type* items.

**text** The attribute value shall be a string. It shall provide the validation evidence depending on the validation method:

- *By analysis*: A statement shall be provided how the requirement is met, by analysing static properties of the *software product*.
- *By inspection*: A statement shall be provided how the requirement is met, by inspection of the *source code*.
- *By review of design*: A rationale shall be provided to demonstrate how the requirement is satisfied implicitly by the software design.

#### 6.2.2.2.47 Runtime Measurement Test Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `runtime-measurement-test`. This set of attributes specifies a runtime measurement test case. All explicit attributes shall be specified. The explicit attributes for this type are:

**params** The attribute value shall be a *Runtime Measurement Parameter Set*.

**test-brief** The attribute value shall be an optional string. If the value is present, then it shall be the test case brief description.

**test-cleanup** The attribute value shall be a *Test Support Method*. If the value is present, then it shall be the measure runtime request cleanup method. The method is called after each measure runtime request.

**test-context** The attribute value shall be a list. Each list element shall be a *Test Context Member*.

**test-context-support** The attribute value shall be an optional string. If the value is present, then it shall be the test context support code. The context support code is placed at file scope before the test context definition.

**test-description** The attribute value shall be an optional string. If the value is present, then it shall be the test case description.

**test-includes** The attribute value shall be a list of strings. It shall be a list of header files included via `#include <...>`.

**test-local-includes** The attribute value shall be a list of strings. It shall be a list of header files included via `#include "..."`.

**test-prepare** The attribute value shall be a *Test Support Method*. If the value is present, then it shall be the measure runtime request prepare method. The method is called before each measure runtime request.

**test-setup** The attribute value shall be a *Test Support Method*. If the value is present, then it shall be the test case setup fixture method.

**test-stop** The attribute value shall be a *Test Support Method*. If the value is present, then it shall be the test case stop fixture method.

**test-support** The attribute value shall be an optional string. If the value is present, then it shall be the test case support code. The support code is placed at file scope before the test case code.

**test-target** The attribute value shall be a string. It shall be the path to the generated test case source file.

**test-teardown** The attribute value shall be a *Test Support Method*. If the value is present, then it shall be the test case teardown fixture method.

#### 6.2.2.2.48 Specification Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `spec`. This set of attributes specifies specification types. All explicit attributes shall be specified. The explicit attributes for this type are:

**spec-description** The attribute value shall be an optional string. It shall be the description of the specification type.

**spec-example** The attribute value shall be an optional string. If the value is present, then it shall be an example of the specification type.

**spec-info** The attribute value shall be a *Specification Information*.

**spec-name** The attribute value shall be an optional string. It shall be the human readable name of the specification type.

**spec-type** The attribute value shall be a *Name*. It shall the specification type.

Please have a look at the following example:

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
enabled-by: true
links:
- role: spec-member
  uid: root
- role: spec-refinement
  spec-key: type
  spec-value: example
  uid: root
spec-description: null
spec-example: null
spec-info:
  dict:
    attributes:
      an-example-attribute:
        description: |
          It shall be an example.
        spec-type: optional-str
      example-number:
        description: |
          It shall be the example number.
        spec-type: int
    description: |
      This set of attributes specifies an example.
    mandatory-attributes: all
spec-name: Example Item Type
spec-type: spec
type: spec
```

#### 6.2.2.2.49 Test Case Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `test-case`. This set of attributes specifies a test case. All explicit attributes shall be specified. The explicit attributes for this type are:

**test-actions** The attribute value shall be a list. Each list element shall be a *Test Case Action*.

**test-brief** The attribute value shall be a string. It shall be the test case brief description.

**test-context** The attribute value shall be a list. Each list element shall be a *Test Context Member*.

**test-context-support** The attribute value shall be an optional string. If the value is present, then it shall be the test context support code. The context support code is placed at file scope before the test context definition.

**test-description** The attribute value shall be an optional string. It shall be the test case description.

**test-header** The attribute value shall be a *Test Header*.

**test-includes** The attribute value shall be a list of strings. It shall be a list of header files included via `#include <...>`.

**test-local-includes** The attribute value shall be a list of strings. It shall be a list of header files included via `#include "..."`.

**test-setup** The attribute value shall be a *Test Support Method*.

**test-stop** The attribute value shall be a *Test Support Method*.

**test-support** The attribute value shall be an optional string. If the value is present, then it shall be the test case support code. The support code is placed at file scope before the test case code.

**test-target** The attribute value shall be a string. It shall be the path to the generated target test case source file.

**test-teardown** The attribute value shall be a *Test Support Method*.

#### 6.2.2.2.50 Test Platform Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `test-platform`. Please note:

> **Warning:** This item type is work in progress.

This set of attributes specifies a test platform. All explicit attributes shall be specified. The explicit attributes for this type are:

**description** The attribute value shall be a string. It shall be the description of the test platform.

**name** The attribute value shall be a string. It shall be the human readable name of the test platform.

#### 6.2.2.2.51 Test Procedure Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `test-procedure`. Please note:

> **Warning:** This item type is work in progress.

This set of attributes specifies a test procedure. All explicit attributes shall be specified. The explicit attributes for this type are:

**name** The attribute value shall be a string. It shall be the human readable name of the test procedure.

**purpose** The attribute value shall be a string. It shall state the purpose of the test procedure.

**steps** The attribute value shall be a string. It shall describe the steps of the test procedure execution.

#### 6.2.2.2.52 Test Suite Item Type

This type refines the *Root Item Type* through the `type` attribute if the value is `test-suite`. This set of attributes specifies a test suite. All explicit attributes shall be specified. The explicit attributes for this type are:

**test-brief** The attribute value shall be a string. It shall be the test suite brief description.

**test-code** The attribute value shall be a string. It shall be the test suite code. The test suite code is placed at file scope in the target source file.

**test-description** The attribute value shall be an optional string. It shall be the test suite description.

**test-includes** The attribute value shall be a list of strings. It shall be a list of header files included via `#include <...>`.

**test-local-includes** The attribute value shall be a list of strings. It shall be a list of header files included via `#include "..."`.

**test-suite-name** The attribute value shall be a string. It shall be the name of the test suite.

**test-target** The attribute value shall be a string. It shall be the path to the generated target test suite source file.

### 6.2.2.3 Specification Attribute Sets and Value Types

#### 6.2.2.3.1 Action Requirement Boolean Expression

A value of this type is a boolean expression.

A value of this type shall be of one of the following variants:

- The value may be a set of attributes. Each attribute defines an operator. Exactly one of the explicit attributes shall be specified. The explicit attributes for this type are:

  **and** The attribute value shall be a list. Each list element shall be an *Action Requirement Boolean Expression*. The *and* operator evaluates to the *logical and* of the evaluation results of the expressions in the list.

  **not** The attribute value shall be an *Action Requirement Boolean Expression*. The *not* operator evaluates to the *logical not* of the evaluation results of the expression.

  **or** The attribute value shall be a list. Each list element shall be an *Action Requirement Boolean Expression*. The *or* operator evaluates to the *logical or* of the evaluation results of the expressions in the list.

  **post-conditions** The attribute value shall be an *Action Requirement Expression Condition Set*. The *post-conditions* operator evaluates to true, if the post-condition states of the associated transition are contained in the specified post-condition set, otherwise to false.

**pre-conditions** The attribute value shall be an *Action Requirement Expression Condition Set*. The *pre-conditions* operator evaluates to true, if the pre-condition states of the associated transition are contained in the specified pre-condition set, otherwise to false.

- The value may be a list. Each list element shall be an *Action Requirement Boolean Expression*. This list of expressions evaluates to the *logical or* of the evaluation results of the expressions in the list.

This type is used by the following types:

- *Action Requirement Boolean Expression*
- *Action Requirement Expression*

#### 6.2.2.3.2 Action Requirement Condition

This set of attributes defines an action pre-condition or post-condition. All explicit attributes shall be specified. The explicit attributes for this type are:

**name** The attribute value shall be an *Action Requirement Name*.

**states** The attribute value shall be a list. Each list element shall be an *Action Requirement State*.

**test-epilogue** The attribute value shall be an optional string. If the value is present, then it shall be the test epilogue code. The epilogue code is placed in the test condition preparation or check before the state-specific code. The code may use a local variable `ctx` which points to the test context, see *Test Context Member*.

**test-prologue** The attribute value shall be an optional string. If the value is present, then it shall be the test prologue code. The prologue code is placed in the test condition preparation or check after the state-specific code. The code may use a local variable `ctx` which points to the test context, see *Test Context Member*.

This type is used by the following types:

- *Action Requirement Item Type*

#### 6.2.2.3.3 Action Requirement Expression

This set of attributes defines an expression which may define the state of a post-condition. The `else` and `specified-by` shall be used individually. The `if` and `then` or `then-specified-by` expressions shall be used together. At least one of the explicit attributes shall be specified. The explicit attributes for this type are:

**else** The attribute value shall be an *Action Requirement Expression State Name*. It shall be the name of the state of the post-condition.

**if** The attribute value shall be an *Action Requirement Boolean Expression*. If the boolean expression evaluates to true, then the state is defined according to the `then` attribute value.

**specified-by** The attribute value shall be an *Action Requirement Name*. It shall be the name of a pre-condition. The name of the state of the pre-condition in the associated transition defines the name of the state of the post-condition.

**then** The attribute value shall be an *Action Requirement Expression State Name*. It shall be the name of the state of the post-condition.

**then-specified-by** The attribute value shall be an *Action Requirement Name*. It shall be the name of a pre-condition. The name of the state of the pre-condition in the associated transition defines the name of the state of the post-condition.

#### 6.2.2.3.4 Action Requirement Expression Condition Set

This set of attributes defines for the specified conditions a set of states. Generic attributes may be specified. Each generic attribute key shall be an *Action Requirement Name*. Each generic attribute value shall be an *Action Requirement Expression State Set*. There shall be at most one generic attribute key for each condition. The key name shall be the condition name. The value of each generic attribute shall be a set of states of the condition.

This type is used by the following types:

- *Action Requirement Boolean Expression*

#### 6.2.2.3.5 Action Requirement Expression State Name

The value shall be a string. It shall be the name of a state of the condition or `N/A` if the condition is not applicable. The value

- shall match with the regular expression "`^[A-Z][a-zA-Z0-9]+$`",
- or, shall be equal to "`N/A`".

This type is used by the following types:

- *Action Requirement Expression*

#### 6.2.2.3.6 Action Requirement Expression State Set

A value of this type shall be of one of the following variants:

- The value may be a list. Each list element shall be an *Action Requirement Expression State Name*. The list defines a set of states of the condition.
- The value may be a string. It shall be the name of a state of the condition or `N/A` if the condition is not applicable. The value
  - shall match with the regular expression "`^[A-Z][a-zA-Z0-9]+$`",
  - or, shall be equal to "`N/A`".

This type is used by the following types:

- *Action Requirement Expression Condition Set*

#### 6.2.2.3.7 Action Requirement Name

The value shall be a string. It shall be the name of a condition or a state of a condition used to define pre-conditions and post-conditions of an action requirement. It shall be formatted in CamelCase. It should be brief and abbreviated. The rationale for this is that the names are used in tables and the horizontal space is limited by the page width. The more conditions you have in an action requirement, the shorter the names should be. The name `NA` is reserved and indicates that a condition is not applicable. The value

- shall match with the regular expression "`^[A-Z][a-zA-Z0-9]+$`",
- and, shall be not equal to "`NA`".

This type is used by the following types:

- *Action Requirement Condition*
- *Action Requirement Expression Condition Set*
- *Action Requirement Expression*
- *Action Requirement Skip Reasons*
- *Action Requirement State*
- *Action Requirement Transition Post-Conditions*
- *Action Requirement Transition Pre-Conditions*

#### 6.2.2.3.8 Action Requirement Skip Reasons

This set of attributes specifies skip reasons used to justify why transitions in the transition map are skipped. Generic attributes may be specified. Each generic attribute key shall be an *Action Requirement Name*. Each generic attribute value shall be a string. The key defines the name of a skip reason. The name can be used in *Action Requirement Transition Post-Conditions* to skip the corresponding transitions. The value shall give a reason why the transitions are skipped.

This type is used by the following types:

- *Action Requirement Item Type*

#### 6.2.2.3.9 Action Requirement State

This set of attributes defines an action pre-condition or post-condition state. All explicit attributes shall be specified. The explicit attributes for this type are:

**name** The attribute value shall be an *Action Requirement Name*.

**test-code** The attribute value shall be a string. It shall be the test code to prepare or check the state of the condition. The code may use a local variable `ctx` which points to the test context, see *Test Context Member*.

**text** The attribute value shall be a *Requirement Text*. It shall define the state of the condition.

This type is used by the following types:

- *Action Requirement Condition*

#### 6.2.2.3.10 Action Requirement Transition

This set of attributes defines the transition from multiple sets of states of pre-conditions to a set of states of post-conditions through an action in an action requirement. The ability to specify multiple sets of states of pre-conditions which result in a common set of post-conditions may allow a more compact specification of the transition map. For example, let us suppose you want to specify the action of a function with a pointer parameter. The function performs an early check that the pointer is NULL and in this case returns an error code. The pointer condition dominates the action outcome if the pointer is NULL. Other pre-condition states can be simply set to `all` for this transition. All explicit attributes shall be specified. The explicit attributes for this type are:

**enabled-by** The attribute value shall be an *Enabled-By Expression*. The transition map may be customized to support configuration variants through this attribute. The default transitions (`enabled-by: true`) shall be specified before the customized variants in the list.

**post-conditions** The attribute value shall be an *Action Requirement Transition Post-Conditions*.

**pre-conditions** The attribute value shall be an *Action Requirement Transition Pre-Conditions*.

This type is used by the following types:

- *Action Requirement Item Type*

#### 6.2.2.3.11 Action Requirement Transition Post-Condition State

A value of this type shall be of one of the following variants:

- The value may be a list. Each list element shall be an *Action Requirement Expression*. The list contains expressions to define the state of the corresponding post-condition.
- The value may be a string. It shall be the name of a state of the corresponding post-condition or `N/A` if the post-condition is not applicable. The value
  - shall match with the regular expression "`^[A-Z][a-zA-Z0-9]+$`",
  - or, shall be equal to "`N/A`".

This type is used by the following types:

- *Action Requirement Transition Post-Conditions*

#### 6.2.2.3.12 Action Requirement Transition Post-Conditions

A value of this type shall be of one of the following variants:

- The value may be a set of attributes. This set of attributes defines for each post-condition the state after the action for a transition in an action requirement. Generic attributes may be specified. Each generic attribute key shall be an *Action Requirement Name*. Each generic attribute value shall be an *Action Requirement Transition Post-Condition State*. There shall be exactly one generic attribute key for each post-condition. The key name shall be the post-condition name. The value of each generic attribute shall be the state of the post-condition or `N/A` if the post-condition is not applicable.
- The value may be a string. It shall be the name of a skip reason. If a skip reason is given instead of a listing of post-condition states, then this transition is skipped and no test code runs for this transition. The value
  - shall match with the regular expression "`^[A-Z][a-zA-Z0-9]+$`",
  - and, shall be not equal to "`NA`".

This type is used by the following types:

- *Action Requirement Transition*

#### 6.2.2.3.13 Action Requirement Transition Pre-Condition State Set

A value of this type shall be of one of the following variants:

- The value may be a list. Each list element shall be an *Action Requirement Name*. The list defines the set of states of the pre-condition in the transition.
- The value may be a string. The value `all` represents all states of the pre-condition in this transition. The value `N/A` marks the pre-condition as not applicable in this transition. The value shall be an element of
  - "`all`", and
  - "`N/A`".

This type is used by the following types:

- *Action Requirement Transition Pre-Conditions*

#### 6.2.2.3.14 Action Requirement Transition Pre-Conditions

A value of this type shall be of one of the following variants:

- The value may be a set of attributes. This set of attributes defines for each pre-condition the set of states before the action for a transition in an action requirement. Generic attributes may be specified. Each generic attribute key shall be an *Action Requirement Name*. Each generic attribute value shall be an *Action Requirement Transition Pre-Condition State Set*. There shall be exactly one generic attribute key for each pre-condition. The key name shall be the pre-condition name. The value of each generic attribute shall be a set of states of the pre-condition.

- The value may be a string. If this name is specified instead of explicit pre-condition states, then the post-condition states of this entry are used to define all remaining transitions of the map. The value shall be equal to “`default`”.

This type is used by the following types:

- *Action Requirement Transition*

#### 6.2.2.3.15 Application Configuration Group Member Link Role

This type refines the *Link* through the `role` attribute if the value is `appl-config-group-member`. It defines the application configuration group membership role of links.

#### 6.2.2.3.16 Application Configuration Option Name

The value shall be a string. It shall be the name of an application configuration option. The value shall match with the regular expression “`^(CONFIGURE_|BSP_)[A-Z0-9_]+$`”.

This type is used by the following types:

- *Application Configuration Option Item Type*

#### 6.2.2.3.17 Boolean or Integer or String

A value of this type shall be of one of the following variants:

- The value may be a boolean.
- The value may be an integer number.
- The value may be a string.

This type is used by the following types:

- *Build Option Action*
- *Interface Return Value*

#### 6.2.2.3.18 Build Assembler Option

The value shall be a string. It shall be an option for the assembler. The options are used to assemble the sources of this item. The options defined by this attribute succeed the options presented to the item by the build item context.

This type is used by the following types:

- *Build Script Item Type*
- *Build Start File Item Type*

#### 6.2.2.3.19 Build C Compiler Option

The value shall be a string. It shall be an option for the C compiler. The options are used to compile the sources of this item. The options defined by this attribute succeed the options presented to the item by the build item context.

This type is used by the following types:

- *Build Ada Test Program Item Type*
- *Build BSP Item Type*
- *Build Library Item Type*
- *Build Objects Item Type*
- *Build Option C Compiler Check Action*
- *Build Script Item Type*
- *Build Test Program Item Type*

#### 6.2.2.3.20 Build C Preprocessor Option

The value shall be a string. It shall be an option for the C preprocessor. The options are used to preprocess the sources of this item. The options defined by this attribute succeed the options presented to the item by the build item context.

This type is used by the following types:

- *Build Ada Test Program Item Type*
- *Build BSP Item Type*
- *Build Library Item Type*
- *Build Objects Item Type*
- *Build Script Item Type*
- *Build Start File Item Type*
- *Build Test Program Item Type*

#### 6.2.2.3.21 Build C++ Compiler Option

The value shall be a string. It shall be an option for the C++ compiler. The options are used to compile the sources of this item. The options defined by this attribute succeed the options presented to the item by the build item context.

This type is used by the following types:

- *Build Library Item Type*
- *Build Objects Item Type*
- *Build Option C++ Compiler Check Action*

- *Build Script Item Type*
- *Build Test Program Item Type*

#### 6.2.2.3.22 Build Dependency Link Role

This type refines the *Link* through the `role` attribute if the value is `build-dependency`. It defines the build dependency role of links.

#### 6.2.2.3.23 Build Include Path

The value shall be a string. It shall be a path to header files. The path is used by the C preprocessor to search for header files. It succeeds the includes presented to the item by the build item context. For an *Build Group Item Type* item the includes are visible to all items referenced by the group item. For *Build BSP Item Type*, *Build Objects Item Type*, *Build Library Item Type*, *Build Start File Item Type*, and *Build Test Program Item Type* items the includes are only visible to the sources specified by the item itself and they do not propagate to referenced items.

This type is used by the following types:

- *Build Ada Test Program Item Type*
- *Build BSP Item Type*
- *Build Group Item Type*
- *Build Library Item Type*
- *Build Objects Item Type*
- *Build Script Item Type*
- *Build Start File Item Type*
- *Build Test Program Item Type*

#### 6.2.2.3.24 Build Install Directive

This set of attributes specifies files installed by a build item. All explicit attributes shall be specified. The explicit attributes for this type are:

**destination** The attribute value shall be a string. It shall be the install destination directory.

**source** The attribute value shall be a list of strings. It shall be the list of source files to be installed in the destination directory. The path to a source file shall be relative to the directory of the `wscript`.

This type is used by the following types:

- *Build BSP Item Type*
- *Build Group Item Type*

- *Build Library Item Type*
- *Build Objects Item Type*

#### 6.2.2.3.25 Build Install Path

A value of this type shall be of one of the following variants:

- There may by be no value (null).
- The value may be a string. It shall be the installation path of a *Build Target*.

This type is used by the following types:

- *Build Configuration File Item Type*
- *Build Configuration Header Item Type*
- *Build Library Item Type*
- *Build Start File Item Type*

#### 6.2.2.3.26 Build Link Static Library Directive

The value shall be a string. It shall be an external static library identifier. The library is used to link programs referenced by this item, e.g. `m` for `libm.a`. The library is added to the build command through the `stlib` attribute. It shall not be used for internal static libraries. Internal static libraries shall be specified through the `use-after` and `use-before` attributes to enable a proper build dependency tracking.

This type is used by the following types:

- *Build Ada Test Program Item Type*
- *Build Script Item Type*
- *Build Test Program Item Type*

#### 6.2.2.3.27 Build Linker Option

The value shall be a string. It shall be an option for the linker. The options are used to link executables. The options defined by this attribute succeed the options presented to the item by the build item context.

This type is used by the following types:

- *Build Ada Test Program Item Type*
- *Build Script Item Type*
- *Build Test Program Item Type*

#### 6.2.2.3.28 Build Option Action

This set of attributes specifies a build option action. Exactly one of the explicit attributes shall be specified. The explicit attributes for this type are:

**append-test-cppflags** The attribute value shall be a string. It shall be the name of a test program. The action appends the action value to the `CPPFLAGS` of the test program. The name shall correspond to the name of a *Build Test Program Item Type* item. Due to the processing order of items, there is no way to check if the name specified by the attribute value is valid.

**assert-aligned** The attribute value shall be an integer number. The action asserts that the action value is aligned according to the attribute value.

**assert-eq** The attribute value shall be a *Boolean or Integer or String*. The action asserts that the action value is equal to the attribute value.

**assert-ge** The attribute value shall be an *Integer or String*. The action asserts that the action value is greater than or equal to the attribute value.

**assert-gt** The attribute value shall be an *Integer or String*. The action asserts that the action value is greater than the attribute value.

**assert-int16** The attribute shall have no value. The action asserts that the action value is a valid signed 16-bit integer.

**assert-int32** The attribute shall have no value. The action asserts that the action value is a valid signed 32-bit integer.

**assert-int64** The attribute shall have no value. The action asserts that the action value is a valid signed 64-bit integer.

**assert-int8** The attribute shall have no value. The action asserts that the action value is a valid signed 8-bit integer.

**assert-le** The attribute value shall be an *Integer or String*. The action asserts that the action value is less than or equal to the attribute value.

**assert-lt** The attribute value shall be an *Integer or String*. The action asserts that the action value is less than the attribute value.

**assert-ne** The attribute value shall be a *Boolean or Integer or String*. The action asserts that the action value is not equal to the attribute value.

**assert-power-of-two** The attribute shall have no value. The action asserts that the action value is a power of two.

**assert-uint16** The attribute shall have no value. The action asserts that the action value is a valid unsigned 16-bit integer.

**assert-uint32** The attribute shall have no value. The action asserts that the action value is a valid unsigned 32-bit integer.

**assert-uint64** The attribute shall have no value. The action asserts that the action value is a valid unsigned 64-bit integer.

**assert-uint8** The attribute shall have no value. The action asserts that the action value is a valid unsigned 8-bit integer.

**check-cc** The attribute value shall be a *Build Option C Compiler Check Action*.

**check-cxx** The attribute value shall be a *Build Option C++ Compiler Check Action*.

**define** The attribute value shall be an optional string. The action adds a define to the configuration set. If the attribute value is present, then it is used as the name of the define, otherwise the `name` of the item is used. The value of the define is the action value. If the action value is a string, then it is quoted.

**define-condition** The attribute value shall be an optional string. The action adds a conditional define to the configuration set. If the attribute value is present, then it is used as the name of the define, otherwise the `name` of the item is used. The value of the define is the action value.

**define-unquoted** The attribute value shall be an optional string. The action adds a define to the configuration set. If the attribute value is present, then it is used as the name of the define, otherwise the `name` of the item is used. The value of the define is the action value. If the action value is a string, then it is not quoted.

**env-append** The attribute value shall be an optional string. The action appends the action value to an environment of the configuration set. If the attribute value is present, then it is used as the name of the environment variable, otherwise the `name` of the item is used.

**env-assign** The attribute value shall be an optional string. The action assigns the action value to an environment of the configuration set. If the attribute value is present, then it is used as the name of the environment variable, otherwise the `name` of the item is used.

**env-enable** The attribute value shall be an optional string. If the action value is true, then a name is appended to the `ENABLE` environment variable of the configuration set. If the attribute value is present, then it is used as the name, otherwise the `name` of the item is used.

**find-program** The attribute shall have no value. The action tries to find the program specified by the action value. Uses the `${PATH}` to find the program. Returns the result of the find operation, e.g. a path to the program.

**find-tool** The attribute shall have no value. The action tries to find the tool specified by the action value. Uses the tool paths specified by the `--rtems-tools` command line option. Returns the result of the find operation, e.g. a path to the program.

**format-and-define** The attribute value shall be an optional string. The action adds a define to the configuration set. If the attribute value is present, then it is used as the name of the define, otherwise the `name` of the item is used. The value of the define is the action value. The value is formatted according to the `format` attribute value.

**get-boolean** The attribute shall have no value. The action gets the action value for subsequent actions from a configuration file variable named by the items `name` attribute. If no such variable exists in the configuration file, then the default value is used. The value is converted to a boolean.

**get-env** The attribute value shall be a string. The action gets the action value for subsequent actions from the environment variable of the configuration set named by the attribute value.

**get-integer** The attribute shall have no value. The action gets the action value for subsequent actions from a configuration file variable named by the items `name` attribute. If no such variable exists in the configuration file, then the default value is used. The value is converted to an integer.

**get-string** The attribute shall have no value. The action gets the action value for subsequent actions from a configuration file variable named by the items `name` attribute. If no such variable exists in the configuration file, then the default value is used. The value is converted to a string.

**script** The attribute value shall be a string. The action executes the attribute value with the Python `eval()` function in the context of the script action handler.

**set-test-state** The attribute value shall be a *Build Option Set Test State Action*.

**set-value** The attribute value shall be a *Build Option Value*. The action sets the action value for subsequent actions to the attribute value.

**split** The attribute shall have no value. The action splits the action value.

**substitute** The attribute shall have no value. The action performs a `${VARIABLE}` substitution on the action value. Use `$$` for a plain `$` character.

This type is used by the following types:

- *Build Option Item Type*

#### 6.2.2.3.29 Build Option C Compiler Check Action

This set of attributes specifies a check done using the C compiler. All explicit attributes shall be specified. The explicit attributes for this type are:

**cflags** The attribute value shall be a list. Each list element shall be a *Build C Compiler Option*.

**fragment** The attribute value shall be a string. It shall be a code fragment used to check the availability of a certain feature through compilation with the C compiler. The resulting object is not linked to an executable.

**message** The attribute value shall be a string. It shall be a description of the feature to check.

This type is used by the following types:

- *Build Option Action*

#### 6.2.2.3.30 Build Option C++ Compiler Check Action

This set of attributes specifies a check done using the C++ compiler. All explicit attributes shall be specified. The explicit attributes for this type are:

**cxxflags** The attribute value shall be a list. Each list element shall be a *Build C++ Compiler Option*.

**fragment** The attribute value shall be a string. It shall be a code fragment used to check the availability of a certain feature through compilation with the C++ compiler. The resulting object is not linked to an executable.

**message** The attribute value shall be a string. It shall be a description of the feature to check.

This type is used by the following types:

- *Build Option Action*

#### 6.2.2.3.31 Build Option Default by Variant

This set of attributes specifies build option default values by variant. All explicit attributes shall be specified. The explicit attributes for this type are:

**value** The attribute value shall be a *Build Option Value*. It value shall be the default value for the matching variants.

**variants** The attribute value shall be a list of strings. It shall be a list of Python regular expression matching with the desired variants.

This type is used by the following types:

- *Build Option Item Type*

#### 6.2.2.3.32 Build Option Name

The value shall be a string. It shall be the name of the build option. The value shall match with the regular expression "`^[a-zA-Z_][a-zA-Z0-9_]*$`".

This type is used by the following types:

- *Build Option Item Type*

#### 6.2.2.3.33 Build Option Set Test State Action

This set of attributes specifies test states for a set of test programs. Generic attributes may be specified. Each generic attribute key shall be a *Name*. Each generic attribute value shall be a *Build Test State*. The keys shall be test program names. The names shall correspond to the name of a *Build Test Program Item Type* or *Build Ada Test Program Item Type* item. Due to the processing order of items, there is no way to check if the name specified by the attribute key is valid.

This type is used by the following types:

- *Build Option Action*

#### 6.2.2.3.34 Build Option Value

A value of this type shall be of one of the following variants:

- The value may be a boolean.
- The value may be an integer number.
- The value may be a list. Each list element shall be a string.
- There may by be no value (null).
- The value may be a string.

This type is used by the following types:

- *Build Option Action*
- *Build Option Default by Variant*
- *Build Option Item Type*

#### 6.2.2.3.35 Build Source

The value shall be a string. It shall be a source file. The path to a source file shall be relative to the directory of the `wscript`.

This type is used by the following types:

- *Build Ada Test Program Item Type*
- *Build BSP Item Type*
- *Build Library Item Type*
- *Build Objects Item Type*
- *Build Start File Item Type*
- *Build Test Program Item Type*

#### 6.2.2.3.36 Build Target

The value shall be a string. It shall be the target file path. The path to the target file shall be relative to the directory of the `wscript`. The target file is located in the build tree.

This type is used by the following types:

- *Build Ada Test Program Item Type*
- *Build Configuration File Item Type*
- *Build Configuration Header Item Type*
- *Build Library Item Type*
- *Build Start File Item Type*

- *Build Test Program Item Type*

#### 6.2.2.3.37 Build Test State

The value shall be a string. This string defines a test state. The value shall be an element of

- “benchmark”,
- “exclude”,
- “expected-fail”,
- “indeterminate”, and
- “user-input”.

This type is used by the following types:

- *Build Option Set Test State Action*

#### 6.2.2.3.38 Build Use After Directive

The value shall be a string. It shall be an internal static library identifier. The library is used to link programs referenced by this item, e.g. `z` for `libz.a`. The library is placed after the use items of the build item context.

This type is used by the following types:

- *Build Ada Test Program Item Type*
- *Build Group Item Type*
- *Build Script Item Type*
- *Build Test Program Item Type*

#### 6.2.2.3.39 Build Use Before Directive

The value shall be a string. It shall be an internal static library identifier. The library is used to link programs referenced by this item, e.g. `z` for `libz.a`. The library is placed before the use items of the build item context.

This type is used by the following types:

- *Build Ada Test Program Item Type*
- *Build Group Item Type*
- *Build Script Item Type*
- *Build Test Program Item Type*

#### 6.2.2.3.40 Constraint Link Role

This type refines the *Link* through the `role` attribute if the value is `constraint`. It defines the constraint role of links. The link target shall be a constraint.

#### 6.2.2.3.41 Copyright

The value shall be a string. It shall be a copyright statement of a copyright holder of the specification item. The value

- shall match with the regular expression "`^\s*Copyright\s+\(C\)\s+[0-9]+,\s*[0-9]+\s+.+\s*$`",
- or, shall match with the regular expression "`^\s*Copyright\s+\(C\)\s+[0-9]+\s+.+\s*$`",
- or, shall match with the regular expression "`^\s*Copyright\s+\(C\)\s+.+\s*$`".

This type is used by the following types:

- *Root Item Type*

#### 6.2.2.3.42 Enabled-By Expression

A value of this type shall be an expression which defines under which conditions the specification item or parts of it are enabled. The expression is evaluated with the use of an *enabled set*. This is a set of strings which indicate enabled features.

A value of this type shall be of one of the following variants:

- The value may be a boolean. This expression evaluates directly to the boolean value.
- The value may be a set of attributes. Each attribute defines an operator. Exactly one of the explicit attributes shall be specified. The explicit attributes for this type are:

  **and** The attribute value shall be a list. Each list element shall be an *Enabled-By Expression*. The *and* operator evaluates to the *logical and* of the evaluation results of the expressions in the list.

  **not** The attribute value shall be an *Enabled-By Expression*. The *not* operator evaluates to the *logical not* of the evaluation results of the expression.

  **or** The attribute value shall be a list. Each list element shall be an *Enabled-By Expression*. The *or* operator evaluates to the *logical or* of the evaluation results of the expressions in the list.

- The value may be a list. Each list element shall be an *Enabled-By Expression*. This list of expressions evaluates to the *logical or* of the evaluation results of the expressions in the list.
- The value may be a string. If the value is in the *enabled set*, this expression evaluates to true, otherwise to false.

This type is used by the following types:

- *Action Requirement Transition*
- *Enabled-By Expression*
- *Interface Include Link Role*
- *Root Item Type*

Please have a look at the following example:

```
enabled-by:
  and:
  - RTEMS_NETWORKING
  - not: RTEMS_SMP
```

#### 6.2.2.3.43 Glossary Membership Link Role

This type refines the *Link* through the `role` attribute if the value is `glossary-member`. It defines the glossary membership role of links.

#### 6.2.2.3.44 Integer or String

A value of this type shall be of one of the following variants:

- The value may be an integer number.
- The value may be a string.

This type is used by the following types:

- *Application Configuration Value Option Item Type*
- *Build Option Action*

#### 6.2.2.3.45 Interface Brief Description

A value of this type shall be of one of the following variants:

- There may by be no value (null).
- The value may be a string. It shall be the brief description of the interface. It should be a single sentence. The value shall not match with the regular expression "`\n\n`".

This type is used by the following types:

- *Interface Compound Item Type*
- *Interface Compound Member Definition*
- *Interface Define Item Type*
- *Interface Enum Item Type*
- *Interface Enumerator Item Type*
- *Interface Function Item Type*

- *Interface Group Item Type*
- *Interface Header File Item Type*
- *Interface Macro Item Type*
- *Interface Typedef Item Type*
- *Interface Variable Item Type*

#### 6.2.2.3.46 Interface Compound Definition Kind

The value shall be a string. It specifies how the interface compound is defined. It may be a typedef only, the struct or union only, or a typedef with a struct or union definition. The value shall be an element of

- “`struct-only`”,
- “`typedef-and-struct`”,
- “`typedef-and-union`”,
- “`typedef-only`”, and
- “`union-only`”.

This type is used by the following types:

- *Interface Compound Item Type*

#### 6.2.2.3.47 Interface Compound Member Compound

This type refines the following types:

- *Interface Compound Member Definition* through the `kind` attribute if the value is `struct`
- *Interface Compound Member Definition* through the `kind` attribute if the value is `union`

This set of attributes specifies an interface compound member compound. All explicit attributes shall be specified. The explicit attributes for this type are:

**definition** The attribute value shall be a list. Each list element shall be an *Interface Compound Member Definition Directive*.

#### 6.2.2.3.48 Interface Compound Member Declaration

This type refines the *Interface Compound Member Definition* through the `kind` attribute if the value is `member`. This set of attributes specifies an interface compound member declaration. All explicit attributes shall be specified. The explicit attributes for this type are:

**definition** The attribute value shall be a string. It shall be the interface compound member declaration. On the declaration a context-sensitive substitution of item variables is performed.

#### 6.2.2.3.49 Interface Compound Member Definition

This set of attributes specifies an interface compound member definition. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an *Interface Brief Description*.

**description** The attribute value shall be an *Interface Description*.

**kind** The attribute value shall be a string. It shall be the interface compound member kind.

**name** The attribute value shall be a string. It shall be the interface compound member name.

This type is refined by the following types:

- *Interface Compound Member Compound*
- *Interface Compound Member Declaration*

This type is used by the following types:

- *Interface Compound Member Definition Directive*
- *Interface Compound Member Definition Variant*

#### 6.2.2.3.50 Interface Compound Member Definition Directive

This set of attributes specifies an interface compound member definition directive. All explicit attributes shall be specified. The explicit attributes for this type are:

**default** The attribute value shall be an *Interface Compound Member Definition*. The default definition will be used if no variant-specific definition is enabled.

**variants** The attribute value shall be a list. Each list element shall be an *Interface Compound Member Definition Variant*.

This type is used by the following types:

- *Interface Compound Item Type*
- *Interface Compound Member Compound*

#### 6.2.2.3.51 Interface Compound Member Definition Variant

This set of attributes specifies an interface compound member definition variant. All explicit attributes shall be specified. The explicit attributes for this type are:

**definition** The attribute value shall be an *Interface Compound Member Definition*. The definition will be used if the expression defined by the `enabled-by` attribute evaluates to true. In generated header files, the expression is evaluated by the C preprocessor.

**enabled-by** The attribute value shall be an *Interface Enabled-By Expression*.

This type is used by the following types:

- *Interface Compound Member Definition Directive*

#### 6.2.2.3.52 Interface Definition

A value of this type shall be of one of the following variants:

- There may by be no value (null).
- The value may be a string. It shall be the definition. On the definition a context-sensitive substitution of item variables is performed.

This type is used by the following types:

- *Interface Definition Directive*
- *Interface Definition Variant*

#### 6.2.2.3.53 Interface Definition Directive

This set of attributes specifies an interface definition directive. All explicit attributes shall be specified. The explicit attributes for this type are:

**default** The attribute value shall be an *Interface Definition*. The default definition will be used if no variant-specific definition is enabled.

**variants** The attribute value shall be a list. Each list element shall be an *Interface Definition Variant*.

This type is used by the following types:

- *Interface Define Item Type*
- *Interface Enumerator Item Type*
- *Interface Macro Item Type*
- *Interface Typedef Item Type*
- *Interface Variable Item Type*

#### 6.2.2.3.54 Interface Definition Variant

This set of attributes specifies an interface definition variant. All explicit attributes shall be specified. The explicit attributes for this type are:

**definition** The attribute value shall be an *Interface Definition*. The definition will be used if the expression defined by the `enabled-by` attribute evaluates to true. In generated header files, the expression is evaluated by the C preprocessor.

**enabled-by** The attribute value shall be an *Interface Enabled-By Expression*.

This type is used by the following types:

- *Interface Definition Directive*

#### 6.2.2.3.55 Interface Description

A value of this type shall be of one of the following variants:

- There may by be no value (null).
- The value may be a string. It shall be the description of the interface. The description should be short and concentrate on the average case. All special cases, usage notes, constraints, error conditions, configuration dependencies, references, etc. should be described in the *Interface Notes*.

This type is used by the following types:

- *Application Configuration Option Item Type*
- *Interface Compound Item Type*
- *Interface Compound Member Definition*
- *Interface Define Item Type*
- *Interface Enum Item Type*
- *Interface Enumerator Item Type*
- *Interface Function Item Type*
- *Interface Group Item Type*
- *Interface Macro Item Type*
- *Interface Parameter*
- *Interface Return Value*
- *Interface Typedef Item Type*
- *Interface Variable Item Type*

#### 6.2.2.3.56 Interface Enabled-By Expression

A value of this type shall be an expression which defines under which conditions an interface definition is enabled. In generated header files, the expression is evaluated by the C preprocessor.

A value of this type shall be of one of the following variants:

- The value may be a boolean. It is converted to 0 or 1. It defines a symbol in the expression.
- The value may be a set of attributes. Each attribute defines an operator. Exactly one of the explicit attributes shall be specified. The explicit attributes for this type are:

  **and** The attribute value shall be a list. Each list element shall be an *Interface Enabled-By Expression*. The *and* operator defines a *logical and* of the expressions in the list.

  **not** The attribute value shall be an *Interface Enabled-By Expression*. The *not* operator defines a *logical not* of the expression.

**or** The attribute value shall be a list. Each list element shall be an *Interface Enabled-By Expression*. The *or* operator defines a *logical or* of the expressions in the list.

- The value may be a list. Each list element shall be an *Interface Enabled-By Expression*. It defines a *logical or* of the expressions in the list.
- The value may be a string. It defines a symbol in the expression.

This type is used by the following types:

- *Interface Compound Member Definition Variant*
- *Interface Definition Variant*
- *Interface Enabled-By Expression*
- *Interface Function Definition Variant*

#### 6.2.2.3.57 Interface Enum Definition Kind

The value shall be a string. It specifies how the enum is defined. It may be a typedef only, the enum only, or a typedef with an enum definition. The value shall be an element of

- “`enum-only`”,
- “`typedef-and-enum`”, and
- “`typedef-only`”.

This type is used by the following types:

- *Interface Enum Item Type*

#### 6.2.2.3.58 Interface Enumerator Link Role

This type refines the *Link* through the `role` attribute if the value is `interface-enumerator`. It defines the interface enumerator role of links.

#### 6.2.2.3.59 Interface Function Definition

This set of attributes specifies a function definition. All explicit attributes shall be specified. The explicit attributes for this type are:

**attributes** The attribute value shall be an optional string. If the value is present, then it shall be the function attributes. On the attributes a context-sensitive substitution of item variables is performed. A function attribute is for example the indication that the function does not return to the caller.

**body** The attribute value shall be an optional string. If the value is present, then it shall be the definition of a static inline function. On the function definition a context-sensitive substitution of item variables is performed. If no value is present, then the function is declared as an external function.

**params** The attribute value shall be a list of strings. It shall be the list of parameter declarations of the function. On the function parameter declarations a context-sensitive substitution of item variables is performed.

**return** The attribute value shall be a string. It shall be the function return type. On the return type a context-sensitive substitution of item variables is performed.

This type is used by the following types:

- *Interface Function Definition Directive*
- *Interface Function Definition Variant*

#### 6.2.2.3.60 Interface Function Definition Directive

This set of attributes specifies a function definition directive. All explicit attributes shall be specified. The explicit attributes for this type are:

**default** The attribute value shall be an *Interface Function Definition*. The default definition will be used if no variant-specific definition is enabled.

**variants** The attribute value shall be a list. Each list element shall be an *Interface Function Definition Variant*.

This type is used by the following types:

- *Interface Function Item Type*

#### 6.2.2.3.61 Interface Function Definition Variant

This set of attributes specifies a function definition variant. All explicit attributes shall be specified. The explicit attributes for this type are:

**definition** The attribute value shall be an *Interface Function Definition*. The definition will be used if the expression defined by the `enabled-by` attribute evaluates to true. In generated header files, the expression is evaluated by the C preprocessor.

**enabled-by** The attribute value shall be an *Interface Enabled-By Expression*.

This type is used by the following types:

- *Interface Function Definition Directive*

#### 6.2.2.3.62 Interface Function Link Role

This type refines the *Link* through the `role` attribute if the value is `interface-function`. It defines the interface function role of links. It is used to indicate that a *Action Requirement Item Type* item specifies functional requirements of an *Interface Function Item Type* or a *Interface Macro Item Type* item.

#### 6.2.2.3.63 Interface Group Identifier

The value shall be a string. It shall be the identifier of the interface group. The value shall match with the regular expression "`^[A-Z][a-zA-Z0-9]*$`".

This type is used by the following types:

- *Design Group Requirement Item Type*
- *Interface Group Item Type*

#### 6.2.2.3.64 Interface Group Membership Link Role

This type refines the *Link* through the `role` attribute if the value is `interface-ingroup`. It defines the interface group membership role of links.

#### 6.2.2.3.65 Interface Include Link Role

This type refines the *Link* through the `role` attribute if the value is `interface-include`. It defines the interface include role of links and is used to indicate that an interface container includes another interface container. For example, one header file includes another header file. All explicit attributes shall be specified. The explicit attributes for this type are:

**enabled-by** The attribute value shall be an *Enabled-By Expression*. It shall define under which conditions the interface container is included.

#### 6.2.2.3.66 Interface Notes

A value of this type shall be of one of the following variants:

- There may by be no value (null).
- The value may be a string. It shall be the notes for the interface.

This type is used by the following types:

- *Application Configuration Option Item Type*
- *Interface Compound Item Type*
- *Interface Define Item Type*
- *Interface Enumerator Item Type*
- *Interface Function Item Type*
- *Interface Macro Item Type*
- *Interface Typedef Item Type*
- *Interface Variable Item Type*

#### 6.2.2.3.67 Interface Parameter

This set of attributes specifies an interface parameter. All explicit attributes shall be specified. The explicit attributes for this type are:

**description** The attribute value shall be an *Interface Description*.

**dir** The attribute value shall be an *Interface Parameter Direction*.

**name** The attribute value shall be a string. It shall be the interface parameter name.

This type is used by the following types:

- *Interface Function Item Type*
- *Interface Macro Item Type*

#### 6.2.2.3.68 Interface Parameter Direction

A value of this type shall be of one of the following variants:

- There may by be no value (null).
- The value may be a string. It specifies the interface parameter direction. The value shall be an element of
  - “`in`”,
  - “`out`”, and
  - “`inout`”.

This type is used by the following types:

- *Interface Parameter*
- *Test Run Parameter*

#### 6.2.2.3.69 Interface Placement Link Role

This type refines the *Link* through the `role` attribute if the value is `interface-placement`. It defines the interface placement role of links. It is used to indicate that an interface definition is placed into an interface container, for example a header file.

#### 6.2.2.3.70 Interface References Set

This set of attributes defines references for the interface. Generic attributes may be specified. Each generic attribute key shall be a *Name*. Each generic attribute value shall be a string. The key defines the reference kind. The value shall be a kind-specific reference target.

This type is used by the following types:

- *Interface Unspecified Item Type*

#### 6.2.2.3.71 Interface Return Directive

This set of attributes specifies an interface return. All explicit attributes shall be specified. The explicit attributes for this type are:

**return** The attribute value shall be an optional string. It shall describe the interface return for unspecified return values.

**return-values** The attribute value shall be a list. Each list element shall be an *Interface Return Value*.

This type is used by the following types:

- *Interface Function Item Type*
- *Interface Macro Item Type*

#### 6.2.2.3.72 Interface Return Value

This set of attributes specifies an interface return value. All explicit attributes shall be specified. The explicit attributes for this type are:

**description** The attribute value shall be an *Interface Description*.

**value** The attribute value shall be a *Boolean or Integer or String*. It shall be the described interface return value.

This type is used by the following types:

- *Interface Return Directive*

#### 6.2.2.3.73 Interface Target Link Role

This type refines the *Link* through the `role` attribute if the value is `interface-target`. It defines the interface target role of links. It is used for interface forward declarations.

#### 6.2.2.3.74 Link

This set of attributes specifies a link from one specification item to another specification item. The links in a list are ordered. The first link in the list is processed first. All explicit attributes shall be specified. The explicit attributes for this type are:

**role** The attribute value shall be a *Name*. It shall be the role of the link.

**uid** The attribute value shall be an *UID*. It shall be the absolute or relative UID of the link target item.

This type is refined by the following types:

- *Application Configuration Group Member Link Role*
- *Build Dependency Link Role*
- *Constraint Link Role*

- *Glossary Membership Link Role*
- *Interface Enumerator Link Role*
- *Interface Function Link Role*
- *Interface Group Membership Link Role*
- *Interface Include Link Role*
- *Interface Placement Link Role*
- *Interface Target Link Role*
- *Placement Order Link Role*
- *Requirement Refinement Link Role*
- *Requirement Validation Link Role*
- *Runtime Measurement Request Link Role*
- *Specification Member Link Role*
- *Specification Refinement Link Role*
- *Unit Test Link Role*

This type is used by the following types:

- *Root Item Type*
- *Test Case Action*
- *Test Case Check*

#### 6.2.2.3.75 Name

The value shall be a string. A string is a valid name if it matches with the `^([a-z][a-z0-9-]*|SPDX-License-Identifier)$` regular expression.

This type is used by the following types:

- *Application Configuration Option Item Type*
- *Build Item Type*
- *Build Option Set Test State Action*
- *Functional Requirement Item Type*
- *Glossary Item Type*
- *Interface Item Type*
- *Interface References Set*
- *Link*
- *Non-Functional Requirement Item Type*
- *Requirement Item Type*

- *Root Item Type*
- *Runtime Measurement Parameter Set*
- *Runtime Performance Parameter Set*
- *Specification Attribute Value*
- *Specification Explicit Attributes*
- *Specification Generic Attributes*
- *Specification Item Type*
- *Specification List*
- *Specification Refinement Link Role*

#### 6.2.2.3.76 Optional String

A value of this type shall be of one of the following variants:

- There may by be no value (null).
- The value may be a string.

#### 6.2.2.3.77 Placement Order Link Role

This type refines the *Link* through the `role` attribute if the value is `placement-order`. This link role defines the placement order of items in a container item (for example an interface function in a header file or a documentation section).

#### 6.2.2.3.78 Requirement Reference

This set of attributes specifies a requirement reference. All explicit attributes shall be specified. The explicit attributes for this type are:

**identifier** The attribute value shall be a string. It shall be the type-specific identifier of the reference target. For *group* references use the Doxygen group identifier.

**type** The attribute value shall be a *Requirement Reference Type*.

This type is used by the following types:

- *Requirement Item Type*

#### 6.2.2.3.79 Requirement Reference Type

The value shall be a string. It specifies the type of a requirement reference. The value shall be an element of

- “`define`”,
- “`file`”,
- “`function`”,
- “`group`”,
- “`macro`”, and
- “`variable`”.

This type is used by the following types:

- *Requirement Reference*

#### 6.2.2.3.80 Requirement Refinement Link Role

This type refines the *Link* through the `role` attribute if the value is `requirement-refinement`. It defines the requirement refinement role of links.

#### 6.2.2.3.81 Requirement Text

The value shall be a string. It shall state a requirement or constraint. The value shall not contain an element of

- “`acceptable`”,
- “`adequate`”,
- “`almost always`”,
- “`and/or`”,
- “`appropriate`”,
- “`approximately`”,
- “`as far as possible`”,
- “`as much as practicable`”,
- “`best`”,
- “`best possible`”,
- “`easy`”,
- “`efficient`”,
- “`e.g.`”,
- “`enable`”,

- “enough”,
- “etc.”,
- “few”,
- “first rate”,
- “flexible”,
- “generally”,
- “goal”,
- “graceful”,
- “great”,
- “greatest”,
- “ideally”,
- “i.e.”,
- “if possible”,
- “in most cases”,
- “large”,
- “many”,
- “maximize”,
- “minimize”,
- “most”,
- “multiple”,
- “necessary”,
- “numerous”,
- “optimize”,
- “ought to”,
- “probably”,
- “quick”,
- “rapid”,
- “reasonably”,
- “relevant”,
- “robust”,
- “satisfactory”,
- “several”,
- “shall be included but not limited to”,

- “simple”,
- “small”,
- “some”,
- “state of the art”,
- “sufficient”,
- “suitable”,
- “support”,
- “systematically”,
- “transparent”,
- “typical”,
- “user friendly”,
- “usually”,
- “versatile”, and
- “when necessary”.

This type is used by the following types:

- *Action Requirement State*
- *Application Configuration Group Item Type*
- *Constraint Item Type*
- *Interface Group Item Type*
- *Requirement Item Type*

#### 6.2.2.3.82 Requirement Validation Link Role

This type refines the *Link* through the `role` attribute if the value is `validation`. It defines the requirement validation role of links.

#### 6.2.2.3.83 Requirement Validation Method

The value shall be a string. This value type characterizes a requirement validation method (except validation by test). The value shall be an element of

- “by-analysis”,
- “by-inspection”, and
- “by-review-of-design”.

This type is used by the following types:

- *Requirement Validation Item Type*

#### 6.2.2.3.84 Runtime Measurement Environment

The value shall be a string. It specifies the runtime measurement environment. The value

- shall be an element of
  - “`FullCache`”,
  - “`HotCache`”, and
  - “`DirtyCache`”,
- or, shall match with the regular expression “`^Load/[1-9][0-9]*$`”.

This type is used by the following types:

- *Runtime Measurement Environment Table*

#### 6.2.2.3.85 Runtime Measurement Environment Table

This set of attributes provides runtime performance limits for a set of runtime measurement environments. Generic attributes may be specified. Each generic attribute key shall be a *Runtime Measurement Environment*. Each generic attribute value shall be a *Runtime Measurement Value Table*.

This type is used by the following types:

- *Runtime Performance Limit Table*

#### 6.2.2.3.86 Runtime Measurement Parameter Set

This set of attributes defines parameters of the runtime measurement test case. All explicit attributes shall be specified. The explicit attributes for this type are:

**sample-count** The attribute value shall be an integer number. It shall be the sample count of the runtime measurement context.

In addition to the explicit attributes, generic attributes may be specified. Each generic attribute key shall be a *Name*. The attribute value may have any type.

This type is used by the following types:

- *Runtime Measurement Test Item Type*

#### 6.2.2.3.87 Runtime Measurement Request Link Role

This type refines the *Link* through the `role` attribute if the value is `runtime-measurement-request`. It defines the runtime measurement request role of links. The link target shall be a *Runtime Measurement Test Item Type* item.

#### 6.2.2.3.88 Runtime Measurement Value Kind

The value shall be a string. It specifies the kind of a runtime measurement value. The value shall be an element of

- “`max-lower-bound`”,
- “`max-upper-bound`”,
- “`mean-lower-bound`”,
- “`mean-upper-bound`”,
- “`min-lower-bound`”, and
- “`min-upper-bound`”.

This type is used by the following types:

- *Runtime Measurement Value Table*

#### 6.2.2.3.89 Runtime Measurement Value Table

This set of attributes provides a set of runtime measurement values each of a specified kind. The unit of the values shall be one second. Generic attributes may be specified. Each generic attribute key shall be a *Runtime Measurement Value Kind*. Each generic attribute value shall be a floating-point number.

This type is used by the following types:

- *Runtime Measurement Environment Table*

#### 6.2.2.3.90 Runtime Performance Limit Table

This set of attributes provides runtime performance limits for BSP variants specified by "<arch>/<bsp>" with <arch> being the architecture of the BSP and <bsp> being the base name of the BSP. Generic attributes may be specified. Each generic attribute key shall be a string. Each generic attribute value shall be a *Runtime Measurement Environment Table*.

This type is used by the following types:

- *Runtime Performance Requirement Item Type*

#### 6.2.2.3.91 Runtime Performance Parameter Set

This set of attributes defines parameters of the runtime performance requirement. Generic attributes may be specified. Each generic attribute key shall be a *Name*. The attribute value may have any type.

This type is used by the following types:

- *Runtime Performance Requirement Item Type*

#### 6.2.2.3.92 SPDX License Identifier

The value shall be a string. It defines the license of the item expressed though an SPDX License Identifier. The value

- shall be equal to “`CC-BY-SA-4.0 OR BSD-2-Clause`”,
- or, shall be equal to “`BSD-2-Clause`”,
- or, shall be equal to “`CC-BY-SA-4.0`”.

This type is used by the following types:

- *Root Item Type*

#### 6.2.2.3.93 Specification Attribute Set

This set of attributes specifies a set of attributes. The following explicit attributes are mandatory:

- `attributes`
- `description`
- `mandatory-attributes`

The explicit attributes for this type are:

**attributes** The attribute value shall be a *Specification Explicit Attributes*. It shall specify the explicit attributes of the attribute set.

**description** The attribute value shall be an optional string. It shall be the description of the attribute set.

**generic-attributes** The attribute value shall be a *Specification Generic Attributes*. It shall specify the generic attributes of the attribute set.

**mandatory-attributes** The attribute value shall be a *Specification Mandatory Attributes*. It shall specify the mandatory attributes of the attribute set.

This type is used by the following types:

- *Specification Information*

#### 6.2.2.3.94 Specification Attribute Value

This set of attributes specifies an attribute value. All explicit attributes shall be specified. The explicit attributes for this type are:

**description** The attribute value shall be an optional string. It shall be the description of the attribute value.

**spec-type** The attribute value shall be a *Name*. It shall be the specification type of the attribute value.

This type is used by the following types:

- *Specification Explicit Attributes*

#### 6.2.2.3.95 Specification Boolean Value

This attribute set specifies a boolean value. Only the `description` attribute is mandatory. The explicit attributes for this type are:

**assert** The attribute value shall be a boolean. This optional attribute defines the value constraint of the specified boolean value. If the value of the assert attribute is true, then the value of the specified boolean value shall be true. If the value of the assert attribute is false, then the value of the specified boolean value shall be false. In case the assert attribute is not present, then the value of the specified boolean value may be true or false.

**description** The attribute value shall be an optional string. It shall be the description of the specified boolean value.

This type is used by the following types:

- *Specification Information*

#### 6.2.2.3.96 Specification Explicit Attributes

Generic attributes may be specified. Each generic attribute key shall be a *Name*. Each generic attribute value shall be a *Specification Attribute Value*. Each generic attribute specifies an explicit attribute of the attribute set. The key of the each generic attribute defines the attribute key of the explicit attribute.

This type is used by the following types:

- *Specification Attribute Set*

#### 6.2.2.3.97 Specification Floating-Point Assert

A value of this type shall be an expression which asserts that the floating-point value of the specified attribute satisfies the required constraints.

A value of this type shall be of one of the following variants:

- The value may be a set of attributes. Each attribute defines an operator. Exactly one of the explicit attributes shall be specified. The explicit attributes for this type are:

  **and** The attribute value shall be a list. Each list element shall be a *Specification Floating-Point Assert*. The *and* operator evaluates to the *logical and* of the evaluation results of the expressions in the list.

  **eq** The attribute value shall be a floating-point number. The *eq* operator evaluates to true, if the value to check is equal to the value of this attribute, otherwise to false.

  **ge** The attribute value shall be a floating-point number. The *ge* operator evaluates to true, if the value to check is greater than or equal to the value of this attribute, otherwise to false.

  **gt** The attribute value shall be a floating-point number. The *gt* operator evaluates to true, if the value to check is greater than the value of this attribute, otherwise to false.

  **le** The attribute value shall be a floating-point number. The *le* operator evaluates to true, if the value to check is less than or equal to the value of this attribute, otherwise to false.

  **lt** The attribute value shall be a floating-point number. The *lt* operator evaluates to true, if the value to check is less than the value of this attribute, otherwise to false.

  **ne** The attribute value shall be a floating-point number. The *ne* operator evaluates to true, if the value to check is not equal to the value of this attribute, otherwise to false.

  **not** The attribute value shall be a *Specification Floating-Point Assert*. The *not* operator evaluates to the *logical not* of the evaluation results of the expression.

  **or** The attribute value shall be a list. Each list element shall be a *Specification Floating-Point Assert*. The *or* operator evaluates to the *logical or* of the evaluation results of the expressions in the list.

- The value may be a list. Each list element shall be a *Specification Floating-Point Assert*. This list of expressions evaluates to the *logical or* of the evaluation results of the expressions in the list.

This type is used by the following types:

- *Specification Floating-Point Assert*
- *Specification Floating-Point Value*

#### 6.2.2.3.98 Specification Floating-Point Value

This set of attributes specifies a floating-point value. Only the `description` attribute is mandatory. The explicit attributes for this type are:

**assert** The attribute value shall be a *Specification Floating-Point Assert*. This optional attribute defines the value constraints of the specified floating-point value. In case the assert attribute is not present, then the value of the specified floating-point value may be every valid floating-point number.

**description** The attribute value shall be an optional string. It shall be the description of the specified floating-point value.

This type is used by the following types:

- *Specification Information*

#### 6.2.2.3.99 Specification Generic Attributes

This set of attributes specifies generic attributes. Generic attributes are attributes which are not explicitly specified by *Specification Explicit Attributes*. They are restricted to uniform attribute key and value types. All explicit attributes shall be specified. The explicit attributes for this type are:

**description** The attribute value shall be an optional string. It shall be the description of the generic attributes.

**key-spec-type** The attribute value shall be a *Name*. It shall be the specification type of the generic attribute keys.

**value-spec-type** The attribute value shall be a *Name*. It shall be the specification type of the generic attribute values.

This type is used by the following types:

- *Specification Attribute Set*

#### 6.2.2.3.100 Specification Information

This set of attributes specifies attribute values. At least one of the explicit attributes shall be specified. The explicit attributes for this type are:

**bool** The attribute value shall be a *Specification Boolean Value*. It shall specify a boolean value.

**dict** The attribute value shall be a *Specification Attribute Set*. It shall specify a set of attributes.

**float** The attribute value shall be a *Specification Floating-Point Value*. It shall specify a floating-point value.

**int** The attribute value shall be a *Specification Integer Value*. It shall specify an integer value.

**list** The attribute value shall be a *Specification List*. It shall specify a list of attributes or values.

**none** The attribute shall have no value. It specifies that no value is required.

**str** The attribute value shall be a *Specification String Value*. It shall specify a string.

This type is used by the following types:

- *Specification Item Type*

#### 6.2.2.3.101 Specification Integer Assert

A value of this type shall be an expression which asserts that the integer value of the specified attribute satisfies the required constraints.

A value of this type shall be of one of the following variants:

- The value may be a set of attributes. Each attribute defines an operator. Exactly one of the explicit attributes shall be specified. The explicit attributes for this type are:

  **and** The attribute value shall be a list. Each list element shall be a *Specification Integer Assert*. The *and* operator evaluates to the *logical and* of the evaluation results of the expressions in the list.

  **eq** The attribute value shall be an integer number. The *eq* operator evaluates to true, if the value to check is equal to the value of this attribute, otherwise to false.

  **ge** The attribute value shall be an integer number. The *ge* operator evaluates to true, if the value to check is greater than or equal to the value of this attribute, otherwise to false.

  **gt** The attribute value shall be an integer number. The *gt* operator evaluates to true, if the value to check is greater than the value of this attribute, otherwise to false.

  **le** The attribute value shall be an integer number. The *le* operator evaluates to true, if the value to check is less than or equal to the value of this attribute, otherwise to false.

  **lt** The attribute value shall be an integer number. The *lt* operator evaluates to true, if the value to check is less than the value of this attribute, otherwise to false.

  **ne** The attribute value shall be an integer number. The *ne* operator evaluates to true, if the value to check is not equal to the value of this attribute, otherwise to false.

  **not** The attribute value shall be a *Specification Integer Assert*. The *not* operator evaluates to the *logical not* of the evaluation results of the expression.

  **or** The attribute value shall be a list. Each list element shall be a *Specification Integer Assert*. The *or* operator evaluates to the *logical or* of the evaluation results of the expressions in the list.

- The value may be a list. Each list element shall be a *Specification Integer Assert*. This list of expressions evaluates to the *logical or* of the evaluation results of the expressions in the list.

This type is used by the following types:

- *Specification Integer Assert*
- *Specification Integer Value*

#### 6.2.2.3.102 Specification Integer Value

This set of attributes specifies an integer value. Only the `description` attribute is mandatory. The explicit attributes for this type are:

**assert** The attribute value shall be a *Specification Integer Assert*. This optional attribute defines the value constraints of the specified integer value. In case the assert attribute is not present, then the value of the specified integer value may be every valid integer number.

**description** The attribute value shall be an optional string. It shall be the description of the specified integer value.

This type is used by the following types:

- *Specification Information*

#### 6.2.2.3.103 Specification List

This set of attributes specifies a list of attributes or values. All explicit attributes shall be specified. The explicit attributes for this type are:

**description** The attribute value shall be an optional string. It shall be the description of the list.

**spec-type** The attribute value shall be a *Name*. It shall be the specification type of elements of the list.

This type is used by the following types:

- *Specification Information*

#### 6.2.2.3.104 Specification Mandatory Attributes

It defines which explicit attributes are mandatory.

A value of this type shall be of one of the following variants:

- The value may be a list. Each list element shall be a *Name*. The list defines the mandatory attributes through their key names.
- The value may be a string. It defines how many explicit attributes are mandatory. If *none* is used, then none of the explicit attributes is mandatory, they are all optional. The value shall be an element of
  - “`all`”,
  - “`at-least-one`”,
  - “`at-most-one`”,
  - “`exactly-one`”, and
  - “none”.

This type is used by the following types:

- *Specification Attribute Set*

#### 6.2.2.3.105 Specification Member Link Role

This type refines the *Link* through the `role` attribute if the value is `spec-member`. It defines the specification membership role of links.

#### 6.2.2.3.106 Specification Refinement Link Role

This type refines the *Link* through the `role` attribute if the value is `spec-refinement`. It defines the specification refinement role of links. All explicit attributes shall be specified. The explicit attributes for this type are:

**spec-key** The attribute value shall be a *Name*. It shall be the specification type refinement attribute key of the specification refinement.

**spec-value** The attribute value shall be a *Name*. It shall be the specification type refinement attribute value of the specification refinement.

#### 6.2.2.3.107 Specification String Assert

A value of this type shall be an expression which asserts that the string of the specified attribute satisfies the required constraints.

A value of this type shall be of one of the following variants:

- The value may be a set of attributes. Each attribute defines an operator. Exactly one of the explicit attributes shall be specified. The explicit attributes for this type are:

  **and** The attribute value shall be a list. Each list element shall be a *Specification String Assert*. The *and* operator evaluates to the *logical and* of the evaluation results of the expressions in the list.

  **contains** The attribute value shall be a list of strings. The *contains* operator evaluates to true, if the string to check converted to lower case with all white space characters converted to a single space character contains a string of the list of strings of this attribute, otherwise to false.

  **eq** The attribute value shall be a string. The *eq* operator evaluates to true, if the string to check is equal to the value of this attribute, otherwise to false.

  **ge** The attribute value shall be a string. The *ge* operator evaluates to true, if the string to check is greater than or equal to the value of this attribute, otherwise to false.

  **gt** The attribute value shall be a string. The *gt* operator evaluates to true, if the string to check is greater than the value of this attribute, otherwise to false.

  **in** The attribute value shall be a list of strings. The *in* operator evaluates to true, if the string to check is contained in the list of strings of this attribute, otherwise to false.

  **le** The attribute value shall be a string. The *le* operator evaluates to true, if the string to check is less than or equal to the value of this attribute, otherwise to false.

**lt** The attribute value shall be a string. The *lt* operator evaluates to true, if the string to check is less than the value of this attribute, otherwise to false.

**ne** The attribute value shall be a string. The *ne* operator evaluates to true, if the string to check is not equal to the value of this attribute, otherwise to false.

**not** The attribute value shall be a *Specification String Assert*. The *not* operator evaluates to the *logical not* of the evaluation results of the expression.

**or** The attribute value shall be a list. Each list element shall be a *Specification String Assert*. The *or* operator evaluates to the *logical or* of the evaluation results of the expressions in the list.

**re** The attribute value shall be a string. The *re* operator evaluates to true, if the string to check matches with the regular expression of this attribute, otherwise to false.

**uid** The attribute shall have no value. The *uid* operator evaluates to true, if the string is a valid UID, otherwise to false.

- The value may be a list. Each list element shall be a *Specification String Assert*. This list of expressions evaluates to the *logical or* of the evaluation results of the expressions in the list.

This type is used by the following types:

- *Specification String Assert*
- *Specification String Value*

#### 6.2.2.3.108 Specification String Value

This set of attributes specifies a string. Only the `description` attribute is mandatory. The explicit attributes for this type are:

**assert** The attribute value shall be a *Specification String Assert*. This optional attribute defines the constraints of the specified string. In case the assert attribute is not present, then the specified string may be every valid string.

**description** The attribute value shall be an optional string. It shall be the description of the specified string attribute.

This type is used by the following types:

- *Specification Information*

#### 6.2.2.3.109 Test Case Action

This set of attributes specifies a test case action. All explicit attributes shall be specified. The explicit attributes for this type are:

**action-brief** The attribute value shall be an optional string. It shall be the test case action brief description.

**action-code** The attribute value shall be a string. It shall be the test case action code.

**checks** The attribute value shall be a list. Each list element shall be a *Test Case Check*.

**links** The attribute value shall be a list. Each list element shall be a *Link*. The links should use the *Requirement Validation Link Role* for validation tests and the *Unit Test Link Role* for unit tests.

This type is used by the following types:

- *Test Case Item Type*

#### 6.2.2.3.110 Test Case Check

This set of attributes specifies a test case check. All explicit attributes shall be specified. The explicit attributes for this type are:

**brief** The attribute value shall be an optional string. It shall be the test case check brief description.

**code** The attribute value shall be a string. It shall be the test case check code.

**links** The attribute value shall be a list. Each list element shall be a *Link*. The links should use the *Requirement Validation Link Role* for validation tests and the *Unit Test Link Role* for unit tests.

This type is used by the following types:

- *Test Case Action*

#### 6.2.2.3.111 Test Context Member

A value of this type shall be of one of the following variants:

- The value may be a set of attributes. This set of attributes defines an action requirement test context member. All explicit attributes shall be specified. The explicit attributes for this type are:

  **brief** The attribute value shall be an optional string. It shall be the test context member brief description.

  **description** The attribute value shall be an optional string. It shall be the test context member description.

  **member** The attribute value shall be a string. It shall be the test context member definition. It shall be a valid C structure member definition without a trailing ;.

- There may by be no value (null).

This type is used by the following types:

- *Action Requirement Item Type*
- *Runtime Measurement Test Item Type*
- *Test Case Item Type*

#### 6.2.2.3.112 Test Header

A value of this type shall be of one of the following variants:

- The value may be a set of attributes. This set of attributes specifies a test header. In case a test header is specified, then instead of a test case a test run function will be generated. The test run function will be declared in the test header target file and defined in the test source target file. The test run function can be used to compose test cases. The test header file is not automatically included in the test source file. It should be added to the includes or local includes of the test. All explicit attributes shall be specified. The explicit attributes for this type are:

  **code** The attribute value shall be an optional string. If the value is present, then it shall be the test header code. The header code is placed at file scope after the general test declarations and before the test run function declaration.

  **includes** The attribute value shall be a list of strings. It shall be a list of header files included by the header file via `#include <...>`.

  **local-includes** The attribute value shall be a list of strings. It shall be a list of header files included by the header file via `#include "..."`.

  **run-params** The attribute value shall be a list. Each list element shall be a *Test Run Parameter*.

  **target** The attribute value shall be a string. It shall be the path to the generated test header file.

- There may by be no value (null).

This type is used by the following types:

- *Action Requirement Item Type*
- *Test Case Item Type*

#### 6.2.2.3.113 Test Run Parameter

This set of attributes specifies a parameter for the test run function. In case this parameter is used in an *Action Requirement Item Type* item, then the parameter is also added as a member to the test context, see *Test Context Member*. All explicit attributes shall be specified. The explicit attributes for this type are:

**description** The attribute value shall be a string. It shall be the description of the parameter.

**dir** The attribute value shall be an *Interface Parameter Direction*.

**name** The attribute value shall be a string. It shall be the parameter name.

**specifier** The attribute value shall be a string. It shall be the complete function parameter specifier. Use `${.:name}` for the parameter name, for example `"int ${.:name}"`.

This type is used by the following types:

- *Test Header*

#### 6.2.2.3.114 Test Support Method

A value of this type shall be of one of the following variants:

- The value may be a set of attributes. This set of attributes defines an action requirement test support method. All explicit attributes shall be specified. The explicit attributes for this type are:

  **brief** The attribute value shall be an optional string. It shall be the test support method brief description.

  **code** The attribute value shall be a string. It shall be the test support method code. The code may use a local variable `ctx` which points to the test context, see *Test Context Member*.

  **description** The attribute value shall be an optional string. It shall be the test support method description.

- There may by be no value (null).

This type is used by the following types:

- *Action Requirement Item Type*
- *Runtime Measurement Test Item Type*
- *Runtime Performance Requirement Item Type*
- *Test Case Item Type*

#### 6.2.2.3.115 UID

The value shall be a string. The string shall be a valid absolute or relative item UID.

This type is used by the following types:

- *Link*

#### 6.2.2.3.116 Unit Test Link Role

This type refines the *Link* through the `role` attribute if the value is `unit-test`. It defines the unit test role of links. For unit tests the link target should be the *Interface Domain Item Type* containing the software unit. All explicit attributes shall be specified. The explicit attributes for this type are:

**name** The attribute value shall be a string. It shall be the name of the tested software unit.

### 6.2.3 Traceability of Specification Items

The standard ECSS-E-ST-10-06C demands that requirements shall be under configuration management, backwards-traceable and forward-traceable [ECS09a]. Requirements are a specialization of specification items in RTEMS.

#### 6.2.3.1 History of Specification Items

The RTEMS specification items should placed in the RTEMS sources using Git for version control. The history of specification items can be traced with Git. Special commit procedures for changes in specification item files should be established. For example, it should be allowed to change only one specification item per commit. A dedicated Git commit message format may be used as well, e.g. use of `Approved-by:` or `Reviewed-by:` lines which indicate an agreed statement (similar to the Linux kernel patch submission guidelines). Git commit procedures may be ensured through a server-side pre-receive hook. The history of requirements may be also added to the specification items directly in a *revision* attribute. This would make it possible to generate the history information for documents without having the Git repository available, e.g. from an RTEMS source release archive.

#### 6.2.3.2 Backward Traceability of Specification Items

Providing backward traceability of specification items means that we must be able to find the corresponding higher level specification item for each refined specification item. A custom tool needs to verify this.

#### 6.2.3.3 Forward Traceability of Specification Items

Providing forward traceability of specification items means that we must be able to find all the refined specification items for each higher level specification item. A custom tool needs to verify this. The links from parent to child specification items are implicitly defined by links from a child item to a parent item.

#### 6.2.3.4 Traceability between Software Requirements, Architecture and Design

The software requirements are implemented in custom YAML files, see *Specification Items*. The software architecture and design is written in Doxygen markup. Doxygen markup is used throughout all header and source files. A Doxygen filter program may be provided to place Doxygen markup in assembler files. The software architecture is documented via Doxygen groups. Each Doxygen group name should have a project-specific name and the name should be unique within the project, e.g. RTEMSTopLevelMidLevelLowLevel. The link from a Doxygen group to its parent group is realized through the `@ingroup` special command. The link from a Doxygen group or *software component* to the corresponding requirement is realized through a `@satisfy{req}` custom command which needs the identifier of the requirement as its one and only parameter. Only links to parents are explicitly given in the Doxygen markup. The links from a parent to its children are only implicitly specified via the link from a child to its parent. So, a tool must process all files to get the complete hierarchy of software requirements,

architecture and design. Links from a software component to another software component are realized through automatic Doxygen references or the `@ref` and `@see` special commands.

### 6.2.4 Requirement Management

#### 6.2.4.1 Change Control Board

Working with requirements usually involves a Change Control Board (*CCB*). The CCB of the RTEMS Project is the RTEMS developer mailing list.

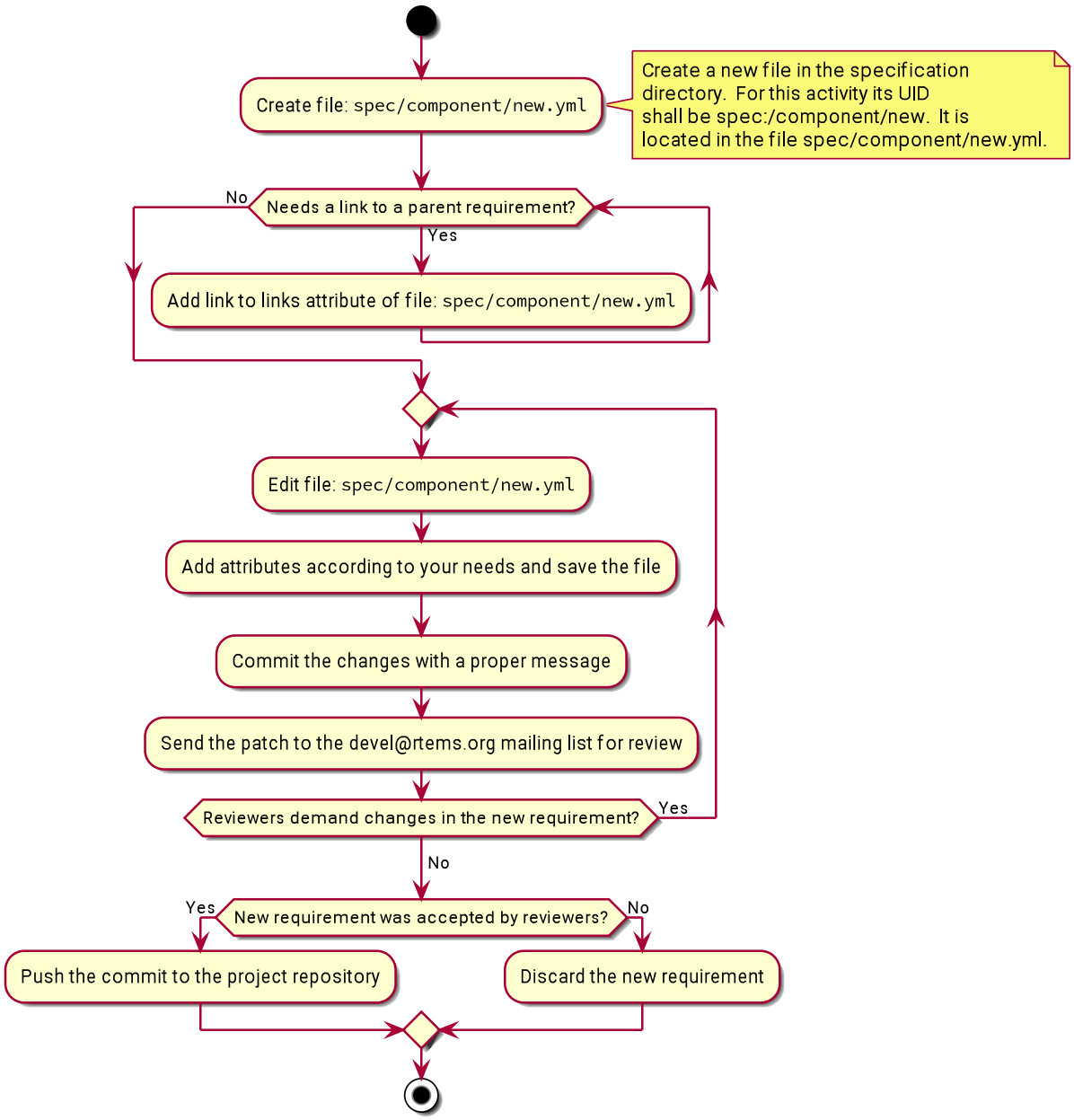
There are the following actors involved:

- *RTEMS users*: Everyone using the RTEMS real-time operating system to design, develop and build an application on top of it.
- *RTEMS developers*: The persons developing and maintaining RTEMS. They write patches to add or modify code, requirements, tests and documentation.
- *RTEMS maintainers*: They are listed in the MAINTAINERS file and have write access to the project repositories.

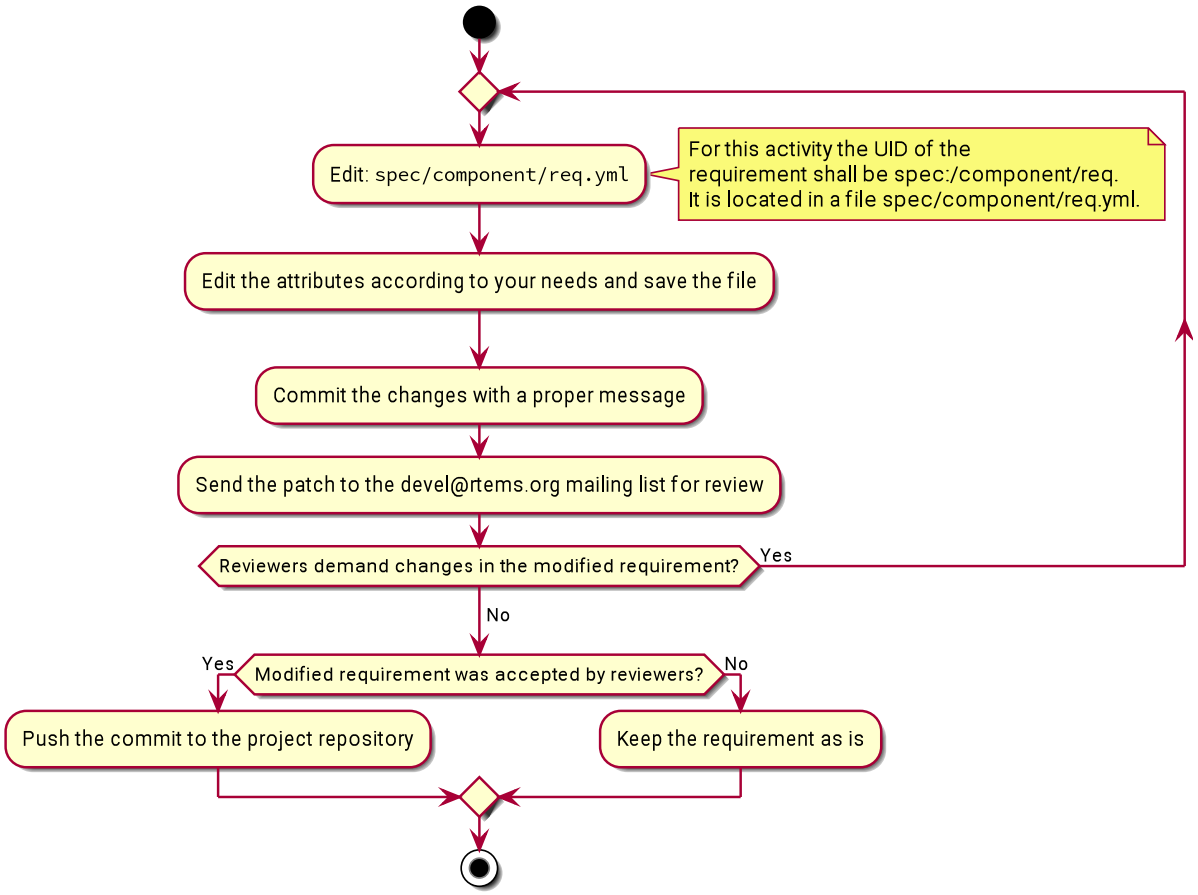
Adding and changing requirements follows the normal patch review process. The normal patch review process is described in the RTEMS User Manual. Reviews and comments may be submitted by anyone, but a maintainer review is required to approve *significant* changes. In addition for significant changes, there should be at least one reviewer with a sufficient independence from the author which proposes a new requirement or a change of an existing requirement. Working in another company on different projects is sufficiently independent. RTEMS maintainers do not know all the details, so they trust in general people with experience on a certain platform. Sometimes no review comments may appear in a reasonable time frame, then an implicit agreement to the proposed changes is assumed. Patches can be sent at anytime, so controlling changes in RTEMS requires a permanent involvement on the RTEMS developer mailing list.

For a qualification of RTEMS according to certain standards, the requirements may be approved by an RTEMS user. The approval by RTEMS users is not the concern of the RTEMS Project, however, the RTEMS Project should enable RTEMS users to manage the approval of requirements easily. This information may be also used by a independent authority which comes into play with an Independent Software Verification and Validation (*ISVV*). It could be used to select a subset of requirements, e.g. look only at the ones approved by a certain user. RTEMS users should be able to reference the determinative content of requirements, test procedures, test cases and justification reports in their own documentation. Changes in the determinative content should invalidate all references to previous versions.

#### 6.2.4.2 Add a Requirement



#### 6.2.4.3 Modify a Requirement




#### 6.2.4.4 Mark a Requirement as Obsolete

Requirements shall be never removed. They shall be marked as obsolete. This ensures that requirement identifiers are not reused. The procedure to obsolete a requirement is the same as the one to *modify a requirement*.

### 6.2.5 Tooling

#### 6.2.5.1 Tool Requirements

To manage requirements some tool support is helpful. Here is a list of requirements for the tool:

- The tool shall be open source.
- The tool should be actively maintained during the initial phase of the RTEMS requirements specification.
- The tool shall use plain text storage (no binary formats, no database).
- The tool shall support version control via Git.

- The tool should export the requirements in a human readable form using the Sphinx documentation framework.
- The tool shall support traceability of requirements to items external to the tool.
- The tool shall support traceability between requirements.
- The tool shall support custom requirement attributes.
- The tool should ensure that there are no cyclic dependencies between requirements.
- The tool should provide an export to *ReqIF*.

#### 6.2.5.2 Tool Evaluation

During an evaluation phase the following tools were considered:

- aNimble
- *Doorstop*
- OSRMT
- Papyrus
- ProR
- ReqIF Studio
- Requirement Heap
- rmToo

The tools aNimble, OSRMT and Requirement Heap were not selected since they use a database. The tools Papyrus, ProR and ReqIF are Eclipse based and use complex XML files for data storage. They were difficult to use and lack good documentation/tutorials. The tools rmToo and Doorstop turned out to be the best candidates to manage requirements in the RTEMS Project. The Doorstop tool was selected as the first candidate mainly due a recommendation by an RTEMS user.

#### 6.2.5.3 Best Available Tool - Doorstop

*Doorstop* is a requirements management tool. It has a modern, object-oriented and well-structured implementation in Python 3.6 under the LGPLv3 license. It uses a continuous integration build with style checkers, static analysis, documentation checks, code coverage, unit test and integration tests. In 2019, the project was actively maintained. Pull requests for minor improvements and new features were reviewed and integrated within days. Each requirement is contained in a single file in *YAML* format. Requirements are organized in documents and can be linked to each other [BA14].

Doorstop consists of three main parts

- a stateless command line tool *doorstop*,
- a file format with a pre-defined set of attributes (YAML), and
- a primitive GUI tool (not intended to be used).

For RTEMS, its scope could be extended to manage specifications in general. The primary reason for a close consideration of Doorstop as the requirements management tool for the RTEMS Project was its data format which allows a high degree of customization. Doorstop uses a directed, acyclic graph (DAG) of items. The items are files in YAML format. Each item has a set of standard attributes (key-value pairs).

The use case for the standard attributes is requirements management. However, Doorstop is capable to manage custom attributes as well. We will heavily use custom attributes for the specification items. Enabling Doorstop to effectively use custom attributes was done specifically for the RTEMS Project in several patch sets which in the end turned out to be not enough to use Doorstop for the RTEMS Project.

A key feature of Doorstop is the fingerprint of items. For the RTEMS Project, the fingerprint hash algorithm was changed from MD5 to SHA256. In 2019, it can be considered cryptographically secure. The fingerprint should cover the normative values of an item, e.g. comments etc. are not included. The fingerprint would help RTEMS users to track the significant changes in the requirements (in contrast to all the changes visible in Git). As an example use case, a user may want to assign a project-specific status to specification items. This can be done with a table which contains columns for

1. the UID of the item,
2. the fingerprint, and
3. the project-specific status.

Given the source code of RTEMS (which includes the specification items) and this table, it can be determined which items are unchanged and which have another status (e.g. unknown, changed, etc.).

After some initial work with Doorstop some issues surfaced (#471). It turned out that Doorstop is not designed as a library and contains too much policy. This results in a lack of flexibility required for the RTEMS Project.

1. Its primary use case is requirements management. So, it has some standard attributes useful in this domain, like derived, header, level, normative, ref, reviewed, and text. However, we want to use it more generally for specification items and these attributes make not always sense. Having them in every item is just overhead and may cause confusion.
2. The links cannot have custom attributes, e.g. role, enabled-by. With link-specific attributes you could have multiple DAGs formed up by the same set of items.
3. Inside a document (directory) items are supposed to have a common type (set of attributes). We would like to store at a hierarchy level also distinct specializations.
4. The verification of the items is quite limited. We need verification with type-based rules.
5. The UIDs in combination with the document hierarchy lead to duplication, e.g. a/b/c/a-b-c-d.yml. You have the path (a/b/c) also in the file name (a-b-c). You cannot have relative UIDs in links (e.g. ../parent-req) . The specification items may contain multiple requirements, e.g. min/max attributes. There is no way to identify them.
6. The links are ordered by Doorstop alphabetically by UID. For some applications, it would be better to use the order specified by the user. For example, we want to use specification items for a new build system. Here it is handy if you can express things like this: A is composed of B and C. Build B before C.

#### 6.2.5.4 Custom Requirements Management Tool

No requirements management tool was available that fits the need of the RTEMS Qualification Project. The decision was to develop a custom requirements management tool written in Python 3.6 or later. The design for it is heavily inspired by Doorstop.

### 6.2.6 How-To

#### 6.2.6.1 Getting Started

The RTEMS specification items and qualification tools are work in progress. The first step to work with the RTEMS specification and the corresponding tools is a clone of the following repository:

```
git clone git://git.rtems.org/rtems-central.git
git submodule init
git submodule update
```

The tools need a virtual Python 3 environment. To set it up use:

```
cd rtems-central
make env
```

Each time you want to use one of the tools, you have to activate the environment in your shell:

```
cd rtems-central
. env/bin/activate
```

#### 6.2.6.2 Application Configuration Options

The application configuration options and groups are maintained by specification items in the directory `spec/if/acfg`. Application configuration options are grouped by *Application Configuration Group Item Type* items which should be stored in files using the `spec/if/acfg/group-*.yml` pattern. Each application configuration option shall link to exactly one group item with the *Application Configuration Group Member Link Role*. There are four application option item types available which cover all existing options:

- The *feature enable options* let the application enable a feature option. If the option is not defined, then the feature is simply not available or active. There should be no feature-specific code linked to the application if the option is not defined. Examples are options which enable a device driver like `CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER`. These options are specified by *Application Configuration Feature Enable Option Item Type* items.
- The *feature options* let the application enable a specific feature option. If the option is not defined, then a default feature option is used. Regardless whether the option is defined or not defined, feature-specific code may be linked to the application. Examples are options which disable a feature if the option is defined such as `CONFIGURE_APPLICATION_DISABLE_FILESYSTEM` and options which provide a stub implementation of a feature by default and a full implementation if the option is defined such

as `CONFIGURE_IMFS_ENABLE_MKFIFO`. These options are specified by *Application Configuration Feature Option Item Type* items.

- The *integer value options* let the application define a specific value for a system parameter. If the option is not defined, then a default value is used for the system parameter. Examples are options which define the maximum count of objects available for application use such as `CONFIGURE_MAXIMUM_TASKS`. These options are specified by *Application Configuration Value Option Item Type* items.
- The *initializer options* let the application define a specific initializer for a system parameter. If the option is not defined, then a default setting is used for the system parameter. An example option of this type is `CONFIGURE_INITIAL_EXTENSIONS`. These options are specified by *Application Configuration Value Option Item Type* items.

Sphinx documentation sources and header files with Doxygen markup are generated from the specification items. The descriptions in the items shall use a restricted Sphinx formatting. Emphasis via one asterisk (“*”), strong emphasis via two asterisk (“**”), code samples via blockquotes (“``”), code blocks (“.. code-block:: c”) and lists are allowed. References to interface items are also allowed, for example “${appl-needs-clock-driver:/name}” and “${../rtems/tasks/create:/name}”. References to other parts of the documentation are possible, however, they are currently provided by hard-coded tables in `rtemsspec/applconfig.py`.

#### 6.2.6.2.1 Modify an Existing Group

Search for the group by its section header and edit the specification item file. For example:

```
$ grep -rl "name: General System Configuration" spec/if/acfg
spec/if/acfg/group-general.yml
$ vi spec/if/acfg/group-general.yml
```

#### 6.2.6.2.2 Modify an Existing Option

Search for the option by its C preprocessor define name and edit the specification item file. For example:

```
$ grep -rl CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER spec/if/acfg
spec/if/acfg/appl-needs-clock-driver.yml
$ vi spec/if/acfg/appl-needs-clock-driver.yml
```

#### 6.2.6.2.3 Add a New Group

Let `new` be the UID name part of the new group. Create the file `spec/if/acfg/group-new.yml` and provide all attributes for an *Application Configuration Group Item Type* item. For example:

```
$ vi spec/if/acfg/group-new.yml
```

#### 6.2.6.2.4 Add a New Option

Let `my-new-option` be the UID name of the option. Create the file `if/acfg/my-new-option.yml` and provide all attributes for an appropriate refinement of *Application Configuration Option Item Type*. For example:

```
$ vi spec/if/acfg/my-new-option.yml
```

#### 6.2.6.2.5 Generate Content after Changes

Once you are done with the modifications of an existing item or the creation of a new item, the changes need to be propagated to generated source files. This is done by the `spec2modules.py` script. Before you call this script, make sure the Git submodules are up-to-date.

```
$ ./spec2dmodules.py
```

The script modifies or creates source files in `modules/rtems` and `modules/rtems-docs`. Create patch sets for these changes just as if these were work done by a human and follow the normal patch review process described in the *RTEMS User Manual*. When the changes are integrated, update the Git submodules and check in the changed items.

### 6.2.6.3 Glossary Specification

The glossary of terms for the RTEMS Project is defined by *Glossary Term Item Type* items in the `spec/glossary` directory. For a new glossary term add a glossary item to this directory. As the file name use the term in lower case with all white space and special characters removed or replaced by alphanumeric characters, for example `spec/glossary/magicpower.yml` for the term *magic power*.

Use `${uid:/attribute}` substitutions to reference other parts of the specification.

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
enabled-by: true
glossary-type: term
links:
- role: glossary-member
  uid: ../glossary-general
term: magic power
text: |
  Magic power enables a caller to create magic objects using a
  ${magicwand:/term}.
type: glossary
```

Define acronyms with the phrase *This term is an acronym for *.* in the `text` attribute:

```
...
term: MP
...
```

(continues on next page)

(continued from previous page)

```
text: |
  This term is an acronym for Magic Power.
...
```

Once you are done with the glossary items, run the script `spec2modules.py` to generate the derived documentation content. Send patches for the generated documentation and the specification to the Developers Mailing List and follow the normal patch review process.

#### 6.2.6.4 Interface Specification

##### 6.2.6.4.1 Specify an API Header File

The RTEMS *API* header files are specified under `spec:/if/rtems/*`. Create a subdirectory with a corresponding name for the API, for example in `spec/if/rtems/foo` for the *foo* API. In this new subdirectory place an *Interface Header File Item Type* item named `header.yml` (`spec/if/rtems/foo/header.yml`) and populate it with the required attributes.

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
enabled-by: true
interface-type: header-file
links:
- role: interface-placement
  uid: /if/domains/api
path: rtems/rtems/foo.h
prefix: cpukit/include
type: interface
```

##### 6.2.6.4.2 Specify an API Element

Figure out the corresponding header file item. If it does not exist, see *Specify an API Header File*. Place a specialization of an *Interface Item Type* item into the directory of the header file item, for example `spec/if/rtems/foo/bar.yml` for the `bar()` function. Add the required attributes for the new interface item. Do not hard code interface names which are used to define the new interface. Use `${uid-of-interface-item:/name}` instead. If the referenced interface is specified in the same directory, then use a relative UID. Using interface references creates implicit dependencies and helps the header file generator to resolve the interface dependencies and header file includes for you. Use *Interface Unspecified Item Type* items for interface dependencies to other domains such as the C language, the compiler, the implementation, or user-provided defines. To avoid cyclic dependencies between types you may use an *Interface Forward Declaration Item Type* item.

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
brief: Tries to create a magic object and returns it.
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
```

(continues on next page)

(continued from previous page)

```
definition:
  default:
    body: null
    params:
    - ${magic-wand:/name} ${.:/params[0]/name}
    return: ${magic-type:/name} *
  variants: []
description: |
  The magic object is created out of nothing with the help of a magic wand.
enabled-by: true
interface-type: function
links:
- role: interface-placement
  uid: header
- role: interface-ingroup
  uid: /groups/api/classic/foo
name: bar
notes: null
params:
- description: is the magic wand.
  dir: null
  name: magic_wand
return:
  return: Otherwise, the magic object is returned.
  return-values:
  - description: The caller did not have enough magic power.
    value: ${/if/c/null}
type: interface
```

#### 6.2.6.5 Requirements Depending on Build Configuration Options

Use the `enabled-by` attribute of items or parts of an item to make it dependent on build configuration options such as `RTEMS_SMP` or architecture-specific options such as `CPU_ENABLE_ROBUST_THREAD_DISPATCH`, see *Enabled-By Expression*. With this attribute the specification can be customized at the level of an item or parts of an item. If the `enabled-by` attribute evaluates to false for a particular configuration, then the item or the associated part is disabled in the specification. The `enabled-by` attribute acts as a formalized *where* clause, see *recommended requirements syntax*.

Please have a look at the following example which specifies the transition map of `rtems_signal_catch()`:

```
transition-map:
- enabled-by: true
  post-conditions:
    Status: Ok
    ASRInfo:
    - if:
        pre-conditions:
          Handler: Valid
      then: New
```

(continues on next page)

(continued from previous page)

```
    - else: Inactive
  pre-conditions:
    Pending: all
    Handler: all
    Preempt: all
    Timeslice: all
    ASR: all
    IntLvl: all
- enabled-by: CPU_ENABLE_ROBUST_THREAD_DISPATCH
  post-conditions:
    Status: NotImplIntLvl
    ASRInfo: NopIntLvl
  pre-conditions:
    Pending: all
    Handler:
    - Valid
    Preempt: all
    Timeslice: all
    ASR: all
    IntLvl:
    - Positive
- enabled-by: RTEMS_SMP
  post-conditions:
    Status: NotImplNoPreempt
    ASRInfo: NopNoPreempt
  pre-conditions:
    Pending: all
    Handler:
    - Valid
    Preempt:
    - 'No'
    Timeslice: all
    ASR: all
    IntLvl: all
```

#### 6.2.6.6 Requirements Depending on Application Configuration Options

Requirements which depend on application configuration options such as `CONFIGURE_MAXIMUM_PROCESSORS` should be written in the following *syntax*:

> **Where** <feature is included>, the <system name> shall <system response>.

Use these clauses with care. Make sure all feature combinations are covered. Using a truth table may help. If a requirement depends on multiple features, use:

> **Where** <feature 0>, **where** <feature 1>, **where** <feature ...>, the <system name> shall <system response>.

For application configuration options, use the clauses like this:

`CONFIGURE_MAXIMUM_PROCESSORS` equal to one

> **Where** the system was configured with a processor maximum of exactly one, ...

`CONFIGURE_MAXIMUM_PROCESSORS` greater than one

**Where** the system was configured with a processor maximum greater than one, ...

Please have a look at the following example used to specify `rtems_signal_catch()`. The example is a post-condition state specification of an action requirement, so there is an implicit set of pre-conditions and the trigger:

**While** <pre-condition(s)>, **when** rtems_signal_catch() is called, ...

The *where* clauses should be mentally placed before the *while* clauses.

```
post-conditions:
- name: ASRInfo
  states:
  - name: NopNoPreempt
    test-code: |
      if ( rtems_configuration_get_maximum_processors() > 1 ) {
        CheckNoASRChange( ctx );
      } else {
        CheckNewASRSettings( ctx );
      }
    text: |
      Where the scheduler does not support the no-preempt mode, the ASR
      information of the caller of ${../if/catch:/name} shall not be
      changed by the ${../if/catch:/name} call.

      Where the scheduler does support the no-preempt mode, the ASR
      processing for the caller of ${../if/catch:/name} shall be done using
      the handler specified by ${../if/catch:/params[0]/name} in the mode
      specified by ${../if/catch:/params[1]/name}.
```

### 6.2.6.7 Action Requirements

*Action Requirement Item Type* items may be used to specify and validate directive calls. They are a generator for event-driven requirements. Event-driven requirements should be written in the following *syntax*:

> **While** <pre-condition 0>, **while** <pre-condition 1>, ..., **while** <pre-condition *n*>, **when** <trigger>, the <system name> shall <system response>.

The list of *while* <pre-condition *i*> clauses for *i* from 1 to *n* in the EARS notation is generated by *n* pre-condition states in the action requirement item, see the `pre-condition` attribute in the *Action Requirement Item Type*.

The <trigger> in the EARS notation is defined for an action requirement item by the link to an *Interface Function Item Type* or an *Interface Macro Item Type* item using the *Interface Function Link Role*. The code provided by the `test-action` attribute defines the action code which should invoke the trigger directive in a particular set of pre-condition states.

Each post-condition state of the action requirement item generates a <system name> shall <system response> clause in the EARS notation, see the `post-condition` attribute in the *Action Requirement Item Type*.

Each entry in the transition map is an event-driven requirement composed of the pre-condition states, the trigger defined by the link to a directive, and the post-condition states. The transition map is defined by a list of *Action Requirement Transition* descriptors.

Use `CamelCase` for the pre-condition names, post-condition names, and state names in action requirement items. The more conditions a directive has, the shorter should be the names. The transition map may be documented as a table and more conditions need more table columns. Use item attribute references in the `text` attributes. This allows context-sensitive substitutions.

#### 6.2.6.7.1 Example

Lets have a look at an example of an action requirement item. We would like to specify and validate the behaviour of the

```
rtems_status_code rtems_timer_create( rtems_name name, rtems_id *id );
```

directive which is particularly simple. For a more complex example see the specification of `rtems_signal_catch()` or `rtems_signal_send()` in `spec:/rtems/signal/req/catch` or `spec:/rtems/signal/send` respectively.

The event triggers are calls to `rtems_timer_create()`. Firstly, we need the list of pre-conditions relevant to this directive. Good candidates are the directive parameters, this gives us the `Name` and `Id` conditions. A system condition is if an inactive timer object is available so that we can create a timer, this gives us the `Free` condition. Secondly, we need the list of post-conditions relevant to this directive. They are the return status of the directive, `Status`, the validity of a unique object name, `Name`, and the value of an object identifier variable, `IdVar`. Each condition has a set of states, see the YAML data below for the details. The specified conditions and states yield the following transition map:

| Entry | Descriptor | Name | Id | Free | Status | Name | IdVar |
|---|---|---|---|---|---|---|---|
| 0 | 0 | Valid | Valid | Yes | Ok | Valid | Set |
| 1 | 0 | Valid | Valid | No | TooMany | Invalid | Nop |
| 2 | 0 | Valid | Null | Yes | InvAddr | Invalid | Nop |
| 3 | 0 | Valid | Null | No | InvAddr | Invalid | Nop |
| 4 | 0 | Invalid | Valid | Yes | InvName | Invalid | Nop |
| 5 | 0 | Invalid | Valid | No | InvName | Invalid | Nop |
| 6 | 0 | Invalid | Null | Yes | InvName | Invalid | Nop |
| 7 | 0 | Invalid | Null | No | InvName | Invalid | Nop |

Not all transition maps are that small, the transition map of `rtems_task_mode()` has more than 8000 entries. We can construct requirements from the clauses of the entries. For example, the three requirements of entry 0 (Name=Valid, Id=Valid, and Free=Yes results in Status=Ok, Name=Valid, and IdVar=Set) are:

> While the `name` parameter is valid, while the `id` parameter references an object of type rtems_id, while the system has at least one inactive timer object available, when rtems_timer_create() is called, the return status of rtems_timer_create() shall be RTEMS_SUCCESSFUL.
>
> While the `name` parameter is valid, while the `id` parameter references an object of

type rtems_id, while the system has at least one inactive timer object available, when rtems_timer_create() is called, the unique object name shall identify the timer created by the rtems_timer_create() call.

While the `name` parameter is valid, while the `id` parameter references an object of type rtems_id, while the system has at least one inactive timer object available, when rtems_timer_create() is called, the value of the object referenced by the `id` parameter shall be set to the object identifier of the created timer after the return of the rtems_timer_create() call.

Now we will have a look at the specification item line by line. The top-level attributes are normally in alphabetical order in an item file. For this presentation we use a structured order.

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
copyrights:
- Copyright (C) 2021 embedded brains GmbH (http://www.embedded-brains.de)
enabled-by: true
functional-type: action
rationale: null
references: []
requirement-type: functional
```

The specification items need a bit of boilerplate to tell you what they are, who wrote them, and what their license is.

```
text: ${.:text-template}
```

Each requirement item needs a `text` attribute. For the action requirements, we do not have a single requirement. There is just a template indicator and no plain text. Several event-driven requirements are defined by the pre-conditions, the trigger, and the post-conditions.

```
pre-conditions:
- name: Name
  states:
  - name: Valid
    test-code: |
      ctx->name = NAME;
    text: |
      While the ${../if/create:/params[0]/name} parameter is valid.
  - name: Invalid
    test-code: |
      ctx->name = 0;
    text: |
      While the ${../if/create:/params[0]/name} parameter is invalid.
  test-epilogue: null
  test-prologue: null
- name: Id
  states:
  - name: Valid
    test-code: |
      ctx->id = &ctx->id_value;
    text: |
      While the ${../if/create:/params[1]/name} parameter references an object
      of type ${../../type/if/id:/name}.
```

(continues on next page)

(continued from previous page)

```
  - name: 'Null'
    test-code: |
      ctx->id = NULL;
    text: |
      While the ${../if/create:/params[1]/name} parameter is
      ${/c/if/null:/name}.
  test-epilogue: null
  test-prologue: null
- name: Free
  states:
  - name: 'Yes'
    test-code: |
      /* Ensured by the test suite configuration */
    text: |
      While the system has at least one inactive timer object available.
  - name: 'No'
    test-code: |
      ctx->seized_objects = T_seize_objects( Create, NULL );
    text: |
      While the system has no inactive timer object available.
  test-epilogue: null
  test-prologue: null
```

This list defines the pre-conditions. Each pre-condition has a list of states and corresponding validation test code.

```
links:
- role: interface-function
  uid: ../if/create
test-action: |
  ctx->status = rtems_timer_create( ctx->name, ctx->id );
```

The link to the `rtems_timer_create()` interface specification item with the `interface-function` link role defines the trigger. The `test-action` defines the how the triggering directive is invoked for the validation test depending on the pre-condition states. The code is not always as simple as in this example. The validation test is defined in this item along with the specification.

```
post-conditions:
- name: Status
  states:
  - name: Ok
    test-code: |
      T_rsc_success( ctx->status );
    text: |
      The return status of ${../if/create:/name} shall be
      ${../../status/if/successful:/name}.
  - name: InvName
    test-code: |
      T_rsc( ctx->status, RTEMS_INVALID_NAME );
    text: |
      The return status of ${../if/create:/name} shall be
      ${../../status/if/invalid-name:/name}.
  - name: InvAddr
```

(continues on next page)

(continued from previous page)

```
    test-code: |
      T_rsc( ctx->status, RTEMS_INVALID_ADDRESS );
    text: |
      The return status of ${../if/create:/name} shall be
      ${../../status/if/invalid-address:/name}.
  - name: TooMany
    test-code: |
      T_rsc( ctx->status, RTEMS_TOO_MANY );
    text: |
      The return status of ${../if/create:/name} shall be
      ${../../status/if/too-many:/name}.
  test-epilogue: null
  test-prologue: null
- name: Name
  states:
  - name: Valid
    test-code: |
      id = 0;
      sc = rtems_timer_ident( NAME, &id );
      T_rsc_success( sc );
      T_eq_u32( id, ctx->id_value );
    text: |
      The unique object name shall identify the timer created by the
      ${../if/create:/name} call.
  - name: Invalid
    test-code: |
      sc = rtems_timer_ident( NAME, &id );
      T_rsc( sc, RTEMS_INVALID_NAME );
    text: |
      The unique object name shall not identify a timer.
  test-epilogue: null
  test-prologue: |
    rtems_status_code sc;
    rtems_id          id;
- name: IdVar
  states:
  - name: Set
    test-code: |
      T_eq_ptr( ctx->id, &ctx->id_value );
      T_ne_u32( ctx->id_value, INVALID_ID );
    text: |
      The value of the object referenced by the ${../if/create:/params[1]/name}
      parameter shall be set to the object identifier of the created timer
      after the return of the ${../if/create:/name} call.
  - name: Nop
    test-code: |
      T_eq_u32( ctx->id_value, INVALID_ID );
    text: |
      Objects referenced by the ${../if/create:/params[1]/name} parameter in
      past calls to ${../if/create:/name} shall not be accessed by the
      ${../if/create:/name} call.
  test-epilogue: null
  test-prologue: null
```

This list defines the post-conditions. Each post-condition has a list of states and corresponding validation test code.

```
skip-reasons: {}
transition-map:
- enabled-by: true
  post-conditions:
    Status:
    - if:
        pre-conditions:
          Name: Invalid
      then: InvName
    - if:
        pre-conditions:
          Id: 'Null'
      then: InvAddr
    - if:
        pre-conditions:
          Free: 'No'
      then: TooMany
    - else: Ok
    Name:
    - if:
        post-conditions:
          Status: Ok
      then: Valid
    - else: Invalid
    IdVar:
    - if:
        post-conditions:
          Status: Ok
      then: Set
    - else: Nop
  pre-conditions:
    Name: all
    Id: all
    Free: all
type: requirement
```

This list of transition descriptors defines the transition map. For the post-conditions, you can use expressions to ease the specification, see *Action Requirement Transition Post-Condition State*. The `skip-reasons` can be used to skip entire entries in the transition map, see *Action Requirement Skip Reasons*.

```
test-brief: null
test-description: null
```

The item contains the validation test code. The validation test in general can be described by these two attributes.

```
test-target: testsuites/validation/tc-timer-create.c
```

This is the target file for the generated validation test code. Make sure this file is included in the build specification, otherwise the test code generation will fail.

```
test-includes:
- rtems.h
- string.h
test-local-includes: []
```

You can specify a list of includes for the validation test.

```
test-header: null
```

A test header may be used to create a parameterized validation test, see *Test Header*. This is an advanced topic, see the specification of `rtems_task_ident()` for an example.

```
test-context-support: null
test-context:
- brief: |
    This member is used by the T_seize_objects() and T_surrender_objects()
    support functions.
  description: null
  member: |
    void *seized_objects
- brief: |
    This member may contain the object identifier returned by
    rtems_timer_create().
  description: null
  member: |
    rtems_id id_value
- brief: |
    This member specifies the ${../if/create:/params[0]/name} parameter for the
    action.
  description: null
  member: |
    rtems_name name
- brief: |
    This member specifies the ${../if/create:/params[1]/name} parameter for the
    action.
  description: null
  member: |
    rtems_id *id
- brief: |
    This member contains the return status of the action.
  description: null
  member: |
    rtems_status_code status
```

You can specify a list of validation test context members which can be used to maintain the state of the validation test. The context is available through an implicit `ctx` variable in all code blocks except the support blocks. The context support code can be used to define test-specific types used by context members. Do not use global variables.

```
test-support: |
  #define NAME rtems_build_name( 'T', 'E', 'S', 'T' )

  #define INVALID_ID 0xffffffff
```

(continues on next page)

(continued from previous page)

```
  static rtems_status_code Create( void *arg, uint32_t *id )
  {
    return rtems_timer_create( rtems_build_name( 'S', 'I', 'Z', 'E' ), id );
  }
```

The support code block can be used to provide functions, data structures, and constants for the validation test.

```
test-prepare: null
test-cleanup: |
  if ( ctx->id_value != INVALID_ID ) {
    rtems_status_code sc;

    sc = rtems_timer_delete( ctx->id_value );
    T_rsc_success( sc );

    ctx->id_value = INVALID_ID;
  }

  T_surrender_objects( &ctx->seized_objects, rtems_timer_delete );
```

The validation test basically executes a couple of nested for loops to iterate over each pre-condition and each state of the pre-conditions. These two optional code blocks can be used to prepare the pre-condition state preparations and clean up after the post-condition checks in each loop iteration.

```
test-setup:
  brief: null
  code: |
    memset( ctx, 0, sizeof( *ctx ) );
    ctx->id_value = INVALID_ID;
  description: null
test-stop: null
test-teardown: null
```

These optional code blocks correspond to test fixture methods, see the *RTEMS Test Framework*.

#### 6.2.6.7.2 Pre-Condition Templates

Specify all directive parameters as separate pre-conditions. Use the following syntax for directive object identifier parameters:

```
- name: Id
  states:
  - name: NoObj
    test-code: |
      ctx->id = 0xffffffff;
    text: |
      While the ${../if/directive:/params[0]/name} parameter is not
      associated with a thing.
```

(continues on next page)

(continued from previous page)

```
  - name: ClassA
    test-code: |
      ctx->id = ctx->class_a_id;
    text: |
      While the ${../if/directive:/params[0]/name} parameter is associated
      with a class A thing.
  - name: ClassB
    test-code: |
      ctx->id = ctx->class_b_id;
    text: |
      While the ${../if/directive:/params[0]/name} parameter is associated
      with a class B thing.
  test-epilogue: null
  test-prologue: null
```

Do not add specifications for invalid pointers. In general, there are a lot of invalid pointers and the use of an invalid pointer is in almost all cases undefined behaviour in RTEMS. There may be specifications for special cases which deal with some very specific invalid pointers such as the `NULL` pointer or pointers which do not satisfy a range or boundary condition. Use the following syntax for directive pointer parameters:

```
- name: Id
  states:
  - name: Valid
    test-code: |
      ctx->id = &ctx->id_value;
    text: |
      While the ${../if/directive:/params[3]/name} parameter references an
      object of type ${../../type/if/id:/name}.
  - name: 'Null'
    test-code: |
      ctx->id = NULL;
    text: |
      While the ${../if/directive:/params[3]/name} parameter is
      ${/c/if/null:/name}.
  test-epilogue: null
  test-prologue: null
```

Use the following syntax for other directive parameters:

```
- name: Name
  states:
  - name: Valid
    test-code: |
      ctx->name = NAME;
    text: |
      While the ${../if/directive:/params[0]/name} parameter is valid.
  - name: Invalid
    test-code: |
      ctx->name = 0;
    text: |
      While the ${../if/directive:/params[0]/name} parameter is invalid.
  test-epilogue: null
```

(continues on next page)

(continued from previous page)

```
  test-prologue: null
```

#### 6.2.6.7.3 Post-Condition Templates

Do not mix different things into one post-condition. If you write multiple sentences to describe what happened, then think about splitting up the post-condition. Keep the post-condition simple and focus on one testable aspect which may be changed by a directive call.

For directives returning an `rtems_status_code` use the following post-condition states. Specify only status codes which may be returned by the directive. Use it as the first post-condition. The first state shall be `Ok`. The other states shall be listed in the order in which they can occur.

```
- name: Status
  states:
  - name: Ok
    test-code: |
      T_rsc_success( ctx->status );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/successful:/name}.
  - name: IncStat
    test-code: |
      T_rsc( ctx->status, RTEMS_INCORRECT_STATE );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/incorrect-state:/name}.
  - name: InvAddr
    test-code: |
      T_rsc( ctx->status, RTEMS_INVALID_ADDRESS );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/invalid-address:/name}.
  - name: InvName
    test-code: |
      T_rsc( ctx->status, RTEMS_INVALID_NAME );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/invalid-name:/name}.
  - name: InvNum
    test-code: |
      T_rsc( ctx->status, RTEMS_INVALID_NUMBER );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/invalid-number:/name}.
  - name: InvSize
    test-code: |
      T_rsc( ctx->status, RTEMS_INVALID_SIZE );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/invalid-size:/name}.
  - name: InvPrio
```

(continues on next page)

(continued from previous page)

```
    test-code: |
      T_rsc( ctx->status, RTEMS_INVALID_PRIORITY );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/invalid-priority:/name}.
  - name: NotConf
    test-code: |
      T_rsc( ctx->status, RTEMS_NOT_CONFIGURED );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/not-configured:/name}.
  - name: NotDef
    test-code: |
      T_rsc( ctx->status, RTEMS_NOT_DEFINED );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/not-defined:/name}.
  - name: NotImpl
    test-code: |
      T_rsc( ctx->status, RTEMS_NOT_IMPLEMENTED );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/not-implemented:/name}.
  - name: TooMany
    test-code: |
      T_rsc( ctx->status, RTEMS_TOO_MANY );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/too-many:/name}.
  - name: Unsat
    test-code: |
      T_rsc( ctx->status, RTEMS_UNSATISFIED  );
    text: |
      The return status of ${../if/directive:/name} shall be
      ${../../status/if/unsatisfied:/name}.
  test-epilogue: null
  test-prologue: null
```

For values which are returned by reference through directive parameters, use the following post-condition states.

```
- name: SomeParamVar
  states:
  - name: Set
    test-code: |
      /* Add code to check that the object value was set to X */
    text: |
      The value of the object referenced by the
      ${../if/directive:/params[0]/name} parameter shall be set to X after
      the return of the ${../if/directive:/name} call.
  - name: Nop
    test-code: |
      /* Add code to check that the object was not modified */
```

(continues on next page)

(continued from previous page)

```
    text: |
      Objects referenced by the ${../if/directive:/params[0]/name}
      parameter in past calls to ${../if/directive:/name} shall not be
      accessed by the ${../if/directive:/name} call.
```

## 6.3 Applicable and Reference Documents

The “Applicable and reference documents” sections of the QDP documentation set will use the following template:

```
Applicable and reference documents
**********************************

Applicable documents
====================
```

If there are no applicable documents, then this template will be used:

```
There are no :term:`applicable documents <applicable document>`.
```

If there are applicable documents, then this template will be used:

```
The following are :term:`applicable documents <applicable document>`:

* :cite:`ABC`

* :cite:`XYZ`
```

```
Reference documents
===================

.. raw:: latex

   For reference documents see the
   {\hyperref[\detokenize{rtemssmp:bibliography}]{\sphinxcrossref{\DUrole{std,std-ref}
↪{bibliography}}}}.

.. only:: html

   For reference documents see the :ref:`bibliography <Bibliography>`.
```

## 6.4 Terms, Definitions and Abbreviated Terms

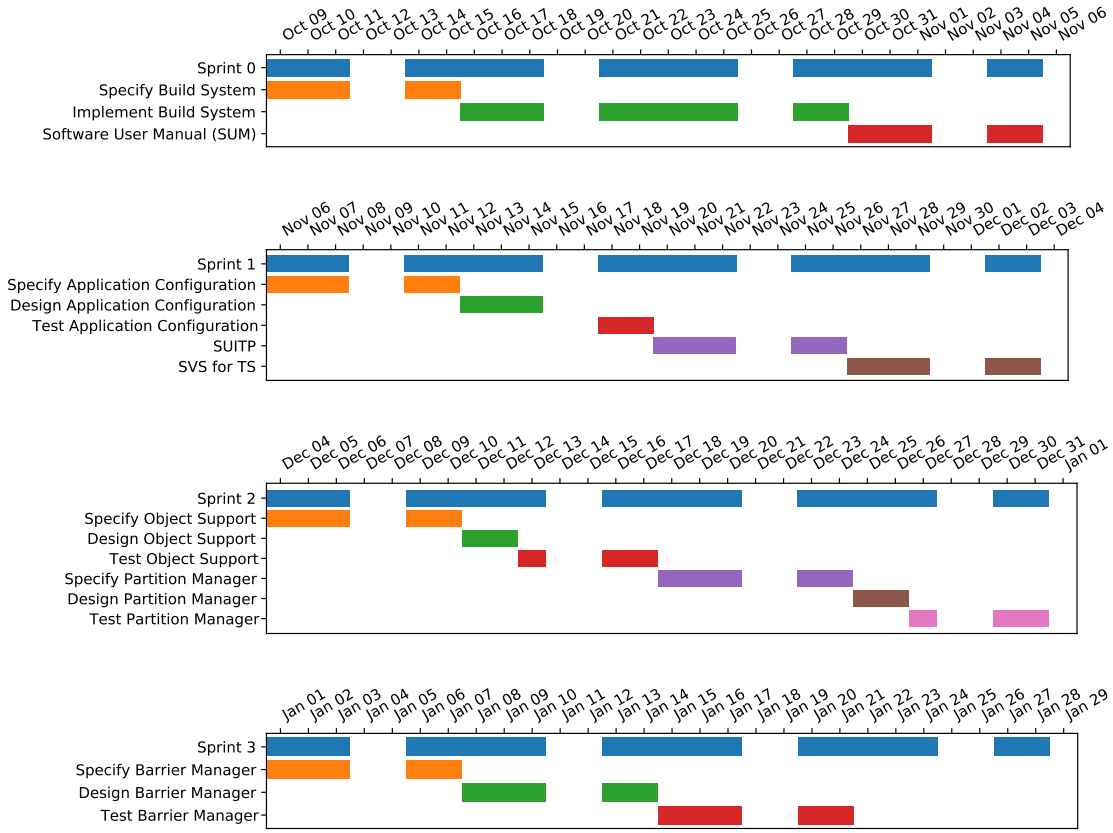
The “Terms, definitions and abbreviated terms” sections of the QDP documentation set will use the following template:

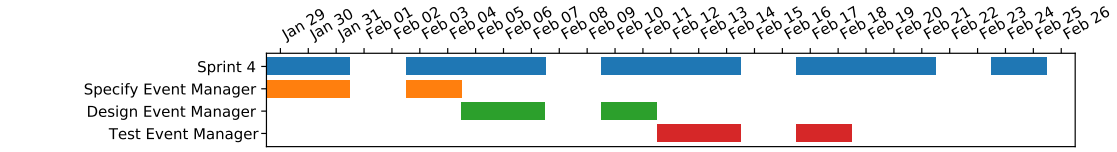
```
.. include:: ../path/to/glossary.rst
```

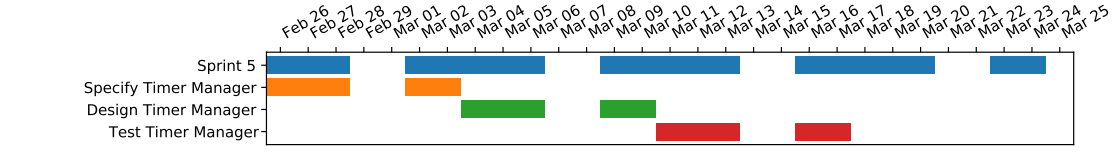
Each document of the QDP will include a QDP-specific glossary.

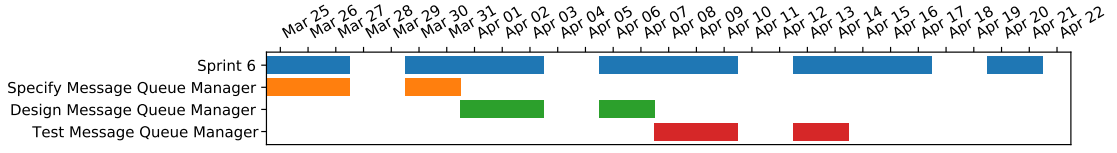
```
Terms, definitions and abbreviated terms
****************************************

.. glossary::
    :sorted:

    ABC
        And so on
```

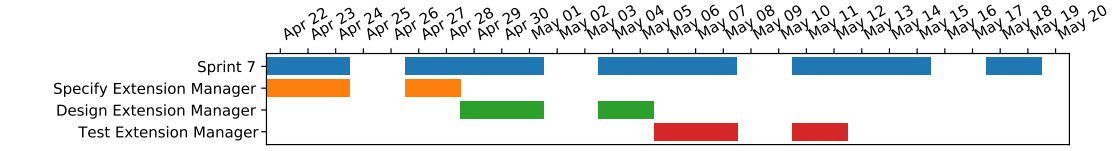
## 6.5 Work Packages

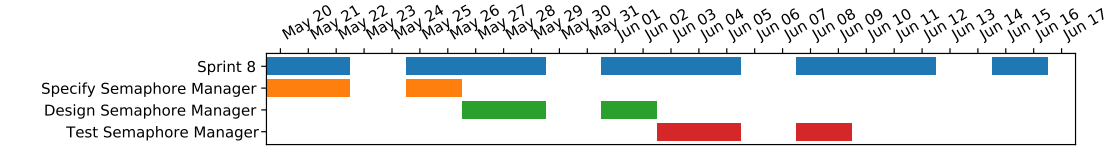


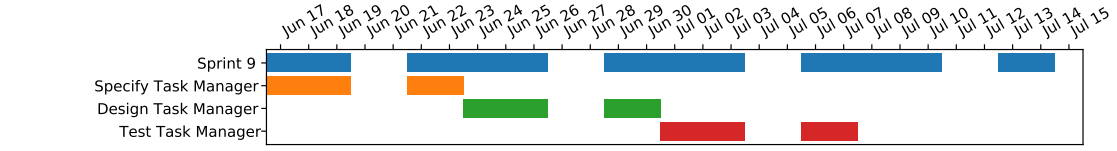

Jan 29 Jan 30 Jan 31 Feb 01 Feb 02 Feb 03 Feb 04 Feb 05 Feb 06 Feb 07 Feb 08 Feb 09 Feb 10 Feb 11 Feb 12 Feb 13 Feb 14 Feb 15 Feb 16 Feb 17 Feb 18 Feb 19 Feb 20 Feb 21 Feb 22 Feb 23 Feb 24 Feb 25 Feb 26
Sprint 4
Specify Event Manager
Design Event Manager
Test Event Manager

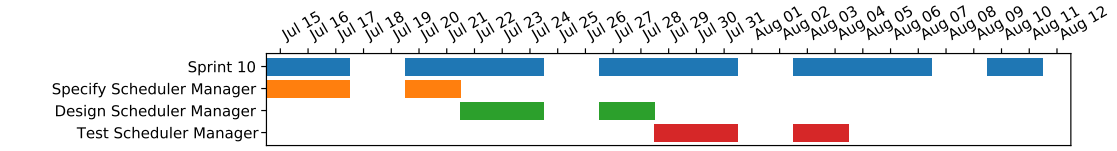

Feb 26 Feb 27 Feb 28 Feb 29 Mar 01 Mar 02 Mar 03 Mar 04 Mar 05 Mar 06 Mar 07 Mar 08 Mar 09 Mar 10 Mar 11 Mar 12 Mar 13 Mar 14 Mar 15 Mar 16 Mar 17 Mar 18 Mar 19 Mar 20 Mar 21 Mar 22 Mar 23 Mar 24 Mar 25
Sprint 5
Specify Timer Manager
Design Timer Manager
Test Timer Manager

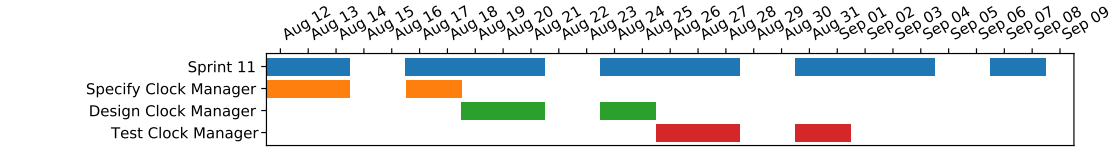

Mar 25 Mar 26 Mar 27 Mar 28 Mar 29 Mar 30 Mar 31 Apr 01 Apr 02 Apr 03 Apr 04 Apr 05 Apr 06 Apr 07 Apr 08 Apr 09 Apr 10 Apr 11 Apr 12 Apr 13 Apr 14 Apr 15 Apr 16 Apr 17 Apr 18 Apr 19 Apr 20 Apr 21 Apr 22
Sprint 6
Specify Message Queue Manager
Design Message Queue Manager
Test Message Queue Manager

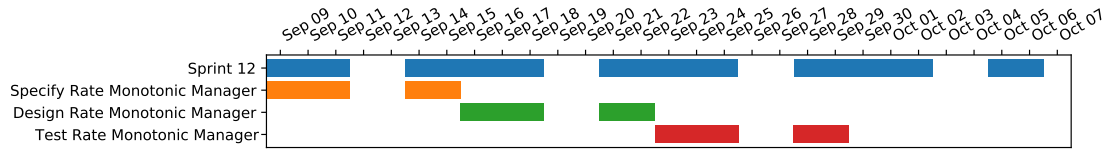

Apr 22 Apr 23 Apr 24 Apr 25 Apr 26 Apr 27 Apr 28 Apr 29 Apr 30 May 01 May 02 May 03 May 04 May 05 May 06 May 07 May 08 May 09 May 10 May 11 May 12 May 13 May 14 May 15 May 16 May 17 May 18 May 19 May 20
Sprint 7
Specify Extension Manager
Design Extension Manager
Test Extension Manager

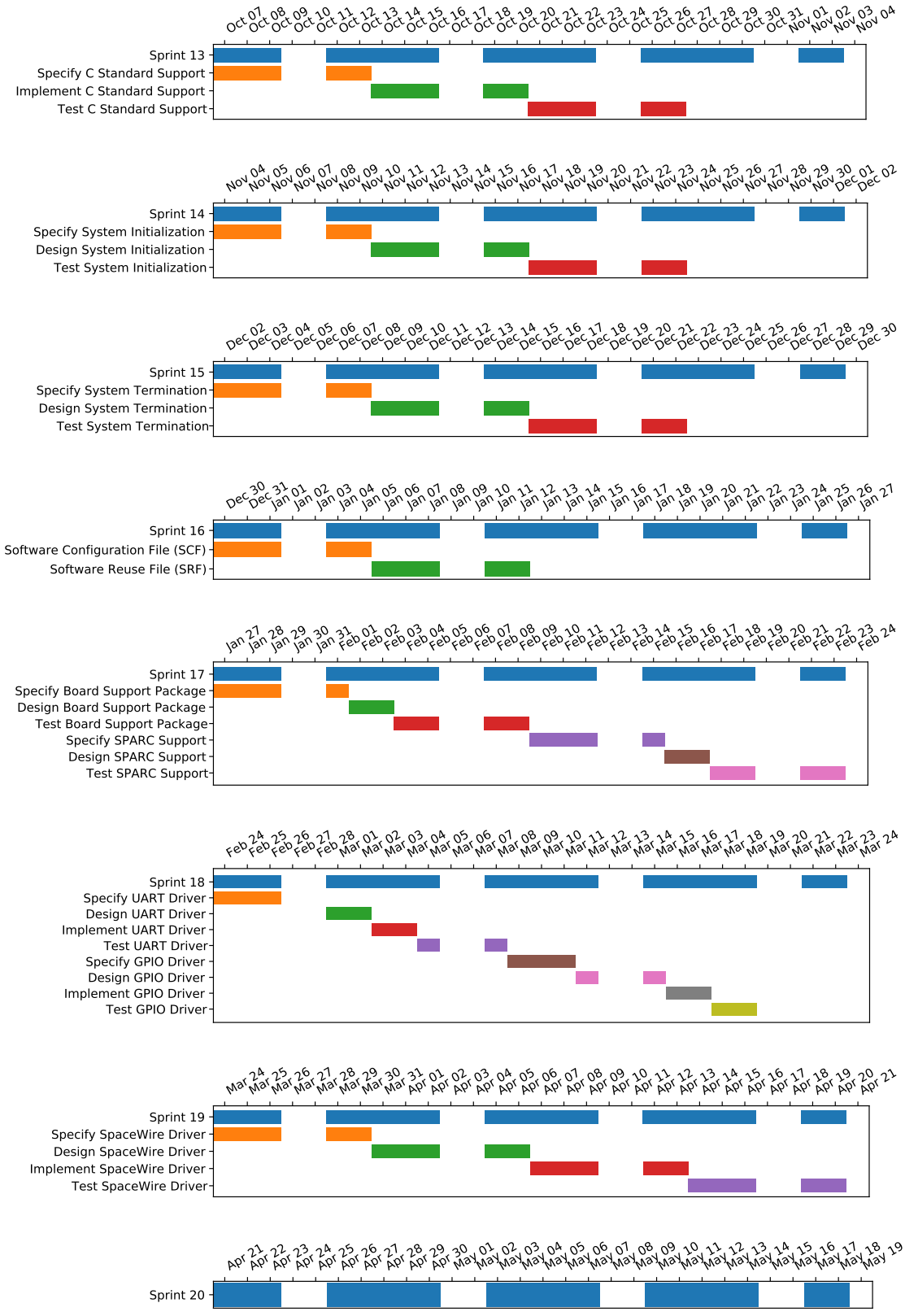

May 20 May 21 May 22 May 23 May 24 May 25 May 26 May 27 May 28 May 29 May 30 May 31 Jun 01 Jun 02 Jun 03 Jun 04 Jun 05 Jun 06 Jun 07 Jun 08 Jun 09 Jun 10 Jun 11 Jun 12 Jun 13 Jun 14 Jun 15 Jun 16 Jun 17
Sprint 8
Specify Semaphore Manager
Design Semaphore Manager
Test Semaphore Manager


Jun 17 Jun 18 Jun 19 Jun 20 Jun 21 Jun 22 Jun 23 Jun 24 Jun 25 Jun 26 Jun 27 Jun 28 Jun 29 Jun 30 Jul 01 Jul 02 Jul 03 Jul 04 Jul 05 Jul 06 Jul 07 Jul 08 Jul 09 Jul 10 Jul 11 Jul 12 Jul 13 Jul 14 Jul 15
Sprint 9
Specify Task Manager
Design Task Manager
Test Task Manager


Jul 15 Jul 16 Jul 17 Jul 18 Jul 19 Jul 20 Jul 21 Jul 22 Jul 23 Jul 24 Jul 25 Jul 26 Jul 27 Jul 28 Jul 29 Jul 30 Jul 31 Aug 01 Aug 02 Aug 03 Aug 04 Aug 05 Aug 06 Aug 07 Aug 08 Aug 09 Aug 10 Aug 11 Aug 12
Sprint 10
Specify Scheduler Manager
Design Scheduler Manager
Test Scheduler Manager


Aug 12 Aug 13 Aug 14 Aug 15 Aug 16 Aug 17 Aug 18 Aug 19 Aug 20 Aug 21 Aug 22 Aug 23 Aug 24 Aug 25 Aug 26 Aug 27 Aug 28 Aug 29 Aug 30 Aug 31 Sep 01 Sep 02 Sep 03 Sep 04 Sep 05 Sep 06 Sep 07 Sep 08 Sep 09
Sprint 11
Specify Clock Manager
Design Clock Manager
Test Clock Manager


Sep 09 Sep 10 Sep 11 Sep 12 Sep 13 Sep 14 Sep 15 Sep 16 Sep 17 Sep 18 Sep 19 Sep 20 Sep 21 Sep 22 Sep 23 Sep 24 Sep 25 Sep 26 Sep 27 Sep 28 Sep 29 Sep 30 Oct 01 Oct 02 Oct 03 Oct 04 Oct 05 Oct 06 Oct 07
Sprint 12
Specify Rate Monotonic Manager
Design Rate Monotonic Manager
Test Rate Monotonic Manager


Oct 07 Oct 08 Oct 09 Oct 10 Oct 11 Oct 12 Oct 13 Oct 14 Oct 15 Oct 16 Oct 17 Oct 18 Oct 19 Oct 20 Oct 21 Oct 22 Oct 23 Oct 24 Oct 25 Oct 26 Oct 27 Oct 28 Oct 29 Oct 30 Oct 31 Nov 01 Nov 02 Nov 03 Nov 04
Sprint 13
Specify C Standard Support
Implement C Standard Support
Test C Standard Support
Nov 04 Nov 05 Nov 06 Nov 07 Nov 08 Nov 09 Nov 10 Nov 11 Nov 12 Nov 13 Nov 14 Nov 15 Nov 16 Nov 17 Nov 18 Nov 19 Nov 20 Nov 21 Nov 22 Nov 23 Nov 24 Nov 25 Nov 26 Nov 27 Nov 28 Nov 29 Nov 30 Dec 01 Dec 02
Sprint 14
Specify System Initialization
Design System Initialization
Test System Initialization
Dec 02 Dec 03 Dec 04 Dec 05 Dec 06 Dec 07 Dec 08 Dec 09 Dec 10 Dec 11 Dec 12 Dec 13 Dec 14 Dec 15 Dec 16 Dec 17 Dec 18 Dec 19 Dec 20 Dec 21 Dec 22 Dec 23 Dec 24 Dec 25 Dec 26 Dec 27 Dec 28 Dec 29 Dec 30
Sprint 15
Specify System Termination
Design System Termination
Test System Termination
Dec 30 Dec 31 Jan 01 Jan 02 Jan 03 Jan 04 Jan 05 Jan 06 Jan 07 Jan 08 Jan 09 Jan 10 Jan 11 Jan 12 Jan 13 Jan 14 Jan 15 Jan 16 Jan 17 Jan 18 Jan 19 Jan 20 Jan 21 Jan 22 Jan 23 Jan 24 Jan 25 Jan 26 Jan 27
Sprint 16
Software Configuration File (SCF)
Software Reuse File (SRF)
Jan 27 Jan 28 Jan 29 Jan 30 Jan 31 Feb 01 Feb 02 Feb 03 Feb 04 Feb 05 Feb 06 Feb 07 Feb 08 Feb 09 Feb 10 Feb 11 Feb 12 Feb 13 Feb 14 Feb 15 Feb 16 Feb 17 Feb 18 Feb 19 Feb 20 Feb 21 Feb 22 Feb 23 Feb 24
Sprint 17
Specify Board Support Package
Design Board Support Package
Test Board Support Package
Specify SPARC Support
Design SPARC Support
Test SPARC Support
Feb 24 Feb 25 Feb 26 Feb 27 Feb 28 Mar 01 Mar 02 Mar 03 Mar 04 Mar 05 Mar 06 Mar 07 Mar 08 Mar 09 Mar 10 Mar 11 Mar 12 Mar 13 Mar 14 Mar 15 Mar 16 Mar 17 Mar 18 Mar 19 Mar 20 Mar 21 Mar 22 Mar 23 Mar 24
Sprint 18
Specify UART Driver
Design UART Driver
Implement UART Driver
Test UART Driver
Specify GPIO Driver
Design GPIO Driver
Implement GPIO Driver
Test GPIO Driver
Mar 24 Mar 25 Mar 26 Mar 27 Mar 28 Mar 29 Mar 30 Mar 31 Apr 01 Apr 02 Apr 03 Apr 04 Apr 05 Apr 06 Apr 07 Apr 08 Apr 09 Apr 10 Apr 11 Apr 12 Apr 13 Apr 14 Apr 15 Apr 16 Apr 17 Apr 18 Apr 19 Apr 20 Apr 21
Sprint 19
Specify SpaceWire Driver
Design SpaceWire Driver
Implement SpaceWire Driver
Test SpaceWire Driver
Apr 21 Apr 22 Apr 23 Apr 24 Apr 25 Apr 26 Apr 27 Apr 28 Apr 29 Apr 30 May 01 May 02 May 03 May 04 May 05 May 06 May 07 May 08 May 09 May 10 May 11 May 12 May 13 May 14 May 15 May 16 May 17 May 18 May 19
Sprint 20

### 6.5.1 Specify Build System

This work package has a planned duration of 5 days.

#### 6.5.1.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.1.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.
- Add ticket for the Build System to the RTEMS ticket system.
- Keep the ticket up to date.

#### 6.5.1.3 Outputs

- Build System Specification

### 6.5.2 Implement Build System

This work package has a planned duration of 10 days.

#### 6.5.2.1 Inputs

- *Build System Specification*

#### 6.5.2.2 Activities

- Implement the solution according to the specification.
- Maintain the ticket for the Build System in the RTEMS ticket system.

#### 6.5.2.3 Outputs

- Build System

### 6.5.3 Software User Manual (SUM)

This work package has a planned duration of 5 days.

#### 6.5.3.1 Inputs

- This technical note.

#### 6.5.3.2 Activities

- Set up a document structure and add it to the documentation build.
- Write the manual content.

#### 6.5.3.3 Outputs

- *Software User Manual (SUM)*

### 6.5.4 Specify Application Configuration

This work package has a planned duration of 5 days.

#### 6.5.4.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.4.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.
- Add ticket for the Application Configuration to the RTEMS ticket system.
- Keep the ticket up to date.

#### 6.5.4.3 Outputs

- Application Configuration Specification

### 6.5.5 Design Application Configuration

This work package has a planned duration of 3 days.

#### 6.5.5.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Application Configuration Specification*

#### 6.5.5.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.
- Maintain the ticket for the Application Configuration in the RTEMS ticket system.

#### 6.5.5.3 Outputs

- Application Configuration Doxygen Markup

### 6.5.6 Test Application Configuration

This work package has a planned duration of 2 days.

#### 6.5.6.1 Inputs

- *Application Configuration Specification*

#### 6.5.6.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.
- Maintain the ticket for the Application Configuration in the RTEMS ticket system.

#### 6.5.6.3 Outputs

- Application Configuration Test Specification
- Application Configuration Test Code

### 6.5.7 SUITP

This work package has a planned duration of 5 days.

#### 6.5.7.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.7.2 Activities

- Set up a document structure and add it to the documentation build.
- Write the manual content.

#### 6.5.7.3 Outputs

- *Software Unit and Integration Test Plan (SUITP)*

### 6.5.8 SVS for TS

This work package has a planned duration of 5 days.

#### 6.5.8.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.8.2 Activities

- Set up a document structure and add it to the documentation build.
- Write the manual content.

#### 6.5.8.3 Outputs

- *Software Validation Specification (SVS) with Respect to TS*

### 6.5.9 Specify Object Support

This work package has a planned duration of 5 days.

#### 6.5.9.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.9.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.9.3 Outputs

- Object Support Specification

### 6.5.10 Design Object Support

This work package has a planned duration of 2 days.

#### 6.5.10.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Object Support Specification*

#### 6.5.10.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.10.3 Outputs

- Object Support Doxygen Markup

### 6.5.11 Test Object Support

This work package has a planned duration of 3 days.

#### 6.5.11.1 Inputs

- *Object Support Specification*

#### 6.5.11.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.11.3 Outputs

- Object Support Test Specification
- Object Support Test Code

### 6.5.12 Specify Partition Manager

This work package has a planned duration of 5 days.

#### 6.5.12.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.12.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.12.3 Outputs

- Partition Manager Specification

### 6.5.13 Design Partition Manager

This work package has a planned duration of 2 days.

#### 6.5.13.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Partition Manager Specification*

#### 6.5.13.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.13.3 Outputs

- Partition Manager Doxygen Markup

### 6.5.14 Test Partition Manager

This work package has a planned duration of 3 days.

#### 6.5.14.1 Inputs

- *Partition Manager Specification*

#### 6.5.14.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.14.3 Outputs

- Partition Manager Test Specification
- Partition Manager Test Code

### 6.5.15 Specify Barrier Manager

This work package has a planned duration of 5 days.

#### 6.5.15.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.15.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.15.3 Outputs

- Barrier Manager Specification

### 6.5.16 Design Barrier Manager

This work package has a planned duration of 5 days.

#### 6.5.16.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Barrier Manager Specification*

#### 6.5.16.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.16.3 Outputs

- Barrier Manager Doxygen Markup

### 6.5.17 Test Barrier Manager

This work package has a planned duration of 5 days.

#### 6.5.17.1 Inputs

- *Barrier Manager Specification*

#### 6.5.17.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.17.3 Outputs

- Barrier Manager Test Specification
- Barrier Manager Test Code

### 6.5.18 Specify Event Manager

This work package has a planned duration of 5 days.

#### 6.5.18.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.18.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.18.3 Outputs

- Event Manager Specification

### 6.5.19 Design Event Manager

This work package has a planned duration of 5 days.

#### 6.5.19.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Event Manager Specification*

#### 6.5.19.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.19.3 Outputs

- Event Manager Doxygen Markup

### 6.5.20 Test Event Manager

This work package has a planned duration of 5 days.

#### 6.5.20.1 Inputs

- *Event Manager Specification*

#### 6.5.20.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.20.3 Outputs

- Event Manager Test Specification
- Event Manager Test Code

### 6.5.21 Specify Timer Manager

This work package has a planned duration of 5 days.

#### 6.5.21.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.21.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.21.3 Outputs

- Timer Manager Specification

### 6.5.22 Design Timer Manager

This work package has a planned duration of 5 days.

#### 6.5.22.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Timer Manager Specification*

#### 6.5.22.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.22.3 Outputs

- Timer Manager Doxygen Markup

### 6.5.23 Test Timer Manager

This work package has a planned duration of 5 days.

#### 6.5.23.1 Inputs

- *Timer Manager Specification*

#### 6.5.23.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.23.3 Outputs

- Timer Manager Test Specification
- Timer Manager Test Code

### 6.5.24 Specify Message Queue Manager

This work package has a planned duration of 5 days.

#### 6.5.24.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.24.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.24.3 Outputs

- Message Queue Manager Specification

### 6.5.25 Design Message Queue Manager

This work package has a planned duration of 5 days.

#### 6.5.25.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Message Queue Manager Specification*

#### 6.5.25.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.25.3 Outputs

- Message Queue Manager Doxygen Markup

### 6.5.26 Test Message Queue Manager

This work package has a planned duration of 5 days.

#### 6.5.26.1 Inputs

- *Message Queue Manager Specification*

#### 6.5.26.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.26.3 Outputs

- Message Queue Manager Test Specification
- Message Queue Manager Test Code

### 6.5.27 Specify Extension Manager

This work package has a planned duration of 5 days.

#### 6.5.27.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.27.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.27.3 Outputs

- Extension Manager Specification

### 6.5.28 Design Extension Manager

This work package has a planned duration of 5 days.

#### 6.5.28.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Extension Manager Specification*

#### 6.5.28.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.28.3 Outputs

- Extension Manager Doxygen Markup

### 6.5.29 Test Extension Manager

This work package has a planned duration of 5 days.

#### 6.5.29.1 Inputs

- *Extension Manager Specification*

#### 6.5.29.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.29.3 Outputs

- Extension Manager Test Specification
- Extension Manager Test Code

### 6.5.30 Specify Semaphore Manager

This work package has a planned duration of 5 days.

#### 6.5.30.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.30.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.30.3 Outputs

- Semaphore Manager Specification

### 6.5.31 Design Semaphore Manager

This work package has a planned duration of 5 days.

#### 6.5.31.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Semaphore Manager Specification*

#### 6.5.31.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.31.3 Outputs

- Semaphore Manager Doxygen Markup

### 6.5.32 Test Semaphore Manager

This work package has a planned duration of 5 days.

#### 6.5.32.1 Inputs

- *Semaphore Manager Specification*

#### 6.5.32.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.32.3 Outputs

- Semaphore Manager Test Specification
- Semaphore Manager Test Code

### 6.5.33 Specify Task Manager

This work package has a planned duration of 5 days.

#### 6.5.33.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.33.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.33.3 Outputs

- Task Manager Specification

### 6.5.34 Design Task Manager

This work package has a planned duration of 5 days.

#### 6.5.34.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Task Manager Specification*

#### 6.5.34.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.34.3 Outputs

- Task Manager Doxygen Markup

### 6.5.35 Test Task Manager

This work package has a planned duration of 5 days.

#### 6.5.35.1 Inputs

- *Task Manager Specification*

#### 6.5.35.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.35.3 Outputs

- Task Manager Test Specification
- Task Manager Test Code

### 6.5.36 Specify Scheduler Manager

This work package has a planned duration of 5 days.

#### 6.5.36.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.36.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.36.3 Outputs

- Scheduler Manager Specification

### 6.5.37 Design Scheduler Manager

This work package has a planned duration of 5 days.

#### 6.5.37.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Scheduler Manager Specification*

#### 6.5.37.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.37.3 Outputs

- Scheduler Manager Doxygen Markup

### 6.5.38 Test Scheduler Manager

This work package has a planned duration of 5 days.

#### 6.5.38.1 Inputs

- *Scheduler Manager Specification*

#### 6.5.38.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.38.3 Outputs

- Scheduler Manager Test Specification
- Scheduler Manager Test Code

### 6.5.39 Specify Clock Manager

This work package has a planned duration of 5 days.

#### 6.5.39.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.39.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.39.3 Outputs

- Clock Manager Specification

### 6.5.40 Design Clock Manager

This work package has a planned duration of 5 days.

#### 6.5.40.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Clock Manager Specification*

#### 6.5.40.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.40.3 Outputs

- Clock Manager Doxygen Markup

### 6.5.41 Test Clock Manager

This work package has a planned duration of 5 days.

#### 6.5.41.1 Inputs

- *Clock Manager Specification*

#### 6.5.41.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.41.3 Outputs

- Clock Manager Test Specification
- Clock Manager Test Code

### 6.5.42 Specify Rate Monotonic Manager

This work package has a planned duration of 5 days.

#### 6.5.42.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.42.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.42.3 Outputs

- Rate Monotonic Manager Specification

### 6.5.43 Design Rate Monotonic Manager

This work package has a planned duration of 5 days.

#### 6.5.43.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Rate Monotonic Manager Specification*

#### 6.5.43.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.43.3 Outputs

- Rate Monotonic Manager Doxygen Markup

### 6.5.44 Test Rate Monotonic Manager

This work package has a planned duration of 5 days.

#### 6.5.44.1 Inputs

- *Rate Monotonic Manager Specification*

#### 6.5.44.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.44.3 Outputs

- Rate Monotonic Manager Test Specification
- Rate Monotonic Manager Test Code

### 6.5.45 Specify C Standard Support

This work package has a planned duration of 5 days.

#### 6.5.45.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- [ISO10]

#### 6.5.45.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.45.3 Outputs

- C Standard Support Specification

### 6.5.46 Implement C Standard Support

This work package has a planned duration of 5 days.

#### 6.5.46.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *C Standard Support Specification*

#### 6.5.46.2 Activities

- Implement the solution according to the specification.

#### 6.5.46.3 Outputs

- C Standard Support Source Code

### 6.5.47 Test C Standard Support

This work package has a planned duration of 5 days.

#### 6.5.47.1 Inputs

- *C Standard Support Specification*

#### 6.5.47.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.47.3 Outputs

- C Standard Support Test Specification
- C Standard Support Test Code

### 6.5.48 Specify System Initialization

This work package has a planned duration of 5 days.

#### 6.5.48.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.48.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.48.3 Outputs

- System Initialization Specification

### 6.5.49 Design System Initialization

This work package has a planned duration of 5 days.

#### 6.5.49.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *System Initialization Specification*

#### 6.5.49.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.49.3 Outputs

- System Initialization Doxygen Markup

### 6.5.50 Test System Initialization

This work package has a planned duration of 5 days.

#### 6.5.50.1 Inputs

- *System Initialization Specification*

#### 6.5.50.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.50.3 Outputs

- System Initialization Test Specification
- System Initialization Test Code

### 6.5.51 Specify System Termination

This work package has a planned duration of 5 days.

#### 6.5.51.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.51.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.51.3 Outputs

- System Termination Specification

### 6.5.52 Design System Termination

This work package has a planned duration of 5 days.

#### 6.5.52.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *System Termination Specification*

#### 6.5.52.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.52.3 Outputs

- System Termination Doxygen Markup

### 6.5.53 Test System Termination

This work package has a planned duration of 5 days.

#### 6.5.53.1 Inputs

- *System Termination Specification*

#### 6.5.53.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.53.3 Outputs

- System Termination Test Specification
- System Termination Test Code

### 6.5.54 Software Configuration File (SCF)

This work package has a planned duration of 5 days.

#### 6.5.54.1 Inputs

- *Specify Build System*

#### 6.5.54.2 Activities

- Set up a document structure and add it to the documentation build.
- Write the manual content.

#### 6.5.54.3 Outputs

- *Software Configuration File (SCF)*

### 6.5.55 Software Reuse File (SRF)

This work package has a planned duration of 5 days.

#### 6.5.55.1 Inputs

- RTEMS Documentation
- RTEMS Sources

#### 6.5.55.2 Activities

- Set up a document structure and add it to the documentation build.
- Write the manual content.

#### 6.5.55.3 Outputs

- *Software Reuse File (SRF)*

### 6.5.56 Specify Board Support Package

This work package has a planned duration of 4 days.

#### 6.5.56.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- [Gai18a]
- [Gai18b]
- [Gai18c]
- [Gai19]

#### 6.5.56.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Discuss the specification with Gaisler.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.56.3 Outputs

- Board Support Package Specification

### 6.5.57 Design Board Support Package

This work package has a planned duration of 2 days.

#### 6.5.57.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *Board Support Package Specification*

#### 6.5.57.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.57.3 Outputs

- Board Support Package Doxygen Markup

### 6.5.58 Test Board Support Package

This work package has a planned duration of 4 days.

#### 6.5.58.1 Inputs

- *Board Support Package Specification*

#### 6.5.58.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.58.3 Outputs

- Board Support Package Test Specification
- Board Support Package Test Code

### 6.5.59 Specify SPARC Support

This work package has a planned duration of 4 days.

#### 6.5.59.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- [SPA91]
- [SPA96]
- [SPA02]

#### 6.5.59.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Discuss the specification with Gaisler.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.59.3 Outputs

- SPARC Support Specification

### 6.5.60 Design SPARC Support

This work package has a planned duration of 2 days.

#### 6.5.60.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *SPARC Support Specification*

#### 6.5.60.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.60.3 Outputs

- SPARC Support Doxygen Markup

### 6.5.61 Test SPARC Support

This work package has a planned duration of 4 days.

#### 6.5.61.1 Inputs

- *SPARC Support Specification*

#### 6.5.61.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.61.3 Outputs

- SPARC Support Test Specification
- SPARC Support Test Code

### 6.5.62 Specify UART Driver

This work package has a planned duration of 3 days.

#### 6.5.62.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- [Gai18a]
- [Gai18b]
- [Gai18c]
- [Gai19]

#### 6.5.62.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Discuss the specification with Gaisler.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.62.3 Outputs

- UART Driver Specification

### 6.5.63 Design UART Driver

This work package has a planned duration of 2 days.

#### 6.5.63.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *UART Driver Specification*

#### 6.5.63.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.63.3 Outputs

- UART Driver Doxygen Markup

### 6.5.64 Implement UART Driver

This work package has a planned duration of 2 days.

#### 6.5.64.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *UART Driver Specification*

#### 6.5.64.2 Activities

- Implement the solution according to the specification.

#### 6.5.64.3 Outputs

- UART Driver Source Code

### 6.5.65 Test UART Driver

This work package has a planned duration of 2 days.

#### 6.5.65.1 Inputs

- *UART Driver Specification*

#### 6.5.65.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.65.3 Outputs

- UART Driver Test Specification
- UART Driver Test Code

### 6.5.66 Specify GPIO Driver

This work package has a planned duration of 3 days.

#### 6.5.66.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- [Gai18a]
- [Gai18b]
- [Gai18c]
- [Gai19]

#### 6.5.66.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Discuss the specification with Gaisler.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.66.3 Outputs

- GPIO Driver Specification

### 6.5.67 Design GPIO Driver

This work package has a planned duration of 2 days.

#### 6.5.67.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *GPIO Driver Specification*

#### 6.5.67.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.67.3 Outputs

- GPIO Driver Doxygen Markup

### 6.5.68 Implement GPIO Driver

This work package has a planned duration of 2 days.

#### 6.5.68.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *GPIO Driver Specification*

#### 6.5.68.2 Activities

- Implement the solution according to the specification.

#### 6.5.68.3 Outputs

- GPIO Driver Source Code

### 6.5.69 Test GPIO Driver

This work package has a planned duration of 2 days.

#### 6.5.69.1 Inputs

- *GPIO Driver Specification*

#### 6.5.69.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.69.3 Outputs

- GPIO Driver Test Specification
- GPIO Driver Test Code

### 6.5.70 Specify SpaceWire Driver

This work package has a planned duration of 5 days.

#### 6.5.70.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- [Gai18a]
- [Gai18b]
- [Gai18c]
- [Gai19]

#### 6.5.70.2 Activities

- Do a reverse engineering of the RTEMS documentation and source code.
- Discuss the specification on the RTEMS developer mailing list.
- Discuss the specification with Gaisler.
- Write *Specification Items*.
- Carry out the patch review process for specification items on the RTEMS development mailing list.

#### 6.5.70.3 Outputs

- SpaceWire Driver Specification

### 6.5.71 Design SpaceWire Driver

This work package has a planned duration of 5 days.

#### 6.5.71.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *SpaceWire Driver Specification*

#### 6.5.71.2 Activities

- Update Doxygen markup for the *Software Design Document (SDD)*.
- Carry out patch review process for documentation changes on the RTEMS development mailing list.

#### 6.5.71.3 Outputs

- SpaceWire Driver Doxygen Markup

### 6.5.72 Implement SpaceWire Driver

This work package has a planned duration of 5 days.

#### 6.5.72.1 Inputs

- RTEMS Documentation
- RTEMS Sources
- *SpaceWire Driver Specification*

#### 6.5.72.2 Activities

- Implement the solution according to the specification.

#### 6.5.72.3 Outputs

- SpaceWire Driver Source Code

### 6.5.73 Test SpaceWire Driver

This work package has a planned duration of 5 days.

#### 6.5.73.1 Inputs

- *SpaceWire Driver Specification*

#### 6.5.73.2 Activities

- Write test designs, see *Specification Items*.
- Write test specifications (includes test plans), see *Specification Items*.
- Write test cases as source code.
- Carry out patch review process for test changes on the RTEMS development mailing list.

#### 6.5.73.3 Outputs

- SpaceWire Driver Test Specification
- SpaceWire Driver Test Code

CHAPTER

SEVEN

# RTEMS IMPROVEMENT QUALIFICATION DATA PACKAGE

This section will perform the analysis of the RTEMS Improvement Data Package content. The following subsections contain a description of each deliverable present in RTEMS Improvement product.

## 7.1 RTEMS Managers Candidate Evaluation Report

This document presents the RTEMS Managers Candidate Evaluation Report. It describes the RTEMS Managers present in the RTEMS Improvement project Statement of Work and proposed a subset of RTEMS Managers that were covered in this project. In addition, inside a proposed RTEMS Manager, there were some functionalities that were not required for space applications and hence they were not covered. In summary, this document present the tailored performed from the original OAR RTEMS 4.8.0 version to the RTEMS Improvement version.

The document is laid out in the following sections:

- Section 1: Introduction, presents the document purpose, scope and overview.
- Section 2: General Description, presents the general description of the RTEMS Managers.
- Section 3: RTEMS Managers Evaluation, presents the evaluation of the RTEMS Managers.

## 7.2 RTEMS Improvement Requirement Document

This document presents the RTEMS Software Requirements. It describes the software requirements of the selected managers of the RTEMS Operating System, according with RTEMS Managers Candidate Evaluation Report document. This document contains the complete list of software requirements for the RTEMS Improvement and its subsequent projects: RTEMS LEON Upgrade project and RTEMS Qualification Extensions.

The requirements placed in this document were gathered from the projects referred above Statements of Work, from the RTEMS 4.8.0 source code, RTEMS 4.8.0 C User's Guide (with the restrictions made in the RTEMS Managers Candidate Evaluation Report) and RTEMS 4.11 source code for the AMBA, SpaceWire and MIL-STD-1553 drivers.

This document is used to assess if the RTEMS Operating System is adequate in terms of functionality, performance, quality, testability, design, implementation, test constraints, installation and dependability to the production of application.

The document is laid out in the following sections:

- Section 1: Introduction, Presents the document purpose, scope and overview.
- Section 2: General Description, presents the general software context and results of the requirement analysis.
- Section 3: Specific Requirements, presents the requirements for the RTEMS product;
- Section 4: Verification and Validation, presents the verification, validation and acceptance requirements of delivered software product. It also presents the verification, validation and acceptance methods for each requirement;
- Section 5: Interface Definition, presents the definition of the interfaces;
- Section 6: Requirements Removed presents the requirements removed from the project;
- Section 7: Traceability Model, makes the traceability between the document requirements with the model of RTEMS;
- Section 8: Traceability Matrix between Software Requirements and RTEMS Improvement Statement of Work Requirements, presents the traceability between SoW and this document requirements;
- Section 9: Traceability Matrix between RLU Proposal and Software Requirements, presents the traceability between the software requirements document and the proposal of RTEMS LEON Upgrade project;
- Section 10: Traceability Matrix between SCAR Requirements and Software Requirements, presents the traceability between the software requirements document and the SCAR requirements.
- Section 11: Traceability Matrix between Software Requirements and RTEMS Qualification Extensions Statement of Work Requirements, presents the traceability between SoW and this document requirements;

## 7.3 RTEMS Improvement User Manual and Design Notes

This document presents the RTEMS Improvement User Manual and Design Notes. It describes how to install, configure and use the RTEMS Operating System, including the device drivers for the 1553 and SpaceWire communication interfaces. This document is limited to the API defined in RTEMS Managers Candidate Evaluation Report, where it is defined the RTEMS Managers Candidates evaluated in the RTEMS Improvement project. This document is a subset of RTEMS C User Guide, truncated by the conclusions taken in RTEMS Managers Candidate Report, reviewed and cross-checked with the RTEMS source code implementation. Some of the information presented in this document has been selected and corrected from RTEMS C User Guide and Getting Started with RTEMS (version 4.8.0).

The document is laid out in the following sections:

- Section 1: Introduction, presents the document purpose, scope and overview.
- Section 2: RTEMS General Description, presents a general overview of the RTEMS Operating System.

- Section 3: Host Requirements, presents the hardware and software requirements of the Host computer
- Section 4: RTEMS Installation and Configuration, presents a guide on how to install and configure RTEMS;
- Section 5: RTEMS API User's Guide, presents an overall look at RTEMS API and how to use it from the user's point of view;
- Section 6: RTEMS Communication Interface drivers, presents an overall look at RTEMS 1553 and SpW drivers and how to use them from the user's point of view;
- Section 7: Interrupts Handlers, Presents the RTEMS default interrupt handlers and SpaceWire and 1553 interrupt handlers;
- Section 8: Notes, presents the API directives that blocks tasks and change the ready task;
- Section 9: Warnings, presents the warnings to be considered by the user;
- Section 10: Configuration of the Application, explains the configuration main topics of an application.
- Section 11: RTEMS Tailoring, presents a short description of the RTEMS Tailoring results;
- Section 12: Compiling the Application, presents an overview of the configuration and compilation of a RTEMS Tailored application
- Section 13: RTEMS Test Suite, describes the organization, compilation and running of the testsuites;

## 7.4 RTEMS Improvement Verification Report

This document presents RTEMS Improvement Verification Report, one of the outputs of Management & Configuration Task. This document follows closely the structure suggested in the Galileo Software Standards, Appendix 18. The report presented in this document is based in the directions provided in the Software Development Plan, section 10 (Software Verification). The current version of this document considers the RTEMS Tailored version that resulted from the code merged of the implemented SW-FMECAs in the scope of the RTEMS LEON Upgrade Project and the code merged of the implemented MIL-STD-1553 and SpaceWire communication interfaces in the scope of the RTEMS Qualification Extensions project.

The document is laid out in the following sections:

- Section 1: Introduction, including this subsection and the acronyms
- Section 2: Documents, including applicable and reference documents;
- Section 3: Software Overview, providing an overview of the software, the architecture of RTEMS, the managers to be analysed and all the tasks defined to develop the work;
- Section 4: Verification Report presenting the results of the verification actions taken in every task of the project;
- Appendix A: Phase Documentation, presenting the phase documentation verification;

- Appendix B: SRD Requirements Verification, presenting the verification of the requirements;
- Appendix C: SRS Verification, presenting the requirement standard verification;
- Appendix D: SDS Verification, presenting the design standard verification;
- Appendix E: SCS Verification, presenting the coding standard verification;
- Appendix F: Source Code modifications Verification, presenting the verification of the source code modifications;
- Appendix G: Code Coverage Verification, presenting the coverage verification.

## 7.5 Software Budget Report

The purpose of the software budget report is to capture all information concerning margins and technical budgets of the software items related with the RTEMS Improvement Project. The objective of this document is to register the Software Budget Report of RTEMS Tailored in the conclusion of every Task related with the first phase of the project. This document is one of the outputs of Validation Phase and Acceptance Phase Tasks. This document follows closely the structure suggested in the Galileo Software Standards, Appendix 10.

The current version of this document considers the RTEMS Tailored version that resulted from the code merged of the implemented SW-FMECAs in the scope of the RTEMS LEON Upgrade Project and of the implemented MIL-STD-1553 and SpaceWire communication interfaces in the scope of the RTEMS Qualification Extensions project.

The document is laid out in the following sections:

- Section 1: Introduction, this section contains a description of the purpose, objective, content and the reason prompting its preparation. It also includes a brief description of the sections of the Document and the additional terms, definition or abbreviated terms that are used in this document;
- Section 2: Documents, this section shall list the applicable and reference documents to support the generation of the document;
- Section 3: Software Overview provides a brief description of the software system and its context. This includes a summary of the software functionality, software configuration, operational environment and external interfaces that describe the context of the software items;
- Section 4: Technical Budget and Margin, describes the analysis done with respect to schedulability. This section is continuously updated with more detailed information as it becomes available and provides budgeting estimation (e.g., memory or processing time allocation), in accordance with the applicable software development life cycle;

## 7.6 Product Software Justification File

The purpose of this document is to describe the Selected Software Products in the RTEMS Improvement project and also give information about the reused software items. This description includes information about the Software Product main characteristics and its purpose in the project. One of the outputs of the RTEMS Improvement project is RTEMS Improvement which is considered to be procured operational software since it preserves its main characteristics (functionality and performance). Each Software Item developed by Edisoft is listed in this document and information about each one is described in terms of availability (this Software Items takes in account the Software Items version and the shortcomings of the Produced Software available). This document is one of the outputs of Planning Phase, Specification Phase, Design Phase, Implementation Phase, Integration Phase, Validation Phase and Acceptance Phase Tasks. The current document considers the RTEMS Tailored version improved with SpaceWire and MIL-STD-1553 communication interfaces and also a RTEMS kernel monitoring tool (not part of RTEMS itself).

The document is laid out in the following sections:

- Section 1: Introduction, describes the document purpose and gives a overview of the context where is produced;
- Section 2: Applicable and reference documents, states the applicable and reference documents used;
- Section 3: Terms, definition and abbreviated terms, describes the terms, acronyms and abbreviation used in the documents;
- Section 4: Presentation of the software, describes the technical and management information of the Software in the RTEMS Improvement Project. This section includes information about the software items of the RTEMS Improvement Project;
- Section 5: Compatibility of existing software with project requirements, describes the reuse software items in terms of availability and quality;
- Section 6: Software reuse analysis conclusion, describes the results of the software items reuse decisions, with a description of the level of reuse and the methods applied when estimating its reuse level;
- Section 7: Detailed results of evaluation, describes the result of the evaluation performed with the information collected in the previous chapters, and presents the developed software by Edisoft to overcome the Procured Software evaluated shortcomings;
- Section 8: Corrective actions, lists the problems reports related to the Procured Software and the decisions taken by Edisoft in relation to each one;
- Section 9: Configuration status, describes the configurations undertaken to ensure the software items are under Software Versioning Configuration;
- Section 10: Annexes, provide a list of the level of reuse of RTEMS documentation and source code and problem reports.

## 7.7 RTEMS Improvement Design Document

This document presents the Software Design Document, one of the outputs of Architectural Phase, Specification Phase, Design Phase, Implementation Phase and Integration Phase Tasks. This document follows closely the structure suggested in the Galileo Software Standards, Appendix A.6. The current version of this document considers the RTEMS Tailored version that resulted from the code merged of the implemented SW-FMECAs in the scope of the RTEMS LEON Upgrade Project and the code merged of the implemented MIL-STD-1553 and SpaceWire communication interfaces in the scope of the RTEMS Qualification Extensions project.

The document is laid out in the following sections:

- Section 1: Introduction, including the acronyms and the applicable and reference documents;
- Section 2: Overview, providing an overview of the system context, the architecture, the processors used, Hardware/Software interfaces and safety features;
- Section 3: System Environment, describing the hardware where the system will operate, the design method followed and the error processing techniques used.
- Section 4: Architectural Design, presenting the RTEMS architectural design, including the Hardware Drivers, BSP, Super Core, API Layer and Support Libraries;
- Section 5: Detailed Design, presenting the RTEMS detailed design, organized by components (modules) or by source files, data structures and functions descriptions for each component identified in the architecture;
- Section 6: Source Code, presenting where to find the source code, including the base releases from OAR and the patches released from this project, as well as a description of the procedures for applying the patches and configuring a development installation;
- Section 7: Traceability, providing the traceability matrices from requirements to architecture, from architecture to detailed design, from detailed design to source code and from requirements to detailed design.

## 7.8 RTEMS Improvement Configuration File

This document contains the RTEMS Improvement Configuration File for the RTEMS Improvement project. It is responsible for defining the items that are used in the development of the RTEMS Improvement project. The purpose of this document is to ensure that all of the crucial software and items used in the RTEMS Improvement project and the respective versions are recorded. This ensures that the working conditions that are used can be reproduced in an easy and clear way. This document is one of the outputs of Management & Configuration Task, it includes the implementations performed in the RQE project, the RTEMS LEON Upgrade project and RTEMS Improvement project. This document follows closely the structure suggested in the Galileo Software Standards, Appendix 8.

The document is laid out in the following sections:

- Section 1: Introduction, this includes the list of the acronyms used;
- Section 2: Documents, this refers to applicable and reference documents used;

- Section 3: Software Release Description, presents the description of how the software shall be released;
- Section 4: Released Documentation, describes the functional, development, product baselines and releases for the documents developed in the RTEMS Improvement project;
- Section 5: Software Facilities Description, describes the development and target environment for the RTEMS Improvement project;
- Section 6: Change Control Status, has a list for the implemented changes and its status in the RTEMS Improvement project, it also has a list of pending or known problems verified that may constrain the environment or operations of the project;
- Section 7: Adaptation Data, this section contains the identification of any data that may vary from one installation to another;
- Section 8: Building, Loading and Installation Instructions, in this section it is described the building, loading and installation instructions for the RTEMS Improvement project;
- Section 9: Software Characteristics, this section describes the software characteristics for RTEMS Improvement project.

## 7.9 RTEMS Improvement Integration Test Plan

This document specifies the Integration Test Plan. It is one of the deliverables of Design Phase, Implementation Phase and Integration Phase Tasks of RTEMS Improvement Project. All this phases are described in Software Development Plan document (referred in section ). The current version of this document considers the RTEMS Tailored version that resulted from the code merged of the implemented SW-FMECAs in the scope of the RTEMS LEON Upgrade Project and the code merged of the implemented MIL-STD-1553 and SpaceWire communication interfaces in the scope of the RTEMS Qualification Extensions project. The Integration test Plan aims to demonstrate that a software component works correctly as a result of the integration of its software elements.

The document is laid out in the following sections:

- Section 1: Introduction, this section presents the document purpose, scope and overview.
- Section 2: Applicable and Reference Documents, this section shows the applicable and reference documents.
- Section 3: Integration Test Plan, description of the Test Designs, Test Case Specifications and Test Procedures. Integration tests verify that a software component works correctly as a result of the integration of its SW elements. This section is divided in Organization and resource, Schedule and Integration Approach.
- Section 4: Integration Test Definition, describes all tests to testing unit against the design. Integration tests are performed with the objective to validate all internal interfaces as defined in the SDD and SWICD as applicable depending on the level of integration testing. This section is composed by Test designs, Test case specifications and Test procedures regarding each test.
- Section 5: Test Coverage Matrix, From/to SDD(DDD) to/from Integration Test Cases.

- Annex A, presenting the test templates used in the document.
- Annex B, presenting the test configuration parameters.

## 7.10 RTEMS Improvement Unit Test Plan

This document specifies the Unit Test Plan. It is one of the deliverables of Design Phase, Implementation Phase and Integration Phase Tasks of RTEMS Improvement Project. All this phases are described in Software Development Plan. The Unit Test Plan aims to identify and specify the Unit Tests to demonstrate that each element of the software component works correctly when isolated from the complementary software elements of the software architecture. The current version of this document considers the RTEMS Tailored version that resulted from the code merged of the implemented SW-FMECAs in the scope of the RTEMS LEON Upgrade Project and the code merged of the implemented MIL-STD-1553 and SpaceWire communication interfaces in the scope of the RTEMS Qualification Extensions project.

The document is laid out in the following sections:

- Section 1: Introduction, this section presents the document purpose, scope and overview.
- Section 2: Applicable and Reference Documents, this section shows the applicable and reference documents.
- Section 3: Unit Test Plan, describes how to verify that each component of the software works as expected when isolated from the remaining software components. This section is subdivided in “Organization and resources” and “Software verification approach” subsections.
- Section 4: Unit Test Definition, Notification for individual software elements or group of related elements of the Test Design, Test Case Specification and Test Procedure. This section is subdivided in Test Designs, Test Cases and Test Procedures.
- Section 5: Test Coverage Matrix, this section defines the traceability between unit test cases and Software code.
- Section 6: Unitary tests configuration parameters, presenting the configuration of the unit tests.
- Annex A, presents the unitary test case and procedure templates.

## 7.11 RTEMS Improvement Validation Testing Specification

This document specifies the Validation Testing Specification. It is one of the deliverables of Design Phase, Implementation Phase and Integration Phase Tasks of RTEMS Improvement Project. All this phases are described in Software Development Plan. The Validation Testing Specification aims to describe all tests to validate that the software is compliant with the requirements defined in RTEMS Improvement Software Requirements Specification. The current version of this document considers the RTEMS Tailored version that resulted from the code merged of the implemented SW-FMECAs in the scope of the RTEMS LEON Upgrade Project and the code merged of the implemented MIL-STD-1553 and SpaceWire communication interfaces in the scope of the RTEMS Qualification Extensions project.

The document is laid out in the following sections:

- Section 1: Introduction, this section presents the document purpose, scope and overview.
- Section 2: Applicable and Reference Documents, this section shows the applicable and reference documents.
- Section 3: Validation Test Definition, describes all tests to validate that the software is compliant with the requirements defined by Software Requirements Document.
- Section 4: Test Coverage Matrix, displays two tables that allow the trace between validation test cases and software requirements, with a case for every requirement.
- Appendix A: Templates, tests and test procedures templates used in the document.
- Appendix B: Additional Sequence Diagrams, supportive sequence diagram to the tests cases.
- Appendix C: Configuration Parameters, tests configuration parameters.

## 7.12 RTEMS Tailoring Plan

This document presents the RTEMS Tailoring Plan and Report. It describes the plan to tailor the RTEMS source code to the specific characteristics of the application. This document also contains the RTEMS Tailoring Report, specifying the actions/SPR/NCR/etc that created the need for the change, the rationale and the change description. The files changed are also presented.

The document is laid out in the following sections:

- Section 1: Introduction, presents the document purpose, scope and overview.
- Section 2: General Description, presents the general software context and results of the requirement analysis.
- Section 3: RTEMS Tailoring, presents the plan to tailor the RTEMS source code to the specific characteristics of the application; presents the tailoring report of the changes made

## 7.13 RTEMS Improvement Validation, Integration and Unit Test Report

This deliverable presents the Validation, Unit and Integration Test Reports, one of the outputs of Implementation Phase, Integration Phase of RTEMS, Validation Phase and Acceptance Phase Tasks of RTEMS Improvement Project. This deliverable is subdivided into three documents presented in the following subsections.

### 7.13.1 RTEMS Improvement Validation Test Report

This document presents the Validation Test Report, containing the results of the validation tests.

The document is laid out in the following sections:

- Section 1: Introduction, presents the document purpose, scope, overview, applicable and reference documents.
- Section 2: Non Regression Approach, describes the approach of non-regression tests for validation and acceptance;
- Section 3: Test Platform Configuration, describes the RTEMS improvement test platform;
- Section 4: Test Scripts used, describes the test scripts used to gather the test reports
- Section 5: Validation Test reports, describes the report of validation test reports;
- Section 6: TS/RB Requirements Coverage Matrix, provides the requirements traceability with the validation tests performed.
- ANNEX: provides the test reports of all platforms.

### 7.13.2 RTEMS Improvement Integration Test Report

This document presents the Integration Test Report, containing the results of the Integration tests.

The document is laid out in the following sections:

- Section 1: Introduction, presents the document purpose, scope, overview, applicable and reference documents.
- Section 2: Non Regression Approach, describes the approach of non-regression tests for validation and acceptance;
- Section 3: Test Platform Configuration, describes the RTEMS improvement test platform;
- Section 4: Test Scripts used, describes the test scripts used to gather the test reports;
- Section 5: Integration Test reports, describes the report of integration test reports;
- Section 6: TS/RB Requirements Coverage Matrix, provides the requirements traceability with the integration tests performed;
- ANNEX: provides the test reports in all platforms.

### 7.13.3 RTEMS Improvement Unit Test Report

This document presents the Unit Test Report, containing the results of the unit tests.

The document is laid out in the following sections:

- Section 1: Introduction, presents the document purpose, scope, overview, applicable and reference documents.
- Section 2: Non Regression Approach, describes the approach of non-regression tests for validation and acceptance;
- Section 3: Test Platform Configuration, describes the RTEMS improvement test platform;
- Section 4: Test Scripts used, describes the test scripts used to gather the test reports;
- Section 5: Unit Test reports, describes the report of unit test reports;
- Section 6: TS/RB Requirements Coverage Matrix, provides the requirements traceability with the unit tests performed;
- ANNEX: attaches the unit test reports for all the platforms.

## 7.14 RTEMS Test Suite

The RTEMS Test Suite contains all the tests and auxiliary files to run the testsuite. It is formed by the following main folders (the folders not described contains makefiles and other files used to compile the testsuite):

- 1553Tests: Auxiliary procedures that run in SCOC3 EGSE which allow to validate the gr712rc MIL-STD-1553 interface
- testsuites/validation: Contains the validation tests and respective makefiles
- testsuites/integration: Contains the integration tests and respective makefiles
- testsuites/unit: Contains the unit tests and respective makefiles
- testsuites/gcov: Contains the files necessary to run the coverage tests
- testsuites/performance: Contains the files necessary to run the performance tests
- testsuites/support: Contains the support files to run the testsuite: test output printing functions, timer for performance measurements, scripts to execute the testsuite and procedures lists.

## 7.15 RTEMS Improvement Acceptance Test Plan

This document specifies the Acceptance Test Plan of the RTEMS Improvement, RTEMS LEON Upgrade and RTEMS Qualification Extensions projects. It is one of the deliverables of the Validation Phase Task, described at Software Development Plan document.

The document is laid out in the following sections:

- Section 1: Introduction, including the acronyms and the applicable and reference documents;
- Section 2: Acceptance Test Plan, providing the organisation and resources to be used in the acceptance test campaign and the software verification approach;
- Section 3: Acceptance Test Definition, describing the test campaign description, design, test cases specification and test procedures;
- Section 4: Test Coverage Matrix, presenting the traceability from/to requirements baseline (defined in the RTEMS Improvement Statement Of Work) to the acceptance test cases;

## 7.16 RTEMS Improvement Maintenance Plan

This document contains the Maintenance Plan for the RTEMS Improvement project. It defines the scope and responsibilities of the maintenance phase 2 of the RTEMS Improvement project. This plan also describes the maintenance procedures to be used. This document is one of the outputs of RTEMS Implementation and Execution and RTEMS Tailoring Test Suite Re-Execution Tasks of RTEMS Improvement Implementation Phase. This document follows closely the structure suggested in the Galileo Software Standards, Appendix A.33. This plan and the referred procedures for its implementation, is applicable to the Maintenance Phase of the RTEMS Improvement Project.

The document is laid out in the following sections:

- Section 1: Introduction, includes the list of the acronyms used;
- Section 2: Documents, refers to applicable and reference documents used;
- Section 3: Scope and Purpose, includes the description of the processes and procedures necessary to provide the corrective and preventive maintenance to the RTEMS Improvement deliverables. The boundaries of the maintenance process are also defined here. In addition, the difference between maintenance and development are addressed at this point;
- Section 4: Application of the Plan, provides the description of the overall flow of work. It includes the description of each process step and their interfaces, the data flow between processes. Definition how each step in the process is controlled and measured, the expected levels of performance are also here addressed;
- Section 5: General Requirements, provide the description of the policies and responsibilities of the project team as its plans for software maintenance are here defined;

- Section 6: Maintenance Concept, provides the description of the maintenance concepts, this includes: the scope of software maintenance; the tailoring of the post-delivery process; the designation of who will provide maintenance; and an estimate of life-cycle costs;
- Section 7: Maintenance Activities, specifies the specific maintenance activities as well as the general software engineering activities that are performed during pre-delivery and post-delivery;
- Section 8: Resources, describes the hardware and software needs for the maintenance activities, this includes a description about the development, maintenance and target environments. Methods adopted are also described here;
- Section 9: Maintenance Process, describes the various phases how modification request are evaluated to determine its classification and handling priority and assignment for implementation as a block of modifications that will be released to the user, reference about the Software Configuration Control Board is also included;
- Section 10: Training Requirements, identifies the training activities necessary to meet the needs of the SMP.
- Section 11: Software Product Assurance Activities, describes the of software product assurance activities for maintenance if not covered by SPAP, including NCRs handling. Description of how qualification status w.r.t. the GSWS requirement is maintained and provided.
- Section 12: Software Configuration Management, describes the software configuration management activities for maintenance, including SPRs/SMRs handling, NCR's, CRs and RFWs handling.
- Section 13. Records, Reports and Sample Request Form describes the rules for submission of maintenance reports.

## 7.17 RTEMS Improvement Installation Report

This document presents the Installation Report of RTEMS Tailored and the results of the installation of the RTEMS testsuite. This document is one of the outputs of RTEMS Tailoring Test Suite Re-execution Task.

The document is laid out in the following sections:

- Section 1: Introduction, presents the document purpose, scope and overview.
- Section 2: Installation Report, presents the installation reports.

## 7.18 RTEMS Improvement Acceptance Data Package

This document presents the Acceptance Data Package, one of the outputs of Management, Reporting, Meetings and Configuration Management Tasks.

The document is laid out in the following sections:

- Section 1: Introduction, presents the document purpose, scope and overview.
- Section 2: Acceptance Data Package, presents the list of items delivered.

## 7.19 RTEMS Tailored

This deliverable contains all the files of RTEMS Improvement software. It is formed by the following main folders (the folders not described contains makefiles and other files used to compile the testsuite):

- c/src: This directory is the source code root for those RTEMS components which must be compiled or linked in a way that is specific to a particular CPU model or board. It encompasses the following subdirectories:
  - c/src/lib: This directory contains the directories “libbsp” and “libcpu” which contain the source code for the Board Support Packages and CPU model platform specific source code for RTEMS. These two directories are organized based upon the CPU family and boards. In RTEMS Improvement project, this folder only contains the source code related to SPARC ERC32, LEON2 and LEON3.
  - c/src/optman: This directory contains stubs for the RTEMS Classic API Managers which are considered optional and whose use may be explicitly forbidden by an application. When RTEMS, as library, is built, each of the dummy managers present in this directory is translated into a no-<manager>.rel object file. When the RTEMS user application is built, and according to the application’s requirements, dummy managers are linked to the executable image file instead of real managers in order to reduce the memory footprint.
- cpukit: this directory contains a set of subdirectories which hold the source files comprising the executive portion of the RTEMS development environment as well as the portable support libraries, such as the C Library. The routines internal to RTEMS, located in the “score” subdirectory are separated from the API specific source code files. The following subdirectories are incorporated in it:
- cpukit/include: This directory contains the header files that are private to RTEMS and are not considered to be owned by any other component.
- cpukit/rtems: This directory contains the implementation of the RTEMS Classic API.
- cpukit/sapi: This directory contains the implementation of RTEMS services which are required but beyond the realm of any standardization efforts. It includes initialization, shutdown, and I/O services.
- cpukit/score: This directory contains the SuperCore part of RTEMS. Classic API is implemented in terms of SuperCore services. This provides a common infrastructure and a high degree of interoperability between the other APIs. For example, services from all APIs may

be used by any task independent of the API used to create it. Within the score directory there is also a subdirectory for each CPU family that contains the CPU dependent code necessary to host RTEMS.

According with the explanation above, RTEMS Improvement is composed by three main components:

- RTEMS API
- RTEMS Core
- RTEMS BSP

The RTEMS API allows the interaction with the application, providing a well-defined interface. This interface allows the management of tasks, inter-task synchronization/communication, error recovery, etc. The RTEMS Core contains all the main functionality of RTEMS, such as the scheduler, dispatcher, time management, inter-task synchronization and communication. The RTEMS BSP contains hardware dependent code, such as handling interrupts, hardware devices (clock device driver) and initialization routines. For this project, there are three relevant BSP architectures: ERC32, LEON2 and LEON3.

The RTEMS Classic APIs considered in the RTEMS Tailored version are:

- Initialization Manager: Responsible for initiating and shutting down RTEMS. Initiating RTEMS involves creating and starting all configured initialization tasks and invoking the initialization routine for each user-supplied device driver. In a multiprocessor configuration, this manager also initializes the inter-processor communications layer.
- Task Manager: Provides a comprehensive set of directives to create, delete, and administer tasks.
- Interrupt Manager: Any real time executive must provide a mechanism for quick response to externally generated interrupts to satisfy the critical time constraints of the application. The interrupt manager provides this mechanism for RTEMS. This manager permits quick interrupt response times by providing the critical ability to alter task execution which allows a task to be preempted upon exit from an ISR.
- Clock Manager: Provides support for time of day and other time related capabilities.
- Timer Manager: Provides support for timer facilities.
- Event Manager: Presents a high performance method of inter-task communication and synchronization.
- Message Queue Manager: Makes available communication and synchronization capabilities using RTEMS message queues.
- Semaphore Manager: This manager uses the standard Dijkstra counting semaphores to provide synchronization and mutual exclusion capabilities.
- Rate Monotonic Manager: Provides facilities to implement tasks which execute in a periodic fashion.
- I/O Manager: The Input/Output interface manager provides a well defined mechanism for accessing device drivers and a structured methodology for organizing device drivers.
- Error Manager: Processes fatal or irrecoverable errors and non-fatal errors.

- User Extension Manager: Allows the application developer to augment the executive by allowing him to supply extension routines which are invoked at critical system events.

## 7.20 Software Development Plan

This document presents RTEMS Improvement Software Development Plan, one of the outputs of Management, Reporting, Meetings and Configuration Management Task of the RTEMS Improvement Implementation Phase. This document follows closely the structure suggested in the Galileo Software Standards, Appendix 39.

The document is laid out in the following sections:

- Section 1: Introduction, including the acronyms and the applicable and reference documents;
- Section 2: System Overview, providing an overview of the system context, the architecture, the processors used, Hardware/Software interfaces and safety features;
- Section 3: Software Overview, describing the software functions with emphasis on the proposed partitioning concepts. It also describes the software to be delivered, major activities and major deliverables;
- Section 4: Development Management, presenting the project organisation and resources (contractor facilities, organisation structure, personnel and responsibilities), schedule and milestones (software reviews planning, activities and activity network), risk management, security, interface with associated contractors, subcontractors, corrective action process, problem/change report and progress reporting;
- Section 5: Software Engineering, describing the software development approach, the software engineering environment (software items, hardware items, installation, control and) and software standards and procedures (requirement standards, design standards, coding standards, standard tailoring, testing, software programmatic and critical software);
- Section 6: Document Plan, listing the documents to be produced, documents approval, delivery processes, time schedule for documents;
- Section 7: Acceptance – Installation, identifying the activities and procedures necessary to integrate the software products into its operational environment for determining whether or not the product satisfies its acceptance criteria, enabling the agency to determine whether or not to accept the product and how the quality staff will check that the activities have been implemented.
- Section 8: Maintenance - Migration, identifying the activities and procedures necessary to modify the software after delivery, to correct faults, improve performance or adapt to a changed environment and how quality staff will check that the activities have been implemented.
- Section 9: Qualification Basis, including means showing the applicability of software DAL-B. Including categories and software level, software safety and potential software contributions to failure conditions and additional considerations.
- Section 10: Software Verification, describes the topics related with verification activities, organisation and resources, schedule and administrative procedures.

- Section 11: Software Validation, describes the topics with verification activities, organisation and resources, schedule and activities.
- Section 12: Software FMECA and Safety Verification, provides a description of Reliability, Availability, Maintainability and Safety.

## 7.21 Review Plan

This document was issued in every review of RTEMS Improvement, providing description of every review. This document follows the points suggested in the Galileo Software Standards, Appendix A.41, and RTEMS Improvement Software Development Plan.

The document is laid out in the following sections (according with each RTEMS Improvement reviews):

- Section 1: Introduction, including this subsection, the acronyms and the applicable and reference documents;
- Section 2: RTEMS Improvement – Preliminary Design Review, providing the full description of the Preliminary Design Review of the RTEMS Improvement project.
- Section 3: RTEMS Improvement – Detailed Design Review, providing the full description of the Detailed Design Review of the RTEMS Improvement project.
- Section 4: RTEMS Improvement – Test Readiness Review, providing the full description of the Test Readiness Review and Delta TRR of the RTEMS Improvement project.
- Section 5: RTEMS Improvement – Critical Design Review, providing the full description of the Critical Design Review of the RTEMS Improvement project.
- Section 6: RTEMS Improvement – Acceptance Review, providing the full description of the Acceptance Review of the RTEMS Improvement project.

## 7.22 Final Report

This document makes a brief presentation of the RTEMS Improvement deliverables and final product, the RTEMS Tailored, the Testsuite, the documentation and the results. The Final Report is one of the outputs of Management, Reporting, Meetings and Configuration Management Tasks of the RTEMS Improvement Implementation Phase.

The document is laid out in the following sections:

- Section 1: Introduction, including this subsection, the acronyms and the applicable and reference documents;
- Section 2: Overview, providing an overview of the system context, the RTEMS overview and RTEMS Improvement (API Managers, BSPs, modifications made, the testsuite and documentation);
- Section 3: Validation, Unit and Integration Tests, providing an overview of the tests results and tests results contents;

- Section 4: Software Budget Report, highlighting the RTEMS Improvement timing characteristics, memory occupancy and CPU usage;
- Section 5: Product Assurance, providing an overview to the product assurance activities and latest quality metrics collected in the RTEMS Improvement deliverables;
- Section 6: Study Conclusions and Future Work, listing the project conclusions, lessons learned, the intended work of the five-year maintenance phase and evolutionary maintenance of the next RTEMS LEON Upgrade project.

## 7.23 RTEMS Improvement Product Assurance Plan

This document presents RTEMS Improvement Software Product Assurance Plan, one of the outputs of Product Assurance and Quality Task of the RTEMS Improvement Implementation Phase. This document follows closely the structure suggested in the Galileo Software Standards, Appendix 31. This plan specifies the policies and objectives for the organisation, implementation and control of the Product Assurance Program for the RTEMS Improvement. The SPAP defines how EDISOFT will fulfil the requirements of the Galileo Software Standards. It covers the following disciplines:

- Quality Assurance;
- Dependability;
- Software PA.

For all the applicable disciplines, this plan defines:

- The PA organisation, including responsibilities, internal and external interfaces, reporting scheme;
- The tasks to be performed in the scope of the applicable phase of the project;
- The procedures to be followed and tools to be used to support the performance of the planned activities.

The document is laid out in the following sections:

- Section 1: Introduction, including the purpose and scope of this plan;
- Section 2: Documents, including the acronyms, applicable and reference documents;
- Section 3: Organization, presenting the project organisation and resources (organisation structure, tasks and responsibilities and the interfaces);
- Section 4: Software Quality Assurance, presenting the processes for the organisation, implementation and control of the PA programme for the RTEMS Improvement;
- Section 5: Traceability Matrix, including a traceability matrix from the GSWS to the SPAP.

## 7.24 RTEMS Improvement Product Assurance Report

This document presents RTEMS Improvement Software Product Assurance Report, one of the outputs of Product Assurance and Quality Task, regarding the RTEMS Improvement project. This document follows closely the structure suggested in the Galileo Software Standards, Appendix 32. The Software Product Assurance Report was prepared for the RTEMS Improvement project reviews, reporting the activities planned and performed for the project's phases.

The document is laid out in the following sections:

- Section 1: Introduction, including the document's purpose, scope and overview;
- Section 2: Documents, including the acronyms and the applicable and reference documents;
- Section 3: Software Overview, presenting the overview of the RTEMS Improvement project;
- Section 4: Software Assessment, presenting the list of Software Assessment activities;
- Section 5: Metrics Reports, presenting the Metrics Report for software products and processes;
- Section 6: Tools, presenting the list of tools used for developing;
- Section 7: Audits, presenting the list of audits performed to date;
- Section 8: FCV and PCV, presenting the FCV and PCV reports;
- Section 9: Dependability, presenting the result of Dependability audits performed to date;
- Section 10: Lessons Learned Report, presenting the Lessons Learned Report.
- Section 11: Training records, presenting the training performed by the team

## 7.25 RTEMS Improvement Configuration Management Plan

This document contains the Configuration and Documentation Management Plan for the RTEMS Improvement project. It allows the identification of the software product through the systematic control of its composing items and changes (due to modifications, evolution, etc.) in order to keep the integrity and traceability of the product through the life cycle phases. This plan describes the Configuration and Documentation Management (CADM) organisation, disciplines and procedures to be used within this project. The purpose of this plan is to ensure that all documents which define the functional and physical characteristics of the end item (RTEMS Improvement-RI) be uniquely identified and related to the end item (RI) throughout its lifecycle.

- The end item (RI) design standard can be defined at any point of the program and can be correlated to the end item (RI) built status;
- An efficient change control is established and maintained;
- All involved participants are aware of the changes impact and participate to their evaluation.

The document herein defined provides for maximum utilisation of the internal existing identification, control, status accounting, and verification procedures. This is one of the outputs of Management, Review, Reporting and Configuration Management Task of the RTEMS Improvement Implementation Phase. This document follows closely the structure suggested in the Galileo Software Standards, Appendix 40.

This plan and the referred procedures for its implementation, is applicable to all configurable items developed/procured during the project life cycle (e.g., Hardware, Software including software configuration files, Documentation and data files, RID, NCRs, SPRs, off-the-shelf products and re-used products).

The document is laid out in the following sections:

- Section 1: Introduction, this includes the list of the acronyms used;
- Section 2: Documentation, this refers to applicable and reference documents used;
- Section 3: Configuration Management for overall SW Engineering Process, presents the description of the Configuration Management concept for the Software Engineering Process throughout all phases of the software lifecycle;
- Section 4: Software Build, Load and Installation, describes the configuration management procedures for building, loading and installing each version of the software produced;
- Section 5: Software Archive, Backup and Retrieval, describes the configuration management procedures for the archive, backup and retrieval of the software code and associated data sets, for both the development and target environment;
- Section 6: Software Release, describes the configuration management procedures for the release of the software code and associated data sets;
- Section 7: Configuration Management Provisions for Software Intended for re-use, describes the configuration management procedures to be applied to the software intended for re-use;

## 7.26 RTEMS Improvement SOC with GSWS

This document contains the compliance matrix of RTEMS Improvement project with GSWS requirements. This document contains the following information for each GSWS requirement:

- GSWS Requirement Id.: GSWS Requirement identifier number
- SW-DAL A: indicates if the requirement is mandatory for software criticality level A
- SW-DAL B: indicates if the requirement is mandatory for software criticality level B
- SW-DAL C: indicates if the requirement is mandatory for software criticality level C
- SW-DAL D: indicates if the requirement is mandatory for software criticality level D
- SW-DAL E: indicates if the requirement is mandatory for software criticality level E
- Description: Requirement text as in GSWS document
- GSWS Document: expected output(s) for the requirement in object
- Planning: indicates if the requirement is applicable to Planning phase of the project (SRR)

- Specification: indicates if the requirement is applicable to Specification phase of the project (SW-PDR)
- Design: indicates if the requirement is applicable to Design phase of the project (SW-DDR)
- Implementation: indicates if the requirement is applicable to Implementation phase of the project (SW-TRR)
- Integration & TS-Validation: indicates if the requirement is applicable to Integration & TS-Validation phase of the project (SW-CDR)
- RB-Validation: indicates if the requirement is applicable to RB-Validation phase of the project (SW-QR)
- Acceptance: indicates if the requirement is applicable to Acceptance phase of the project (SW-AR)
- Maintenance: indicates if the requirement is applicable to Maintenance phase of the project
- Operation: indicates if the requirement is applicable to Operation phase of the project
- OtherReviews: indicates if the requirement is applicable to any other phase of the project not specified previously
- TXT: indicates if the RTEMS Improvement project is compliant with the requirement text
- DAL: indicates if the RTEMS Improvement project is compliant with the requirement DAL level
- Ver: indicates if the RTEMS Improvement project is compliant with the target milestone
- DoC: indicates if the RTEMS Improvement project is compliant with the target deliverable
- Compliance Justification/Comments: Justification of the RTEMS Improvement compliance to the GSWS requirement

## 7.27 RTEMS Improvement Software Criticality Analysis

The SCAR Software Criticality Analysis Report is a Software component document whose contents are described in GSWS. This document contains the report of the software criticality analyses carried out as part of RAMS activities. Their conclusions relative to safety recommendations are included in this document. As part of the software development, SW DAL allocation was carried out using the SW-FMECA to assign a DAL to each component failure. The conclusions of this process are also included in this document. The preliminary SW-FMECA analyses are provided at the second design decomposition level of the Software architecture (SW Components and Sub-Components) identified in Software Design Document.

The document is laid out in the following sections:

- Section 1: Introduction, including the scope and the acronyms.
- Section 2: Documents, presents the list of applicable and reference documents.
- Section 3: Context, providing an overview of the System context, the architecture, the processors used, Hardware/Software interfaces and safety features. Provides also, an

overview of the Software, describing the software functions with emphasis on the proposed safety and partitioning concepts.

- Section 4: Criticality Analysis, presents the Safety Analysis of the SW Items, including the approach for the criticality analysis, SW-FMECA Analysis and results with the DAL Allocation of RTEMS Improvement SW Components/Sub-Components.
- Section 5: Safety Recommendations: presenting the Safety Recommendations derived from RAMS activities and their status.

## 7.28 EDILIB

Although RTEMS Tailored has no dependencies to external libraries, the testsuite, and some applications, has external dependencies to some libc functions, like memset or memcpy, GCC uses these functions to copy or initialize variables like structs and arrays. To cover these dependencies, in the testsuite, Edisoft provides the edilib package which contains a small subset of, well tested, libc functions. The edilib is built independently from RTEMS Improvement and then it is linked with RTEMS application.

The following list presents the functions and signatures contained in the edilib:

- void *memcpy(void *dst0 , const void *src0 , size_t len0)
- void *memset(void *m , int c , size_t n)
- struct tm *_mktm_r(const time_t *tim_p , struct tm *res)
- int strncmp(const char *s1 , const char *s2 , size_t n)

## 7.29 Conclusion

It is noted that RTEMS SMP project will follow a different deliverable structure than the RTEMS Improvement, because RTEMS Improvement project followed the GSWS standard. Also, since RTEMS SMP diverged from original RTEMS (and hence from RTEMS Improvement), most of the technical information in RTEMS Improvement deliverables will not be applicable to RTEMS SMP. As a consequence of the points above, it was decided to not reuse RTEMS Improvement deliverables into RTEMS SMP project.

CHAPTER

EIGHT

# ANALYSIS OF OTHER STANDARDS

The aim of the Qualification toolchain is automate as much as possible the pre-qualification of the RTEMS project. The use-case for this tool will be RTEMS SMP, which is a subset of RTEMS for multicore processors, which operate in Symmetric MultiProcessing configuration. This subset of RTEMS will be pre-qualified using the produced toolchain for space applications. This document describes the study undertaken to compare the content of the applicable standards for the RTEMS SMP Project, ECSS-E-ST-40C and ECSS-Q-ST-80C, with the following standards, highlighting the major differences:

- GSWS, [GSW04]
- DO-178C, [S20511b]
- DO-330, [S20511c]
- DO-333, [S20511a]
- IEC 60158-1, [IEC10a]
- IEC 60158-3, [IEC10b]
- ISO 26262, [ISO11]

This aims to guarantee the minimization of future possible qualification in these standards or if there is an impediment to do so.

In order to allow for a better understanding of how each standard deals with (functional) safety aspects, throughout this document, specifically, in the respective sections where the analysis of the standards are performed, the titles of the sections and subsections were kept identical or similar to the ones found in each of the standards. These titles and the quoted sentences are surrounded by a square through this sections.

## 8.1 GSWS Analysis

In this section it is presented the study performed to assess the compatibility of GSWS with ECSS. The study was performed considering the GSWS sections and assessing if each section's requirements are/are not compatible with ECSS. The equivalent section in ECSS is pointed out (when applicable) and the differences between the two standards are highlighted. The study is presented below:

```
3 SYSTEM LEVEL CONSIDERATIONS WRT SOFTWARE
```

This section does not contain requirements.

### 3.1 GALILEO SYSTEM DECOMPOSITION

This section does not contain requirements.

### 3.2 UPPER-LEVEL LIFE CYCLE AND SW LIFE CYCLE SYNCHRONIZATION

This section does not contain requirements.

### 3.3 UPPER-LEVEL RAMS

The requirement in this section are according with ECSS standard (see 6.2.2 ECSS-Q-ST-80), which also defines the necessary relations between System and Software teams regarding the RAMS requirements.

#### 3.3.1 SW-DALs Allocation

The requirements in this section are according with ECSS standard (see 6.2.8 and 5.6 ECSS-Q-ST-80), which also specifies that automatic development/verification/validation tools shall be in accordance with the project required standards and rules.

#### 3.3.2 Contribution of Software to upper-level RAMS Analyses

This section does not contain requirements.

### 3.4 SECURITY REQUIREMENTS

The ECSS defines also the need for security requirements (see 6.3.2.4 ECSS-Q-ST-80). However, it does not define the rules to define these security requirements, being up to the SW supplier to do it. The GSWS is more specific regarding this point, pointing to security standards. If GSWS qualification is required for RTEMS SMP, these security standards shall be applied to the project.

### 3.5 SOFTWARE ACTIVITIES AT UPPER-LEVEL

This section does not contain requirements.

#### 3.5.1 Upper-level Database Definition

The ECSS defines that the costumer shall specify a database (see 5.2.4.4 ECSS-E-ST-40). However, the GSWS is more detailed in what kind of information shall be in the database specification.

#### 3.5.2 RB Definition as part of the Upper-level Requirements Folder

The ECSS also defines requirements for the Requirement Baseline (Upper-Level) definition (see 5.2 ECSS-E-ST-40).

### 3.6 COMMON SOFTWARE APPROACH AT SYSTEM AND SEGMENT LEVEL

These requirements are related with interface/integration between the software and the whole system. This is defined in the ECSS (see 5.2 ECSS-E-ST-40, more specifically 5.2.4).

# 4 SOFTWARE LIFE CYCLES

## 4.1 INTRODUCTION

This section does not contain requirements.

## 4.2 SELECTING SW LIFE CYCLE MODELS

ECSS provides requirements on life cycle definition and management (see 5.3.2 ECSS-E-ST-40C, 6.1 ECSS-Q-ST-80C and ECSS-M-ST-10C), however it does not specify which life cycle the project should take as in GSWS.

## 4.3 SW LIFE CYCLE MODELS REQUIREMENTS

### 4.3.1 Reference Life Cycles for Generic Software

This section does not contain requirements.

### 4.3.2 Waterfall Model

The ECSS does not define the model for Waterfall lifecycle, but instead defines a model applicable to all lifecycles which is slightly different from the one presented in GSWS in therms on when begin producing and baseline certain deliverables (see Table A-1 ECSS-E-ST-40).

### 4.3.3 Incremental Model

The ECSS does not define the model for Incremental lifecycle, but instead defines a model applicable to all lifecycles which is slightly different from the one presented in GSWS in therms on when begin producing and baseline certain deliverables (see Table A-1 ECSS-E-ST-40).

### 4.3.4 Reference Life Cycle for Databases

The ECSS does not define life cycle for Databases.

### 4.3.5 Reference Life Cycle for the Development of MMI Applications

The MMI requirements are defined in ECSS (see 5.2.2.3 ECSS-E-ST-40C, which redirects to ECSS-E-ST-10-11). The ECSS does not define life cycle development for MMI Applications.

### 4.3.6 Reference Life Cycle for Test Software

The ECSS does not define life cycle development for Test Software.

### 4.3.7 Reference Life Cycle for Simulators

The ECSS specifies simulators requirements (see ECSS-E-TM-40-07A) with more detail than GSWS. The ECSS does not define life cycle development for Simulators.

### 4.3.8 Algorithms

The ECSS does not contain requirements specific to algorithms implementation and life cycle.

# 5 SOFTWARE REVIEWS

## 5.1 GENERAL REQUIREMENTS FOR SW REVIEWS

### 5.1.1 Review Description and Scope

The ECSS specifies the review plan (see 5.3.3.2 ECSS-E-ST-40, which also redirects to ECSS-M-ST-10-01).

### 5.1.2 Review Schedule

The ECSS Review Plan (see 5.3.3.2 ECSS-E-ST-40) describes the existence of a review schedule. However, GSWS is more detailed for what activities and their dates should be on the Review Plan.

### 5.1.3 Review Objectives

The ECSS Review Plan (see 5.3.3.2 ECSS-E-ST-40) describes the review objectives.

#### 5.1.3.1 System Requirements Review (SRR)

The ECSS defines the System Requirements Review objectives (see 5.3.4.1 ECSS-E-ST-40C), however with less level of detail rather than GSWS. See also the comment in sections 4.3.2 and 4.3.3, regarding the deliverables discrepancy in project phases.

#### 5.1.3.2 SW Preliminary Design Review (SW-PDR)

The ECSS defines the SW Preliminary Design Review objectives (see 5.3.4.2 ECSS-E-ST-40C). GSWS details more points rather than ECSS, but it is missing a reference that the interfaces shall also be reviewed (this is a typo, since this is implicitly referred in sections 4.3.2 and 4.3.3. The ECSS also states that in case the software requirements are baselined before the start of the architectural design, the part of the PDR addressing the software requirements specification and the interfaces specification shall be held in a separate joint review anticipating the PDR, in a software requirements review (SWRR). See also the comment in sections 4.3.2 and 4.3.3, regarding the deliverables discrepancy in project phases.

#### 5.1.3.3 SW Detailed Design Review (SW-DDR)

In the ECSS this phase shall be only taken in case the software detailed design is baselined before the start of the coding, otherwise, it will be merged in the CDR (see 5.3.4.3 ECSS-E-ST-40C). GSWS is more detailed concerning the objectives of the DDR, however it does not refer to address the software budget as in ECSS. See also the comment in sections 4.3.2 and 4.3.3, regarding the deliverables discrepancy in project phases.

#### 5.1.3.4 SW Test Readiness Review (SW-TRR)

The ECSS does not specify when the TRR shall be held. It is just is stated that it shall be held before the start of test activities and after the test activities, a TRB (Test Review Board) shall take place to approve the test results in the end of test activities (see 5.3.5 ECSS-E-ST-40). However, it is implicit that the TRR shall take place between DDR and CDR, which is in accordance with

GSWS. The TRR objectives are more detailed in GSWS. See also the comment in sections 4.3.2 and 4.3.3, regarding the deliverables discrepancy in project phases.

`5.1.3.5 SW Critical Design Review (SW-CDR)`

The ECSS defines the Critical Design Review objectives (see 5.3.4.3 ECSS-E-ST-40C), however with less level of detail rather than GSWS. See also the comment in sections 4.3.2 and 4.3.3, regarding the deliverables discrepancy in project phases.

`5.1.3.6 SW Qualification Review (SW-QR)`

The ECSS defines the Qualification Review objectives (see 5.3.4.4 ECSS-E-ST-40C), however with less level of detail rather than GSWS. See also the comment in sections 4.3.2 and 4.3.3, regarding the deliverables discrepancy in project phases.

`5.1.3.7 SW Acceptance Review (SW-AR)`

The ECSS defines the Acceptance Review objectives (see 5.3.4.5 ECSS-E-ST-40C), however with less level of detail rather than GSWS. See also the comment in sections 4.3.2 and 4.3.3, regarding the deliverables discrepancy in project phases.

`5.1.4 Review Participants`

The ECSS defines the review participants (see 5.2 ECSS-M-ST-10-01C) similar as in GSWS.

`5.1.5 Review Data Package`

As stated in 4.3.2 and 4.3.3 the ECSS defines slightly different the data package delivery and review, but these differences have no impact in project structure.

`5.1.6 Review Success Criteria`

The ECSS defines the reviews success criteria (see Annex P, P.2 <7> ECSS-E-ST-40C) in a similar form as GSWS. The GSWS requires compliance verification of the project at every review, whereas in the ECSS, the compliance matrix is elaborated and then the project shall comply with it (see 5.3.9 ECSS-E-ST-40 and 5.2.1.5 ECSS-Q-ST-80). In practice, GSWS requires an extra effort to show evidence of its requirements compliance.

`5.1.7 Review Conclusion`

The ECSS defines the reviews conclusion (see Annex P, P.2 <7> ECSS-E-ST-40C) in a similar form as GSWS.

`5.2 GENERAL REQUIREMENTS FOR BUILD PROGRESS POINTS`

This section does not contain requirements.

`5.2.1 Build Progress Points Description and Scope`

This section does not contain requirements.

5.2.2 Build Progress Points Participants

The ECSS specifies that the progress meetings shall be between the costumers and suppliers (see 5.2.2 ECSS-M-ST-10C). The GSWS has further level of detail regarding this, but there is no incompatibility with ECSS.

5.2.3 Build Progress Points Data Package

The ECSS does not define data package for progress report (see 5.2.2 ECSS-M-ST-10C).

5.2.4 Build Progress Points Success Criteria

The ECSS does not define success criteria for progress report (see 5.2.2 ECSS-M-ST-10C).

5.2.5 Build Progress Point Conclusion

The ECSS does not define conclusion type for progress report (see 5.2.2 ECSS-M-ST-10C). In ECSS, the results of the meeting shall be documented, including the actions raised from the meeting to be implemented. However, as already stated, the progress reports in ECSS have no objective to pass/fail any work development, but instead to make clear what is the current progress of the project and the necessary actions to be undertaken until the next progress meeting/review.

6 SW ENGINEERING

6.1 INTRODUCTION

There is no currently a ECSS standard for ISVV. However, there is an guide - ESA Guide for Independent Software Verification and Validation - which defines guidelines to perform an ISVV. See next GSWS section 11 for the comparison between the ESA Guide and the GSWS requirements for ISVV.

6.2 SOFTWARE PLANNING PHASE

6.2.1 Phase Description

The ECSS defines the planning phase which includes the System Requirements review Planning activities definition (see 5.2 and 5.3 ECSS-E-ST-40). However in some points, the GSWS is more detailed. See the sections below.

6.2.2 Methods, Standards and Tools

6.2.2.1 Galileo Methods List

This section does not contain requirements.

6.2.2.2 SW Requirements Standard

The ECSS states that the development standard shall be defined by the costumer (see 5.2.4.5 ECSS-E-ST-40), but there is total freedom in the choice of the Requirement Standards. If a

future qualification for the GSWS is required, at least the Standards chosen should be the more closest possible with GSWS to reduce the effort on a possible GSWS qualification.

#### 6.2.2.3 SW Design Standards

The ECSS states that the development standard shall be defined by the costumer (see 5.2.4.5 ECSS-E-ST-40), but there is total freedom in the choice of the Design Standards. If a future qualification for the GSWS is required, at least the Standards chosen should be the more closest possible with GSWS to reduce the effort on a possible GSWS qualification, specially it shall be ensured that the software developed (RTEMS) will cope with the following GSWS requirements:

- [GSWS-SWENG-0930]
- [GSWS-SWENG-0940]
- [GSWS-SWENG-1000]
- [GSWS-SWENG-1010]
- [GSWS-SWENG-1040]
- [GSWS-SWENG-1050] (unless the database design can be translated to Entity Relationship modelling)

Otherwise, these GSWS requirements will be violated, preventing the qualification into this standard.

#### 6.2.2.4 Tools

In the ECSS is specified that the tools and supporting environment shall be defined (see 5.6 ECSS-Q-ST-80C) and is according with written in the GSWS.

### 6.2.3 Verification requirements

The ECSS verifications for procedures for this phase (see 5.8.3 ECSS-E-ST-40C and 6.2.6 ECSS-Q-ST-80) are according with the GSWS.

## 6.3 SOFTWARE SPECIFICATION PHASE

### 6.3.1 Phase Description

The Phase requirements described for the Specification phase are according with the ECSS requirements (see 5.4 and 5.8.3 ECSS-E-ST-40C). Note that the User Manual, as indicated in [GSWS-SWENG-1090], is not to be specified in this phase, according to the ECSS.

### 6.3.2 Methods, Standards and Tools

#### 6.3.2.1 Galileo Methods List

This section does not contain requirements.

#### 6.3.2.2 SW Coding Standards

#### 6.3.2.2.1 Requirements

The ECSS states that the Coding Standards shall be defined (see 6.3.4 ECSS-Q-ST-80C), however there are no specification of how these coding standards should be. Also there are requirements for critical software (see 6.2.3 ECSS-Q-ST-80C), but in ECSS it is up to the user the definitions of the measures for the critical software. As a consequence, the following GSWS requirements may become incompatible with software developed under ECSS requirement, which are less restrictive:

- [GSWS-SWENG-1180]
- [GSWS-SWENG-1200]
- [GSWS-SWENG-1210] - this requirement may work as a turn around to the requirement [GSWS-SWENG-1180]
- [GSWS-SWENG-1240] - referred as a suggestion in ECSS, not mandatory
- [GSWS-SWENG-1250]
- [GSWS-SWENG-1260]
- [GSWS-SWENG-1270]
- [GSWS-SWENG-1280]
- [GSWS-SWENG-1290]
- [GSWS-SWENG-1310]
- [GSWS-SWENG-1330]
- [GSWS-SWENG-1340]
- [GSWS-SWENG-1360]
- [GSWS-SWENG-1370]
- [GSWS-SWENG-1380]
- [GSWS-SWENG-1390]
- [GSWS-SWENG-1400]
- [GSWS-SWENG-1410]
- [GSWS-SWENG-1420]
- [GSWS-SWENG-1430]
- [GSWS-SWENG-1440]
- [GSWS-SWENG-1450]
- [GSWS-SWENG-1460]

#### 6.3.2.2.2 Recommendations

This section does not contain requirements.

#### 6.3.3 Verification Requirements

The ECSS describes the verification requirements for the specification phase (see 5.8.3.2 and 5.8.3.3 of ECSS-E-ST-40C). The Conformance with GSWS templates defined in [GSWS-SWENG-1470] may be not assured following the ECSS. In the remaining topics, the ECSS and GSWS are in accordance.

##### 6.3.3.1 Verification of the TS

The ECSS describes the verification requirements for the TS (see 5.8.3.2 of ECSS-E-ST-40C). The ECSS requirements are more exhaustive in what shall be assured in TS, but they are according with GSWS.

##### 6.3.3.2 Verification of the SDD (architectural design section)

The ECSS describes the verification requirements for the SDD - Architectural Design section (see 5.8.3.3 of ECSS-E-ST-40C) and they are according with GSWS. In this section it is not referred the verification of compliance with design standards ([GSWS-SWENG-1580]), in ECSS this is only required at Detailed Design level.

### 6.4 SOFTWARE DESIGN PHASE

#### 6.4.1 Phase Description

The Phase requirements described for the Design phase are according with the ECSS requirements (see 5.5.2 and 5.8.3 ECSS-E-ST-40C). Note that the User Manual (referred in [GSWS-SWENG-1620]), following the ECSS, is begun at this phase.

#### 6.4.2 Methods, Standards and Tools

This section does not contain requirements.

#### 6.4.3 Verification Requirements

The ECSS Verification requirements described for the Design Phase are in accordance with GSWS (see 5.8.3.4 of ECSS-E-ST-40C). GSWS templates and Configuration management procedures slightly differ from ECSS.

##### 6.4.3.1 Verification of the SDD (detailed design section)

The ECSS describes the verification requirements for the SDD (see 5.8.3.4 of ECSS-E-ST-40C) and they are according with GSWS.

### 6.5 SOFTWARE IMPLEMENTATION PHASE

#### 6.5.1 Phase Description

The Phase requirements described for the Implementation phase are according with the ECSS requirements (see 5.5.3, 5.5.4 and 5.8.3 ECSS-E-ST-40C). In ECSS it is not mandatory to define the traceability between SDD (ADD section) and integration test ([GSWS-SWENG-1770])

and traceability between TS and TS-Validation test cases ([GSWS-SWENG-1780]) at implementation phase, although the Integration Test Plan shall be already available at this phase (as Release).

`6.5.2 Methods, Standards and Tools`

This section does not contain requirements.

`6.5.2.1 Structural Coverage Requirements`

The Structural coverage requirements defined in ECSS are the same for GSWS, concerning high critical software, DAL A and DAL B, (see 5.8.3.5 of ECSS-E-ST-40C). For less critical software there are slight differences in source code coverage specifications.

`6.5.2.2 Tests Methods Requirements`

The ECSS defines Test Methods requirements according with GSWS. See 5.5.3 and 5.8.3.6 of ECSS-E-ST-40C (although some equivalent GSWS requirements may be also scattered through 5.5 and 5.8.3 remaining subsections). See also 6.2.3, 6.2.6.5 and 7.3.6 ECSS-Q-ST-80C, which contains requirements traced to this section.

`6.5.3 Verification Requirements`

The ECSS Verification requirements described for the Implementation Phase are in accordance with GSWS (see 5.8.3, more specifically 5.8.3.5, 5.8.3.6 and 5.8.3.10 of ECSS-E-ST-40C).

`6.5.3.1 Verification of Source and Executable Object Code`

The ECSS defines the Verification requirements for the source/object code mostly according with ECSS (see 5.8.3, specially 5.8.3.5 ECSS-E-ST-40C). In ECSS, there is no separation between source and object code definitions, except for the coverage, when traceability between source code and object code cannot be verified. There are some remarks in the following requirements:

- [GSWS-SWENG-1990] - ECSS allows the existence the deactivated code, providing that it is shown that the code cannot be executed (see 6.2.3.2 ECSS-Q-ST-80C)
- [GSWS-SWENG-2000] to [GSWS-SWENG-2050] - The ECSS does not define the percentage of SCS verifications, so it is assumed that even, if the SCS verifications are made by hand, they need to be done for all software. In RTEMS, they will be done automatically, so the compatibility with GSWS will be automatically met.
- [GSWS-SWENG-2060] to [GSWS-SWENG-2090] - The ECSS does not define the percentage for implementation of detail design verification. It is assumed that for all criticality levels it has to 100 %, according to ECSS, so these requirements will be automatically met.
- [GSWS-SWENG-2110] - No explicit reference in ECSS, but covered by the verification of security in source code.
- [GSWS-SWENG-2120] and [GSWS-SWENG-2130] - No requirements on ECSS on when/how frequently static code analysis, inspection and walkthroughs shall be performed, except for that they shall be made before the reviews. However, this does not constitute an incompatibility between GSWS and ECSS, since for both standards, before

each review the verifications shall be performed. These requirements constitute indeed a common practice of verifying code to correct possible errors as upstream as possible.

#### 6.5.3.2 Verification of SUP

These requirements are according with ECSS (see 5.8.3.6 ECSS-E-ST-40C).

#### 6.5.3.3 Verification of SIP

These requirements are according with ECSS (see 5.8.3.7 ECSS-E-ST-40C).

#### 6.5.3.4 Verification of VTS-TS

These requirements are according with ECSS (see 5.8.3.8 ECSS-E-ST-40C).

#### 6.5.3.5 Verification of GTR(UT)

These requirements are according with ECSS (see 5.8.3.5 and 5.8.3.6 ECSS-E-ST-40C).

## 6.6 SOFTWARE INTEGRATION AND TS-VALIDATION PHASE

### 6.6.1 Phase Description

The requirements described for Integration and Validation are in accordance with ECSS (see 5.6, 5.8.3.7, 5.8.3.8 and 5.8.3.9 of ECSS-E-ST-40C).

The Maintenance Planning requirements they are according with ECSS (see 5.10.2 ECSS-E-ST-40C and 6.3.8 ECSS-Q-ST-80C).

### 6.6.2 Methods, Standards and Tools

Already analyzed, regarding requirement [GSWS-SWENG-2305], it is according with ECSS (see 6.2.2.10 ECSS-Q-ST-80C). ECSS goes further stating that if failures from lower criticality components cannot be isolated from other components, then these components criticality shall be promoted to highest criticality among the software components.

#### 6.6.2.1 Non-regression Req.'s during Integration and TS-Validation

The non-regression requirements presented are in accordance with ECSS (see 5.6.2 of ECSS-E-ST-40C and 6.2.3.4, 6.3.5.15, 6.3.5.16, 6.3.5.17 and 6.3.5.18 of ECSS-Q-ST-80C).

### 6.6.3 Verification Requirements

The requirements presented here are in accordance with ECSS (see 5.6, 5.8.3.5, 5.8.3.7, 5.8.3.8, 5.8.3.9 and 5.8.3.10 of ECSS-E-ST-40C and see 6.3.5 ECSS-Q-ST-80). A remark on requirements [GSWS-SWENG-2400] and [GSWS-SWENG-2410] - As stated above in ECSS there is no distinction between source and object code. Since, in practice, it is the object code that will be tested, these requirements are automatically according with ECSS.

#### 6.6.3.1 Verification of VTS-RB

These requirements are according with ECSS (see 5.6 and 5.8.3.8 ECSS-E-ST-40C).

#### 6.6.3.2 Verification of GTR(IT)

These requirements are according with ECSS (see 5.8.3.7 ECSS-E-ST-40C).

#### 6.6.3.3 Verification of GTR (TS)

These requirements are according with ECSS (see 5.8.3.8 ECSS-E-ST-40C).

## 6.7 SOFTWARE RB-VALIDATION PHASE

### 6.7.1 Phase Description

These requirements are according with ECSS (scattered in 5.6.4, 5.7, 5.8.3.8, 5.8.3.9, 5.8.3.12, 5.10 ECSS-E-ST-40C).

### 6.7.2 Methods, Standards and Tools

This section does not contain requirements.

#### 6.7.2.1 Non-regression Req.'s during RB-Validation

The non-regression requirement presented is in accordance with ECSS (see 5.6.2 of ECSS-E-ST-40C and 6.2.3.4, 6.3.5.15, 6.3.5.16, 6.3.5.17 and 6.3.5.18 of ECSS-Q-ST-80C).

### 6.7.3 Verification requirements

The requirements presented here are in accordance with ECSS (see 5.8.3.8 of ECSS-E-ST-40C and see 6.3.5 ECSS-Q-ST-80). GSWS templates and Configuration management procedures slightly differ from ECSS.

#### 6.7.3.1 Verification of SATP

These requirements are according with ECSS (see 5.8.3.8 ECSS-E-ST-40C).

#### 6.7.3.2 Verification of GTR (RB)

These requirements are according with ECSS (see 5.8.3.8 ECSS-E-ST-40C).

## 6.8 SOFTWARE ACCEPTANCE PHASE

The requirements presented here are in accordance with ECSS (see 5.6.4, 5.7, 5.8.3.12 and 5.10 ECSS-E-ST-40C).

### 6.8.2 Methods, Standards and Tools

This section does not contain requirements.

#### 6.8.2.1 Non-regression Req.'s during Acceptance

The non-regression requirement presented is in accordance with ECSS (see 6.2.3.4, 6.3.5.15, 6.3.5.16, 6.3.5.17 and 6.3.5.18 of ECSS-Q-ST-80C, this is also applicable for the Acceptance tests).

#### 6.8.3 Verification requirements

The requirements presented here are in accordance with ECSS (see 5.7 ECSS-E-ST-40C, 6.3.5 and 6.3.6 ECSS-Q-ST-80C, Testing and validation procedures are applicable for acceptance testing).

##### 6.8.3.1 Verification of GTR (AT)

This requirement is in accordance with ECSS (see 5.7.3 ECSS-E-ST-40C).

### 6.9 SOFTWARE OPERATION PHASE

#### 6.9.1 Phase Description

The requirements in this section are in accordance with ECSS (5.9, 5.10.6 and 5.10.7 ECSS-E-ST-40C). Note that migration ([GSWS-SWENG-2700]) and retirement ([GSWS-SWENG-2710]) is described in Maintenance phase in ECSS standard.

#### 6.9.2 Methods, Standards and Tools

This section does not contain requirements.

#### 6.9.3 Verification Requirements

This requirement is implicitly according with ECSS (see 5.9 ECSS-E-ST-40C). There are no explicit requirements for the verification of operation phase documentation, but they shall comply with ECSS standard and they shall be verified in Operational Readiness Review phase. GSWS templates and Configuration management procedures slightly differ from ECSS.

### 6.10 SOFTWARE MAINTENANCE PHASE

#### 6.10.1 Phase Description

This requirement is in accordance with ECSS (see 5.10 ECSS-E-ST-40C).

#### 6.10.2 Methods, Standards and Tools

This section does not contain requirements.

##### 6.10.2.1 Non-regression for Revisions

The non-regression requirements presented are in accordance with ECSS (see 6.2.3.4, 6.3.5.15, 6.3.5.16, 6.3.5.17 and 6.3.5.18 of ECSS-Q-ST-80C, this ia also applicable for maintenance). In ECSS there is no reference on the possibility to reduce the Non-regression activities on maintenance (as in [GSWS-SWENG-2760]), however this does not have impact in qualifying ECSS software to GSWS standard.

#### 6.10.3 Verification Requirements

These requirements are according with ECSS (see 5.10 ECSS-E-ST-40C). Regarding [GSWS-SWENG-2770] there is no explicit requirements for the verification of maintenance phase documentation, but they shall e verified each time a maintenance activity is undertaken.

# 7 SOFTWARE CONFIGURATION MANAGEMENT

## 7.1 GENERAL

These requirements are according with ECSS (see 5.3.7, 5.2.1 and 5.2.2 ECSS-M-ST-40C).

## 7.2 CONFIGURATION IDENTIFICATION AND VERSION CONTROL

### 7.2.1 SW Configuration Items

These requirements are partially in accordance with ECSS (see 5.3.1 ECSS-M-ST-40C). There are differences on the criteria to consider at item Configuration Item and on the identifiers to be used (ECSS does not define an identifier format).

### 7.2.2 Versions and Revisions

The ECSS does not specify a format to represent version and revision numbers (see 5.3.1 ECSS-M-ST-40C).

### 7.2.3 Documentation Release and Baseline

The requirement [GSWS-CM-2920] is according with ECSS (see 5.3.1 ECSS-M-ST-40C). [GSWS-CM-2900] and [GSWS-CM-2910] were already analyzed, see against Table A-1: ECSS-E-ST-40 and ECSS-Q-ST-80 Document requirements list (DRL).

## 7.3 PROBLEM REPORTING AND CHANGE CONTROL

### 7.3.1 SW Change Control Board

The ECSS does define the need for Configuration change procedure (see 5.3.2.1 ECSS-M-ST-40C), but it is up to the stakeholder the procedures definitions.

### 7.3.2 SPR and SMR Handling Process

These requirements are in accordance with ECSS (see 5.2.5 and 6.2.4 ECSS-Q-ST-80C and 5.3.2 ECSS-M-ST-40C). There are slight differences as depicted below:

- There is no SMR in ECSS, it is merged into SPR as stated in GSWS:

  Note: SPR and SMR can be merged in one single document.

- [GSWS-CM-3000] and [GSWS-CM-3010] - The GSWS is more strict requiring that there shall be specific verifications/tests to verify that the SPR has been solved. ECSS requires only re-execution of the tests of affected software.

## 7.4 CONFIGURATION STATUS ACCOUNTING

The requirements presented here are in accordance with ECSS (see ECSS-M-ST-40C). [GSWS-CM-3080] is implicit in ECSS, since

```
The purpose of configuration status accounting reports is to provide a reliable source of␣
↪configuration information to support all programme or project activities.
```

```
7.5 CONFIGURATION VERIFICATION
```

The ECSS defines also the need for FCV and PCV. However in ECSS the FCV shall be performed at QR and the PCV shall be performed at AR (see 5.3.4.2 ECSS-M-ST-40C).

```
7.5.1 Functional Configuration Verification
```

ECSS does not define what should be verified at FCV, which shall be agreed between the supplier and costumer (see 5.3.1.3 and 5.3.1.4 ECSS-M-ST-40C).

```
7.5.2 Physical Configuration Verification
```

ECSS does not define what should be verified at PCV, which shall be agreed between the supplier and costumer (see 5.3.1.3 and 5.3.1.4 ECSS-M-ST-40C).

```
7.6 SCM TOOLS AND TECHNIQUES
```

This requirement is according with ECSS (see 5.2.1.1 which redirects to <6.2> Annex A ECSS-M-ST-40C).

```
7.7 RETENTION AND ARCHIVE
```

These requirements are according with ECSS (see 5.3.7.6 and all topics of ECSS-M-ST-40C, which describe the necessary archiving features). There is no ECSS specification on what concerns the requirement [GSWS-CM-3240].

```
7.8 DELIVERY
```

- Regarding requirements [GSWS-CM-3270] to [GSWS-CM-3290], the ECSS is more specific and restrictive on the format of deliveries and the electronic transmission requirements (see 5.3.7.4 ECSS-M-ST-40C) than GSWS, hence the GSWS can use the ECSS formats.
- Regarding requirements [GSWS-CM-3295] to [GSWS-CM-3390], most of requirements are according with ECSS (see 6.3.6 and 6.3.8 ECSS-Q-ST-80C and 5.9 and 5.10 ECSS-E-ST-40C), with exceptions:
- [GSWS-CM-3310] - Certificate of Conformance for developed software not defined in ECSS.
- [GSWS-CM-3320] - Inputs to Upper-level Operation Manual and Migration Justification File not defined in ECSS. Migration Plan and Retirement Plan does not have a defined phase to be delivered (see Table A-1: ECSS-E-ST-40 and ECSS-Q-ST-80 Document requirements list (DRL)).
- [GSWS-CM-3340] - According with ECSS, any changes performed during the Maintenance phase shall follow the same procedures as used in development (see 5.10.2.1 ECSS-E-ST-40C). Work-around solutions may given to the product users before a permanent solution is delivered, but ECSS does not define a revision change in the software when doing this (see 5.9.5.3 ECSS-E-ST-40C).

- [GSWS-CM-3350] - According with ECSS (see 6.3.5.20 ECSS-Q-ST-80C)
- [GSWS-CM-3360] - There is no revision concept in ECSS.
- [GSWS-CM-3370] and [GSWS-CM-3390] - In ECSS there is no concept of non formal deliveries.

# 8 DEPENDABILITY AND SAFETY MANAGEMENT

## 8.1 SW RAMS ANALYSES

In ECSS it is defined that the RAMS analysis need shall be assessed depending the type of the project (see section 8 ECSS-Q-ST-30C and Annex F ECSS-Q-ST-40C) and then the analysis will derive the software criticality. [GSWS-RAMS-3410] is according with ECSS (see 6.2.2 ECSS-Q-ST-80C).

## 8.2 SW-DAL REDUCTION PROCEDURE

In ECSS measures to reduce the number of critical components shall be applied and justified (see 6.2.2.4, which redirects to 6.2.3 ECSS-Q-ST-80C). Regarding [GSWS-RAMS-3420] and [GSWS-RAMS-3430], they are according with ECSS (see 7.3 ECSS-Q-HB-80-01A and 6.4 ECSS-Q-HB-80-03A).

# 9 SW PRODUCT ASSURANCE

## 9.1 INTRODUCTION

This section does not contain requirements.

## 9.2 ORGANIZATION AND RESPONSIBILITIES

These requirements are according with ECSS (see 5.1.2, 5.1.4 and 5.2.1.3 ECSS-Q-ST-80C).

## 9.3 SW PRODUCT ASSURANCE ACTIVITIES

### 9.3.1 Contractual Participation

The ECSS states implicitly that the Product Assurance representative has the authority to propose and maintain a software product assurance programme (see 5.1.4.2 and 5.2.1 ECSS-Q-ST-80C). ECSS does not specify that the SPA representative shall shall participate in the review of all changes to the contractual requirements ([GSWS-PA-3520]).

### 9.3.2 Software Product Assurance Planning

These requirements are according with ECSS, apart that the plan/compliance is according with GSWS and not ECSS (see 5.2.1 and 5.1.5 ECSS-Q-ST-80C).

### 9.3.3 Software Product Assurance Verification Tasks

#### 9.3.3.1 Recurrent Phase Independent SW PA Verification Tasks

The requirements presented here are almost according with ECSS (see 6.2.6 ECSS-Q-ST-80C), apart that the verifications shall be made against GSWS standard and not ECSS. The requirement [GSWS-PA-3570] is not specified in ECSS.

`9.3.3.2 Phase Dependent SW PA Verification Tasks`

The requirement [GSWS-PA-3670] is not specified in ECSS.

`9.3.3.2.1 SW Specification Phase`

The ECSS is not specific on what shall be verified by SPA representative in TS. It is stated what shall be verified (see 5.8.3.2 ECSS-E-ST-40C) and that SPA representative shall verify the outputs of this activity (see 6.2.6.2 ECSS-Q-ST-80C). The TS sampling verification is not defined in ECSS.

`9.3.3.2.2 SW Design Phase`

The requirement is according with ECSS, except that ECSS does not define level of sampling (see 6.3.3.4 ECSS-Q-ST-80C). All requirements shall be verified.

`9.3.3.2.3 SW Implementation Phase`

The requirement is according with ECSS (see 6.3.4 ECSS-Q-ST-80C), apart that the verification shall be made against GSWS standard and not ECSS.

`9.3.3.2.4 SW Integration and TS-Validation Phase`

These requirements are partially according with ECSS (see 6.3.5 ECSS-Q-ST-80C). The following requirements are not according with ECSS:

- [GSWS-PA-3730] - ECSS does not forbid having major NCRs/SPRs open at CDR
- [GSWS-PA-3740] - ECSS states that whenever a change is made to the software, all affected areas shall be retested (see 6.3.5.15 ECSS-Q-ST-80C). Hence ECSS does not define the possibility of not performing a validation test campaign in an updated software subject to delivery.
- [GSWS-PA-3760] - No FCV/PCV enviasaged for CDR in ECSS (see 5.3.4.2 ECSS-M-ST-40C).

`9.3.3.2.5 SW RB-Validation Phase`

These requirements are partially according with ECSS (see 6.3.5 ECSS-Q-ST-80C). The following requirements are not according with ECSS:

- [GSWS-PA-3770] - ECSS does not forbid having major NCRs/SPRs open at QR
- [GSWS-PA-3780] - ECSS states that whenever a change is made to the software, all affected areas shall be retested (see 6.3.5.15 ECSS-Q-ST-80C). Hence ECSS does not define the possibility of not performing a validation test campaign in an updated software subject to delivery.
- [GSWS-PA-3800] - ECSS enviasages the need for only FCV for QR (see 5.3.4.2 ECSS-M-ST-40C).

- [GSWS-PA-3810] - The ECSS does not define the need for a test for the User Manual

```
9.3.3.2.6 SW Acceptance Phase
```

These requirements are partially according with ECSS (see 6.2.4, 6.3.5, 6.3.6 and 7.3.7 ECSS-Q-ST-80C). The following requirements are not according with ECSS:

- [GSWS-PA-3820] - ECSS does not forbid having major NCRs/SPRs open at AR
- [GSWS-PA-3850] - The ECSS does not define the need for a verification that the User Manual is executed by the end-user.

```
9.3.3.2.7 SW Maintenance Phase
```

These requirements are according with ECSS (see 5.10.2 ECSS-E-ST-40C and 6.3.8 ECSS-Q-ST-80C), except that these maintenance activities are performed according with GSWS and not ECSS.

```
9.3.4 SW Product Assurance Sub-contractors Monitoring
```

These requirements are according with ECSS (see 5.4.3 ECSS-Q-ST-80C), except that the conformance shall be verified against GSWS and ECSS. Some remarks on the following requirements:

- [GSWS-PA-3930] - Not specified in ECSS, but may be considered as normal activity of Contractor SPA
- [GSWS-PA-3940] - Not specified in ECSS
- [GSWS-PA-3950] - Not specified in ECSS

```
9.3.5 Non Conformances
```

These requirements are mostly according with ECSS (see 5.2.5 and 5.2.6 ECSS-Q-ST-80C), apart that the non-conformance control system is compliant with GSWS and not ECSS. Some remarks on the following requirements:

- [GSWS-PA-3990] and [GSWS-PA-4000] - In ECSS is not mandatory to have a SW tool to manage non conformances, but it is a recommendation (see 5.5.2 ECSS-Q-ST-10-09).

```
9.3.6 Risk Management and Critical Item Control
```

ECSS defines its own standards for Risk management and critical item control (see 5.3 ECSS-Q-ST-80C).

```
9.3.7 Alert Procedure
```

ECSS defines its own requirements for Alerts (see 5.2.4 ECSS-Q-ST-80C).

```
9.3.8 SW Product Assurance Reporting
```

These requirements are according with ECSS (see 5.2.2 ECSS-Q-ST-80C). Some remarks:

- [GSWS-PA-4030] and [GSWS-PA-4040] - There are slight discrepancies between the GSWS and ECSS Software product Assurance Report

- [GSWS-PA-4050] - The ECSS does not define lessons learned report for SW.

### 9.4 SOFTWARE METRICS

This section does not contain requirements.

#### 9.4.1 The Galileo Software Quality Model Framework

This section does not contain requirements.

#### 9.4.2 Galileo Software Quality Model

This section does not contain requirements.

##### 9.4.2.1 Metrics and Target Value w.r.t. SW-DALs

This section does not contain requirements.

##### 9.4.2.2 Metrication Planning

The ECSS defines Metrics to be used, with slight differences from GSWS also, the values presented are just recommendations (see 6.2.5, 7.1.4 ECSS-Q-ST-80C, recommended values are defined in 5.3 ECSS-Q-HB-80-04A).

##### 9.4.2.3 Metric Collecting and Reporting

These requirements are according with ECSS (see 6.2.5 and 7.1.6 ECSS-Q-ST-80C), except [GSWS-PA-4090] which is not specified in ECSS.

##### 9.4.2.4 Procedures, Tools, Resources and Methods

This requirement is according with ECSS (see 6.2.5, 7.1.4 ECSS-Q-ST-80C), except this information in ECSS is placed in Software Product Assurance Plan.

### 9.5 SUB-CONTRACTOR AUDITS

This section does not contain requirements.

#### 9.5.1 Audit Policy

ECSS defines its own requirements for Audits (see 5.2.3 ECSS-Q-ST-80C).

#### 9.5.2 Audit Checklists

The ECSS does not define Audits template.

### 9.6 SOFTWARE PROCESS ASSESSMENT AND IMPROVEMENT

This section does not contain requirements.

#### 9.6.1 Process Assessment

- [GSWS-PA-4130] - The ECSS also defines a S4S assessment model, but it is presented as a suggestion (see 5.7.1 and 5.7.2 ECSS-Q-ST-80C, which redirects to ECSS-Q-HB-80-02), the user may choose its own model, as long as it is in conformance with ISO/IEC 15504.
- [GSWS-PA-4140] - The ECSS does not require the assessment to be made by a higher level contractor or a third party, as long as performed by a competent assessor (see 5.7.2.4 ECSS-Q-ST-80C)
- [GSWS-PA-4150] - Nothing specified in ECSS

```
9.6.2 Process Improvement
```

These requirements are according with ECSS (see 5.7.3 ECSS-Q-ST-80C), except that in ECSS the results of assessment/evidence of improvement shall be Software Process Assessment Records document.

```
10 PROCUREMENT AND REUSE OF SOFTWARE PRODUCTS
```

This section does not contain requirements.

```
10.1 APPLICABILITY
```

These requirements are according with ECSS (see 6.2.7 ECSS-Q-ST-80C). In ECSS there is no specific requirement for algorithm prototypes ([GSWS-PR-4200]), but it shall follow the same procedures as Procured Operational Software.

```
10.2 DEFINITION OF ROLES
```

The ECSS does not specify Purchasing ContractorContractor relationships.

```
10.3 PROCURED SOFTWARE RELATED DOCUMENTS
```

These requirements are partially in conformance with ECSS (see 5.5, 5.6 and 6.2.7 ECSS-Q-ST-80C). There are some aspects specified in GSWS, specified in ECSS and vice versa.

- [GSWS-PR-4220] - ECSS defines the criteria to decide if a tool needs or not to be qualified (see 6 ECSS-Q-HB-80-01A). Regarding the Operational Software, in ECSS, this information is placed in Software Reuse File.
- [GSWS-PR-4230] - Not specified in ECSS.
- [GSWS-PR-4240] - The ECSS does not specify that a list of procured Operational Software should be part of SDD.
- [GSWS-PR-4250] - Not specified in ECSS.

```
10.4 PROCESS PHASES
```

```
10.4.1 Requirement Phase
```

These requirements are partially according with ECSS:

- [GSWS-PR-4260] - Not specified in ECSS.
- [GSWS-PR-4270] - According to ECSS, but more detailed (see 5.5 ECSS-Q-ST-80C).

- [GSWS-PR-4280] - According to ECSS, but more detailed (see 5.5 ECSS-Q-ST-80C).
- [GSWS-PR-4290] - Not specified in ECSS.
- [GSWS-PR-4300] - According with ECSS (see 6.2.7.8 ECSS-Q-ST-80C)

`10.4.2 Procurement Phase`

ESCROW policy and agreement not specified in ECSS.

`10.4.2.1 Short list selection phase`

This phase and its requirements are not specified in ECSS. The ECSS process just requires that the procured software is analysed and the choice justified (see 5.5, 5.6 and 6.2.7 ECSS-Q-ST-80C).

`10.4.2.2 Final selection phase`

This phase and its requirements are not specified in ECSS. The ECSS process just requires that the procured software is analysed and the choice justified (see 5.5, 5.6 and 6.2.7 ECSS-Q-ST-80C).

`10.4.3 Transfer Phase`

This phase is not explicitly specified in ECSS. The ECSS process requires that the procured software is inspected and, for Procured Operational Software, corrective actions and reverse engineering techniques shall be applied to reach the required verification and validation level (see 5.5, 5.6 and 6.2.7 ECSS-Q-ST-80C).

`10.4.4 Qualification phase`

This section does not contain requirements.

`10.4.4.1 Procured Software-Tools`

Requirements not specified by ECSS.

`10.4.4.2 Procured Operational Software`

Requirements not specified by ECSS.

`10.4.5 Maintenance Phase`

This phase is not explicitly specified in ECSS.

`10.5 PROCESS MILESTONES`

This section does not contain requirements.

`10.6 PRODUCT SERVICE HISTORY`

This section is according with ECSS (see 6.2.7 ECSS-Q-ST-80C, 7.6 and Annex B ECSS-Q-HB-80-01A).

# 11 INDEPENDENT SW V&V PROCESS

The ECSS contains the requirements 5.6.2.2 ECSS-E-ST-40C and 6.2.6.13 6.3.5.28 ECSS-Q-ST-80C which state that an Independent organization shall be selected to perform the ISVV, when the risks associated with the project justify the costs involved. Besides these requirements there are no official requirements for ISVV. Instead, ESA has developed a guide, ESA Guide for Independent Software Verification and Validation, which contains the detailed guidelines to perform the ISVV activity.

## 11.1 GENERAL

This section does not contain requirements.

## 11.2 ISVV POLICY

This section is according with ESA ISVV Guide, except the following requirements:

- [GSWS-ISVV-4640] - Not specified
- [GSWS-ISVV-4680] - GSWS template differ from ECSS template

## 11.3 ISVV ACTIVITIES AND MILESTONES

This section is according with ESA ISVV Guide, except the following requirements:

- [GSWS-ISVV-4750] - Not required by ECSS
- [GSWS-ISVV-4780] - According to ECSS, the ISVV activity shall finish before AR (it lasts beyond CDR)

### 11.3.1 Software Specification Activities

This section is according with ESA ISVV Guide, except the following requirements:

- [GSWS-ISVV-4850] - In ECSS this verification is applicable to all software criticality levels
- [GSWS-ISVV-4860] - In ECSS this verification is applicable to all software criticality levels

### 11.3.2 Software Design

This section is according with ESA ISVV Guide, except the following requirement:

- [GSWS-ISVV-4900] - In ECSS this activity is performed during code analysis.

### 11.3.3 Implementation Activities

- [GSWS-ISVV-4920] - In ECSS there is no need to re-execute unit tests, but just to verify their specification and results
- [GSWS-ISVV-4930] - The ECSS does not specify specific verification to partitioning integrity

### 11.3.4 Software Integration and TS-Validation Activities

This section is according with ESA ISVV Guide, note however that the ECSS ISVV is more focused in the ISVV supplier to specify and ru nits own validation tests (Independent Validation) rather than verify the Validation Specification of the costumer.

12 DOCUMENTATION REQUIREMENTS

12.1 GENERAL DOCUMENTATION STANDARDS

Although similar, there are few differences in documentation between GSWS and ECSS (see Table A-1: ECSS-E-ST-40 and ECSS-Q-ST-80 Document requirements list (DRL) and Annexes, which contains each document, if any, template).

12.2 PROJECT ARCHIVE

12.2.1 Requirements Baseline

This section does not contain requirements.

12.2.2 Technical Specification

This section does not contain requirements.

12.2.3 Design Definition File

This section does not contain requirements.

12.2.4 Design Justification File

This section does not contain requirements.

12.2.5 Algorithms File

This section does not contain requirements.

12.2.6 Product Assurance File

This section does not contain requirements.

12.2.7 Maintenance File

This section does not contain requirements.

12.2.8 Management File

This section does not contain requirements.

12.3 CONTRACTUAL APPLICABILITY OF SW DELIVERABLES

This section does not contain requirements.

### 8.1.1 Conclusions

As a generic conclusion, it was found that both standards are compatible, apart of minor differences and what is explained next. GSWS is more specific and contains more restrictions in what approach to be used in certain topics (ex: Design and Coding standards), whereas the ECSS just states that these topics shall be covered, but it is up to the software developer the definition how it will be covered. That means that in ECSS there is more freedom in certain requirements application. As a result, some effort to make the software compatible with GSWS may be required if the application of the ECSS requirement is not according with the correspondent, more specific GSWS requirement.

## 8.2 DO Analysis

In this section it is presented the study performed to assess the compatibility of DO standard with ECSS. The study was performed considering the DO sections and assessing if each section's requirements are/are not compatible with ECSS. The equivalent section in ECSS is pointed out (when applicable) and the differences between the two standards are highlighted. The study is presented below:

### 8.2.1 DO-178

```
2.0 SYSTEM ASPECTS RELATING TO SOFTWARE DEVELOPMENT
```

Introductory section. No requirements here.

```
2.1 System Requirements Allocation to Software
```

This section is according with ECSS (see section 5.2 ECSS-E-ST-40C). Note however that ECSS does not explicitly define the need Certification requirements, although a need for a specific Certification could be considered as a constraint.

```
2.2 Information Flow Between System and Software Life Cycle Processes
```

This section does not contain requirements.

```
2.2.1 Information Flow from System Processes to Software Processes
```

This section is partially according with ECSS (see section 5.2 ECSS-E-ST-40C). In ECSS there is no possibility of having iterations in software caused by the system during software life-cycle, after SRR (the Software System Specification shall be baselined at this stage).

```
2.2.2 Information Flow from Software Processes to System Processes
```

In ECSS, it is referred an interaction from Software Processes to System Processes as sending the results of the software dependability and safety analysis for integration into the system-level (see 6.2.2.7 ECSS-Q-ST-80C). This is the only case of such interaction specified in ECSS.

```
2.2.3 Information Flow between Software Processes and Hardware Processes
```

In ECSS, the hardware it is considered to be part of the system and hence, the considerations described for sections 2.2.1 and 2.2.2 apply here.

```
2.3 System Safety Assessment Process and Software Level
```

This section is according with ECSS (see section 6.2.2 of ECSS-Q-ST-80C).

```
2.3.1 Relationship between Software Errors and Failure Conditions
```

This section is according with ECSS (see section 6.2.2 of ECSS-Q-ST-80C).

```
2.3.2 Failure Condition Categorization
```

There are slight differences in DO and ECSS Failure Condition Category (see the table presented here, against Table 5-1: Severity categories, ECSS-Q-ST-30C). The equivalence of severity categories between both standards is as follows:

Table 1: DO and ECSS severity correspondence

| DO-178 | ECSS |
|---|---|
| Catastrophic, Hazardous | Catastrophic |
| Major | Critical |
| Minor | Major |
| No Safety Effect | Minor or Negligible |

```
2.3.3 Software Level Definition
```

Apart from the differences outlined for section 2.3.2, this section is in accordance with ECSS (see Table D-1 of ECSS-Q-ST-80C, also 5.4.2 ECSS-Q-ST-30C). ECSS is more detailed with the relation of software criticality evaluation in relation with the remaining system. Note also that DO-178 requires proper certification of Software Level definition

```
The applicant should always consider the appropriate certification...
```

, which is not required by ECSS.

```
2.3.4 Software Level Determination
```

This section is according with ECSS (see section 6.2.2 of ECSS-Q-ST-80C).

```
2.4 Architectural Considerations
```

This section is according with ECSS (see sections 6.2.3 and 6.2.2.10 of ECSS-Q-ST-80C).

```
2.4.1 Partitioning
```

This section is according with ECSS (see 7.3 ECSS-Q-HB-80-01A), except that the point e. is not addressed, since it is at hardware level.

```
2.4.2 Multiple-Version Dissimilar Software
```

The ECSS does not contain specific requirements for Multiple-Version Dissimilar Software. As stated in ECSS

```
The use of N-version programming as a fault tolerance mechanism is controversial (see␣
↪[IEEE-Trans-99], [DASC-93]) and it is not listed among the methods recommended by the␣
↪ECSS Standards.
```

(see 6.4 ECSS-Q-HB-80-03A).

```
2.4.3 Safety Monitoring
```

The ECSS does not contain specific requirements for Safety Monitoring. Note however, according with ECSS, safety monitoring is mandatory for manned space systems (see section 6.3.6 ECSS-Q-ST-40C).

```
2.5 Software Considerations in System Life Cycle Processes
```

This section does not contain requirements.

```
2.5.1 Parameter Data Items
```

The Parameter Data Item is equivalent to the ECSS configurable code (see 3.2.5 configurable code ECSS-Q-ST-80). The expression

```
should be addressed
```

should be more specified in this standard. However, it can be considered that the requirements in this section to be in accordance with ECSS (see sections 6.2.6.6 and 6.3.5.7 of ECSS-Q-ST-80C). This will be re-assessed on section 6.6 analysis.

```
2.5.2 User-Modifiable Software
```

There is no concept of User-Modifiable Software/Component in ECSS at the same sense of this standard. More details are given when analysing section 5.2.3 of DO-178.

```
2.5.3 Commercial-Off-The-Shelf Software
```

This section is according with ECSS (see section 6.2.7 of ECSS-Q-ST-80C). The actions to overcome possible deficiencies in COTS will be analysed in the respective sections (12.1.4 and 12.3.4).

```
2.5.4 Option-Selectable Software
```

The guidance for deactivated code will be analysed further. This section is more restrictive rather than the ECSS. In ECSS admits the an

```
accidental activation
```

of deactivated code, but this cannot harm the system (see sections 6.2.6.5 and 6.3.5.30 of ECSS-Q-ST-80C).

`2.5.5 Field-Loadable Software`

The ECSS does not define requirements or considerations for software loading function.

`2.5.6 Software Considerations in System Verification`

This section does not constitute any requirement. Still it is according with ECSS (see 5.2.3.1 ECSS-E-ST-40C).

`2.6 System Considerations in Software Life Cycle Processes`

In ECSS it is found a specific particular case of the described in this section. It concerns with complementary system level validation (see 5.8.3.9 ECSS-E-ST-40C), when there are software requirements which needs to be validated at system level. This activity shall follow the same procedures for the validation of these requirements, so in this case this standard is according with ECSS.

`3.0 SOFTWARE LIFE CYCLE`

This section does not contain requirements

`3.1 Software Life Cycle Processes`

This section is according with ECSS. All these life cycle processes are also defined in ECSS:

- software planning considerations are present in ECSS-M-ST-10C, ECSS-E-ST-40C and ECSS-Q-ST-80C
- software development processes considerations are present in ECSS-E-ST-40C and ECSS-Q-ST-80C
- integral processes considerations are present in ECSS-E-ST-40C (SOFTWARE VERIFICATION PROCESS), ECSS-M-ST-40C (CONFIGURATION MANAGEMENT) and ECSS-Q-ST-80C (SOFTWARE QUALITY ASSURANCE PROCESS), except CERTIFICATION LIAISON PROCESS. Certification is not defined in ECSS.

A more detailed analysis of the compatibility with ECSS for all these processes will be provided in the next sections.

`3.2 Software Life Cycle Definition`

This is a descriptive section, but this content is in line with ECSS (see 4.4.1 ECSS-M-ST-10C descriptive section), apart from that the example provided differs. Note that the definition of a Life Cycle is mandatory in ECSS (see 6.1.1 ECSS-Q-ST-80C).

`3.3 Transition Criteria Between Processes`

This requirement is according with ECSS (see 6.2.6.2 ECSS-Q-ST-80C). There is no specific reference to partial inputs, but it is implicit that a partial input as a complete input should provide enough and correct information to proceed the next process.

`4.0 SOFTWARE PLANNING PROCESS`

This section does not contain requirements. As referred in Annex A, the tables presented do not provide complete information (the body of the section should be used as a complete description of the standard).

`4.1 Software Planning Process Objectives`

This section is according with ECSS (see 5.3.2.1 ECSS-E-ST-40C and 5.1 ECSS-M-ST-10C).

`4.2 Software Planning Process Activities`

This section is according with ECSS (see 5.3 ECSS-E-ST-40C). The points f, h, i and j are not referred in ECSS to be part of Software Planning Process Activities. Regarding point l, the supplier oversight is performed by Joint reviews.

`4.3 Software Plans`

The Plan for Software Aspects of Certification is not defined in ECSS. The Software Development Plan, Software Verification Plan, Software Configuration Management Plan and Software Quality Assurance Plan (equivalent to ECSS Software product assurance plan) are defined in ECSS with the same objectives as this standard. The content of these four documents against ECSS will be analysed in section 11. For the remaining content of this section, it is according with ECSS (see each of the above four documents content in ECSS-E-ST-40C, ECSS-M-ST-40 and ECSS-Q-ST-80C). Regarding point c, the ECSS does not contain specific requirements for certified products.

`4.4 Software Life Cycle Environment Planning`

This section does not contain requirements.

`4.4.1 Software Development Environment`

This section is partially according with ECSS (see 5.6.2 ECSS-Q-ST-80C). The points from b to f are not explicitly required in ECSS.

`4.4.2 Language and Compiler Considerations`

This section is according with ECSS (see 5.8.3.5 ECSS-E-ST-40C and 6.2.3.4/6.3.5 ECSS-Q-ST-80C).

`4.4.3 Software Test Environment`

The ECSS does not specify that the emulator or simulator shall be qualified nor that differences between emulator/simulator should be considered. However, according to ECSS, prior delivering the product, it shall be validated under conditions similar to the application environment (see 6.3.5.26 ECSS-Q-ST-80C). This means that, although simulators may be used in ECSS, the software shall be also validated in the hardware platform.

`4.5 Software Development Standards`

The ECSS defines the need for definition of Software Development Standards (see 5.2.4.5 and 5.3.2.1 ECSS-E-ST-40C). However, the ECSS does not provide any norms to define these standards as opposed to this standard.

`4.6 Review of the Software Planning Process`

This section is according with ECSS (see 5.1 ECSS-M-ST-10C). However, the ECSS does not provide any restriction regarding the content of the Planning Process output.

`5.0 SOFTWARE DEVELOPMENT PROCESSES`

This is an introductory section, which is according with ECSS, except that the text starting at

`Each software development process may produce derived requirements.`

is not compatible with ECSS. In ECSS, the high-level requirements and low-level requirements are baselined respectively at SRR and PDR. They cannot change in later phases of the project, which means that, contrary to this standard, no requirement can be derived from later phases of software development.

`5.1 Software Requirements Process`

This is an introductory section. This is according with ECSS (see 5.2 ECSS-E-ST-40C). Note that regarding requirements, the following correspondence between DO-178 and ECSS applies:

Table 2: DO and ECSS requirements definitions correspondence

| DO-178 | ECSS |
|---|---|
| System Requirements | *SYSTEM* (no specific definition) |
| High-level requirements | System requirements |

`5.1.1 Software Requirements Process Objectives`

This section is according with ECSS (see 5.2.2.1 ECSS-E-ST-40C).

`5.1.2 Software Requirements Process Activities`

This section is partially according with ECSS. See the following comments:

a. ECSS does not assign the responsibility to analyse the system inputs.

b. ECSS does not assign the responsibility to analyse the system inputs.

c. ECSS does not specify the need to have traceability between high-level requirements and system.

d. This is according with ECSS (see 5.2.2.1 ECSS-E-ST-40C).

e. This is according with ECSS (see 5.2.4.5 ECSS-E-ST-40C and 5.8.3.1 ECSS-Q-ST-80C).

f. This is according with ECSS (see 7.2.1.2 ECSS-Q-ST-80C). Note that referring that the requirements should be expressed in quantitative terms is equivalent to referring that they are unambiguous. Tolerances for requirements is not explicitly specified in ECSS. However, some requirements needs tolerance to be correct, hence ECSS implicitly covers this topic.

g. ECSS does not forbid the description of design or verification detail.

h. ECSS does not require explicit justification for derived high-level requirements.

i. ECSS does not require that the derived requirements to be provided to the system level.

j. ECSS does not specify requirements for data items.

```
5.2 Software Design Process
```

This section is in accordance with ECSS (see 5.4 and 5.5.2 ECSS-E-ST-40C).

```
5.2.1 Software Design Process Objectives
```

This section is partially according with ECSS (see 5.4 and 5.5.2 ECSS-E-ST-40C). The differences between DO-178 and ECSS regarding this phase are outlined below:

- in ECSS the architecture is resulted from low level requirements, whereas in DO-178, the architecture is resulted from high-level requirements.
- in DO-178 there is no division into Architectural and Detail Design phase as in ECSS. Both of these phases belong to Software Design Process.

Regarding point b., ECSS admits the existence of derived low-level requirements as

```
the software requirements that are not traced to the system requirements allocated to␣
↪software are justified;
```

(see 5.8.3.2 ECSS-E-ST-40C).

```
5.2.2 Software Design Process Activities
```

This section is partially according with ECSS. See below:

a. This is according with ECSS (see 5.8.3 ECSS-E-ST-40C and 6.3 ECSS-Q-ST-80C).

b. This is according with ECSS (see 5.8.3.2 ECSS-E-ST-40C). In ECSS it is implicit that the low-level requirements should not compromise the high-level requirements. This would result that there would be inconsistent low-level requirements.

c. This is according with ECSS, which states that dependability and safety analysis at each software development milestone shall be updated and the results provided to the system level (see 6.2.2 ECSS-Q-ST-80C).

d. This is according with ECSS (see 5.4.3.5 and 5.5.2.2 ECSS-E-ST-40C).

e. This is not specified in ECSS. If not implemented in ECSS software, it may be incompatible with DO-178.

f. This according with ECSS (see 5.8.3.3 ECSS-E-ST-40C).

g. Not specified in ECSS. ECSS assumes that the inputs to this phase are correct.

```
5.2.3 Designing for User-Modifiable Software
```

User-Modifiable Software not defined in ECSS at the same sense of this standard. ECSS does not allow software changes by the user and any changes in general without following the regression testing (see 6.3.5.15 ECSS-Q-ST-80C and 5.10.2.1 ECSS-E-ST-40C). These changes can occur at

Maintenance (5.10 ECSS-E-ST-40C) or in flight modification (see 5.2.2.1 and 5.4.2.2 ECSS-E-ST-40C), but they shall follow the regression approach.

`5.2.4 Designing for Deactivated Code`

This section is partially in accordance with ECSS. In ECSS admits the an

`accidental activation`

of deactivated code, but this cannot harm the system (see sections 6.2.6.5 and 6.3.5.30 of ECSS-Q-ST-80C). Regarding point c., the ECSS does not contain an explicit similar requirement. However, looking at the definition of deactivated code in ECSS, it can be inferred that it is in accordance with the point c.

`5.3 Software Coding Process`

This section does not contain requirements.

`5.3.1 Software Coding Process Objectives`

This section in not in accordance with ECSS, since in ECSS, the source code is produced from detailed design (see 5.5.3.1 ECSS-E-ST-40C).

`5.3.2 Software Coding Process Activities`

This section is partially according with ECSS. See below:

a. This is according with ECSS (see 5.5.3.1 and 5.8.3.5 ECSS-E-ST-40C).

b. This is according with ECSS (see 5.8.3.5 ECSS-E-ST-40C).

c. Not specified in ECSS. ECSS assumes that the inputs to this phase are correct.

d. This is according with ECSS (see 5.3.2.4 ECSS-E-ST-40C and 6.2.8 ECSS-Q-ST-80C).

`5.4 Integration Process`

This section does not contain requirements. The term integration has a different meaning in this standard from the ECSS, as depicted in the below table:

Table 3: DO Integration definition ECSS correspondence

| DO-178 | ECSS |
|---|---|
| Integration | Installation |

`5.4.1 Integration Process Objectives`

This section is according with ECSS (see 5.7.2 ECSS-E-ST-40C).

`5.4.2 Integration Process Activities`

This section is partially according with ECSS. See below:

a. This is according with ECSS (see 5.7.2.4 ECSS-E-ST-40C).

b. This is according with ECSS (see 5.7.2.1 ECSS-E-ST-40C).

c. This is according with ECSS (see 5.7.2.1 and 5.7.2.3 ECSS-E-ST-40C). Note: ECSS does not separate the integration and loading activities.

d. Not specified in ECSS. ECSS assumes that the inputs to this phase are correct.

e. This is implicitly according with ECSS (see 6.2.3.4, 6.2.3.5 and 6.3.5 ECSS-Q-ST-80C). When the software or target/compiling environment is changed, the software shall undergo the regression approach

f. ECSS does not specify the need for the points 1 and 3. However, as depicted in point e. the patched software shall undergo the regression approach, meaning that the point 2 is met.

```
5.5 Software Development Process Traceability
```

This section is partially according with ECSS. See below:

a. ECSS does not require traceability between high-level and system requirements.

b. This is according with ECSS (see 5.8.3.2 ECSS-E-ST-40C).

c. ECSS requires that source code to be traced to units (see 5.8.3.5 ECSS-E-ST-40C) and not to the requirements. However, following ECSS traceability flow: low-level requirements to design to to source code, it is possible to derive the traceability of source code to low-level requirements.

```
6.0 SOFTWARE VERIFICATION PROCESS
```

This section does not contain requirements.

```
6.1 Purpose of Software Verification
```

This section is according with ECSS (see 5.8 ECSS-E-ST-40C and 6.2.6 ECSS-Q-ST-80C). Note that regarding point b., ECSS does not specify more than 1 level for low-level requirements.

```
6.2 Overview of Software Verification Process Activities
```

This section is partially according with ECSS. See below:

a. ECSS does not allow that the airborne software to be not identical to the tested software.

b. This is according with ECSS (see 5.6.3.1 and 5.6.4.1 ECSS-E-ST-40C).

c. Not specified in ECSS. ECSS does not specify that errors from one phase to be feedbacked to previous phases.

d. This is according with ECSS (see 6.3.5.15 ECSS-Q-ST-80C).

e. This is according with ECSS (see 6.3.5.19 ECSS-Q-ST-80C). As a side note, it seems that there is a gap in both standards in the sense that independence of alternative methods (ex: inspection, analysis) for requirements verification is not covered.

```
6.3 Software Reviews and Analyses
```

This section is according with ECSS (see 5.8.2). Note that the terms inspection and analysis definitions are according with ECSS (see ECSS-S-ST-00-01C).

`6.3.1 Reviews and Analyses of High-Level Requirements`

This section is according with ECSS (see 5.8.3.1 and 5.8.3.13 ECSS-E-ST-40C). Point e. can be implicitly derived from ECSS (see 5.3.3.1 ECSS-E-ST-40C and 6.2.1 ECSS-Q-ST-80C). Point f. is not applicable (see analysis of DO-178 5.1.2).

`6.3.2 Reviews and Analyses of Low-Level Requirements`

This section is according with ECSS (see 5.8.3.2 and 5.8.3.13 ECSS-E-ST-40C). Point e. can be implicitly derived from ECSS (see 5.3.3.1 ECSS-E-ST-40C and 6.2.1 ECSS-Q-ST-80C).

`6.3.3 Reviews and Analyses of Software Architecture`

This section is according with ECSS (see 5.8.3.3, 5.8.3.4 ECSS-E-ST-40C and 6.3.3.4 ECSS-Q-ST-80C). Some remarks:

a. In ECSS, the architecture is verified against the low-level requirements.

b. In ECSS the protection mechanisms are referred at code level (see 5.8.3.5 ECSS-E-ST-40C).

`6.3.4 Reviews and Analyses of Source Code`

This section is according with ECSS (see 5.8.3.5, 5.8.3.11 and 5.8.3.12 ECSS-E-ST-40C).

`6.3.5 Reviews and Analyses of the Outputs of the Integration Process`

This section is according with ECSS (see 6.3.6 ECSS-Q-ST-80C).

`6.4 Software Testing and corresponding subsections`

This overall approach is different ECSS (see against 5.5.3.2, 5.5.4, 5.6 and 5.8.3 ECSS-E-ST-40C). The following list will outline the main observations taken from analysing this subsection (from 6.4.1 to 6.4.4):

- Contrary to ECSS, DO-178 admits some requirements to not be tested in the target computer environment (case of Software integration testing and Low-level testing), although this is still the preferred solution according with 6.4.1.
- As for 6.4.2, 6.4.2.1 and 6.4.2.2, also ECSS specifies the need for Nominal and Robustness tests.
- ECSS approach for testing is to start by Unit Tests with the goal to test all the software Units. Then, integration tests are developed and exercised in the software to analyse the interactions between software components. Note that regarding integration tests, ECSS does not impose any goal to be achieved. The level of integration test to perform on the software depends on what was agreed between the supplier and customer and is documented in the Integration Plan. Finally, the validation tests shall be specified and they shall cover all low-level requirements (ECSS Technical specification) and high-level requirements (ECSS Requirements baseline). This means that in ECSS the tests follow the

software development cycle, by starting with the less complex type of tests (unit) and following to more complex type of tests (integration and then validation). This philosophy assures that the next step of software can only start when the previous is successful (integration testing starts when unit test is successful and validation starts when integration is met). ECSS states that both requirement and code coverage shall be 100 %, but does not provide guidelines to achieve this goal.

The DO-178 approach is to define Hardware/Software Integration Testing, Software/Software Integration Testing for testing high-level level requirements and achieve code coverage (see 6.4.3). These tests are equivalent to the ECSS validation/integration (although in ECSS integration do not aim to cover requirements) tests. Low-level testing (see also 6.4.3) targets to verify the implementation of low-level requirements, by testing each software Unit (similar to ECSS unit tests, although they do not aim to cover requirements). Although not preferred, DO-178 offers the possibility to substitute Low-level requirement coverage by high-level test(s). DO-178 emphasis is on the correct coverage of the requirements by the tests from start of test specification. This standard also states that both requirement and code coverage shall be 100 % and provides guidelines to achieve that goal (see 6.4.4 and subsections).

Note that since the final goal is the same, it is possible to translate the work done in test specifications from ECSS to DO-178, as explained following. The validation and integration tests are equivalent to Hardware/Software Integration Testing or Software/Software Integration Testing (ECSS validation tests may be seen as a more complex ECSS integration tests). The traceability of integration tests to requirements has to be performed. The unit tests are equivalent to Low-level testing. The traceability of unit tests to low-level requirements has to be performed.

```
6.4.5 Reviews and Analyses of Test Cases, Procedures, and Results
```

This section is according with ECSS (see 5.8.3 ECSS-E-ST-40C).

```
6.5 Software Verification Process Traceability
```

This section is partially according with ECSS (see 5.8.3, Software [unit/integration] test plan and Software validation specification templates in ECSS-E-ST-40C). This point c is not required by ECSS.

```
6.6 Verification of Parameter Data Items
```

This section is partially according with ECSS, being ECSS more restrictive. ECSS does not offer the possibility to test parameter data items separately (see 6.2.6.6, 6.3.5.7 and 6.3.5.31 ECSS-Q-ST-80C). Hence, software developed under ECSS, will met this section.

```
7.0 SOFTWARE CONFIGURATION MANAGEMENT PROCESS
```

This section is an introductory section and does not contain requirements. Its content is according with ECSS description for Software Configuration Management Process (see 4.1 ECSS-M-ST-40C).

```
7.1 Software Configuration Management Process Objectives
```

This section is a descriptive section and does not contain requirements. Its content is according with ECSS Software Configuration Management Process Objectives (see 4.3 ECSS-M-ST-40C).

### 7.2 Software Configuration Management Process Activities

This section is a descriptive section and does not contain requirements. Its content is according with ECSS Software Configuration Management Process Activities (see 4.1.1 and 4.1.2 ECSS-M-ST-40C).

#### 7.2.1 Configuration Identification

This section is partially according with ECSS (see 5.3.1 Configuration management plan - DRD and ECSS-M-ST-40C). Points a. and b. are according with ECSS. Points c. and d. are implicit in ECSS. Point e. is not covered by ECSS.

#### 7.2.2 Baselines and Traceability

This section is according with ECSS (see 5.3.1 ECSS-M-ST-40C).

#### 7.2.3 Problem Reporting, Tracking, and Corrective Action

This section is in accordance with ECSS, although the Problem Report content differ slightly (see 5.2.5.2 ECSS-Q-ST-80C, 5.9.2.1 and 5.10.2.1 ECSS-E-ST-40C). Problem Report content will be analyzed further in 11.17.

#### 7.2.4 Change Control

This section is according with ECSS (see 5.3.1.5, 5.3.2 ECSS-M-ST-40C and 6.2.4 ECSS-Q-ST-80C). Point b. is implicit (5.3.1.5 ECSS-M-ST-40C) in ECSS, since any change will cause the change of Version and/or revision number.

#### 7.2.5 Change Review

This section is according with ECSS (see 5.3.2.4 ECSS-M-ST-40C).

#### 7.2.6 Configuration Status Accounting

This section is according with ECSS (see 5.3.3 ECSS-M-ST-40C).

#### 7.2.7 Archive, Retrieval, and Release

This section is according with ECSS (see 5.3.7 ECSS-M-ST-40C, the need for back-ups is referenced on the descriptive section 4.3.1).

### 7.3 Data Control Categories

ECSS does not define Software Control Categories for which the some activities of Configuration Management does not apply. However, ECSS allow the Configuration Management Standards to be tailored, according with project needs (see section 1 ECSS-M-ST-40C).

### 7.4 Software Load Control

The software Load Control is not defined in ECSS. Note however that the point b. is automatically assured by ECSS (see 6.3.5.26 ECSS-Q-ST-80C).

### 7.5 Software Life Cycle Environment Control

This section is partially according with ECSS (see 5.3.1.3 ECSS-M-ST-40C, 5.3.2.1 ECSS-E-ST-40C and 7.3.5 ECSS-Q-ST-80C). ECSS states that the description development environment shall be under configuration control. ECSS does not define specific Configuration Management Process for (qualified) environment tools.

## 8.0 SOFTWARE QUALITY ASSURANCE PROCESS

This section does not contain requirements. It is an introductory section, which is according with ECSS (see 4.1 ECSS-Q-ST-80C).

### 8.1 Software Quality Assurance Process Objectives

This is a descriptive section and does not contain requirements. The Software Quality Assurance Process Objectives for this standard are in line with the ECSS (see 4.1 and, in general, all ECSS-Q-ST-80C document). A note that the ECSS equivalent to conformity review is as follows:

Table 4: DO and ECSS conformity review correspondence

| DO-178 | ECSS |
|---|---|
| Conformity review | Acceptance review |

### 8.2 Software Quality Assurance Process Activities

The points below analyse the compatibility of this section with ECSS:

a. This is implicitly according with ECSS (see 6.1 ECSS-Q-ST-80C).

b. This is according with ECSS (see 5.2.1.5 and 6.2.6 ECSS-Q-ST-80C), part that the verification of compliance is made against ECSS.

c. This is according with ECSS (see 5.8.2.1 ECSS-E-ST-40C and 6.2.6 ECSS-Q-ST-80C).

d. This list specify with more detail the Audit activities rather than in ECSS (see 5.2.3 ECSS-Q-ST-80C). However, it is implicit that ECSS shall also perform these listed activities.

e. This is according with ECSS (see 6.2.6 ECSS-Q-ST-80C).

f. This is partially according with ECSS (see 6.2.4 and 6.3 ECSS-Q-ST-80C). ECSS states that configuration management of life cycle data shall be controlled, but there is no definition of control categories as in this standard.

g. This is partially according with ECSS (see 5.3.4.5 ECSS-E-ST-40C). In ECSS the software is delivered to the customer, which shall install and run the acceptance tests. Upon successful completion of this activity, the software is considered accepted.

h. This is according with ECSS (see 5.2.2 ECSS-Q-ST-80C).

i. This is according with ECSS (see 5.2.1 and 6.2.6 ECSS-Q-ST-80C).

### 8.3 Software Conformity Review

This section is according with ECSS (see 6.3.6 ECSS-Q-ST-80C). The point i. is not specified in ECSS in terms of certification, but in ECSS the re-used software shall be complemented in order to met the project qualification needs (see 6.2.7 ECSS-Q-ST-80C).

```
9.0 CERTIFICATION LIAISON PROCESS
```

Certification Liaison Process is not defined in ECSS. This means that a software developed under ECSS shall undergo this certification to be compliant with DO-178.

```
10.0 OVERVIEW OF CERTIFICATION PROCESS
```

Certification not specified in ECSS.

```
11.0 SOFTWARE LIFE CYCLE DATA
```

The points below analyse the compatibility of this section with ECSS:

a. This is implicitly according with ECSS (see 5.8.3 ECSS-E-ST-40C). No generic characteristics for life cycle data are defined, but the correctness of the software outputs imply that the listed characteristics are met.

b. This is according with ECSS (see 5.3.7 ECSS-M-ST-40C).

c. No configuration control levels is specified in ECSS. All life cycle data shall follow the same configuration management procedures.

d. To be analysed against ECSS DRD - Document Requirements Definition in the following subsections. The following subsections will be analysed against the ECSS DRD

```
11.1 Plan for Software Aspects of Certification
```

This document is not specified in ECSS. However, in ECSS some of these topics are covered by other documents, as following:

a. Covered by ECSS (for example see SSS Annex B ECSS-E-ST-40C).

b. Covered by ECSS (for example see SSS Annex B ECSS-E-ST-40C). Note that it is not explicitly specified that the this introductory section shall emphasise safety and partitioning concepts.

c. Certification not covered by ECSS. Software level and safety assessment are covered by Software dependability and safety analysis report - Criticality classification of software components (see 6.2.2 ECSS-Q-ST-80C).

d. Covered by ECSS (see SDP Annex O ECSS-E-ST-40C), except certification liaison.

e. Covered by ECSS (see SDP Annex O ECSS-E-ST-40C), except information regarding certification.

f. Covered by ECSS (see SDP Annex O ECSS-E-ST-40C).

g. Covered by ECSS (in ECSS standard these additional considerations shall be part of SDP, Annex O ECSS-E-ST-40C).

h. Covered by ECSS (see SDP Annex O ECSS-E-ST-40C).

### 11.2 Software Development Plan

The SDP is defined as a project deliverable in ECSS (see Annex O ECSS-E-ST-40C). Below is the compatibility assessment between DO-178 and ECSS regarding the SDP topics:

a. Covered by ECSS (see Annex O ECSS-E-ST-40C), except references to the standards for previously developed software.

b. Covered by ECSS (see Annex O ECSS-E-ST-40C).

c. Covered by ECSS (see Annex O ECSS-E-ST-40C).

### 11.3 Software Verification Plan

The SVP is defined as a project deliverable in ECSS (see Annex I ECSS-E-ST-40C). Below is the compatibility assessment between DO-178 and ECSS regarding the SVP topics:

a. Covered by ECSS (see Annex I ECSS-E-ST-40C).

b. Covered by ECSS (see Annex I ECSS-E-ST-40C).

c. Covered by ECSS (see Annex I ECSS-E-ST-40C).

d. Covered by ECSS (see Annex I ECSS-E-ST-40C).

e. Covered by ECSS (see Annex I ECSS-E-ST-40C).

f. ECSS does not specify the need for an explicit section covering Partitioning Considerations.

g. In ECSS this information is not in SVP, but in SDP (see Annex O ECSS-E-ST-40C).

h. This is implicitly covered by ECSS (see Annex I ECSS-E-ST-40C taking into account 6.3.5 ECSS-Q-ST-80C). Since the modified areas should be subjected to regression approach (i.e reverified), they shall follow the methods described for the verification of software.

i. In ECSS this information is not in SVP, but in SRF (see Annex N ECSS-E-ST-40C).

j. Not explicitly specified by ECSS.

### 11.4 Software Configuration Management Plan

The SCM is defined as a project deliverable in ECSS (see Annex A ECSS-M-ST-40C). Below is the compatibility assessment between DO-178 and ECSS regarding the SCM topics:

a. Covered by ECSS (see Annex A ECSS-M-ST-40C).

b. Covered by ECSS (see Annex A ECSS-M-ST-40C), apart that ECSS does not define Configuration Control levels.

c. Covered by ECSS (see Annex A ECSS-M-ST-40C).

d. Covered by ECSS (see Annex A ECSS-M-ST-40C). The later subsections will analyse if ECSS also defines each of the SCM data items as in DO-178.

e. Covered by ECSS (see Annex A ECSS-M-ST-40C).

### 11.5 Software Quality Assurance Plan

The SQA is defined as a project deliverable in ECSS, but designated as Software Product Assurance Plan (see Annex B ECSS-Q-ST-80C). Below is the compatibility assessment between DO-178 and ECSS regarding the SQA topics:

a. Covered by ECSS (see Annex B ECSS-Q-ST-80C).

b. Covered by ECSS (see Annex B ECSS-Q-ST-80C).

c. Covered by ECSS (see Annex B ECSS-Q-ST-80C).

d. This is implicit in ECSS (see 5.2 ECSS-Q-ST-80C). The SQA activities shall start along with the project activities.

e. Covered by ECSS (see Annex B ECSS-Q-ST-80C).

f. Covered by ECSS (see Annex B ECSS-Q-ST-80C).

g. Covered by ECSS (see Annex B ECSS-Q-ST-80C).

### 11.6 Software Requirements Standards

In ECSS the Software Requirements Standards to be used in the project are to be placed in SDP (see Annex O ECSS-E-ST-40C) and SPAP (see Annex B ECSS-Q-ST-80C). The points referred in this section are not explicitly outlined in ECSS.

### 11.7 Software Design Standards

In ECSS the Software Design Standards to be used in the project are to be placed in SDP (see Annex O ECSS-E-ST-40C), SPAP (see Annex B ECSS-Q-ST-80C) and SDD (see Annex F ECSS-E-ST-40C). The points a., b. (see Annex F ECSS-E-ST-40C) and f. (see Annex B ECSS-Q-ST-80C) are specified in ECSS. The points c. to e. are not specified in ECSS.

### 11.8 Software Code Standards

In ECSS the Software Code Standards to be used in the project are to be placed in SDP (see Annex O ECSS-E-ST-40C) and SPAP (see Annex B ECSS-Q-ST-80C). The point a. is specified in ECSS (see Annex O ECSS-E-ST-40C). The points b. to e. are not specified in ECSS.

### 11.9 Software Requirements Data

The Software Requirements Data is defined as a project deliverable in ECSS (see Annex B and Annex C ECSS-E-ST-40C). Below is the compatibility assessment between DO-178 and ECSS regarding the Software Requirements Data topics:

a. Covered by ECSS (see Annex B ECSS-E-ST-40C).

b. Covered by ECSS (see Annex B ECSS-E-ST-40C).

c. Covered by ECSS (see Annex B ECSS-E-ST-40C).

d. Covered by ECSS (see Annex B ECSS-E-ST-40C).

e. Covered by ECSS (see Annex B ECSS-E-ST-40C).

f. Covered by ECSS (see Annex B and Annex C ECSS-E-ST-40C).

g. Covered by ECSS (see Annex B ECSS-E-ST-40C).

h. Not explicitly specified by ECSS.

11.10 Design Description

The Design Description is defined as a project deliverable in ECSS (see Annex D, Annex E and Annex F ECSS-E-ST-40C). Below is the compatibility assessment between DO-178 and ECSS regarding the Software Requirements Data topics:

a. Covered by ECSS (see Annex D ECSS-E-ST-40C). As stated in ECSS

```
SRS shall provide where applicable the link between the requirements and the system␣
↪states and modes.
```

b. Covered by ECSS (see Annex F ECSS-E-ST-40C).

c. Covered by ECSS (see Annex E and Annex F ECSS-E-ST-40C).

d. Covered by ECSS (see Annex F ECSS-E-ST-40C).

e. Covered by ECSS (see Annex D ECSS-E-ST-40C).

f. Covered by ECSS (see Annex F ECSS-E-ST-40C).

g. Covered by ECSS (see Annex F ECSS-E-ST-40C).

h. Not specified in ECSS.

i. Covered by ECSS (see Annex F ECSS-E-ST-40C).

j. Covered by ECSS (see Annex D ECSS-E-ST-40C). It is explicit from requirements traceability.

k. Not specified in ECSS Design Description. The deactivated code is verified in the ECSS Software Verification Report (see Annex M ECSS-E-ST-40C).

l. Covered by ECSS (see Annex F ECSS-E-ST-40C).

11.11 Source Code

The requirement in this section is implicit in ECSS (see Annex E ECSS-M-ST-40C).

11.12 Executable Object Code

This section does not contain requirements.

11.13 Software Verification Cases and Procedures

The Software Verification Cases and Procedures can be found in the ECSS documents Software validation plan (see Annex J ECSS-E-ST-40C), Software [unit/integration] test plan (see Annex K ECSS-E-ST-40C) and Software validation specification (see Annex L ECSS-E-ST-40C). The content in this section is according with ECSS (see the ECSS Annexes outlined above).

11.14 Software Verification Results

The Software Verification Results is defined as a project deliverable in ECSS (see Software verification report - Annex M ECSS-E-ST-40C). The content of this section is according with ECSS (see the ECSS Annex outlined above).

### 11.15 Software Life Cycle Environment Configuration Index

The Software Life Cycle Environment Configuration Index information is on the ECSS Software Configuration File (see Annex E ECSS-M-ST-40C). The content of this section is according with ECSS (see the ECSS Annex outlined above).

### 11.16 Software Configuration Index

The Software Configuration Index information is on the ECSS Software Configuration File (see Annex E ECSS-M-ST-40C). The content of this section is according with ECSS (see the ECSS Annex outlined above).

### 11.17 Problem Reports

ECSS does not define a template (DRD) for the software problem reports format. However, the information which shall be in a problem report is defined (see 5.2.5 ECSS-Q-ST-80C) and is according with this standard.

### 11.18 Software Configuration Management Records

ECSS does not define Software Configuration Management Records document. However some contents for the Software Configuration Management Records are defined in ECSS (see ECSS-M-ST-40C).

### 11.19 Software Quality Assurance Records

The Software Quality Assurance Records information is present in ECSS in form of Software Product Assurance Reports (see 5.2.2 ECSS-Q-ST-80C), including the Software Product Assurance Milestone Reports (see Annex C ECSS-Q-ST-80C).

### 11.20 Software Accomplishment Summary

This document is not specified in ECSS.

### 11.21 Trace Data

Below is the compatibility assessment between DO-178 and ECSS regarding the Trace Data:

a. Not specified in ECSS.

b. Specified in ECSS (see Annex D and Annex E ECSS-E-ST-40C).

c. Not specified in ECSS.

d. Specified in ECSS (see Annex L ECSS-E-ST-40C).

e. Specified in ECSS (see Annex K and Annex L ECSS-E-ST-40C).

f. Not specified in ECSS.

Note that ECSS defines other traceabilities which can be used to derive the traceabilities not specified.

```
11.22 Parameter Data Item File
```

This is not in accordance with ECSS. ECSS only requires that each configuration to be tested (see 6.3.5 ECSS-Q-ST-80C). In addition, in ECSS, the Parameter Data Item File is not considered a separated Software Life Cycle Data, but as part of the code.

```
12.0 ADDITIONAL CONSIDERATIONS
```

This is an introductory section. The requirement in this section

```
The use of additional considerations and the proposed impact on the guidance provided in␣
↪the other sections of this document should be agreed on a case-by-case basis with the␣
↪certification authorities.
```

cannot be linked to ECSS, since no certification is defined there.

```
12.1 Use of Previously Developed Software
```

The requirement in this section is according with ECSS (see 6.2.7 ECSS-Q-ST-80C).

```
12.1.1 Modifications to Previously Developed Software
```

This section is according with ECSS (see 6.2.7 ECSS-Q-ST-80C). Analysis against section 12.1.4 of this standard is done below.

```
12.1.2 Change of Aircraft Installation
```

This section is according with ECSS (see 6.2.3, 6.3.5 and 6.2.7 ECSS-Q-ST-80C).

```
12.1.3 Change of Application or Development Environment
```

Although this section is more detailed, it is according with with ECSS (see 5.6, 6.2.3 and 6.3.5 ECSS-Q-ST-80C). The points not specified in ECSS are implicit that they shall be met.

```
12.1.4 Upgrading a Development Baseline
```

This section is according with ECSS (see 6.2.7 ECSS-Q-ST-80C), apart that ECSS does not specify certification aspects.

```
12.1.5 Software Configuration Management Considerations
```

Not specified in ECSS.

```
12.1.6 Software Quality Assurance Considerations
```

This section is according with ECSS (see 6.2.7 ECSS-Q-ST-80C), although ECSS just requires that the changes to be documented in the reuse file.

```
12.2 Tool Qualification
```

No text in this section.

#### 12.2.1 Determining if Tool Qualification is Needed

This is according with ECSS (see 6 ECSS-Q-HB-80-01A). ECSS text is based on this standard.

#### 12.2.2 Determining the Tool Qualification Level

This is according with ECSS (see 6 ECSS-Q-HB-80-01A). ECSS text is based on this standard.

#### 12.2.3 Tool Qualification Process

To be analysed in DO-330 section.

### 12.3 Alternative Methods

This is according with ECSS (see 5.6 ECSS-Q-ST-80C), in the sense that any method to be used shall be evaluated regarding its suitability. In ECSS there is no concept of

Alternative Method

Note that, also, the methods in ECSS are agreed by the customer (no certification authority involved, as in this standard).

#### 12.3.1 Exhaustive Input Testing

This is according with ECSS (see 6.3.5.29 ECSS-Q-ST-80C).

#### 12.3.2 Considerations for Multiple-Version Dissimilar Software Verification

Multiple-Version Dissimilar Software is not specified in ECSS. The subsections of 12.3.2 are skipped.

#### 12.3.3 Software Reliability Models

This section does not contain requirements. Also in ECSS, ECSS advises to not use reliability models for space software (see ECSS-Q-HB-80-03A).

#### 12.3.4 Product Service History

This section is according with ECSS (see 6.2.7.8 ECSS-Q-ST-80C and 7.6 ECSS-Q-HB-80-01A). Note that more details are in ECSS HandBook.

##### 12.3.4.1 Relevance of Service History

This section is partially according with ECSS (see 7.6 and Annex B ECSS-Q-HB-80-01A). The points a. and f. are not pointed out by ECSS.

##### 12.3.4.2 Sufficiency of Accumulated Service History

This section is according with ECSS (see 7.6 ECSS-Q-HB-80-01A). ECSS is more complete in what is required to assess the Sufficiency of the accumulated Service History.

```
12.3.4.3 Collection, Reporting, and Analysis of Problems Found During Service History
```

This section is partially according with ECSS (see 7.6 ECSS-Q-HB-80-01A). The points a.1.vi., a.2. and a.3. are not specified in ECSS. The point c. is implicit in ECSS (these safety-related problems are the major problems).

```
12.3.4.4 Service History Information to be Included in the Plan for Software Aspects of␣
↪Certification
```

The Plan for Software Aspects of Certification is not specified in ECSS. In ECSS the rationale for the applicability for the parameters of the Product Service History of a software product to be used shall be defined in the Software Reuse File (see 7.6 ECSS-Q-HB-80-01A and 6.2.7.8 ECSS-Q-ST-80C). Below is the compatibility assessment between DO-178 and ECSS regarding the points of this section.

a. Partially covered by ECSS
b. Covered by ECSS
c. Not covered by ECSS
d. Not covered by ECSS
e. Covered by ECSS
f. Implicitly covered by ECSS
g. Covered by ECSS
h. Implicitly covered by ECSS.

#### 8.2.1.1 Conclusions

Both standards are compatible, except for the main differences pointed out below:

- Life-cycle differ slightly between DO and ECSS:
  - In ECSS the low-level requirements definition is separate from Design phase.
  - In ECSS earlier phases in the project cannot be reentred (once a document is baselined, it shall not change).
  - in ECSS no derived requirements can come from later phases after low-level requirements definition.
- Verification approach: the DO focuses on testing the integrated product first and then testing by unit, when coverage cannot be achieved by a validation test, ECSS starts testing at unit level and going-up as soon as the units begin to integrate and ending in the validation tests (following the product incremental development). Note also in DO every test needs to be traced for a requirement, whereas in ECSS this is only required for the validation tests.
- In ECSS there is no definition of Configuration Control levels
- In ECSS certification is not considered

- DO-178 covers (has requirements) for some software-specific topics not covered by ECSS (ex: Multiple-Version Dissimilar Software)

The differences in Life-cycle and verification approach does not have implications in the final product of a DO software against ECSS software (these are different paths to achieve the same goal). Hence for a qualification of a ECSS software to DO-178 standard, it is necessary to take into account the topics not covered by ECSS and that the ECSS software, in this case, will require certification as described in DO-178.

### 8.2.2 DO-330

ECSS provides considerations on Tool Qualification, particularly the level assignment and qualification method to be applied according with this standard (see section 6 ECSS-Q-HB-80-01A), but there is no specific standard for tool qualification in ECSS. In ECSS at tool to be qualified shall follow the normal software development standards and guidelines (ECSS-E-ST-40C and ECSS-Q-ST-80C). Since this standard follows the philosophy of the DO-178 standard, the overall main differences between this one and ECSS are the same as outlined for the DO-178, in the previous section. Hence, the considerations for qualifying a Tool developed under ECSS to DO are the same for the tool's project qualification from ECSS to DO.

### 8.2.3 DO-333

In ECSS, formal methods are described as a mean to apply the ECSS-E-ST-40C standard (see Annex B ECSS-E-HB-40A). ECSS does not provide specific requirements on formal methods usage, but only states the possibility to use this approach on the following situations:

- Logical Model Description (see 6.2.3 ECSS-Q-ST-80C and Annex D ECSS-E-ST-40C).
- Software quality requirements verification (see 6.2.6 ECSS-Q-ST-80C and Annex I ECSS-E-ST-40C).

This standard does not contain requirements which are contradictory to the ECSS standard. However, software developed under ECSS standard and using formal methods, shall have in consideration this standard, if a qualification for DO is foreseen.

## 8.3 ISO 26262 Analysis

The increase of technological complexity in electrical and/or electronic (E/E) systems raises the risks from systematic and random failures. Thus, a standardized set of practices emerged to avoid risk by providing a set of requirements and processes that allows one to minimize failures. ISO 26262 appeared as the adaptation of IEC 61508 (the generic functional safety standard for electrical and electronic (E/E) systems) to comply with the needs of the automotive industry with respect to electrical and/or electronic (E/E) systems within road vehicles.

This section presents the analysis of compatibility between ISO 26262 and ECSS. The applied methodology was to analyse each section of the ISO 26262 standard and evaluate the corresponding clauses, if there is correspondence, on the ECSS. Furthermore, the compatibility of ISO 26262 work products (that represents the expected output of one or more associated requirements) with the ECSS artifacts is presented at the end of each corresponding part.

```
ISO 26262 lifecycle
```

ISO 26262 is divided into three phases: concept phase, product development and start of production as depicted in figure *ISO 26262 Lifecycle*.

During the concept phase, the item shall be defined, hazard analysis and risk assessment performed based on the item (hazards are classified and assigned with an Automotive Safety Integrity Level). Safety goals are derived from the hazards and are assigned with the hazard Automotive Safety Integrity Level (ASIL) classification. Functional safety requirements are established based on the safety goals (inheriting the ASIL classification) and are allocated to preliminary architectural elements of the item or to external measures.

In the product development phase at the system level, the item is developed from a system level perspective upon the V-model approach. Concerning the V-model, on the left branch of the V, the technical safety requirements, system architecture, system design and implementation are specified. On the right branch of the V, the system integration, verification, validation and functional safety assessment are specified.

Being developed at the system level, it is implied that both software and hardware are considered. Again a V-model is used in which, the specific (software or hardware) requirements, design and implementation are on the left branch, and on the right branch, specific (software or hardware) integration, testing and verification are performed.

Finally, in the start of production phase the production processes related to the functional safety goals of the item are specified, e.g., safety-related special characteristics, and the development and management of instructions for the maintenance, repair and decommissioning of the item.

```
1 ISO 26262 - Part 1: Vocabulary
```

This part of the standard describes the terms and definitions used in all parts of ISO 26262. The equivalent of this part in ECSS standards is found in ECSS-S-ST-00-01C [ECS12b] and specific definitions for software systems are detailed in Section 3 of ECSS-E-ST-40C [ECS09b] and ECSS-Q-ST-80C [ECS17d].

```
2 ISO 26262 - Part 2: Management of functional safety
```

This part of ISO 26262 specifies the requirements for functional safety management for automotive applications, covering the following clauses (each of the clauses is detailed in the next subsections):

- Overall safety management;
- Safety management during the concept phase and the product development;
- Safety Management after the item's release for production.

```
2.1 Overall safety management
```

This clause defines the requirements for the organizations that either are responsible for the safety lifecycle or that perform safety activities in the safety lifecycle. The organizations shall comply with the following sub-clauses: safety culture, competence management, quality management during the safety lifecycle and project tailoring of the safety lifecycle.

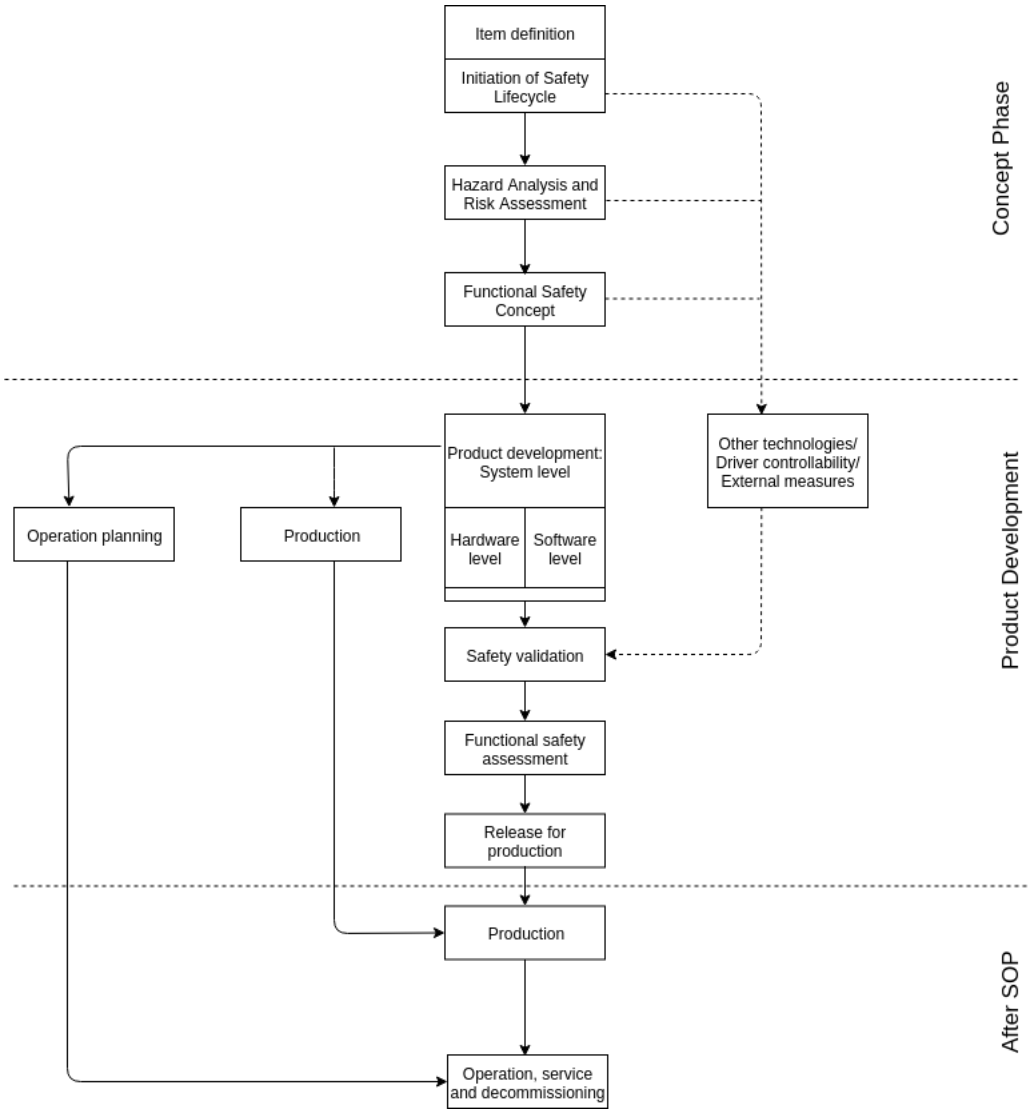



Fig. 1: ISO 26262 Lifecycle

#### 2.1.1 Safety culture

ISO 26262 presents requirements to the organizations and personnel involved in safety activities in order to foster support and achieve the functional safety of the specified items.

ECSS follows the same mindset and specifies requirements for organizations and personnel involved in product development, such as: (i) responsibilities of the involved personnel and organizations; and (ii) guaranteeing that the resources involved in the project have the required skills, etc. (see 5.1.1 ECSS-Q-ST-10C [ECS16]).

#### 2.1.2 Competence management

Both ECSS and ISO 26262 define competence requirements (i.e., skills, competences and qualifications).

ECSS specifies the personnel working in all product disciplines (see 5.1.1.1 ECSS-Q-ST-10C). Additionally, an assessment shall be performed jointly by the customer and the supplier to verify if additional skills, resources, or facilities are needed to complete the project (see 4.1.5 ECSS-M-ST-10C [ECS09c]).

#### 2.1.3 Quality management during the safety lifecycle

For this sub-clause, ISO 26262 defines that the organizations executing a safety lifecycle shall have an operational quality management system that complies with a quality standard (as for instance ISO 9001).

Concerning the quality assurance requirements, ECSS specifies them in ECSS-Q-ST-20C [ECS18b].

#### 2.1.4 Project-independent tailoring of the safety lifecycle

ISO 26262 allows the tailoring of the safety lifecycle across different item developments. However, such tailoring does not contemplate the removal of sub-phases, tasks or activities from the lifecycle.

With respect to tailoring, ECSS allows the tailoring of activities as long as in the case of requirements not being applicable without change, the modifications must be recorded and justified (see 7.3 ECSS-S-ST-00C [ECS08e]). Moreover, the tailoring of activities can be performed based on software criticality (see annex R ECSS-E-ST-40C).

### 2.2 Safety management during the concept phase and the product development

This clause has the following two objectives: (i) define the safety management roles and responsibilities; and (ii) specify the requirements for the safety management during the concept phase and the development phases.

#### 2.2.1 Roles and responsibilities in safety management

ISO 26262 specifies that a project manager shall be appointed at the initiation of the item development and shall be given the corresponding authority and responsibility to perform safety management.

In ECSS, each supplier shall appoint a product assurance manager with sufficient authority and independence to ensure their responsibilities (see 5.1.1 ECSS-Q-ST-10C). Moreover, a safety manager shall equally be appointed by the supplier (see 5.2.1.2 ECSS-M-ST-10C and 5.4.2 ECSS-Q-ST-40C [ECS17c]). The safety manager role can be fulfilled by the product assurance manager as long as the customer agrees.

#### 2.2.2 Planning and coordination of the safety activities

ISO 26262 specifies the safety manager responsibilities concerning the planning and coordination of the functional safety activities in the development phases of the safety lifecycle. Besides the safety manager responsibilities, this sub-clause additionally presents what shall be included in the safety plan.

ECSS specifies partially the same for both the safety manager responsibilities (see 5.4 ECSS-Q-ST-40C) and what shall be included in the safety plan (see 5 ECSS-Q-ST-40C). However, in ECSS the safety plan is referred as the safety programme plan and may be included as part of an overall project product assurance plan.

#### 2.2.3 Progression of the safety life cycle

ISO 26262 specifies that safety activities in subsequent sub-phases of the safety lifecycle shall be started only when there is sufficient information from the pertinent sub-phases.

ECSS defines that all plans concerning the lifecycle shall be finalized before the start of the related activities. Additionally, any activity shall start only when each procedure and/or standard of an activity shall be reviewed (see 6.2.1 ECSS-Q-ST-80C). Moreover, ECSS specifies a progress report (see annex E ECSS-M-ST-10C), where the actual information concerning the status of the project shall be provided to all actors.

#### 2.2.4 Tailoring of the safety activities

Just as mentioned in Section 3.1.4 Project-independent tailoring of the safety lifecycle, the tailoring of safety activities is supported in ECSS as long as the modifications are recorded and justified. In this sub-clause, the modifications shall be performed according to the change management (see 5.3.2 ECSS-M-ST-40C [ECS09d]).

#### 2.2.5 Safety case

In this sub-clause, ISO 26262 establishes that item's with at least one safety goal with an ASIL classification shall have a safety case.

ECSS complies with this requirement through 7 ECSS-Q-ST-40C (Safety analysis requirements and techniques), specifically through 7.3 Assessment and allocation of requirements.

#### 2.2.6 Confirmation measures: types, independency and authority

ISO 26262 specifies the confirmation measures required by the safety plan to review, verify and validate if the safety plan is sufficiently complete and complies with the standard requirements.

Concerning the confirmation measures, ECSS specifies them in the safety and dependability standards (ECSS-Q-ST-40C and ECSS-Q-ST-30C [ECS17b]) and defines that to successfully complete the safety process, a positive evaluation shall be given to all verification items associated with a given hazard.

### 2.2.7 Functional safety audit

ECSS specifies safety audits (see 5.4.3 ECSS-Q-ST-40C). However, safety audits shall be in compliance with ECSS-Q-ST-10 and the general requirements of project audits (ECSS-M-ST-10C).

### 2.2.8 Functional safety assessment

ISO is in accordance to what is specified for the safety risk assessment (5.5 ECSS-Q-ST-40C) and with the assessment of safety requirements (see 7.3 ECSS-Q-ST-40C).

## 2.3 Safety Management after the item's release for production

The intention of this clause is to define the responsibilities of the organizations and persons responsible for functional safety after the item's release for production.

### 2.3.1 Responsibilities, planning and required processes

In ECSS, it is described that the supplier shall identity the necessary personnel for all product assurance disciplines, including the product maintenance (see 5.1.1 ECSS-Q-ST-10C). Moreover, the safety manager shall have the required authority to maintain the safety programme plan (see 5.4 ECSS-Q-ST-40C).

## 2.4 Work products

ISO 26262 Part 2: Management for functional safety requires the following work products:

- Organization-specific rules and processes for functional safety. In ECSS, the objective of this work product is described in the safety programme plan (see annex B ECSS-Q-ST-40C). However, general requirements for the organizations are defined in the PAP (annex A ECSS-Q-ST-10C) and the PMP (annex A ECSS-M-ST-10C).
- Evidence of competence. In the PAP, the organization shall specify the personnel working on product assurance disciplines and their respective skills, demonstrating their competence for the execution of the proposed activities.
- Evidence of quality management. In ECSS, the evidence of quality management is ensured through the quality assurance plan (see annex A ECSS-Q-ST-20C).
- Safety plan. In ECSS, the correspondent artifact is the safety programme plan, where in annex B of ECSS-Q-ST-40C its content is specified. Moreover, the safety programme plan may be incorporated within the product assurance plan if agreed between the relevant parties.
- Safety case. In ECSS, the objective of this work product is spread throughout the safety programme plan, safety verification tracking log (see annex C ECSS-Q-ST-40C) and the safety analysis report (see annex D ECSS-Q-ST-40C).
- Project plan. ECSS specifies the project management plan (PMP) to state the purpose and provide an introduction for the project management system, thus, following the same mindset as ISO 26262.
- Functional safety assessment plan. In ECSS, the objective of this work product is covered by the safety programme plan, the risk management plan (see annex B ECSS-M-ST-80C

[ECS08d]), safety analysis report (see annex D ECSS-Q-ST-40C) and the risk assessment report (see annex C ECSS-M-ST-80C).

- Confirmation measures reports. In ECSS, this work product is spread throughout the following work products: (i) safety programme plan; (ii) dependability plan (see annex C ECSS-Q-ST-30C); (iii) Audit plan and schedule (see PAF); and (iv) the risk management plan (see annex B ECSS-M-ST-80C).
- Evidence of field monitoring. ECSS complies with this work product by specifying on-board monitoring, where the capability to monitor on-board a set of on-board parameters specified by ground segment shall be provided (see 5.8.6 ECSS-E-ST-70-11C [ECS08a]).

```
3 ISO 26262 - Part 3: Concept phase
```

Part 3 of ISO 26262 specifies the requirements for the concept phase of the automotive applications development. The following clauses are detailed in this part:

- Item definition;
- Initiation of the safety lifecycle;
- Hazard analysis and risk assessment;
- Functional safety concept.

```
3.1 Item Definition
```

This clause specifies how an item shall be described such that subsequent activities in the lifecycle can progress. This includes the description of the functional and non-functional requirements, its dependencies with other items and the environment.

For software items, ECSS follows the same approach in the system requirement process by considering the following activities: system requirements analysis, system verification and system integration and control. These are defined in 5.2 ECSS-E-ST-40C, 5.4.2 ECSS-E-ST-40C and 6.3.2.4 ECSS-Q-ST-80C.

```
3.2 Initiation of the safety life cycle
```

In this clause, ISO 26262 establishes the difference between the development of a new item and the modification of an existing item, by defining additional requirements in case of an item modification.

ECSS covers the modification of a software product in 5.10.4 ECSS-E-ST-40C and the reuse of existing software in the development of a new item in 5.4.3.7 ECSS-E-ST-40C.

```
3.3 Hazard Analysis and Risk Assessment
```

The goal of this clause is to identify and categorize the hazards that malfunctions in the item under definition can trigger and formulate safety goals that allows one to prevent or mitigate such events. ISO 26262 divides the analysis of hazards and risk assessment into different topics, namely:

- Initiation of the hazard analysis and risk assessment;
- Situation analysis and hazard identification;

- Classification of hazardous events;
- Determination of ASIL and safety goals;
- Verification.

In ISO 26262 the risk assessment of hazardous events considers three parameters, namely severity, likelihood and controllability, while in ECSS the parameters considered are severity and likelihood. The results of this analysis are taken into account into the determination of the ASIL level (mapped from the list of possible values in Table 4) and the respective safety goals for the item under analysis, which inherit the highest ASIL level of the hazardous event.

ECSS-Q-ST-80C (see 5.3.1) specifies that risk management for software shall be performed by cross-referencing to the project risk policy, as specified in ECSS-M-ST-80 (this standard defines the principles and requirements for integrated risk management of space projects, where in section 5.2 the steps and tasks are identified).

The analysis and classification of hazardous events shall be performed according to the following standards: (i) in ECSS-M-ST-80 (see 5.2.1), the risk index and magnitude are established through the combination of the severity and the likelihood of events; (ii) ECSS-Q-ST-40C (see 5.5), the steps to perform the safety risk assessment and control are detailed.

In ECSS the severity of potential consequences of the hazardous events are identified and categorized in Table 6-1 of ECSS-Q-ST-40C (see 6.4) (the criteria shall be agreed between the costumer and supplier). Furthermore, the criticality of the functions of critical functions, hardware and operations must be assigned following the categories defined in Table 5-2 of ECSS-Q-ST-30, in which criticality is related to the severity of hazardous events. Then, the criticality category of a software product (A,B,C,D) is assigned based on the criticality assigned to the most critical function it implements and meeting the criteria defined in Table 5-3 of ECSS-Q-ST-30 and the requirements in 5.4.2.

### 3.4 Functional Safety Concept

ISO 26262 expects that functional safety requirements are derived from the safety goals and then allocated to preliminary architectural elements or to external measures.

The correspondent clause in ECSS is detailed in ECSS-Q-ST-40C, chapter 6 and 7, where the requirements for Safety Engineering are detailed, for instance requiring their identification and traceability from system level into the design. Concerning software, the safety requirements must be specified in the Software System Specification (SSS) and Software Requirements Specification documents, as specified in ECSS-E-ST-40C.

#### 3.4.1 Validation criteria

ISO 26262 specifies that the acceptance criteria for safety validation shall be specified considering the functional safety requirements.

ECSS validation is performed by tests that shall include the demonstration of nominal, contingency and emergency operational modes (see 8.4.1 ECSS-Q-ST-40C).

Furthermore, the acceptance criteria should be defined in the project risk management policy and considered in the tailoring of the project specific safety requirements.

### 3.4.2 Verification of the functional safety concept

ECSS follows the same mindset as ISO 26262 for the verification of the functional safety concept (see 8 ECSS-Q-ST-40C).

## 3.5 Work products

ISO 26262 Part 3: Concept Phase requires the following work products:

- Item definition. In this work product, the item functional and non-functional requirements and dependencies between other items and the environment are specified. In ECSS, this information is covered by the requirements baseline. For software, this is part of the software system specification (annex B ECSS-E-ST-40C) where the system requirements are defined by the customer and delivered to the supplier, and the technical specification, specifically the software requirements specification (annex D ECSS-E-ST-40C) where the software requirements are specified by the supplier as a response for the requirements baseline.
- Hazard analysis and risk assessment. In ECSS, hazard analysis and risk assessment are specified throughout the following artefacts: (i) safety programme plan, where the planning and execution of safety analysis is specified; (ii) safety analysis report including hazards reports (annex D ECSS-Q-ST-40C), which contains the results of the systematic identification, evaluation, reduction, verification and tracking of hazards; (iii) the risk management plan (annex B ECSS-M-ST-80C), where it is described the identification and assessment process and procedures for evaluating the critical risk items and domains; and (iv) the risk assessment report (annex C ECSS-M-ST-80C) that describes what was done during the identification and assessment exercise.
- Impact analysis. In ISO 26262, this work product specifies the analysis carried out to identify, describe and assess the impact a modification has on the item. In ECSS, this is part of the following artefacts: (i) the configuration management plan (see annex A ECSS-M-ST-40C), where change assessments shall be carried out; (ii) the software dependability and safety analysis report; and (iii) the modification documentation (included in the maintenance file).
- Safety plan. See safety plan work product in Section 2.4.
- Safety goals. The term safety goal encompasses the top-level safety requirements that are derived from the safety analysis and risk assessments. Thus, in ECSS the safety goals are specified in the safety analysis report and hazards reports and the risk assessment report.
- Verification review report of the hazard analysis and risk assessment and the safety goals. In ECSS, although the verification process of the hazard analysis and risk assessment is specified in the safety programme plan and in the risk management plan, the correspondent verification reports and follow up actions are presented in the risk assessment report and in the safety verification tracking log (annex C ECSS-Q-ST-40C).
- Functional safety concept. In ECSS, the objectives of the functional safety concept is spread throughout various artefacts such as: (i) the safety programme plan; and (ii) safety analysis report including hazard reports. In these artefacts, the safety requirements are derived from the safety analysis, specified and then traced from system level into the design. Concerning software, the following artefacts specify the safety requirements and

trace them to software elements: (i) the software requirements specification; and (ii) the software system specification; and (iii) the software design document.

- Verification report of the functional safety concept. This ISO 26262 work product is covered in ECSS throughout the safety verification tracking log (annex C ECSS-Q-ST-40C). Concerning software, this work product is covered by the software verification report (see annex M ECSS-E-ST-40C).

```
4 ISO 26262 - Part 4: System level
```

This part specifies the requirements for product development at system level for automotive applications, it includes the following clauses:

- Initiation of product development at the system level;
- Specification of the technical safety requirements;
- System design;
- Item integration and testing;
- Safety validation;
- Functional safety assessment;
- Release for Production.

ECSS follows a project based approach that is defined in ECSS-M-ST.10C (see 4), which is equivalent to the lifecycle proposed by ISO 26262 for product development at the system level, in order to plan and execute a space project from initiation to completion at all levels in the customer-supplier chain.

```
4.1 Initiation of product development at the system level
```

This clause focuses on the determination and planning of the functional safety activities during the individual sub-phases of the system development.

ECSS follows the same concept and includes the specification and planning of safety activities that shall be performed during the defined lifecycle (see 5.7 ECSS-Q-ST-40C where each project phase and the respective safety activities that shall be done in each phase are specified).

```
4.2 Specification of the technical safety requirements
```

In this clause, the goal is to specify the technical safety requirements based on the functional safety concept and the preliminary architectural assumptions. Moreover, it also contemplates the verification that the technical safety requirements comply with the functional safety requirements.

ECSS contemplates this clause through the specification of the technical safety requirements (see 6 ECSS-Q-ST-40C).

```
4.2.1 Safety mechanisms
```

In ISO 26262, this clause defines that the technical safety requirements shall specify the system response to events that may affect the accomplishment of safety goals.

Concerning the system response to hazardous events, ECSS specifies a set of mechanisms and requirements to erase, minimize and control these events (see 6.3.3 ECSS-Q-ST-40C and 6.2.3 ECSS-Q-ST-80C).

#### 4.2.2 ASIL decomposition

ECSS standards do not contemplate the ASIL decomposition. However, ECSS criticality level categorization shall be applied (see Section 4.3 Hazard Analysis and Risk Assessment).

#### 4.2.3 Avoidance of latent faults

The avoidance of latent faults, hazard elimination, mitigation and control shall be performed according to the clauses mentioned in Section 5.2.1 Safety mechanisms and 7 ECSS-Q-ST-40C.

#### 4.2.4 Production, operation, maintenance and decommissioning

ECSS specifies general and safety requirements for the processes of production, operation, maintenance and decommissioning. See 4.4.3 ECSS-M-ST-10C for the processes' requirements and recommendations throughout the ECSS lifecycle (specially phases D, E and F). Regarding the software and safety requirements for the concerned processes, see 5.7, 5.9, 5.10 ECSS-E-ST-40C; 6.3.6, 6.3.7, 6.3.8 ECSS-Q-ST-80C and 5.7.1 ECSS-Q-ST-40C.

#### 4.2.5 Verification and validation

The criteria for safety validation and the verification of the compliance between the technical safety requirements and both the safety concept and the preliminary architectural design assumptions is specified in clause 8 of ECSS-Q-ST-40C.

### 4.3 System Design

#### 4.3.1 System design specification and technical safety concept

In this subclause, the system design shall be established according to the functional concept, the preliminary architectural assumptions and the technical safety requirements.

ECSS specifies that the system design shall be developed based on the functional architecture, the allocation requirements and the selected technology (see 5.4.1 ECSS-E-ST-10C [ECS17a]). Regarding software requirements and architectural design, see 5.4 and 5.5.2 ECSS-E-ST-40C.

#### 4.3.2 Measures for the avoidance of systematic failures

In ISO 26262, this subclause specifies the requirements and recommendations for both the design and the execution of safety analysis over the system design to identify and mitigate systematic failures. Considering safety analyses, they shall be performed in compliance with deductive and inductive analysis methods in order to assist the design.

ECSS define safety analysis that complies with deductive and inductive methods to mitigate systematic faults. Concerning their requirements, safety analysis requirements are presented in clause 7.5 of ECSS-Q-ST-40C and the analysis methods in 6.4.2 of ECSS-Q-ST-30C. For system design requirements and properties see Section 5.3.1 System design specification and technical safety concept.

```
4.3.3 Measures for control of random hardware failures during operation
```

In this clause, ISO 26262 specifies requirements for the detection and control of random hardware failures considering the system design.

ECSS follows partially the same mindset, as measures shall be provided as result of safety analysis (see clause 6 and 7 of ECSS-Q-ST-40C ). As part of the analyses, ECSS specifies a hardware-software interaction analysis (HSIA) to ensure that the software reacts in an acceptable way to hardware failures (see 6.4.2.3 ECSS-Q-ST-30C). For specific hardware details and its analysis, 5 ISO 26262 - Part 5: Hardware level.

```
4.3.4 Allocation to hardware and software
```

This clause specifies the allocation of the technical safety requirements to hardware and/or software elements.

ECSS specifies that requirements shall be identified and traced from the system level into the design and allocate them to lower level (see 6.2 ECSS-Q-ST-40C). Moreover, in 4.2 ECSS-E-ST-40C (Overview of space system software engineering processes) it is stated that the customer is responsible for deriving the functional and performance requirements for the hardware and software according to system engineering principles and methods. Considering the allocation of software requirements to software elements see 5.2.2.1 ECSS-E-ST-40C.

Moreover, in ISO 26262 this clause also specifies that an adequate development process combining hardware and software requirements shall be defined for custom hardware elements that incorporate programmable behaviour.

In the same line of thought, ECSS defines a set of requirements for the development of digital, analog and mixed analog-digital custom designed integrated circuits (see ECSS-Q-ST-60-02C).

```
4.3.5 Hardware-software interface specification
```

In ISO 26262, this clause presents the requirements to the specification of the hardware and software interaction and its compliance with the technical safety concept.

According to 5.3 ECSS-E-ST-10-24C [ECS15], the interface requirements should be conformant with Annex A of the same document. Moreover, with respect to hardware and software interfaces, in 6.4.2.3 ECSS-Q-ST-30C it is mentioned how hardware to software interaction analysis shall be performed, which is further specified in 6.2.2.8 and 6.2.2.9 ECSS-Q-ST-80C. Moreover, in Annex K ECSS-E-ST-40C it is also mention how the integration testing activities (as part of the SUITP) shall be carried.

```
4.3.6 Requirements for production, operation, service and decommissioning
```

For requirements for the processes of production, operation, service and decommissioning see Section 5.2.4 Production, operation, maintenance and decommissioning.

```
4.3.7 Verification of system design
```

ECSS specifies various verification activities, see general verification requirements in ECSS-E-ST-10-02C [ECS18a], product verification in 5.5 ECSS-E-ST-10C, safety verification in 8 ECSS-Q-ST-40C and software verification in 8.8 ECSS-E-ST-40C and 6.2.6 ECSS-Q-ST-80C.

```
4.4 Item integration and testing
```

In this clause, ISO 26262 specifies requirements for the following three integration phases: (i) integration of the hardware and software elements that compose the item; (ii) integration of the elements that compose an item to form a complete system; and finally, (iii) the item integration with other systems within the vehicle and with the vehicle itself.

Furthermore, this clause contains the following objectives: (i) verify that the system design covering the safety requirements are correctly implemented by the entire item; (ii) test the compliance between each safety requirements and its specification and ASIL specification.

In ECSS, testing requirements are described in ECSS-E-ST-10-03C [ECS12a]. In particular, test management is described in 4.3 of the same standard. In this clause, it is referenced that there should be an assembly, integration and test plan (AITP) (see annex A ECSS-E-ST-10-03C). The AITP shall be established by the supplier and must provide a complete description of the AIT processes and shall determine along with the verification plan (the verification plan and the AITP can be combined in one single AIV Plan) how the requirements are verified through inspection and test.

The AIT processes and their safety activities (e.g., the monitoring and control of project assembly, integration, testing and handling operations which are potentially hazardous to personnel or hardware) are included in phase C of the ECSS lifecycle (see 5.7.1.4 ECSS-Q-ST-40C).

Moreover, the integration capability performing its intended function in terms of performance and calibration shall be verified as part of the overall integration (see 5.6.2, 5.7.4.2 and 5.7.4.3 ECSS-Q-ST-20-07C [ECS14]). Concerning software integration, see 5.5.4 ECSS-E-ST-40C and 5.6 for software validation (see Section 5.3.5 Hardware-software interface specification for more details).

```
4.5 Safety validation
```

This clause contains the following two objectives: (i) provide evidence of compliance with the safety goals and that the functional safety concepts are appropriate for the functional safety of the item; and (ii) provide evidence that the safety goals are correct, complete and fully achieved at the vehicle level. The goal is to provide evidence that the results of each activity complies with the specified requirements.

ECSS-E-ST-10-03C provides the requirements for verification using testing for space elements and space segment equipment prior to launch. Statement of compliance between space system elements and safety requirements shall be demonstrated as required by 5.8 ECSS-Q-ST-40C.

For software, the validation and verification processes are described in 4.2.6 and 4.2.8 ECSS-E-ST-40C, respectively, and the requirements for each phase are detailed in 5.6 and 5.8 of ECSS-E-ST-40C. Moreover, 6.2.6 ECSS-Q-ST-80C describes the activities that are part of the verification plan and 6.3.5 ECSS-Q-ST-80C presents the requirements for testing and validation.

```
4.6 Functional safety assessment
```

For functional safety assessment requirements see Section 3.2.8 Functional safety assessment.

```
4.7 Release for production
```

In ISO 26262, this clause specifies the requirements for the specification of the release for production criteria at the completion of the item development and the confirmation that the item complies with the requirements for functional safety at the vehicle level.

ECSS follows the same approach through the execution of the acceptance process (see phase D of ECSS lifecycle, 4.4.3.6 ECSS-M-ST-10C), where the customer shall perform an acceptance review (AR) with the supplier support. In 4.4.3.6.4 of ECSS-M-ST-10C, ECSS specifies the AR objectives. Regarding software acceptance, see 5.7.3 ECSS-E-ST-40C and 6.3.6 ECSS-Q-ST-80C.

### 4.8 Work products

This section presents the study concerning the compatibility of ISO 26262 work products related with product development at system level with ECSS artefacts. The study is shown below:

- Project plan. See Section 2.4.
- Safety plan. See Section 2.4.
- Validation plan and report. ECSS follows roughly the same mindset for these work products though the specification of the safety validation in the safety verification activity (see the safety artefacts, such as the safety programme plan and the safety verification tracking log. For software, it specifies these expected output for software items in the software validation plan (see annex J ECSS-E-ST-40C) and the software validation report (see annex M ECSS-E-ST-40C).
- Item integration and testing plan and report. In ECSS, the item integration and testing activities are specified in the assembly, integration and test plan (annex A ECSS-E-ST-10-03C). However, software integration and testing is covered by the software [unit/integration] test plan (SUITP) (see annex K ECSS-E-ST-40C) and the software integration strategy (included in the design justification file).
- Functional safety assessment report. In ECSS, this work product is covered through the safety analysis report (see annex D ECSS-Q-ST-40C) and the risk assessment report (see annex C ECSS-M-ST-80C).
- System verification report. The system verification report is covered in ECSS through the verification report (see annex F ECSS-E-ST-10-02C).
- Technical safety concept and requirements specification. ECSS specifies this work product objective throughout its artefacts, mainly in the safety programme plan (see annex B ECSS-Q-ST-40C) and in the technical specification. Moreover, the verification of the technical safety requirements is covered in the safety verification tracking log (see annex C ECSS-Q-ST-40C) and the safety analysis report including hazard reports (see annex ECSS-Q-ST-40C). Considering the verification of software, this work product is covered in the software verification plan (see annex I ECSS-E-ST-40C) and report (see annex M ECSS-E-ST-40C).
- System design specification. In ECSS, system and software design specification and justification activities are defined in the design definition file (see annex G ECSS-E-ST-10C) and in the design justification file (see annex K ECSS-E-ST-10C).
- HSI specification. Software and hardware interfaces are specified in the interface control document (annex E ECSS-E-ST-40C) and in the software interface requirements document (annex C ECSS-E-ST-40C).

- Safety analysis report. The ISO 26262 safety analysis report is in accordance with ECSS safety analysis report including hazards reports (see annex D ECSS-Q-ST-40C).
- Release for production. In ECSS, this work product is specified in the acceptance review and the acceptance test plan and report. Concerning software, ECSS specifies the software release document.

```
5 ISO 26262 - Part 5: Hardware level
```

ISO 26262 specifies requirements recommendations for the following hardware development stages:

- Initiation of product development at the hardware level;
- Specification of hardware safety requirements;
- Hardware design;
- Evaluation of the hardware architectural metrics;
- Evaluation of safety goal violations due to random hardware failures;
- Hardware integration and testing.

ECSS defines the processes and the respective requirements related to hardware development in ECSS-E-ST-60-XXX and ECSS-Q-ST-60-XXX, where XXX varies according to the type of hardware being developed. As hardware development is out of scope of RTEMS-SMP and is very specific for each of the domains (automotive vs. space), this part of ISO was not directly compared with ECSS standards.

```
6 ISO 26262 - Part 6: Software level
```

This part presents the requirements concerning product development at the software level for automotive applications. The following clauses are detailed in the following subsections:

- Initiation of product development at the software level.
- Specification of the software safety requirements.
- Software architectural design.
- Software unit design and implementation.
- Software unit testing.
- Software integration and testing.
- Verification of software safety requirements.

ECSS has two documents that focus on software development, namely ECSS-E-ST-40C and ECSS-Q-ST-80C. The software related processes in ECSS standards are mapped in figure 4.1 of ECSS-E-ST-40C and the software life cycle process is depicted in figure 4.2 of the same standard.

```
6.1 Initiation of product development at the software level
```

This sub-clause has the objective of planning and initiating the functional safety activities for the sub-phases of the software development. This involves the determination of the methods (which include guidelines and tools) that have to comply with requirements and integrity levels.

In ECSS, the software management process is described in clause 5.3 ECSS-E-ST-40C. This process specifies among other tasks, the software life cycle management process in sec. 5.3.2. Moreover, the requirements that are applicable to all software engineering processes are described in 6.2 ECSS-Q-ST-80C, which include guidelines for handling critical software (see 6.2.3 ECSS-Q-ST-80C).

```
6.2 Specification of software safety requirements
```

In ISO 26262, this clause specifies the software safety requirements and details the hardware-software interfaces (HSI) and verifies the both the software safety requirements and hardware-software interfaces are consistent with with the technical safety concept and system design specification.

ECSS follows a similar approach as the specification of the software safety requirements and its verification are performed according to the requirements defined in 5.4 and of 5.8.3.2 ECSS-E-ST-40C, 6.3.2 ECSS-Q-ST-80C and clause 6 and 8 of ECSS-Q-ST-40C. Regarding HSI specification, see the software interface requirement document (annex C ECSS-E-ST-40C).

```
6.3 Software architectural design
```

In this clause, ISO 26262 specifies the requirements the development of the software architectural design (which includes all software components and their interactions) that accomplishes the software safety requirements and the verification of the software architectural design.

The software architectural design activities are described in 5.4.3 ECSS-E-ST-40C. This subclause covers the transformation of software requirements into a software architecture; the software design method (a method to produce software components and their interfaces); the software model and their behaviour. Moreover, the software verification process requirements and activities are described in 5.8 ECSS-E-ST-40C, specifically, in 5.8.3, verification activities are described.

Concerning the safety design objectives and the hazard reduction, the activities in 6.3 ECSS-Q-ST-40C should be considered. In addition, the software process assurance should also be considered, as described in 6 ECSS-Q-ST-80C, and in particular for this ISO sub-clause, 6.3.3 (Software architectural design and design of software items) should be taken into account.

In ECSS-Q-HB-80-02_Part2A [ECS10], the process ENG.5 Software design, detailed in 4.1.2.4.5, should also be considered.

```
6.4 Software unit design and implementation
```

This clause specifies the design, implementation and verification of the software units according to the software architectural design.

ECSS follows the same mindset as ISO 26262, specifying in 5.5 ECSS-E-ST-40C (Software design and implementation engineering process) and in 6.3 ECSS-Q-ST-80C the processes for software item design (6.3.3) and implementation (6.3.4). The design and implementation processes are under the supplier responsibility, however, the customer shall agree in both on the adopted coding standards. Regarding the verification, ECSS specifies the detailed design and coding verification in 5.8.3.4 and 5.8.3.5 ECSS-E-ST-40C.

In ECSS-Q-HB-80-02_Part2A, the process ENG.6 Software construction, detailed in 4.1.2.4.6, should also be considered.

### 6.5 Software unit testing

In this clause, ISO 26262 specifies requirements and recommendations to demonstrate that software units obey to their design specifications and do not contain undesired functionality. It focus on the following topics: (i) planning of software unit testing; (ii) methods that shall be applied; (iii) use of structural metrics to evaluate the test cases completeness; and (iv) for the test environment.

In ECSS, these requirements are defined 5.5.3 ECSS-E-ST-40C, where the process of software unit testing is described (see 5.5.3.2), and in 6.3.5 ECSS-Q-ST-80C where the requirements corresponding to those defined in ISO 26262 are specified. ECSS additionally dictates that it is the supplier responsibility to perform and verify the software unit tests under the customer acceptance (for critical software, witnessed or independent testing shall be performed according to 6.2.3 ECSS-Q-ST-80C).

In ECSS-Q-HB-80-02_Part2A, the process ENG.8 Software testing, detailed in 4.1.2.4.8, should also be considered.

### 6.6 Software integration and testing

Such as the previous clause, ISO 26262 specifies requirements for the following topics: (i) the planning of software integration and testing; (ii) methods that shall be applied for the integration and to the derivation of the test cases; (iii) structural coverage metrics; and (iv) test environment.

ECSS follows the same mindset as ISO 26262, specifying the planning and methods that shall be applied for the software integration and testing (see 5.5.4 of ECSS-E-ST-40C and ECSS-E-HB-40A [ECS13]), the methods to be applied during the integration testing are defined in 6.4 ECSS-E-HB-40A.

Regarding the software verification report refinement, ECSS defines the verification of software integration in 5.8.3.7 ECSS-E-ST-40C.

In ECSS-Q-HB-80-02_Part2A, the process ENG.7 Software integration, detailed in 4.1.2.4.7, should also be considered.

### 6.7 Verification of software safety requirements

In ISO 26262, the purpose of this clause is to demonstrate that the embedded software fulfils the software safety requirements.

Regarding ECSS, the correspondent to this clause is 5.8.3.8 of ECSS-E-ST-40C, where it is specified that the supplier shall verify that all software requirements of the technical specification and/or the requirements baseline are covered, and the clause 8.4 of ECSS-Q-ST-40C, that defines that all safety-critical functions shall be verified and qualified. As a side note, the verification process requirements for software, as an engineering process applied to the software development life cycle, are detailed in 6.2.6 ECSS-Q-ST-80C.

### 6.8 Work products

This section specifies the compatibility of the ISO 26262 work products related to the software level with the ECSS artefacts. The study is presented below:

- Safety plan. See Section 2.4.
- Design and coding guidelines for modeling, programming languages and tool application guidelines. In ECSS, these work products are included in the specification of the software engineering standards and techniques of the software development process artefact (see annex Q ECSS-E-ST-40C), where it is described the applied methodologies and standards for each software development process.
- Software safety requirements specification. In ECSS, the software safety requirements are specified in the software requirements specification (see annex D ECSS-E-ST-40C) and in the software system specification (see annex B ECSS-E-ST-40C).
- HSI specification. See Section 4.8.
- Software architectural design and software unit design specification. In ECSS, the artefact that specifies the software architectural design and unit design specification is the design definition file, specifically the software design document (see annex F ECSS-E-ST-40C).
- Safety analysis report. ECSS defines roughly the same as ISO 26262 concerning safety analysis report by specifying the safety analysis report including hazard reports (see annex D ECSS-Q-ST-40C). Concerning software safety analysis, ECSS specifies the software dependability and safety analysis report that is included in the SPAP (see annex B ECSS-Q-ST-80C).
- Dependent failures analysis report. In ECSS, this work product is included in the common-cause analysis (see annex I ECSS-Q-ST-30C) and the safety analysis report, as the analysis of dependent failures is integrated in the safety analysis.
- Software unit implementation. In ECSS, the software source code shall be provided in the design definition file (see annex G ECSS-E-ST-10C), specifically in the software source code and media labels clause.
- Software verification. ECSS follows the same mindset for the software verification activities, as it specifies the software verification plan (SVerP) (annex I ECSS-E-ST-40C), where software verification activities are specified and their approach and the organization aspects in order to be performed are described, and the software verification report (SVR) (annex M ECSS-E-ST-40C) that gathers the results of all software verification activities executed according to the SVerP.
- Embedded software. In ISO 26262, this work product consists in the planning of software integration, where it shall be described the steps for integrating the individual software units hierarchically into software components until the embedded software is fully integrated. In ECSS, this work product corresponds to the following artefacts: (i) Software integration strategy; (ii) the software integration test plan; and (iii) the software integration test report.
- Configuration data, configuration data specification, calibration data and calibration data specification. In ECSS, all project items subject to the configuration management process shall be included in the configuration item list (see annex B ECSS-M-ST-40C) and in the configuration item data list (see annex C ECSS-M-ST-40C), where the current design status of a configuration item is provided. Concerning specific software configuration, ECSS specifies the software configuration file (who is a constituent of the design definition file, see annex E ECSS-M-ST-40C), where all software content related with configuration and calibration data is presented. Moreover, the software static architecture clause in the

design definition file (see annex F ECSS-E-ST-40C) shall specify the separated mission and configuration data, such as: (i) data resulting from the mission (data specific to each mission); (ii) reference data which are specific to a family or software product; (iii) reference data that do not change from mission to mission; (iv) data depending only on the specific mission requirements (e.g. calibration of sensors); and (v) data required for the software operation which only vary the higher level system design is changed.

# 7 ISO 26262 - Part 7: Production and operation

## 7.1 Production

In this clause, ISO 26262 specifies the production process requirements to achieve the following objectives:

- Develop and maintain a production process for safety-related elements or item that will be installed in road vehicles;
- Ensure that the responsible entities for the production process achieve functional safety during its execution.

ISO 26262 production process objectives are covered in the phase D(Qualification and Production) of ECSS lifecycle (see 4.4.3.6 ECSS-M-ST-10C). Regarding specific software product requirements for software delivery, installation and acceptance process see 5.7 ECSS-E-ST-40C and 6.3.6 ECSS-Q-ST-80C. In 4.3.2.4 ECSS-M-ST-40, ECSS specifies the product configuration baseline (PCB) (established at the functional configuration verification and the physical configuration verification) for serial production.

## 7.2 Operation service (maintenance and repair), and decommissioning

ECSS follows partially the same concept as ISO 26262 for the planning and execution of the processes of operation, maintenance, repair and decommissioning concerning the item, system or element. ECSS requirements and recommendations for these processes are specified in the phase E and F of the lifecycle (see 4.4.3.7 and 4.4.3.8 of ECSS-M-ST-10C). Concerning specific software requirements for these processes, see 5.9 and 5.10 of ECSS-E-ST-40C, and 6.3.7 and 6.3.8 of ECSS-Q-ST-80C.

## 7.3 Work products

ISO 26262 Part 7: Production and operation specifies the following work products:

- Safety-related content of the production and production control plan. In ECSS, general and software content of the production plan is specified in the production plan (referred in the system engineering plan, annex D ECSS-E-ST-10C), the software release document (see annex G ECSS-E-ST-40C) and the installation procedures and report. Regarding specific safety-related content, see the safety programme plan (annex B ECSS-Q-ST-40C) where safety activities are defined for each step of the product lifecycle.
- Control measures report. The equivalent artifact in ECSS is the Quality Assurance Plan (see annex A ECSS-Q-ST-20C).
- Assessment report for capability of the production process. ECSS, in 5.5.1a ECSS-Q-ST-20C, states that there should exist a manufacturing plan or flow chart for the product, where the information required by this ISO artifact shall be detailed.

- Safety-related content of the maintenance plan. In ECSS, although the maintenance plan is specified in the maintenance file, most of the safety content is described in the safety programme plan.
- Repair instruction. In ECSS, this work product is covered by the maintenance file.
- Safety-related content of the information made available to the user. In ECSS, all safety warnings that shall be made available to the end user are detailed in the product user manual (see annex P ECSS-E-ST-10C).
- Instruction regarding field observations. In ECSS, the artefact that has a similar purpose is the Operations Anomaly Report (see annex H ECSS-E-ST-70C [ECS08b]) which exists to document a departure from expected performance during operation of the system, either in the space and ground segment.
- Safety-related content of the instructions for decommissioning. This work product is covered in ECSS through the maintenance file where the decommissioning phase is defined. In addition, in the safety programme plan the safety concerns and activities regarding this phase are specified.
- Specification of requirements on the producibility, operation, service and decommissioning at system, hardware or software development level. Although these processes are described throughout the artefacts mentioned in this section, their individual requirements specification for product development at system or software level are also included in the following ECSS artefacts: (i) the system engineering plan (see annex D ECSS-E-ST-10C); (ii) the PAP (see annex A ECSS-Q-ST-10C) and SPAP (see annex B ECSS-Q-ST-80C); (iii) the software system specification (see annex B ECSS-Q-ST-40C); (iv) the software requirements specification (see annex D ECSS-Q-ST-40C); (v) the software operational support plan; and (vi) the software maintenance plan (see annex C ECSS-E-HB-40A). Regarding specific requirements for the product development at hardware level, see Section 5 ISO 26262 - Part 5: Hardware level.

```
8 ISO 26262 - Part 8: Supporting processes
```

This section contains the supporting processes requirements, covering the following clauses (each of the clauses is detailed in the next subsections):

- Interfaces within distributed developments;
- Specification and management of safety requirements;
- Configuration management;
- Change management;
- Verification;
- Documentation;
- Confidence in the use of software tools;
- Qualification of software components;
- Qualification of hardware components;
- Proven in use argument.

### 8.1 Interfaces within distributed developments

This clause objective is to describe the procedures and to allocate associated responsibilities within distributed developments for items and elements, presenting requirements for the following topics:

- Supplier selection criteria;
- Initiation and planning of distributed development;
- Execution of distributed development;
- Functional safety assessment at supplier's premise;
- After release for production.

These requirements are covered throughout ECSS-M-ST-10C, where the requirements for the customer and the supplier regarding the project planning and organization are specified. Business agreements (RFQ's,ITT's or RFP's), shall be specified in the project requirements documents and shall be included in the project planning as the customer responsibility (see 4.1.10 and 5.1 ECSS-M-ST-10C). In the acquisition process group of ECSS-Q-HB-80-02_Part2A, the supplier selection process is specified.

Concerning the interfaces between customer and supplier see 5.2.2 ECSS-M-ST-10C, where the requirements for communication and reporting between customer-supplier are defined. Moreover, each supplier in the costumer-supplier chain shall prepare and submit the PMP to his costumer for approval. The PMP shall state the purpose and provide a introduction to the project management system covering the topics defined in annex A of ECSS-M-ST-10C.

### 8.2 Specification and management of safety requirements

This clause defines the requirements for the correct specification and management of the safety requirements throughout the complete safety lifecycle.

In 5 of ECSS-Q-ST-40C, ECSS specifies the organizations safety responsibilities and the safety activities that shall be performed during the lifecycle. These ECSS requirements ensure a consistent management of the safety requirements throughout the lifecycle. Moreover, requirements and recommendations for a correct specification of the safety requirements are defined throughout ECSS-Q-ST-40C, specially clause 6 and 7.

### 8.3 Configuration and change management

ECSS specifies both the configuration and the change management in the project management plan (see 4.1.11 ECSS-M-ST-10C). ECSS-M-ST-40C present the requirements for these management processes, covering the ISO 26262 requirements (see clause 5 for the configuration management and clause 5.3.2 for the changes). ECSS specifies additionally that changes may be initiated by the customer (a change request shall be established) or by the supplier (change proposal). Regarding specific software implementation modification and configuration, requirements specified in 5.10.4 ECSS-E-ST-40C and in 6.2.4 ECSS-Q-ST-80C shall also be applied.

### 8.4 Verification

ECSS is in accordance with ISO 26262 for the verification process, specifying partially the same general requirements for the planning, specification, execution and evaluation of verification ac-

tivities (see ECSS-E-ST-10-02C). Regarding software verification, see the requirements specified in 6.2.6 ECSS-Q-ST-80C and 5.8 ECSS-E-ST-40C.

### 8.5 Documentation

ECSS specifies the documentation management as defined in ISO 26262 (see 4.3.8 and 5.3.7 ECSS-M-ST-40C). Moreover, this management process shall also be included in the project management plan.

### 8.6 Confidence in the use of software tools

ECSS specifies slightly different requirements for this clause, as ISO 26262 is more strict regarding the software tool evaluation and qualification. Tools involved in any activity of the development phase shall be identified by the supplier and agreed by the customer (see 5.6.1 ECSS-Q-ST-80C and 5.3.4 ECSS-E-ST-10C).

### 8.7 Qualification of software components

In ECSS, the qualification of components is specified in phase D of ECSS lifecycle (see 4.4.3.6 ECSS-M-ST-10C), where it is specified that a complete qualification testing and associated verification activities shall be performed, e.g., a qualification review. Regarding qualification reviews for software components see 5.3.4.4 ECSS-E-ST-40C.

### 8.8 Qualification of hardware components

For more information about the requirements regarding hardware components see Section 5 ISO 26262 - Part 5: Hardware level.

### 8.9 Proven in use argument

This clause specifies an alternative method of compliance with ISO 26262 for the reuse of existing items or elements when field data is available, i.e., the compliance with ISO 26262 can be applied to any type of product whose definition and conditions of use are identical or highly similar to a product that is already released and in operation.

In ECSS, regarding the reuse of existing software clause 6.2.7 of ECSS-Q-ST-80C shall be applied. This ECSS clause present the requirements for the reuse of existing software, specifying the analysis, assessments and methods that shall be applied to identify the level of compliance with the project requirements.

### 8.10 Work products

ISO 26262 Part 8: Supporting processes specifies the following work products:

- Supplier selection report. In ECSS, the supplier selection process and all the required and specified work products are described in the acquisition process group (see 4.1.2.1 ECSS-Q-HB-80-02_Part2A).
- Development interface agreement (DIA). For interface definition, ECSS requires the Interface Control Document which is specified in annex B ECSS-E-ST-10-24.
- Supplier's project and safety plan. These work products correspond to the supplier's PMP and safety programme plan.

- Functional safety assessment report. See the acquisition process group of 4.1.2.1 ECSS-Q-HB-80-02_Part2A.
- Supply agreement. See the acquisition process group of 4.1.2.1 ECSS-Q-HB-80-02_Part2A.
- Configuration management plan. ECSS follows the same approach as ISO 26260 for its configuration management plan (see annex A ECSS-M-ST-40C).
- Change management, request plan and report. In ECSS, the change management plan is integrated in the configuration management plan and it follows the same approach as ISO 26262. Moreover, Change requests and proposals are specified in their respective artefacts (see annexes G and H ECSS-M-ST-40C).
- Impact analysis. See section 3.5.
- Change request plan. In ISO 26262, this work product corresponds to the change evaluation in accordance with the impact analysis and follow up action, such as who shall perform the change and when. ECSS covers this work product in the configuration management plan, as it describes the change evaluation and follow up actions (change approval and implementation).
- Verification plan, specification and report. The objective of these ISO 26262 work products is in accordance with the following ECSS artefacts: (i) the verification plan (see annex A ECSS-E-ST-10-02C), where the verification activities are planned and specified; and (ii) the verification report (see annex F ECSS-E-ST-10-02C).
- Documentation management plan. In the configuration management plan, ECSS specifies roughly the same as ISO 26262 concerning the documentation management.
- Document guideline requirements. ECSS follows the same mindset as ISO 26262 as it specifies document guidelines in the information/documentation management clause of the configuration management plan.
- Software tool criteria evaluation report and software tool qualification report. ECSS follows a distinct mindset concerning software tools, as it shall be identified in the product assurance file all the tools that will be used for the product development lifecycle (the tools are identified by the supplier and agreed by the customer). Moreover, the tools selection shall also be justified in the product assurance file, where the following items shall be demonstrated though testing or documented assessment: (i) the development team has the appropriate experience or training to apply them; (ii) the tools are appropriate for the functional and operational characteristics of the product; and (iii) the tools are available in a appropriate environment for the entire lifetime of the development and maintenance processes of the product. As a side note, the product assurance file shall also include the verification and report for the correct use of the software tools.
- Software component documentation. This work product specifies all the documents related to the software component. In ECSS, this corresponds to annex A of ECSS-E-ST-40, where all software documents are identified.
- Software component qualification report. In ISO 26262, this work product shall document a set of information that is covered in the following ECSS artefacts and activities: (i) qualification review; (ii) software verification report; (iii) software configuration file; and (iv) in the configuration item.

- Safety plan. See Section 2.4.
- Qualification plan and report. In this ISO 26262 part, these work products are specific to the qualification of hardware components. Thus, see Section 5 ISO 26262 - Part 5: Hardware level.
- Hardware component test plan. For specific hardware information see Section 5 ISO 26262 - Part 5: Hardware level.
- Description of a candidate for proven in use argument. In ECSS, this work product is covered by the software reuse file (see annex N ECSS-E-ST-40C), where it is described the technical and management information about each item intended to be reused.
- Proven in use analysis reports. In ECSS, the software reuse file (SRF) documents the results of the software reuse analysis, where it shall be provided for each software item the information related with the decision to reuse or not, the level of reuse and the assumptions and methods applied when estimating the level of reuse. Moreover, the SRF shall also detail the evaluation results.

```
9 ISO 26262 - Part 9: Automotive Safety Integrity Level (ASIL) - oriented and safety␣
↪oriented analyses
```

```
9.1 Requirements decomposition with respect to ASIL tailoring
```

When performing ECSS criticality classification, continuing functional decomposition into lower-level functions is not considered as creating compensating provisions. Furthermore, software product criticality category is assigned according to the criticality of the most critical function that it implements (see 5.4 ECSS-Q-ST-40C).

```
9.2 Criteria for coexistence of elements
```

For the coexistence of mixed criticality components, ECSS specifies that all the involved components shall be classified at the highest criticality category among them (see 6.2.2.10 ECSS-Q-ST-80C).

```
9.3 Analysis of dependent failures
```

ISO 26262 specifies the analysis of dependent failures in order to identify single events or single causes that could neglect or invalidate a given requirements. ISO 26262 additionally defines that the potential of dependent failures shall be shall identified according with the safety analysis results.

ECSS dictates in the safety analysis the identification and analyse of multiple failures resulting from common-cause or common-mode (6.4.2 ECSS-Q-ST-40C) and due to failure propagation, all components shall be assigned with the highest criticality category among them (see 6.2.2.10 ECSS-Q-ST-80C).

As a side note, for analyses lower than system level, the severity due to possible failure propagation shall be identified as level 1 according to the dependability criteria (see 5.3.2 ECSS-Q-ST-30C).

```
9.4 Safety analysis
```

ECSS covers the ISO 26262 requirements concerning the safety analyses (see 7 ECSS-Q-ST-40C) by specifying requirements and methods that achieve the following ISO 26262 objectives:

- Evaluation of the consequences of faults and failures on the functions, behaviour and design of items and elements;
- Produce information on conditions and causes that could trigger the violation of safety goals or safety requirements;
- Identification of new hazards that were not identified during hazard analysis and risk assessment.

```
9.5 Work products
```

ISO 26262 Part 9: ASIL oriented and safety oriented analysis specifies the following work products:

- Update of architectural information. In ECSS, upon the modification of architectural information, the design definition file and design justification file shall be updated.
- Update of ASIL as attribute of safety requirements and elements, and of sub-elements of elements. In ECSS, although ASIL classification is not applied, ECSS criticality categorization level shall be updated upon mixed criticality and decomposition. Thus, the safety analysis report, the risk assessment report and the FMEA/FMECA report shall be updated.
- Safety analyses. This work product is an activity defined by the safety programme plan (see annex B ECSS-Q-ST-40C) and its results are presented in the safety analysis report (see annex D ECSS-Q-ST-40C).
- Analysis of dependent failures. In ECSS, this work product is included in the safety analysis activity (see Section 6.8).

### 8.3.1 Conclusions

Even though the application domain is different, there is a high degree of compatibility between both standards. ISO 26262 lifecycle is centered around the concept of an item, which can be interpreted as a system or a system of systems that implement a function at the vehicle level, while ECSS focuses on all the aspects pertaining to a space project from initiation to completion at all levels in the customer-supplier chain. Safety aspects are covered in both standards, however, ISO 26262 is more focused into the safety activities and requirements of each item than ECSS standards. Besides safety activities, ECSS also focuses on the processes.

Regarding the lifecycle tailoring, a minor discrepancy between both standards is observed, as ISO 26262 does not allow the removal of sub-phases, tasks or activities from the lifecycle (only the modification or the inclusion of new sub-phases, tasks or activities is allowed). Whereas ECSS allows their removal as long as it correctly recorded and justified.

In ISO 26262, the level of concern regarding the safety mechanisms and measures for the avoidance and control of systematic failures and random hardware failures during operation is higher when compared with the ECSS requirements. Moreover, in the glossary/vocabulary part ISO 26262 differs from ECSS by being more precise and less ambiguous (this can be observed in the definitions of similar terms).

Concerning the development of hardware parts, even though it was not analysed, in part because of the differences in the application domain and the target of this project, ISO 26262 is more focused than ECSS standards. This happens because in space projects the hardware used involves many components that are part of different segments (i.e., ground segment and space segment).

Besides the specific application domain and the development of hardware parts, another major discrepancy between both standards is the criticality classification. ISO 26262 follows the ASIL classification, where the following parameters are considered: (i) severity; (ii) likelihood; and (iii) controllability. ECSS specifies the function criticality according to the severity and likelihood of the related hazardous events.

In conclusion, despite the above mentioned differences, a final product that complies with ISO 26262 will not diverge from an ECSS final product with respect to safety aspects.

## 8.4 IEC Analysis

This section will perform the compatibility assessment between IEC 61508 and ECSS. It will start with the general requirements applicable to electrical/electronic/programmable electronic systems developed under IEC standard (IEC 61508-1) and then it will describe the specific requirements for software (IEC 61508-3). The IEC 61508-1 section requirements analysis is made in a summary manner, unless an higher level of detailed is required (when IEC 61508-3 does not cover the specific topic and it is also addressed in ECSS-E-ST-40C/ECSS-Q-ST-80C).

### 8.4.1 IEC 61508-1

```
4 Conformance to this standard
```

These clauses states that compliance to the relevant requirements of this standard shall be demonstrated. And also states the possibility for a tailoring of the standard, according with the project complexity. This is addressed in ECSS at system level (see ECSS-S-ST-00C, ECSS-E-ST-10C, ECSS-Q-ST-10C and ECSS-M-ST-10C). For software, this is according with ECSS, which has similar requirements, requesting for compliance with its own requirements (see 5.3.9 ECSS-E-ST-40C and 5.2.1.5 ECSS-Q-ST-80C). Also, in ECSS software standards, it is presented the possible tailoring, according with the software criticality (see Annex R of ECSS-E-ST-40C and Annex D.2 of ECSS-Q-ST-80C).

```
5 Documentation
```

```
5.1 Objectives
```

This section does not contain requirements.

```
5.2 Requirements
```

These clauses relate with the quality of the produced documentation and generic configuration management requirements. They are according with with ECSS, which addresses this topic

together for both system and software level (see 5 ECSS-M-ST-40C). For software, ECSS specify additional requirements (see 5.8.3.10 ECSS-E-ST-40C), which are also according with this section.

```
6 Management of functional safety
```

```
6.1 Objectives
```

This section does not contain requirements.

```
6.2 Requirements
```

These clauses relate with the project organization with emphasis on the measures to be undertaken to assure that the project is developed with the necessary safety level. This is addressed in ECSS at system level (see 5 ECSS-Q-ST-10C). Also, at software level, all these requirements are according with ECSS, which also presents similar requirements to assure the safety aspects of the software (see section 5 ECSS-Q-ST-80C).

```
7 Overall safety lifecycle requirements
```

```
7.1 General
```

```
7.1.1 Introduction
```

This section does not contain requirements.

```
7.1.2 Objectives and requirements \- general
```

This section does not contain requirements.

```
7.1.3 Objectives
```

This section does not contain requirements.

```
7.1.4 Requirements
```

These requirements specify the necessary steps for the safety life-cycle of the project at system level. This is addressed in ECSS at system level (see 5.4 ECSS-E-ST-10C, which re-directs to ECSS-M-ST-10C). See analysis in 8.4.2, section 7.1.2 for the specific requirements for the software life-cycle.

```
7.2 Concept
```

```
7.2.1 Objective
```

This section does not contain requirements.

```
7.2.2 Requirements
```

These requirements state that the environment in which the system operates shall be assessed and taken into account when performing the hazard and possible failures analysis. At system

level this is addressed in ECSS (see 5.3.2 ECSS-E-ST-10C and standards ECSS-Q-ST-30C, ECSS-Q-ST-40C and ECSS-Q-ST-40-02C). For software this analysis is also required in ECSS (see 6.2.2 ECSS-Q-ST-80C, which also re-directs to ECSS-Q-ST-30C and ECSS-Q-ST-40C).

```
7.3 Overall scope definition
```

```
7.3.1 Objectives
```

This section does not contain requirements.

```
7.3.2 Requirements
```

These requirements are in line with section 7.2.2, but with more emphasis on the interactions between the environment and the system to be developed. This is according with ECSS at system (see 5.3.2 ECSS-E-ST-10C, ECSS-Q-ST-30C and ECSS-Q-ST-40C) and software levels (see 6.2.2 ECSS-Q-ST-80C, which also re-directs to ECSS-Q-ST-30C and ECSS-Q-ST-40C).

```
7.4 Hazard and risk analysis
```

```
7.4.1 Objectives
```

This section does not contain requirements.

```
7.4.2 Requirements
```

These requirements define what content shall be present in hazard analysis. This is according with ECSS (see ECSS-Q-ST-40-02C). Note also the hazard analysis shall include the software contribution (see 6.2.2 ECSS-Q-ST-80C).

```
7.5 Overall safety requirements
```

```
7.5.1 Objective
```

This section does not contain requirements.

```
7.5.2 Requirements
```

This section describes the requirements to specify the overall system requirements, with emphasis in hazard functions. This is addressed in ECSS at system level (see 5.2 ECSS-E-ST-10C and ECSS-E-ST-10-06C). For software level see analysis in 8.4.2, sections 7.1.2 and 7.2.2.

```
7.6 Overall safety requirements allocation
```

```
7.6.1 Objectives
```

This section does not contain requirements.

```
7.6.2 Requirements
```

This section describes the necessary requirements to translate the system requirements into system design. This addressed in ECSS at system level (see 5.4 ECSS-E-ST-10C). For software

level see analysis in 8.4.2, sub-sections of 7.4. Note that ECSS does not define targets for failures probability after applying the hazard reduction measures. This has to be taken into account, if a future qualification to this standard is foreseen.

`7.7 Overall operation and maintenance planning`

`7.7.1 Objective`

This section does not contain requirements.

`7.7.2 Requirements`

This section targets the necessary requirements to be taken into account for operation and maintenance of the system planning. This is addressed in ECSS at system level (see ECSS-E-ST-10C Annexes, which require the operation and maintenance to be described in the system documents). For software level see analysis in 8.4.2, section 7.8.2.

`7.8 Overall safety validation planning`

`7.8.1 Objective`

This section does not contain requirements.

`7.8.2 Requirements`

This section requirements address the validation planning. This is addressed in ECSS at system level (see 5.5 ECSS-E-ST-10C and ECSS-E-ST-10-03C). For software level see analysis in 8.4.2, section 7.3.2.

`7.9 Overall installation and commissioning planning`

`7.9.1 Objectives`

This section does not contain requirements.

`7.9.2 Requirements`

This section specifies the necessary requirements for the system installation and commissioning (acceptance) planning. This is addressed in ECSS at system level (see ECSS-E-ST-10C Annexes, which require the installation and commissioning to be described in the system documents). Since IEC 61508-3 does not provide specific requirements for installation, these system-level requirements are analyzed against the ECSS software requirements:

- 7.9.2.1 - This is according with ECSS (see 5.7.2.3 ECSS-E-ST-40C and 6.3.6.1 and 6.3.6.1 ECSS-Q-ST-80C).
- 7.9.2.2 - This is according with ECSS (see 5.7.3.1 ECSS-E-ST-40C and 6.3.6.3 ECSS-Q-ST-80C), although more detailed in this standard.
- 7.9.2.3 - This is according with ECSS (see 5.7.2.3 and 5.7.3.1 ECSS-E-ST-40C).

7.10 E/E/PE system safety requirements specification

7.10.1 Objective

This section does not contain requirements.

7.10.2 Requirements

This section specify the requirements for the requirements which result of a refinement (lower-level requirements) of the project high-level requirements (section 7.6). This is addressed in ECSS at system level (see 5.2 ECSS-E-ST-10C and ECSS-E-ST-10-06C). For software level see analysis in 8.4.2, section 7.2.2.

7.11 E/E/PE safety-related systems \- realisation

7.11.1 Objective

This section does not contain requirements.

7.11.2 Requirements

See section 8.4.2.

7.12 Other risk reduction measures  specification and realisation

7.12.1 Objective

This section does not contain requirements.

7.12.2 Requirements

This section does not contain requirements. As stated in this section other risk reduction measures, which are based on other technology which is not electrical/electronic/programmable electronic equipment, is out of scope of this standard and also is not covered by ECSS software standards.

7.13 Overall installation and commissioning

7.13.1 Objectives

This section does not contain requirements.

7.13.2 Requirements

This section specifies the necessary requirements for the system installation and commissioning (acceptance) execution. This is addressed in ECSS at system level (see ECSS-E-ST-10C Annexes - the installation and commissioning shall follow the plans referred in these Annexes). Again, since IEC 61508-3 does not provide specific requirements for installation, these system-level requirements are analyzed against the ECSS software requirements:

- 7.13.2.1 - This is according with ECSS (see 6.3.6.2 ECSS-Q-ST-80C).

- 7.13.2.2 - This is according with ECSS (see 5.7.2.4 ECSS-E-ST-40C).
- 7.13.2.3 - This is according with ECSS (see 6.3.6.4 ECSS-Q-ST-80C).
- 7.13.2.4 - This is according with ECSS (see 6.3.6.7 and 6.3.6.9 ECSS-Q-ST-80C).

### 7.14 Overall safety validation

#### 7.14.1 Objective

This section does not contain requirements.

#### 7.14.2 Requirements

This section describes the requirements for the validation activities. This is addressed in ECSS at system level (see 5.5 ECSS-E-ST-10C and ECSS-E-ST-10-03C). For software level see analysis in 8.4.2, section 7.7.2.

### 7.15 Overall operation, maintenance and repair

#### 7.15.1 Objective

This section does not contain requirements.

#### 7.15.2 Requirements

This section describe the requirements to be applied during operation and maintenance activities. This is addressed in ECSS at system level (see ECSS-E-ST-10C Annexes - the operation, maintenance and repair shall follow the procedures described in the system documents). For software level see analysis in 8.4.2, section 7.8.2.

### 7.16 Overall modification and retrofit

#### 7.16.1 Objective

This section does not contain requirements.

#### 7.16.2 Requirements

This section specifies the requirements when performing modifications to the product. In ECSS this falls also in maintenance. This is addressed in ECSS at system level (see ECSS-E-ST-10C Annexes - the modification and retrofit shall follow the procedures described in the system documents). For software level see analysis in 8.4.2, section 7.8.2.

### 7.17 Decommissioning or disposal

#### 7.17.1 Objective

This section does not contain requirements.

#### 7.17.2 Requirements

This section specifies the requirements when performing decommissioning (retirement) of the product. This is addressed in ECSS at system level (see 4.3.8 ECSS-E-ST-10C). In ECSS, at system level, this activity is referred as end of life Since IEC 61508-3 does not provide specific requirements for Decommissioning/disposal, these system-level requirements are analyzed against the ECSS software requirements (in ECSS software standard, this activity is referred as Software retirement):

- 7.17.2.1 - From a software point of view, this is according with ECSS (see 5.10.7.1 ECSS-E-ST-40C).
- 7.17.2.2 - This is according with ECSS (see 5.10.7.1 ECSS-E-ST-40C).
- 7.17.2.3 - This is implicit in ECSS (see 5.10.7.2 ECSS-E-ST-40C, note the output of this clause is a “Retirement notification”).
- 7.17.2.4 - This is implicit in ECSS (see 5.10.7.2 ECSS-E-ST-40C).
- 7.17.2.5 - From a software point of view, this is according with ECSS (see 5.10.7.1 ECSS-E-ST-40C).
- 7.17.2.6 - This is implicit in ECSS. Although ECSS (software) only specifies that a replacing software to be identified (see 5.10.7.2 ECSS-E-ST-40C), but does not provide any requirement which relates the software with other systems.
- 7.17.2.7 - This is implicit in ECSS (see 5.10.7.2 ECSS-E-ST-40C).

```
7.18 Verification
```

```
7.18.1 Objective
```

This section does not contain requirements.

```
7.18.2 Requirements
```

This section describes the requirements for the verification planning and execution. This is addressed in ECSS at system level (see 5.5 ECSS-E-ST-10C and ECSS-E-ST-10-02C). For software level see analysis in 8.4.2, section 7.9.2.

```
8 Functional safety assessment
```

```
8.1 Objective
```

This section does not contain requirements.

```
8.2 Requirements
```

This section specifies the requirements to perform independent verification of the product. For system level it was not found any document addressing this topic. From software point of view, this section is according with ECSS (see 6.2.6.13 ECSS-Q-ST-80C and ESA Guide for Independent Software Verification and Validation). Note that ECSS standard is more strict and specific in what concerns the necessary independence and the necessary work to be performed in terms of verification. The IEC consequences B, C and D all of them fall in the ECSS Catastrophic category and consequence A (the least severe in IEC) falls into ECSS Critical category (see 4.2

ECSS-Q-ST-30-02C). ECSS also defines levels of independent verification, which require different level of verification depth (for example ECSS ISVV level 2 requires the requirements consistency to be verified using modeling or formal methods, whereas in level 1 inspection is enough) when performing the analysis. For the remaining topics, as stated above, they are according with ECSS, with the ECSS being more complete and specific on what work shall be done. As a conclusion, since ECSS is more restrict, an independent verification performed under ECSS will be compatible with an IEC independent verification.

### 8.4.2 IEC 61508-3

```
4 Conformance to this standard
```

See section 8.4.1.

```
5 Documentation
```

See section 8.4.1.

```
6 Additional requirements for management of safety-related software
```

```
6.1 Objectives
```

This section does not contain requirements.

```
6.2 Requirements
```

These clauses specify the requirements for the Configuration Management of a software project. See below the analysis of each of these clauses against the ECSS standard.

- 6.2.1 - See 8.4.1, section 6.2 analysis.
- 6.2.2 - This is according with ECSS (see 5.3 ECSS-E-ST-40C), which also requires a planning for the software development. In ECSS, this planning is realised in the Software Development Plan (see Annex O ECSS-E-ST-40C).
- 6.2.3 - See each point below:
  a. This is according with ECSS (see 5.3.2.1, 5.3.2.4 and 5.3.2.5 ECSS-M-ST-40C and 6.2.4.6 ECSS-Q-ST-80C), which also requires software changes procedures to be defined and applied through the software project life-cycle.
  b. This is according with ECSS (see 5.3.4 ECSS-M-ST-40C), which also requires the verification of the software product from a configuration management point of view.
  c. This is according with ECSS (see 5.3.1 ECSS-M-ST-40C and 6.2.4.11 ECSS-Q-ST-80C), which also requires correct labeling of all software items.
  d. This is according with ECSS (see 5.3.2.1 ECSS-M-ST-40C and 6.2.4.6 ECSS-Q-ST-80C), which also requires software changes to be authorized and controlled.
  e. This is implicit in ECSS (see 5.3.1.3 ECSS-M-ST-40C and 6.2.4.7 ECSS-Q-ST-80C), which requires that the information on how to install/operate shall be available.

f. This is according with ECSS, although more detailed. The ECSS only states that the necessary information for performing audits shall be available (see 5.3.5 and <4.8>, Annex A ECSS-M-ST-40C and 5.2.3 ECSS-Q-ST-80C).

g. This according with ECSS (see 5.3.7.5 and 5.3.7.6 ECSS-M-ST-40C and 6.2.4.2 ECSS-Q-ST-80C), which also requires means of permanent storage of the software product.

# 7 Software safety lifecycle requirements

## 7.1 General

### 7.1.1 Objective

This section does not contain requirements.

### 7.1.2 Requirements

These clauses specify the requirements to define and apply a life cycle to software development. See below the analysis of each of these clauses against the ECSS standard. Note that ECSS specifies, in addition, specific life cycle requirements for autocoded software.

- 7.1.2.1 - See 8.4.1, section 6.2 analysis.
- 7.1.2.2 - This is according with ECSS (see 5.3.2.1 ECSS-E-ST-40C), which states that the life-cycle shall be chosen according with it is more appropriate for the project.
- 7.1.2.3 - This is according with ECSS (see 5.3.2.1 ECSS-E-ST-40C and 6.1.1 ECSS-Q-ST-80C), which also requires each phase of the project life-cycle to be detailed specified.
- 7.1.2.4 - This is implicit in ECSS (see 5.3.2.1 ECSS-E-ST-40C), the software life-cycle may be adapted according with the project needs, but still the ECSS requirements shall be meet within the project.
- 7.1.2.5 - This is implicit in ECSS (see 5.3.2.1 ECSS-E-ST-40C), the tailoring of the software life-cycle shall be justified taking into account the project criticality level.
- 7.1.2.6 - This is according with ECSS (see 5.2.2.1, 5.2.4.8 and 5.4.2.1 ECSS-E-ST-40C and 5.2.7, 6.1.2 and 6.2.2 ECSS-Q-ST-80C), which also requires the project to have quality and safety requirements and procedures.
- 7.1.2.7 - This is according with ECSS (see 5.3.2.1 ECSS-E-ST-40C and 6.2.3 ECSS-Q-ST-80C), which states also that the techniques and measures should be defined, with special emphasis to to critical software. However ECSS does not specify explicitly which techniques are applicable, depending on the software criticality level, as in Annexes A and B of this document.
- 7.1.2.8 - This is according with ECSS (see 5.3.2.1 ECSS-E-ST-40C), which requires the documentation of the outcome of each life cycle activity.
- 7.1.2.9 - This is implicit in ECSS (see in general 5.3.2 ECSS-E-ST-40C and 6.1 ECSS-Q-ST-80C). Since some outputs of life cycle phases are inputs of later phases, then a re-assessment of a later phase activity may require the respective adaptation of its dependent(s) earlier phases.

## 7.2 Software safety requirements specification

### 7.2.1 Objectives

This section does not contain requirements.

### 7.2.2 Requirements

These clauses specify the requirements for the software requirements with emphasis in safety requirements. See below the analysis of each of these clauses against the ECSS standard.

- 7.2.2.1 - This is implicit in ECSS (see 5.2.2.1 ECSS-E-ST-40C), if the software system requirements are already specified in the system requirements, the Software System Specification (see Annex B) should point to the System Requirements. However, it should be verified if the software requirements in System requirements covers what is required by ECSS-E-ST-40C Annex B.
- 7.2.2.2 - This is according with ECSS (see 5.2.2.1 ECSS-E-ST-40C).
- 7.2.2.3 - This is according with ECSS (see 5.4.2.1 ECSS-E-ST-40C), referring to the Technical specification.
- 7.2.2.4 - This is according with ECSS (see 5.2.2.1 ECSS-E-ST-40C and 6.3.1.3 ECSS-Q-ST-80C), which requires also a failure analysis.
- 7.2.2.5 - This clause states the type of requirements that should present in the requirements specification. These type of requirements are also present in ECSS (see ), as follows:
  a. <5.13> Software safety requirements (Annex D ECSS-E-ST-40C)
  b. <5.7> Design requirements and implementation constraints (Annex D ECSS-E-ST-40C)
  c. <5.5> Operational requirements/<5.6> Resources requirements (Annex D ECSS-E-ST-40C). Although in ECSS it is not intended to provide pure hardware requirements, but requirements which concerns the relation between hardware and software.
  d. <5.2> Functional requirements (Annex D ECSS-E-ST-40C)
  e. <5.3> Performance requirements (Annex D ECSS-E-ST-40C)
  f. <5.4> Interface requirements/<5.16> Human factors related requirements (Annex D ECSS-E-ST-40C)
- 7.2.2.6 - This is according with ECSS (see 5.4.2.1 and Operational requirements in Annex D, ECSS-E-ST-40C)
- 7.2.2.7 - This is according with ECSS (see 5.4.2.1 and safety requirements and Design requirements and implementation constraints in Annex D, ECSS-E-ST-40C)
- 7.2.2.8 - The points a. and b. are in accordance with ECSS (see 5.2.2.2 and 5.8.3.1 ECSS-E-ST-40C and 6.2.3 ECSS-Q-ST-80C) The points c., d. and e. are partially according with ECSS (see same sections above). ECSS states for the need for capabilities monitoring and fault recovery mechanisms, but does not specify that the software shall have testing functions embedded in the software itself (i.e: testing code in the software).

- 7.2.2.9 - This is according with ECSS (see 6.3.1.3 and 6.3.2.4 ECSS-Q-ST-80C), although ECSS states this requirement in the opposite direction: the safety requirements shall be clearly identified.
- 7.2.2.10 - See analysis below
  a. This is according with ECSS (see 5.8.3.1 ECSS-E-ST-40C), although more detailed in this standard.
  b. This is according with ECSS (see 6.2.2.10 ECSS-Q-ST-80C), in addition in ECSS it is stated if this independence is not possible, “then all the involved components shall be classified at the highest criticality category among them”.
- 7.2.2.11 - This is according with ECSS (see 5.8.3.1 ECSS-E-ST-40C). In ECSS, this is materialized in the interface definitions (see 5.4.3.5 and Annex E ECSS-E-ST-40C)
- 7.2.2.12 - This according with ECSS (see 5.2.4.4, 5.4.3.5 and Annex E ECSS-E-ST-40C). The points c., d. and e. are implicit in ECSS.
- 7.2.2.13 - This is according with ECSS (see 6.2.3.2 and 6.2.4 ECSS-Q-ST-80C), being ECSS more generic and not specifying this measures only to operational parameters.

## 7.3 Validation plan for software aspects of system safety

### 7.3.1 Objective

This section does not contain requirements.

### 7.3.2 Requirements

These clauses specify the requirements to for the software validation plan. See below the analysis of each of these clauses against the ECSS standard.

- 7.3.2.1 - This is according with ECSS (see 5.6.3.1 and 5.6.4.1 ECSS-E-ST-40C and 7.2.3 ECSS-Q-ST-80C). Note that ECSS requires a test plan for both Technical Specification (lower level requirements) and Requirements Baseline (upper level requirements).
- 7.3.2.2 - See analysis below:
  a. In ECSS it is specified that the validation shall take place in specific phases of the software development:
     – CDR - Technical Specification validation (see 5.6.3.4 ECSS-E-ST-40C)
     – QR - Requirements Baseline validation (see 5.6.4.4 ECSS-E-ST-40C)

     Having the possibility to have other validation life-cycle strategy is acceptable, as long as in the end of the project the validation activity goals are achieved.
  b. ECSS is more specific and restrictive on the personal who shall perform the validation, specially on what concerns the level of independence (see 5.6.2.1 and 5.6.2.2 ECSS-E-ST-40C and 6.3.5.19 ECSS-Q-ST-80C)
  c. This is according with ECSS (see 7.2.3.5 ECSS-Q-ST-80C)
  d. This is according with ECSS. In ECSS the validation takes place before the commissioning (operational phase), see point a. analysis. In addition ECSS states that the

validation shall take place in a configuration which replicates the operational scenario (see 5.6.3.1 and 5.6.4.1 ECSS-E-ST-40C)

e. This is according with ECSS (see 5.6.3.1 and 5.6.4.1 ECSS-E-ST-40C)

f. See point e.

g. This is according with ECSS (see 5.6.3.1 and 5.6.4.1 ECSS-E-ST-40C and 7.2.3.5 ECSS-Q-ST-80C)

h. This is according with ECSS (see 5.6.3.1 and 5.6.4.1 ECSS-E-ST-40C)

i. This is according with ECSS (see 6.3.5.6, 6.3.5.8 and 7.2.3.3 ECSS-Q-ST-80C)

- 7.3.2.3 - This is according with ECSS (see 5.6.2.1 ECSS-E-ST-40C and 7.2.3.1 ECSS-Q-ST-80C), although this standard is more specific on the option strategies which may be followed.
- 7.3.2.4 - This is according with ECSS, in general all activities workflow shall be agreed with the customer/assessor (see 5.6.1.1 ECSS-Q-ST-80C). ECSS foresees an Acceptance Test phase, where the customer/assessor performs the tests itself (see 5.7.3.1 and 5.7.3.2 ECSS-E-ST-40C)
- 7.3.2.5 - This is according with ECSS (see 5.6.3.1, 5.6.4.1 and Annex J ECSS-E-ST-40C )

```
7.4 Software design and development
```

```
7.4.1 Objectives
```

This section does not contain requirements

```
7.4.2 General requirements
```

These clauses specify the guidelines for software development and reuse. See below the analysis of each of these clauses against the ECSS standard.

- 7.4.2.1 - This requirement is not specified in ECSS. Although, this is implicit, since the project may decide for each functionality whether it is more appropriate to implement at hardware or at software level
- 7.4.2.2 - This is according with ECSS (see 6.3.3 ECSS-Q-ST-80C), although more detailed in this standard what should be considered in the design of software
- 7.4.2.3 - This is implicit in ECSS (see 6.3.3 ECSS-Q-ST-80C), the software shall be testable and programming practices which difficult software maintenance shall be avoided.
- 7.4.2.4 - This is implicit in ECSS (see 5.2.7.2 ECSS-Q-ST-80C). The software shall be designed to allow maintainability, this is part of ECSS project's Quality Model.
- 7.4.2.5 - This is implicit in ECSS for design (see 6.3.3.2 ECSS-Q-ST-80C) and referred for the coding phase (see 6.3.4.1 ECSS-Q-ST-80C).
- 7.4.2.6 - This is according with ECSS (see 6.3.3.5 and 6.3.3.6 ECSS-Q-ST-80C).
- 7.4.2.7 - This is according with ECSS (see 6.2.3.2 ECSS-Q-ST-80C), for the critical software design techniques. Note that ECSS provides this measure as a suggestion, although

it is implicit that for critical software it shall be applied, unless justified that it may/can not be applied and the necessary measures are taken to compensate.

- 7.4.2.8 - This is according with ECSS (see 6.2.2.10 ECSS-Q-ST-80C)
- 7.4.2.9 - This is related with the previous point, and again according with ECSS (see 6.2.2.10 ECSS-Q-ST-80C). Note however, that ECSS requires that the critical components to be reduced the minimum possible (see 6.2.2.4 ECSS-Q-ST-80C).
- 7.4.2.10 - This is according with ECSS (see 6.2.2.1 ECSS-Q-ST-80C, which redirects to 6.5.6.3 ECSS-Q-ST-40).
- 7.4.2.11 - This project does not have access to IEC 61508-2, however this clause is related to the previous one, so it can be assumed that the IEC 61508-2 requirements are in accordance with ECSS in what concerns using elements to compensate software modules with level lower than the required by the project. This is a common practice in ESA projects, since in most projects, implementing SCC A (maximum criticality) software is much resource consuming.
- 7.4.2.12 - See analysis below:
  a. Each route is treated in ECSS as below:
     – For route1, this is according with ECSS (see see 6.2.7.3, 6.2.7.4 and 6.2.7.6 ECSS-Q-ST-80C)
     – For route2, this is according with ECSS, with ECSS requiring additionally generation of “validation and verification documents” and “execution of tests in order to achieve the required level of test coverage” (see 6.2.7.8 ECSS-Q-ST-80C)
     – For route3, this standard specifies in clause 7.4.2.13, the minimum requirements for the non-compliant software to be still admissible to use. ECSS is more restrict, by requiring for a non-compliant software to perform reverse engineering, in order for the reused software to meet the project requirements (see 6.2.7.7).
  b. This is according with ECSS (see 6.2.7.5 ECSS-Q-ST-80C). The reused software shall be documented in the Software Reuse File document (see Annex N ECSS-E-ST-40), which contains all relevant information of the reused software.
- 7.4.2.13 - This clause was already discussed in point above (see 7.4.2.12 a. third point).
- 7.4.2.14 - This specific case is not addressed explicitly by ECSS. However, any configuration data for hardware shall be considered as software and follow the same rules as “normal” software.

### 7.4.3 Requirements for software architecture design

These clauses specify the guidelines for the software architectural. See below the analysis of each of these clauses against the ECSS standard.

- 7.4.3.1 - This clause is repeated of 7.4.4.19. See analysis in respective section.
- 7.4.3.2 - See analysis below:
  a. This is according with ECSS (see 5.4.3.1 and 5.4.3.2 ECSS-E-ST-40C and 6.3.3.2 and 6.3.3.5 ECSS-Q-ST-80C).

b. This is implicit in ECSS (see 5.4.3.1 and 5.8.3.3 ECSS-E-ST-40C). See specific comments to the points below:

   1. This is not explicitly stated in ECSS, however the information about external elements/subsystems shall be available and comply with the whole software specifications (see also 5.4.3.6 and 5.4.3.7 for activities for software reuse)

   2. This can be deduced from the requirements allocated to the elements. Components implementing critical requirements inherit the criticality of the respective requirements.

   3. According with ECSS, as stated above.

c. This is according with ECSS (see 5.4.3.1 ECSS-E-ST-40C).

d. This is implicit in ECSS, with the choice of design standard (see 5.4.3.2 ECSS-E-ST-40C and 6.3.3.2 and 6.3.3.4 ECSS-Q-ST-80C).

e. This according with ECSS (see 5.8.3.3 ECSS-E-ST-40C).

f. This is according with ECSS (see 5.4.3.8 ECSS-E-ST-40C).

- 7.4.3.3 - This is implicit in ECSS, which allows changes to baseline to be performed, but controlled by the configuration management (see 5.3.2.5 and 5.3.3.1 ECSS-E-ST-40C).

`7.4.4 Requirements for support tools, including programming languages`

These clauses specify the requirements for the software support tools.

Introductory note: in the absence of the IEC 61508-4 for this work, a search in google was performed to clarify the meaning of on-line, off-line tools and the *Tx* categories used. This is what was taken from EngineerZone:

- "Online tools which run as part of the application and offline tools used during the development or manufacturing phases."
- "T1 - tools which have no impact on the executable code. The examples given in IEC 61508-4:2010 include text editors and requirements management tools. Perhaps a description more consistent with the examples given in the text are tools that are not used to produce the code or verify the code but even then it is hard to argue that a text editor is only a T1 tool."
- "T2 - tools which only impact on the verification of the executable code and can't inject an error into the code but could cause an error to be missed e.g. static timing analysis tools"
- "T3 - tools which can put an error in the code e.g. compilers"

See below the analysis of each of these clauses against the ECSS standard.

- 7.4.4.1 - This is according with ECSS (see 6.2.7.1 ECSS-Q-ST-80C). Software on-line tools are considered as reused software (clause 6.2.7.1 applies to all software except tools and software development environment).
- 7.4.4.2 - This is according with ECSS (see 5.6.1.2 ECSS-Q-ST-80C).
- 7.4.4.3 - This is according with ECSS (see 5.6.1.2 ECSS-Q-ST-80C).
- 7.4.4.4 - This is according with ECSS (see 5.6.2.1 ECSS-Q-ST-80C).

- 7.4.4.5 - This is according with ECSS (see 5.6.1.2 and 5.6.2.1 ECSS-Q-ST-80C).
- 7.4.4.6 - This is according with ECSS (see 5.6.2.1 ECSS-Q-ST-80C).
- 7.4.4.7 - This standard is more specific in what information should be available on the tool performed validation (see against 5.6.2.1 ECSS-Q-ST-80C). However, since this information is in line with the ECSS necessary validation information for a normal software development.
- 7.4.4.8 - This clause is not foreseen in ECSS (i.e: ECSS does not foresees compensating measures to be applicable to the software itself due to a potential bug in the tool). If the tool has potential failures that could impact the software, techniques as described in 6.3 ECSS-Q-HB-80-01A shall be applied, in order to make the tool compliant with the software under development.
- 7.4.4.9 - This is according with ECSS (see 5.6.2.1 ECSS-Q-ST-80C).
- 7.4.4.10 - This is according with ECSS, although more detailed in this standard (see 5.6.1.2, 5.6.2.1, 5.6.2.2 and 6.2.3.2 ECSS-Q-ST-80C). Note in addition, ECSS requires the justification of low-level languages usage (see 6.3.4.5 ECSS-Q-ST-80C)
- 7.4.4.11 - This is implicit in ECSS (see 5.6.1.2, 5.6.2.1, 5.6.2.2 ECSS-Q-ST-80C). If the language by itself is not alone fully adequate for the software project, the necessary compensating measures shall be identified and justified.
- 7.4.4.12 - This is according with ECSS (see 6.3.4.1 and 6.3.4.2 ECSS-Q-ST-80C).
- 7.4.4.13 - This is according with ECSS (see 6.3.4.1, 6.3.4.2 and 6.2.3.2 ECSS-Q-ST-80C), except that for the specific required information in the source code listed in this requirement, ECSS is not specific.
- 7.4.4.14 - This is according with ECSS (see 6.2.8.1 ECSS-Q-ST-80C).
- 7.4.4.15 - This according with ECSS (see 5.3.1.3 ECSS-M-ST-40C and 6.2.4.5 ECSS-Q-ST-80C), although less detailed in ECSS. Since the goal of configuration management is to allow to reproduce any software version (see 6.2.4.2 ECSS-Q-ST-80C), all means necessary to do it, shall be available.
- 7.4.4.16 - In ECSS this requirement does not fall into the scope of the configuration management, but instead on the quality assurance, in addition, they should be agreed with the customer (see 5.6.1.1, 5.6.1.2 and 5.6.2.1 ECSS-Q-ST-80C)
- 7.4.4.17 - As with clause 7.4.4.16, in ECSS this task is within the quality assurance domain (see 5.6.2.1 ECSS-Q-ST-80C)
- 7.4.4.18 - This is not explicitly stated in ECSS. However, the updated tool shall be considered as new tool and follow the same verification as the old version tool. This however, does not eliminate the possibility of reusing verification done for the old version tool. Note that a word is worthwhile to provide here: the re-verification process of updated tools shall be done with the same rigor as done for the earlier version. Even an apparently minor change may have the impact to introduce bugs or change the behavior of the tool.
- 7.4.4.19 - This is implicit in ECSS, where the definition of the tools and the necessary verification to make sure they meet the project needs is made by the engineering team. The quality assurance team shall verify if the choices and the necessary measures to make sure the tools meet the project needs are applied.

More detailed information about tools can be also found in ECSS-Q-HB-80-01A.

`7.4.5 Requirements for detailed design and development  software system design`

These clauses specify the guidelines for the software detailed design and development. See below the analysis of each of these clauses against the ECSS standard.

- 7.4.5.1 - This is implicit in ECSS: the engineering team is responsible to apply the necessary measures for the correct detailed design and the quality assurance to verify the compliance of detailed design with the project specifications (see 6.3.3.5 and 6.3.3.6 ECSS-Q-ST-80C).
- 7.4.5.2 - This is according with ECSS: the specification of requirements (see 5.4.2.1 ECSS-E-ST-40C), the software architecture design (see 5.4.3.1 ECSS-E-ST-40C) and the validation plan (see 5.6.2.1 ECSS-E-ST-40C) shall be made available in the Preliminary Design Review phase which is held immediately before the Detailed Design Review phase (in the indicated ECSS sections see “EXPECTED OUTPUT”, where it is indicated the document output of the activities and the respective phase).
- 7.4.5.3 - This is according with ECSS (see 5.8.3.4 ECSS-E-ST-40C).
- 7.4.5.4 - This is according with ECSS (see 5.5.2.1 and 5.5.2.9 ECSS-E-ST-40C).
- 7.4.5.5 - This is according with ECSS (see 5.5.4.1 ECSS-E-ST-40C).

`7.4.6 Requirements for code implementation`

This clause specifies the guideline for the software coding. See below the analysis of each of these clauses against the ECSS standard.

- 7.4.6.1 - This is according with ECSS, although less detailed in this standard (see 5.8.3.5 ECSS-E-ST-40C). For 7.4.4 and 7.4.2 analysis report, see the respective sections.

`7.4.7 Requirements for software module testing`

These clauses specify the guidelines for the software module testing. See below the analysis of each of these clauses against the ECSS standard.

- 7.4.7.1 - This is according with ECSS (see 5.5.3.2 ECSS-E-ST-40C)
- 7.4.7.2 - This is according with ECSS (see 5.5.3.2 ECSS-E-ST-40C)
- 7.4.7.3 - This is according with ESSS (see 5.8.3.6 ECSS-E-ST-40C)
- 7.4.7.4 - This is according with ECSS (see 7.2.3.3 ECSS-Q-ST-80C)

`7.4.8 Requirements for software integration testing`

These clauses specify the guidelines for the software integration testing. See below the analysis of each of these clauses against the ECSS standard.

- 7.4.8.1 - This is according with ECSS (see 5.5.4.1 ECSS-E-ST-40C, this activity is within section 5.5 Software design and implementation engineering process)
- 7.4.8.2 - This is according with ECSS (see 5.5.4.1, 5.5.4.2 and 5.8.3.7 ECSS-E-ST-40C)
- 7.4.8.3 - This is according with ECSS (see 5.5.4.2 and 5.8.3.7 ECSS-E-ST-40C)

- 7.4.8.4 - This is according with ECSS (see 5.8.3.7 ECSS-E-ST-40C and 7.2.3 ECSS-Q-ST-80C)
- 7.4.8.5 - This is implicit in ECSS (see 6.2.1.3, 6.2.4.5, 6.2.4.6 and 6.3.2.5 ECSS-Q-ST-80C), note specially all changed components shall be re-tested (see 6.2.3.4 ECSS-Q-ST-80C).

### 7.5 Programmable electronics integration (hardware and software)

#### 7.5.1 Objectives

This section does not contain requirements.

#### 7.5.2 Requirements

These clauses specify the guidelines for the software integration with hardware.

Note: The ECSS integration clauses apply both to software-software and software-hardware (ECSS definition of integration testing: “testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them”). However, for the hardware, the ECSS software documents needs to be complemented other discipline ECSS documents (e.x: E-10 and E-20). See below the analysis of each of these clauses against the ECSS standard.

- 7.5.2.1 - This is according with ECSS (see 5.5.4.1 ECSS-E-ST-40C, this activity is within section 5.5 Software design and implementation engineering process)
- 7.5.2.2 - This is according with ECSS (see 5.5.4.1, 5.5.4.2 and 5.8.3.7 ECSS-E-ST-40C)
- 7.5.2.3 - This is according with ECSS (see 5.8.3.9 ECSS-Q-ST-80C)
- 7.5.2.4 - This is implicit in ECSS, the integration testing shall distinguish between hardware and software integration.
- 7.5.2.5 - This is according with ECSS (see 5.5.4.1, 5.5.4.2 and 5.8.3.7 ECSS-E-ST-40C). Note that although the integration with hardware is not explicitly stated in these clauses, hardware is a possible interface for the software (see <5.3> Annex C ECSS-E-ST-40C).
- 7.5.2.6 - The same analysis made in 7.4.8.5 apply to this clause, with the addition that ECSS states that a change in hardware requires also re-testing (see 6.2.3.4, 6.2.3.5 and 6.3.5.17 ECSS-Q-ST-80C)
- 7.5.2.7 - This is according with ECSS (see 5.5.4.1 ECSS-E-ST-40C 6.3.5.11 ECSS-Q-ST-80C)
- 7.5.2.8 - This is according with ECSS (see 5.5.4.1 ECSS-E-ST-40C 6.3.5.11 ECSS-Q-ST-80C)

### 7.6 Software operation and modification procedures

#### 7.6.1 Objective

This section does not contain requirements.

### 7.6.2 Requirements

See analysis of this standard, section 7.8

## 7.7 Software aspects of system safety validation

### 7.7.1 Objective

This section does not contain requirements.

### 7.7.2 Requirements

These clauses specify the guidelines for the software validation. See below the analysis of each of these clauses against the ECSS standard.

- 7.7.2.1 - This is not according with ECSS. ECSS is more restrict, requiring both a system validation and software validation (see 5.2.3.1, 5.2.3.2).
- 7.7.2.2 - This is according with ECSS (see 5.6.3.2 and 5.6.4.2 ECSS-E-ST-40C).
- 7.7.2.3 - This is according with ECSS (see 5.6.2.1 ECSS-E-ST-40C), with ECSS being more strict in this point, requiring that validation to be taken by personnel not involved in the software coding (see 6.3.5.19 ECSS-Q-ST-80C).
- 7.7.2.4 - This is according with ECSS (see 5.8.3.8 ECSS-E-ST-40C).
- 7.7.2.5 - This is according with ECSS (see 6.3.5.11, 6.3.5.12, 6.3.5.21, 6.3.5.22, 6.3.5.23, 6.3.5.24 and 6.3.5.25 ECSS-Q-ST-80C).
- 7.7.2.6 - This is according with ECSS (see 6.3.5.6 and 6.3.5.8 ECSS-Q-ST-80C).
- 7.7.2.7 - See analysis below:
  a. This is according with ECSS (see 5.6.3.1 and 5.6.4.1 ECSS-E-ST-40C).
  b. This is according with ECSS (see 5.6.3.1 and 5.6.4.1 ECSS-E-ST-40C).
  c. This is implicit in ECSS, the validation results shall be documented (see 6.3.5.11 ECSS-Q-ST-80C) and the results shall be made available to the interested parts. Typically, the results would be available in CDR (for technical specification validation, see 5.6.3.4 ECSS-E-ST-40C) and QR (for requirements baseline, see 5.6.4.4 ECSS-E-ST-40C).
- 7.7.2.8 - This is according with ECSS (see 6.3.5.23 and 6.3.5.24 ECSS-Q-ST-80C).
- 7.7.2.9 - See analysis below:
  a. This is according with ECSS (see 5.6.3.1, 5.6.4.1 and 5.8.3.8 ECSS-E-ST-40C).
  b. This is implicit in ECSS (see 5.6.2.2 ECSS-E-ST-40C and 6.3.5.28 ECSS-Q-ST-80C), the independent assessment shall have access to all relevant documentation.
  c. This is according with ECSS (see 6.3.5.11 and 6.3.5.13 ECSS-Q-ST-80C).

## 7.8 Software modification

#### 7.8.1 Objective

This section does not contain requirements.

#### 7.8.2 Requirements

These clauses specify the guidelines for the software modification (maintenance of the software product). See below the analysis of each of these clauses against the ECSS standard.

- 7.8.2.1 - This is according with ECSS (see 5.10.2.1 ECSS-E-ST-40C).
- 7.8.2.2 - This is according with ECSS (see 5.10.3.1 ECSS-E-ST-40C and 6.3.8.7 ECSS-Q-ST-80C).
- 7.8.2.3 - This is implicit in ECSS (see 5.10.3.1 ECSS-E-ST-40C and 6.3.8.4 ECSS-Q-ST-80C).
- 7.8.2.4 - This is according with ECSS (see 5.10.3.1 ECSS-E-ST-40C and 6.3.8.7 ECSS-Q-ST-80C).
- 7.8.2.5 - This is according with ECSS (see 5.10.4.3 ECSS-E-ST-40C).
- 7.8.2.6 - This is according with ECSS (see 6.3.8.4 and 6.3.8.7 ECSS-Q-ST-80C).
- 7.8.2.7 - This is implicit in ECSS (see 6.3.8.4 ECSS-Q-ST-80C).
- 7.8.2.8 - This is according with ECSS (see 6.3.8.6 and 6.3.8.7 ECSS-Q-ST-80C).
- 7.8.2.9 - This is according with ECSS (see 5.10.4.3 ECSS-E-ST-40C and 6.3.8.7 ECSS-Q-ST-80C).
- 7.8.2.10 - This is according with ECSS (see 5.10.3.1 ECSS-E-ST-40C).

### 7.9 Software verification

#### 7.9.1 Objective

This section does not contain requirements.

#### 7.9.2 Requirements

These clauses specify the guidelines for the software verification. See below the analysis of each of these clauses against the ECSS standard.

- 7.9.2.1 - This is according with ECSS (see 5.8.2.1 ECSS-E-ST-40C and 6.2.6.2 ECSS-Q-ST-80C).
- 7.9.2.2 - This is according with ECSS (see 5.8.2.1 ECSS-E-ST-40C).
- 7.9.2.3 - This is according with ECSS (see 6.2.6.7 ECSS-Q-ST-80C).
- 7.9.2.4 - This is according with ECSS (see 6.2.6.2 and 6.2.6.7 ECSS-Q-ST-80C).
- 7.9.2.5 - This is according with ECSS (see 6.2.6.10 and 6.2.6.11 ECSS-Q-ST-80C).
- 7.9.2.6 - This is according with ECSS (see 6.2.6.2 and 6.2.6.7 ECSS-Q-ST-80C), although more detailed in this standard. It is implicit that the information described in this clause for phase N shall be also available for phase N+1 in ECSS standard.

- 7.9.2.7 - See below the report for the verification activities:
  a. Verification activity foreseen by ECSS (see 5.8.3.1 and 5.8.3.2 ECSS-E-ST-40C).
  b. Verification activity foreseen by ECSS (see 5.8.3.3 ECSS-E-ST-40C).
  c. Verification activity foreseen by ECSS (see 5.8.3.4 ECSS-E-ST-40C).
  d. Verification activity foreseen by ECSS (see 5.8.3.5 ECSS-E-ST-40C).
  e. Verification activity foreseen by ECSS (see 5.8.3.5 ECSS-E-ST-40C).
  f. Verification activity foreseen by ECSS (see 5.8.3.5 ECSS-E-ST-40C).
  g. Verification activity foreseen by ECSS (see 5.8.3.11 and 5.8.3.12 ECSS-E-ST-40C).
  h. Verification activity foreseen by ECSS (see 5.8.3.6 ECSS-E-ST-40C).
  i. Verification activity foreseen by ECSS (see 5.8.3.7 ECSS-E-ST-40C).
  j. Verification activity foreseen by ECSS (see 5.8.3.7, 5.8.3.8 and 5.8.3.9 ECSS-E-ST-40C).
  k. Verification activity foreseen by ECSS (see 5.8.3.8 ECSS-E-ST-40C).
- 7.9.2.8 - This is according with ECSS (see 5.8.3.1 and 5.8.3.2 ECSS-E-ST-40C).
- 7.9.2.9 - This is according with ECSS (see 5.8.3.3 and 5.8.3.7 ECSS-E-ST-40C).
- 7.9.2.10 - This is according with ECSS (see 5.8.3.4 and 5.8.3.7 ECSS-E-ST-40C).
- 7.9.2.11 - This is according with ECSS (see 5.8.3.5 and 5.8.3.6 ECSS-E-ST-40C).
- 7.9.2.12 - This is according with ECSS (see 5.8.3.5 ECSS-E-ST-40C).
- 7.9.2.13 - This is according with ECSS (see 5.8.3.5 ECSS-E-ST-40C), but more detailed in this standard.
- 7.9.2.14 - This is according with ECSS (see 5.8.3.11 and 5.8.3.12 ECSS-E-ST-40C). Note that in ECSS clauses, it is required also the memory budget verification.

```
8 Functional safety assessment
```

These clauses specify the guidelines for the functional safety assessment (independent verification). See below the analysis of each of these clauses against the ECSS standard.

- 8.1 - See Clause 8 analysis of IEC 61508-1 in section 8.4.1.
- 8.2 - See Clause 8 analysis of IEC 61508-1 in section 8.4.1.
- 8.3 - The techniques presented in table A.10 are in accordance with the ECSS techniques for the functional safety assessment (see ESA Guide for Independent Software Verification and Validation).

### 8.4.3 Conclusions

The philosophy of IEC standard is the same as the ECSS: the quality and assurance targets that the software products under development shall achieve is the same. This standard focuses more in safety, which is also an important aspect of the ECSS standards, but which is more generic regarding this topic and assumes that the safety is a consequence of applying correctly the standard (ECSS does not provide as much specific safety requirements as this standard). The main differences found between the two standards are the following:

- Software criticality classification, although there is no impact in a possible ECSS to IEC qualification, since ECSS is more strict in criticality assignment (see analysis in 8.2 IEC 61508-1);
- There are topics that are either more detailed in ECSS or in this standard, but it should be considered that the level of compliance is equivalent in these topics.
- Life-cycle definition (see analysis in 7.1.4 IEC 61508-1);
- Failures probability specification (see analysis in 7.6.2, IEC 61508-1);
- Responsibility of the application software of meeting the requirements for software development (see analysis in 7.4.2, IEC 61508-3);
- ECSS does not recommend specific activities on software depending on the Software Integrity Level (as in IEC 61508-3 Annex A tables), being that some activities are always necessary regardless the Software Integrity Level (ex: traceability), whereas others are under the decision of software supplier/customer (ex: the use of formal methods).

Despite the indicated differences, both standards share a common guideline and hence, it can be concluded that an ECSS developed product can be adapted to this standard without major effort.

CHAPTER

NINE

# TAILORING OF ECSS STANDARDS FOR THE QDP

The following sections present the tailoring of the ECSS standards ECSS-E-ST-40C [ECS09b] and ECSS-Q-ST-80C Rev.1 [ECS17d] for the QDP.

For the standard tailoring status the following notation is used:

**Y:** The product is fully compliant to the clause.

**Ye:** The product is compliant to the clause with exceptions.

**N:** The product is not compliant at all to the clause.

**N/A:** The clause is not applicable to the product.

**US:** The status User-Specified (US) indicates that compliance to this clause is out of scope of the product. A user of the product may have to ensure compliance to the clause if the product is used to build a complete system.

## 9.1 Tailoring of ECSS-E-ST-40C

Table 1: Compliance Matrix of ECSS-E-ST-40C to QDP

| Clause | QDP Status | Clause | QDP Status | Clause | QDP Status |
|---|---|---|---|---|---|
| 5.2.2.1a | *US* | 5.2.2.2a | *US* | 5.2.2.3a | *US* |
| 5.2.3.1a | *US* | 5.2.3.2a | *US* | 5.2.3.3a | *US* |
| 5.2.4.1a | *US* | 5.2.4.1b | *US* | 5.2.4.2a | *US* |
| 5.2.4.3a | *US* | 5.2.4.4a | *US* | 5.2.4.5a | *US* |
| 5.2.4.6a | *US* | 5.2.4.7a | *US* | 5.2.4.8a | *US* |
| 5.2.4.9a | *US* | 5.2.5a | *US* | 5.3.2.1a | *Y* |
| 5.3.2.1b | *Y* | 5.3.2.1c | *Y* | 5.3.2.1d | *Y* |
| 5.3.2.2a | *US* | 5.3.2.3a | *N/A* | 5.3.2.4a | *N* |
| 5.3.2.4b | *N* | 5.3.2.4c | *N* | 5.3.2.4d | *N* |
| 5.3.2.4e | *N* | 5.3.2.5a | *Y* | 5.3.3.1a | *Y* |
| 5.3.3.2a | *Y* | 5.3.3.2b | *Y* | 5.3.3.3a | *Ye* |
| 5.3.3.3b | *Y* | 5.3.3.3c | *Ye* | 5.3.4.1a | *US* |
| 5.3.4.2a | *Ye* | 5.3.4.2b | *N/A* | 5.3.4.3a | *Ye* |
| 5.3.4.3b | *N/A* | 5.3.4.4a | *Ye* | 5.3.4.5a | *Ye* |
| 5.3.5.1a | *N* | 5.3.5.2a | *N* | 5.3.6.1a | *N/A* |

continues on next page

Table 1 – continued from previous page

| Clause | QDP Status | Clause | QDP Status | Clause | QDP Status |
|---|---|---|---|---|---|
| 5.3.6.1b | *N/A* | 5.3.6.2a | *N/A* | 5.3.7.1a | *US* |
| 5.3.8.1a | *Ye* | 5.3.8.2a | *Ye* | 5.3.9.1a | *Y* |
| 5.3.9.2a | *Y* | 5.4.2.1a | *Ye* | 5.4.2.2a | *N/A* |
| 5.4.2.3a | *N* | 5.4.2.3b | *N* | 5.4.2.3c | *N* |
| 5.4.2.4a | *N/A* | 5.4.3.1a | *Ye* | 5.4.3.2a | *Ye* |
| 5.4.3.3a | *N* | 5.4.3.4a | *Ye* | 5.4.3.5a | *Ye* |
| 5.4.3.6a | *Ye* | 5.4.3.6b | *Ye* | 5.4.3.6c | *Ye* |
| 5.4.3.7a | *US* | 5.4.3.8a | *Ye* | 5.4.4a | *Ye* |
| 5.5.2.1a | *Ye* | 5.5.2.1b | *Ye* | 5.5.2.1c | *Ye* |
| 5.5.2.2a | *Ye* | 5.5.2.3a | *Ye* | 5.5.2.4a | *Ye* |
| 5.5.2.5a | *N* | 5.5.2.5b | *Ye* | 5.5.2.5c | *Ye* |
| 5.5.2.5d | *Ye* | 5.5.2.5e | *Ye* | 5.5.2.6a | *Ye* |
| 5.5.2.7a | *N* | 5.5.2.8a | *Ye* | 5.5.2.9a | *Ye* |
| 5.5.2.10a | *N/A* | 5.5.3.1a | *Ye* | 5.5.3.2a | *Ye* |
| 5.5.3.2b | *Ye* | 5.5.3.2c | *Ye* | 5.5.4.1a | *Ye* |
| 5.5.4.2a | *Ye* | 5.6.2.1a | *Ye* | 5.6.2.1b | *Ye* |
| 5.6.2.1c | *Ye* | 5.6.2.2a | *N* | 5.6.2.2b | *N* |
| 5.6.3.1a | *Ye* | 5.6.3.1b | *Y* | 5.6.3.1c | *Y* |
| 5.6.3.2a | *Y* | 5.6.3.3a | *Y* | 5.6.3.4a | *Ye* |
| 5.6.4.1a | *US* | 5.6.4.1b | *US* | 5.6.4.1c | *US* |
| 5.6.4.2a | *US* | 5.6.4.2b | *US* | 5.6.4.3a | *US* |
| 5.6.4.4a | *Ye* | 5.7.2.1a | *Ye* | 5.7.2.2a | *US* |
| 5.7.2.3a | *Ye* | 5.7.2.4a | *US* | 5.7.2.4b | *US* |
| 5.7.2.4c | *US* | 5.7.2.4d | *US* | 5.7.3.1a | *US* |
| 5.7.3.2a | *US* | 5.7.3.3a | *US* | 5.7.3.4a | *US* |
| 5.7.3.4b | *US* | 5.7.3.5a | *US* | 5.7.3.6a | *Ye* |
| 5.8.2.1a | *Ye* | 5.8.2.1b | *Ye* | 5.8.2.1c | *Ye* |
| 5.8.2.1d | *Ye* | 5.8.2.2a | *N* | 5.8.2.2b | *N* |
| 5.8.3.1a | *US* | 5.8.3.2a | *Ye* | 5.8.3.3a | *Ye* |
| 5.8.3.4a | *Ye* | 5.8.3.5a | *Ye* | 5.8.3.5b | *Ye* |
| 5.8.3.5c | *Y* | 5.8.3.5d | *Y* | 5.8.3.5e | *N/A* |
| 5.8.3.5f | *Y* | 5.8.3.6a | *Ye* | 5.8.3.7a | *Ye* |
| 5.8.3.8a | *Ye* | 5.8.3.8b | *Ye* | 5.8.3.9a | *US* |
| 5.8.3.10a | *Ye* | 5.8.3.11a | *N/A* | 5.8.3.11b | *N/A* |
| 5.8.3.11c | *N/A* | 5.8.3.12a | *Ye* | 5.8.3.12b | *Ye* |
| 5.8.3.12c | *Ye* | 5.8.3.13a | *N* | 5.8.3.13b | *N* |
| 5.8.3.13c | *N* | 5.9.2.1a | *US* | 5.9.2.2a | *US* |
| 5.9.2.3a | *US* | 5.9.3.1a | *US* | 5.9.3.2a | *US* |
| 5.9.3.3a | *US* | 5.9.4.1a | *US* | 5.9.4.2a | *US* |
| 5.9.5.1a | *US* | 5.9.5.1b | *US* | 5.9.5.2a | *US* |
| 5.9.5.2b | *US* | 5.9.5.2c | *US* | 5.9.5.3a | *US* |
| 5.9.5.3b | *US* | 5.10.2.1a | *US* | 5.10.2.1b | *US* |
| 5.10.2.1c | *US* | 5.10.2.1d | *US* | 5.10.2.1e | *US* |
| 5.10.2.2a | *US* | 5.10.3.1a | *US* | 5.10.3.1b | *US* |
| 5.10.3.1c | *US* | 5.10.3.1d | *US* | 5.10.3.1e | *US* |

continues on next page

Table 1 – continued from previous page

| Clause | QDP Status | Clause | QDP Status | Clause | QDP Status |
|---|---|---|---|---|---|
| 5.10.4.1a | *US* | 5.10.4.2a | *US* | 5.10.4.3a | *US* |
| 5.10.4.3b | *US* | 5.10.4.3c | *US* | 5.10.4.3d | *US* |
| 5.10.4.3e | *US* | 5.10.5.1a | *US* | 5.10.5.2a | *US* |
| 5.10.6.1a | *US* | 5.10.6.2a | *US* | 5.10.6.3a | *US* |
| 5.10.6.4a | *US* | 5.10.6.5a | *US* | 5.10.6.5b | *US* |
| 5.10.6.6a | *US* | 5.10.6.6b | *US* | 5.10.6.7a | *US* |
| 5.10.7.1a | *US* | 5.10.7.2a | *US* | 5.10.7.3a | *US* |
| 5.10.7.4a | *US* | | | | |

### 9.1.1 Specification of system requirements allocated to software (5.2.2.1a)

**ECSS-E-ST-40C Clause 5.2.2.1a**

The customer shall derive system requirements allocated to software from an analysis of the specific intended use of the system, and from the results of the safety and dependability analysis.

**Expected Output**: a. Functions and performance system requirements allocated to software [RB, SSS; SRR]; b. Verification and validation product requirements [RB, SSS; SRR]; c. Software operations requirements [RB, SSS; SRR]; d. Software maintenance requirements [RB, SSS; SRR]; e. Requirements for in flight modification capabilities [RB, SSS; SRR]; f. Requirements for real-time [RB, SSS; SRR]; g. Requirements for security [RB, SSS, SRR]; h. Quality requirements [RB, SSS, SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.2 Identification of observability requirements (5.2.2.2a)

**ECSS-E-ST-40C Clause 5.2.2.2a**

The customer shall specify all software observability requirements to monitor the software behaviour and to facilitate the system integration and failure investigation.

**Expected Output**: System and software observability requirements [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.3 Specification of HMI requirements (5.2.2.3a)

**ECSS-E-ST-40C Clause 5.2.2.3a**

The customer shall specify HMI requirements, following the human factor engineering process specified in ECSS-E-ST-10-11.

**Expected Output**: HMI requirements [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.4 Verification and validation process requirements (5.2.3.1a)

**ECSS-E-ST-40C Clause 5.2.3.1a**

The customer shall specify the requirements needed for planning and setting up the system verification and validation process related to software.

**Expected Output**: Verification and validation process requirements [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.4.1a (Development and documentation of a software validation specification with respect to the requirements baseline)*
- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.5 System input for software validation (5.2.3.2a)

**ECSS-E-ST-40C Clause 5.2.3.2a**

The customer shall specify requirements for the validation of the software against the requirements baseline and technical specification, in particular mission representative data and scenarios, and operational procedures to be used.

**Expected Output**: Validation requirements and scenario [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.6 System input for software installation and acceptance (5.2.3.3a)

**ECSS-E-ST-40C Clause 5.2.3.3a**

The customer shall specify requirements for the installation and acceptance of the software.

**Expected Output**: Installation and acceptance requirements at the operational and maintenance sites [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.7 Identification of software versions for software integration into the system (5.2.4.1a)

**ECSS-E-ST-40C Clause 5.2.4.1a**

The customer shall identify the software versions to be delivered and associate each requirement of the requirements baseline to a version.

**Expected Output**: Association of requirements to versions [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.8 Identification of software versions for software integration into the system (5.2.4.1b)

**ECSS-E-ST-40C Clause 5.2.4.1b**

The customer shall specify the content and media of the delivery.

**Expected Output**: Delivery content and media [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.9 Supplier support to system integration (5.2.4.2a)

**ECSS-E-ST-40C Clause 5.2.4.2a**

The customer shall specify the support to be provided by the software supplier in order to integrate the software at system level. {NOTE: For example: Raining, maintenance, configuration and test support.}

**Expected Output**: System level integration support requirements [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.10 Interface requirement specification (5.2.4.3a)

**ECSS-E-ST-40C Clause 5.2.4.3a**

The customer shall specify the external interfaces of the software, including the static and dynamic aspects, for nominal and degraded modes.

**Expected Output**: External interface requirements specification [RB, IRD; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.11 System database (5.2.4.4a)

**ECSS-E-ST-40C Clause 5.2.4.4a**

The customer shall specify the content of the system database for the supplier in order to ensure the consistency of common data and to define the allowed operational range of the data.

**Expected Output**: System database content and allowed operational range [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.12 Development constraints (5.2.4.5a)

**ECSS-E-ST-40C Clause 5.2.4.5a**

The customer shall define specific development and design constraints on the supplier, including the use of development standards.

**Expected Output**: Design and development constraints [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.13 On board control procedures (5.2.4.6a)

**ECSS-E-ST-40C Clause 5.2.4.6a**

The customer shall specify the requirements to be implemented by OBCP. {NOTE: See ECSS-E-ST-70-01.}

**Expected Output**: OBCP requirements [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.14 Development of software to be reused (5.2.4.7a)

**ECSS-E-ST-40C Clause 5.2.4.7a**

The customer shall specify the reusability requirements that apply to the development, to enable the future reuse of the software (including models used to generate the software), or customization for mission (e.g. in a family of spacecraft or launcher).

**Expected Output**: Requirements for 'software to be reused' [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*
- *ECSS-Q-ST-80C-R1 7.3.1a (Software reuse/Customer requirements)*

### 9.1.15 Software safety and dependability requirements (5.2.4.8a)

**ECSS-E-ST-40C Clause 5.2.4.8a**

The customer shall specify the software safety and dependability requirements in accordance with ECSS-Q-ST-80 clauses *5.4.4*, *6.2.2* and *6.2.3*, based on the results of the safety and dependability analysis performed at system level.

**Expected Output**: Software safety and dependability requirements [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 5.4.4a (Criticality classification)*
- *ECSS-Q-ST-80C-R1 6.2.2.1a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.2a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.3a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.3b (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.5a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.6a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.7a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.8a (Software dependability and safety)*

- *ECSS-Q-ST-80C-R1 6.2.2.9a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.10a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.3.1a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.1b (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.2a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.3a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.4a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.5a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.6a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.7a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.8a (Handling of criticality software)*

This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.16 Format and data medium (5.2.4.9a)

**ECSS-E-ST-40C Clause 5.2.4.9a**

The customer shall specify the format and the delivery medium of the exchanged data, in particular the interface and the system database.

**Expected Output**: Format and delivery medium of exchanged data [RB, SSS; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.17 System requirements review (5.2.5a)

**ECSS-E-ST-40C Clause 5.2.5a**

The customer shall conduct a system requirements review (SRR) in accordance with *5.3.4.1a*.

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.1.1a (Software related system requirements process)*

### 9.1.18 Software life cycle identification (5.3.2.1a)

**ECSS-E-ST-40C Clause 5.3.2.1a**

The software supplier shall define and follow a software life cycle including phases, their inputs and outputs, and joint reviews, in accordance with the overall project constraints and with ECSS-M-ST-10.

**Expected Output**: Software life cycle definition [MGT, SDP; SRR, PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.19 Software life cycle identification (5.3.2.1b)

**ECSS-E-ST-40C Clause 5.3.2.1b**

The life cycle shall be chosen, assessing the specifics of the project technical approaches and the relevant project risks.

**Expected Output**: Software life cycle definition [MGT, SDP; SRR, PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.20 Software life cycle identification (5.3.2.1c)

**ECSS-E-ST-40C Clause 5.3.2.1c**

The software supplier shall define the development strategy, the software engineering standards and techniques, the software development and the software testing environment.

**Expected Output**: Development strategy, standards, techniques, development and testing environment [MGT, SDP; PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.21 Software life cycle identification (5.3.2.1d)

**ECSS-E-ST-40C Clause 5.3.2.1d**

The output of each phase and their status of completion, submitted as input to joint reviews, shall be specified in the software life cycle definition, including documents in complete or outline versions, and the results of verification of the outputs of the phase.

**Expected Output**: Software life cycle definition [MGT, SDP; SRR, PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.22 Identification of interfaces between development and maintenance (5.3.2.2a)

**ECSS-E-ST-40C Clause 5.3.2.2a**

The interfaces between development and maintenance (e.g. documents to be handed over, tools to be kept for maintenance) shall be identified in the software life cycle.

**Expected Output**: Identification of interface between development and maintenance [MGT, SDP; PDR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.23 Software procurement process implementation (5.3.2.3a)

**ECSS-E-ST-40C Clause 5.3.2.3a**

The supplier shall document and implement the software procurement process as specified in ECSS-Q-ST-80 clause *5.5*.

**Expected Output**: Software procurement process documentation and implementation [MGT, SDP; SRR, PDR]

**QDP Status (N/A):** Software procurement is not forecasted in the execution of this project.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 5.5.1a (Procurement documents)*
- *ECSS-Q-ST-80C-R1 5.5.2a (Review of procured software component list)*
- *ECSS-Q-ST-80C-R1 5.5.3a (Procurement details)*
- *ECSS-Q-ST-80C-R1 5.5.4a (Identification)*
- *ECSS-Q-ST-80C-R1 5.5.5a (Inspection)*
- *ECSS-Q-ST-80C-R1 5.5.6a (Exportability)*

This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.24 Automatic code generation (5.3.2.4a)

**ECSS-E-ST-40C Clause 5.3.2.4a**

The autocode input models shall be reviewed together with the rest of the software specification, architecture and design. {NOTE: The autocode input models are integral part of the software specification, architecture and design.}

**Expected Output**: Autocode input model review [MGT, SDP; SRR, PDR]

**QDP Status (N):** No Automatic code generation

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.25 Automatic code generation (5.3.2.4b)

**ECSS-E-ST-40C Clause 5.3.2.4b**

In the case of coexisting autocoded and manually written parts, the software development plan shall include the definition of a clear interface definition and resource allocation (memory, CPU) at PDR.

**Expected Output**: Autocode interface definition and resource allocation [MGT, SDP; SRR, PDR]

**QDP Status (N):** No Automatic code generation

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.26 Automatic code generation (5.3.2.4c)

**ECSS-E-ST-40C Clause 5.3.2.4c**

The input model management, the code generation process and supporting tools shall be documented in the SDP.

**Expected Output**: Automatic code generation development process and tools [MGT, SDP; SRR, PDR]

**QDP Status (N):** No Automatic code generation

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.27 Automatic code generation (5.3.2.4d)

**ECSS-E-ST-40C Clause 5.3.2.4d**

The supplier shall define in the SDP the verification and validation strategy for automatic code generation as a result of the trade off between the qualification of the code generation toolchain and the end to end validation strategy of the software item, or any combination thereof, in relation with ECSS-Q-ST-80 clause *6.2.8*.

**Expected Output**: Automatic code generation verification and validation strategy [MGT, SDP; SRR, PDR]

**QDP Status (N):** No Automatic code generation

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.8.1a (Automatic code generation)*
- *ECSS-Q-ST-80C-R1 6.2.8.2a (Automatic code generation)*
- *ECSS-Q-ST-80C-R1 6.2.8.3a (Automatic code generation)*
- *ECSS-Q-ST-80C-R1 6.2.8.4a (Automatic code generation)*
- *ECSS-Q-ST-80C-R1 6.2.8.5a (Automatic code generation)*
- *ECSS-Q-ST-80C-R1 6.2.8.6a (Automatic code generation)*
- *ECSS-Q-ST-80C-R1 6.2.8.7a (Automatic code generation)*

This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.28 Automatic code generation (5.3.2.4e)

**ECSS-E-ST-40C Clause 5.3.2.4e**

The configuration management of the automatic code generation related elements shall be defined in the SCMP.

**Expected Output**: Automatic code generation configuration management [MGT, SCMP; SRR, PDR]

**QDP Status (N):** No Automatic code generation

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.29 Changes to baselines (5.3.2.5a)

**ECSS-E-ST-40C Clause 5.3.2.5a**

Changes to baselines shall be handled by the configuration management process described in clause *6.2.4* of ECSS-Q-ST-80.

**Expected Output**: Changes to baselines procedures [MGT, SCMP; SRR, PDR]

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.4.1a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.2a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.3a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.4a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.5a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.5b (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.6a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.7a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.8a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.9a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.10a (Software configuration management)*
- *ECSS-Q-ST-80C-R1 6.2.4.11a (Software configuration management)*

This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.30 Joint reviews (5.3.3.1a)

**ECSS-E-ST-40C Clause 5.3.3.1a**

Joint reviews shall be held to evaluate the progress and outputs of a project process or activity and provide evidence that: 1. the output are complete; 2. the output conforms to applicable standards and specifications; 3. any changes are properly implemented and impact only those areas identified by the configuration management process; 4. the output conforms to applicable schedules; 5. the output are in such a status that the next activity can start; 6. the activity is being conducted according to the plans, schedules, standards, and guidelines laid down for the project. {NOTE: The joint review process is a process for evaluating the

status and products of an activity of a project as appropriate. This process is employed by two parties, where one party (reviewing party) reviews another party (reviewed party).For project reviews, the two parties are the customer and the supplier. Joint reviews are held throughout the life cycle of the software.}

**Expected Output**: Joint review reports [DJF, - ; SRR, PDR, CDR, QR, AR]

**QDP Status (Y):** -

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.31 Software project reviews (5.3.3.2a)

**ECSS-E-ST-40C Clause 5.3.3.2a**

Software project reviews (i.e. joint reviews organized under the responsibility of the customer aiming at defining a customer approved technical baseline) shall be included in the software life cycle, with as a minimum SRR, PDR, CDR, QR and AR as specified in 5.3.4.

**Expected Output**: Software project reviews included in the software life cycle definition [MGT, SDP; SRR, PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.32 Software project reviews (5.3.3.2b)

**ECSS-E-ST-40C Clause 5.3.3.2b**

The review process specified in ECSS-M-ST-10-01 shall apply to all software project reviews, including the agreement on a review plan before the review process is started.

**Expected Output**: Review Plan [MGT, SRevP; SRR, PDR]

**QDP Status (Y):** See [EDI19f].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.33 Software technical reviews (5.3.3.3a)

**ECSS-E-ST-40C Clause 5.3.3.3a**

In addition to the software project reviews, software technical reviews (i.e. joint reviews organized under the responsibility of the customer or the supplier aiming at defining a technical baseline) shall be defined.

**Expected Output**: Software technical reviews included in the software life cycle definition [MGT, SDP; SRR, PDR]

**QDP Status (Ye):** See [EDI19c] and [EDI19h].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.34 Software technical reviews (5.3.3.3b)

**ECSS-E-ST-40C Clause 5.3.3.3b**

The applicable technical review process shall be specified by the supplier.

**Expected Output**: Technical reviews process [MGT; SDP; SRR, PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.35 Software technical reviews (5.3.3.3c)

**ECSS-E-ST-40C Clause 5.3.3.3c**

The supplier shall use the software technical reviews to implement intermediate reviews, in particular for incremental life cycle. {AIM: - this enables to cope with any alternative life cycle not necessarily waterfall. - in the case of incremental life cycle in particular, this

enables to hold formal reviews on the last increments, and to have those technical reviews in anticipation for each of the increment.}

**Expected Output**: Software technical reviews included in the software life cycle definition [MGT, SDP; SRR, PDR]

**QDP Status (Ye):** See [EDI19c] and [EDI19h].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.9a (Verification)*

### 9.1.36 System requirement review (5.3.4.1a)

**ECSS-E-ST-40C Clause 5.3.4.1a**

After completion of the software requirements baseline specification, a system requirements review (SRR) shall take place. {AIM: Reach the approval of the software requirements baseline by all stakeholders.}

**Expected Output**: Approved requirements baseline [RB; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.37 Preliminary design review (5.3.4.2a)

**ECSS-E-ST-40C Clause 5.3.4.2a**

After completion of the software requirement analysis and architectural design, and the verification and validation processes implementation, a preliminary design review (PDR) shall take place. {AIM: To review compliance of the technical specification (TS) with the requirements baseline, to review the software architecture and interfaces, to review the development, verification and validation plans.}

**Expected Output**: Approved technical specification and interface, architecture and plans [TS, DDF, DJF, MGT; PDR]

**QDP Status (Ye):** The technical specification of the QDP is the QT-109, see also [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.4.4a (Conducting a preliminary design review)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.38 Preliminary design review (5.3.4.2b)

**ECSS-E-ST-40C Clause 5.3.4.2b**

In case the software requirements are baselined before the start of the architectural design, the part of the PDR addressing the software requirements specification and the interfaces specification shall be held in a separate joint review anticipating the PDR, in a software requirements review (SWRR). {AIM: e.g. in case of software intensive system or when an early baseline of the requirements is required.}

**Expected Output**: Approved technical specification and interface [TS; PDR]

**QDP Status (N/A):** The software requirements are not baselined before the start of the architectural design.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.4.2.4a (Conducting a software requirement review)*
- *ECSS-E-ST-40C 5.4.4a (Conducting a preliminary design review)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.39 Critical design review (5.3.4.3a)

**ECSS-E-ST-40C Clause 5.3.4.3a**

After completion of the design of software items, coding and testing, integration and validation with respect to the technical specification, a critical design review (CDR) shall take place. {AIM: -To review the design definition file, including software architectural design, detailed design, code and users manual; - To review the design justification file, including the completeness of the software unit testing, integration and validation with respect to the technical specification.}

**Expected Output**: Approved design definition file and design justification file [DDF, DJF; CDR]

**QDP Status (Ye):** See [EDI19c] and [EDI19h].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.3.4a (Conducting a critical design review)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.40 Critical design review (5.3.4.3b)

**ECSS-E-ST-40C Clause 5.3.4.3b**

In case the software detailed design is baselined before the start of the coding, the part of the CDR addressing the software detailed design, the interfaces design and the software budget shall be held in a separate joint review anticipating the CDR, in a detailed design review (DDR).

**Expected Output**: Approved detailed design, interface design and budget [DDF, DJF; CDR]

**QDP Status (N/A):** The software detailed design is not baselined before the start of the coding.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.5.2.10a (Conducting a detailed design review)*
- *ECSS-E-ST-40C 5.6.3.4a (Conducting a critical design review)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.41 Qualification review (5.3.4.4a)

**ECSS-E-ST-40C Clause 5.3.4.4a**

After completion of the software validation against the requirements baseline, and the verification activities, a qualification review (QR) shall take place.

**Expected Output**: Qualified software product [RB, TS, DDF, DJF, MGT, MF; QR]

**QDP Status (Ye):** See *No Requirements Baseline (RB)*, *No Maintenance (MF)*, [EDI19c], and [EDI19h].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.4.4a (Conducting a qualification review)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.42 Acceptance review (5.3.4.5a)

**ECSS-E-ST-40C Clause 5.3.4.5a**

After completion of the software delivery and installation, and software acceptance, an acceptance review (AR) shall take place. {AIM: To accept the software product in the intended operational environment.}

**Expected Output**: Accepted software product [RB, TS, DDF, DJF, MGT, MF; AR]

**QDP Status (Ye):** See *No Requirements Baseline (RB)*, *No Maintenance (MF)*, [EDI19c], and [EDI19h].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.7.3.6a (Conducting an acceptance review)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.43 Test readiness reviews (5.3.5.1a)

**ECSS-E-ST-40C Clause 5.3.5.1a**

Test readiness reviews (TRR) shall be held before the beginning of test activities, as defined in the software development plan.

**Expected Output**: Confirmation of readiness of test activities [DJF; TRR]

**QDP Status (N):** TRR will not be performed in this project.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.1.5a (Software validation process schedule)*

### 9.1.44 Test review board (5.3.5.2a)

**ECSS-E-ST-40C Clause 5.3.5.2a**

The test review board (TRB) shall approve test results at the end of test activities, as defined in the software development plan.

**Expected Output**: Approved test results[DJF; TRB]

**QDP Status (N):** TRR will not be performed in this project.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.45 Review phasing for flight software (5.3.6.1a)

**ECSS-E-ST-40C Clause 5.3.6.1a**

For flight software, the phasing of the software life cycle to the system life cycle shall be chosen, assessing the following driving aspects: 1. the system model philosophy (e.g. proto-flight model, versus utilization of engineering qualification model) 2. the system verification and qualification approach and constraints 3. the capability to baseline the system design at system CDR, by knowing enough information about software design, in particular consolidated sizing and timing budgets, consistent hardware design and software design.

**Expected Output**: Flight software review phasing [MGT, SDP;SRR, PDR]

**QDP Status (N/A):** Phasing of software life cycle to system life cycle not part of this project.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.46 Review phasing for flight software (5.3.6.1b)

**ECSS-E-ST-40C Clause 5.3.6.1b**

For flight software the following software versus system level reviews synchronisation shall be planned as follows: 1. the software SRR not later than the system PDR 2. the software PDR between the system PDR and the system CDR 3. the detailed design of the software reviewed before the system CDRe.g. in a DDR 4. the software CDR before the system QR 5. the software QR within system QR

**Expected Output**: Flight software review phasing [MGT, SDP;SRR, PDR]

**QDP Status (N/A):** Phasing of software life cycle to system life cycle not part of this project.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.47 Review phasing for ground software (5.3.6.2a)

**ECSS-E-ST-40C Clause 5.3.6.2a**

For ground segment software, the software life cycle shall be chosen assessing the following constraints for the ground reviews phasing: 1. the baseline of the software requirements (e.g. SWRR) is performed before the ground segment PDR , 2. the software PDR is performed before the ground segment CDR 3. all the other software reviews are performed before the ground segment QR.

**Expected Output**: Ground software review phasing [MGT, SDP;SRR, PDR]

**QDP Status (N/A):** There is no ground software in the QDP.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.48 Interface management procedures (5.3.7.1a)

**ECSS-E-ST-40C Clause 5.3.7.1a**

Interface management procedures shall be defined in accordance with ECSS-M-ST-40 requirements. {AIM: Define procedures that guarantee the consistency of the system interfaces.}

**Expected Output**: Interface management procedures [MGT, - ; SRR];

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.49 Software technical budget and margin philosophy definition (5.3.8.1a)

**ECSS-E-ST-40C Clause 5.3.8.1a**

Technical budget targets and margin philosophy dedicated to the software shall be specified by the customer in the requirements baseline in order to define the limits of software budgets associated with computer and network resources (such as: CPU load, maximum memory size, deadline fulfilment, communication, archiving needs, remote access needs) and performance requirements (such as data throughput). {AIM: This allows anticipating: - Expected changes

in the requirements baseline - Requirements on reprogramming of the system during operational use - Required budget for temporary copies of software images - Constraints on state transitions, especially when recovery from a faulty state is concerned - Constraints on physical CPU type and memory (e.g. driven by radiation levels) and expected processor capacity, wait states, interfaces, caching and pipelining, etc - Equipment, communication and performances aspects (e.g. buses, protocols, acceptable errors, bus capacity usage by other sources, etc) - Accuracy aspects, such as conversion to/from analogue signals, and accuracy of timing signals - Budgets for OS kernel characterisation, such as context switch latency or deadlines for tasking - Mission and system operation characteristics and reference operational scenarios} {NOTE 1: The following CPU load margin reference values are often considered: 50 % at PDR, 35 % at DDR or TRR, 25 % at CDR, QR or AR.} {NOTE 2: The following memory margin reference values in RAM or EEPROM are often considered: 50 % at PDR, 40 % at DDR or TRR, 35 % at CDR and 25 % at QR or AR.}

**Expected Output**: Technical budgets and margin philosophy for the project [RB, SSS; SRR]

**QDP Status (Ye):** See *No Requirements Baseline (RB)* and *Resources and Performance*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.50 Technical budget and margin computation (5.3.8.2a)

**ECSS-E-ST-40C Clause 5.3.8.2a**

The way to compute the technical budgets and margin shall be agreed between the customer and the supplier.

**Expected Output**: Technical budgets and margin computation [DJF, SVR; SRR, PDR]

**QDP Status (Ye):** See *Resources and Performance*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.51 Compliance matrix (5.3.9.1a)

**ECSS-E-ST-40C Clause 5.3.9.1a**

The supplier shall provide a compliance matrix documenting conformance with the individual software engineering process requirements (Clause 5) applicable to the project or business agreement, as per ECSS-S-ST-00.

**Expected Output**: ECSS-E-ST-40 compliance matrix [MGT, SDP; SRR, PDR]

**QDP Status (Y):** A compliance matrix for the QDP will be provided with respect to ECSS-E-ST-40C and ECSS-Q-ST-80C Rev.1 in the QT-109 with detailed tailoring information and in [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.52 Documentation compliance (5.3.9.2a)

**ECSS-E-ST-40C Clause 5.3.9.2a**

The compliance to each of the individual software engineering process requirements shall make reference to the document where the expected output is placed, if it is not placed in this Standard's DRDs in annexes of this document. {NOTE: A general statement of compliance to this Standard's DRDs is acceptable.}

**Expected Output**: ECSS-E-ST-40 compliance matrix [MGT, SDP; SRR, PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*

### 9.1.53 Establishment and documentation of software requirements (5.4.2.1a)

**ECSS-E-ST-40C Clause 5.4.2.1a**

The supplier shall establish and document software requirements, including the software quality requirements, as part of the technical specification.

**Expected Output**: a. Functional and performance specifications, including hardware characteristics, and environmental conditions under which the software item executes, including budgets requirements [TS, SRS; PDR]; b. Operational, reliability, safety, maintainability, portability, configuration, delivery, adaptation and installation requirements, design constraints [TS, SRS; PDR]; c. Software product quality requirements (see ECSS-Q-ST-80 clause 7.2) [TS, SRS; PDR]; d. Security specifications, including those related to factors which can compromise sensitive information [TS, SRS; PDR]; e. Human factors engineering (ergonomics including HMI usability) specifications, following the human factor engineering process specified in ECSS-E-ST-10-11 [TS, SRS; PDR]; f. Data definition and database requirements [TS, SRS; PDR]; g. Validation requirements [TS, SRS, ICD; PDR] h. Interfaces external to the software item [TS, ICD; PDR]; i. Reuse requirements (see ECSS-Q-ST-80 clause 6.2.7) [TS, SRS; PDR]

**QDP Status (Ye):** No human factors, no database, no security and no reuse requirements shall be established. See *Specification Items*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.54 Definition of functional and performance requirements for in flight modification (5.4.2.2a)

**ECSS-E-ST-40C Clause 5.4.2.2a**

When in flight modification is specified for flight software, the supplier shall perform analysis of the specific implications for the software design and validation processes and include the functional and performance requirements in the technical specification, including in case of use of automatic code generation.

**Expected Output**: Specifications for in flight software modifications [TS, SRS; PDR]

**QDP Status (N/A):** Software product does not provide in flight modification.

For an overview of all clauses, see the *tailoring table*.

### 9.1.55 Construction of a software logical model (5.4.2.3a)

**ECSS-E-ST-40C Clause 5.4.2.3a**

The supplier shall construct a logical model of the functional requirements of the software product.

**Expected Output**: Software logical model [TS, SRS; PDR]

**QDP Status (N):** See *No Logical and Computational Model*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.56 Construction of a software logical model (5.4.2.3b)

**ECSS-E-ST-40C Clause 5.4.2.3b**

The supplier shall use a method to support the construction of the logical model.

**Expected Output**: Software logical model method [TS, SRS; PDR]

**QDP Status (N):** See *No Logical and Computational Model*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.57 Construction of a software logical model (5.4.2.3c)

**ECSS-E-ST-40C Clause 5.4.2.3c**

The logical model shall include a behavioural view.

**Expected Output**: Behavioural view in software logical model [TS, SRS; PDR]

**QDP Status (N):** See *No Logical and Computational Model*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.8.3.13a (Behaviour modelling verification)*

### 9.1.58 Conducting a software requirement review (5.4.2.4a)

**ECSS-E-ST-40C Clause 5.4.2.4a**

The supplier shall conduct a software requirement review (SWRR) as anticipation of the PDR, in conformance with *5.3.4.2b*.

**QDP Status (N/A):** The software requirements are not baselined before the start of the architectural design.

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.3.4.2b (Preliminary design review)*

### 9.1.59 Transformation of software requirements into a software architecture (5.4.3.1a)

**ECSS-E-ST-40C Clause 5.4.3.1a**

The supplier shall transform the requirements for the software item into an architecture that: 1. describes its top-level structure; 2. identifies the software components, ensuring that all the requirements for the software item are allocated to its software components and later refined to facilitate detailed design; 3. covers as a minimum hierarchy, dependency, interfaces and operational usage for the software components; 4. documents the process, data and control aspects of the product; 5. describes the architecture static decomposition into software elements such as packages, classes or units; 6. describes the dynamic architecture, which involves the identification of active objects such as threads, tasks and processes; 7. describes the software behaviour.

**Expected Output**: Software architectural design [DDF, SDD; PDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.60 Software design method (5.4.3.2a)

**ECSS-E-ST-40C Clause 5.4.3.2a**

The supplier shall use a method (e.g. object oriented or functional) to produce the static and dynamic architecture including: 1. software elements, their interfaces, and; 2. software elements relationships.

**Expected Output**: Software architectural design method [DDF, SDD; PDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.61 Selection of a computational model for real-time software (5.4.3.3a)

**ECSS-E-ST-40C Clause 5.4.3.3a**

The dynamic architecture design shall be described according to an analysable computational model.

**Expected Output**: Computational model [DDF, SDD; PDR]

**QDP Status (N):** See *No Logical and Computational Model*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.62 Description of software behaviour (5.4.3.4a)

**ECSS-E-ST-40C Clause 5.4.3.4a**

The software architecture design shall also describe the behaviour of the software, by means of description techniques using automata and scenarios.

**Expected Output**: Software behaviour [DDF, SDD; PDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.8.3.13b (Behaviour modelling verification)*

### 9.1.63 Development and documentation of the software interfaces (5.4.3.5a)

**ECSS-E-ST-40C Clause 5.4.3.5a**

The supplier shall develop and document a software preliminary design for the interfaces external to the software item and between the software components of the software item.

**Expected Output**: a. Preliminary external interfaces design [TS, ICD; PDR]; b. Preliminary internal interfaces design [DDF, SDD; PDR]

**QDP Status (Ye):** See *Software Interface Control Document (ICD)*, *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.64 Definition of methods and tools for software intended for reuse (5.4.3.6a)

**ECSS-E-ST-40C Clause 5.4.3.6a**

The supplier shall define procedures, methods and tools for reuse, and apply these to the software engineering processes to comply with the reusability requirements for the software development.

**Expected Output**: Software intended for reuse - justification of methods and tools [DJF, SRF; PDR]

**QDP Status (Ye):** See *Software Reuse File (SRF)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.3.1a (Software reuse/Customer requirements)*

### 9.1.65 Definition of methods and tools for software intended for reuse (5.4.3.6b)

**ECSS-E-ST-40C Clause 5.4.3.6b**

An evaluation of the reuse potential of the software shall be performed at PDR and CDR.

**Expected Output**: Software intended for reuse - evaluation of reuse potential [DJF, SRF; PDR, CDR]

**QDP Status (Ye):** See *Software Reuse File (SRF)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.3.1a (Software reuse/Customer requirements)*

### 9.1.66 Definition of methods and tools for software intended for reuse (5.4.3.6c)

**ECSS-E-ST-40C Clause 5.4.3.6c**

The supplier shall design the software such that mission and configuration dependant data are separatede.g. in a database.

**Expected Output**: Software architectural design with configuration data - [DDF, SDD; PDR, CDR]

**QDP Status (Ye):** Mission data will not be used by the software product. The application configuration of the software product is defined at link-time through a user provided configuration object. See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.3.1a (Software reuse/Customer requirements)*

### 9.1.67 Reuse of existing software (5.4.3.7a)

**ECSS-E-ST-40C Clause 5.4.3.7a**

The analysis of the potential reusability of existing software components shall be performed through: 1. identification of the reuse components and models with respect to the functional requirements baseline, and; 2. a quality evaluation of these components, applying ECSS-Q-ST-80 clause *6.2.7*.

**Expected Output**: Justification of reuse with respect to requirements baseline [DJF, SRF; PDR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.7.2a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.3a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.4a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.5a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.6a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.7a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.8a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.8b (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.9a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.10a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.11a (Reuse of existing software)*

### 9.1.68 Definition and documentation of the software integration requirements and plan (5.4.3.8a)

**ECSS-E-ST-40C Clause 5.4.3.8a**

The supplier shall define and document the preliminary software integration strategy in terms of responsibility and schedule, control procedures and testing approach (goals to be achieved, sequence, environment and criteria).

**Expected Output**: Software integration strategy [DJF, SUITP; PDR]

**QDP Status (Ye):** See *Software Unit and Integration Test Plan (SUITP)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.69 Conducting a preliminary design review (5.4.4a)

**ECSS-E-ST-40C Clause 5.4.4a**

The supplier shall conduct a preliminary design review (PDR) in accordance with clause *5.3.4.2*. {NOTE: The successful completion of the review establishes a baseline for the development of the software item.}

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.3.4.2a (Preliminary design review)*
- *ECSS-E-ST-40C 5.3.4.2b (Preliminary design review)*

### 9.1.70 Detailed design of each software component (5.5.2.1a)

**ECSS-E-ST-40C Clause 5.5.2.1a**

The supplier shall develop a detailed design for each component of the software and document it.

**Expected Output**: Software components design documents [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.71 Detailed design of each software component (5.5.2.1b)

**ECSS-E-ST-40C Clause 5.5.2.1b**

Each software component shall be refined into lower levels containing software units that can be coded, compiled, and tested.

**Expected Output**: Software components design documents [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.72 Detailed design of each software component (5.5.2.1c)

**ECSS-E-ST-40C Clause 5.5.2.1c**

It shall be ensured that all the software requirements are allocated from the software components to software units.

**Expected Output**: Software components design documents [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.73 Development and documentation of the software interfaces detailed design (5.5.2.2a)

**ECSS-E-ST-40C Clause 5.5.2.2a**

The supplier shall develop and document a detailed design for the interfaces external to the software item, between the software components, and between the software units, in order to allow coding without requiring further information.

**Expected Output**: a. External interfaces design (update) [TS, ICD; CDR]; b. Internal interfaces design (update) [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Interface Control Document (ICD)*, *Software Design Document (SDD)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.74 Production of the detailed design model (5.5.2.3a)

**ECSS-E-ST-40C Clause 5.5.2.3a**

The supplier shall produce the detailed design model of the software components defined during the software architectural design, including their static, dynamic and behavioural aspects.

**Expected Output**: a. Software static design model [DDF, SDD; CDR]; b. Software dynamic design model [DDF, SDD; CDR]; c. Software behavioural design model [DDF, SDD; CDR];

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.8.3.13c (Behaviour modelling verification)*

### 9.1.75 Software detail design method (5.5.2.4a)

**ECSS-E-ST-40C Clause 5.5.2.4a**

The supplier shall use a design method (e.g. object oriented or functional method) to produce the detailed design including: 1. software units, their interfaces, and; 2. software units relationships.

**Expected Output**: Software design method [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.76 Detailed design of real-time software (5.5.2.5a)

**ECSS-E-ST-40C Clause 5.5.2.5a**

The dynamic design model shall be compatible with the computational model selected during the software architectural design model.

**Expected Output**: Real-time software dynamic design model [DDF, SDD; CDR]

**QDP Status (N):** See *No Logical and Computational Model*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.77 Detailed design of real-time software (5.5.2.5b)

**ECSS-E-ST-40C Clause 5.5.2.5b**

The supplier shall document and justify all timing and synchronization mechanisms.

**Expected Output**: Real-time software dynamic design model [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.78 Detailed design of real-time software (5.5.2.5c)

**ECSS-E-ST-40C Clause 5.5.2.5c**

The supplier shall document and justify all the design mutual exclusion mechanisms to manage access to the shared resources.

**Expected Output**: Real-time software dynamic design model [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.79 Detailed design of real-time software (5.5.2.5d)

**ECSS-E-ST-40C Clause 5.5.2.5d**

The supplier shall document and justify the use of dynamic allocation of resources.

**Expected Output**: Real-time software dynamic design model [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.80 Detailed design of real-time software (5.5.2.5e)

**ECSS-E-ST-40C Clause 5.5.2.5e**

The supplier shall ensure protection against problems that can be induced by the use of dynamic allocation of resources,e.g. memory leaks.

**Expected Output**: Real-time software dynamic design model [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.81 Utilization of description techniques for the software behaviour (5.5.2.6a)

**ECSS-E-ST-40C Clause 5.5.2.6a**

The behavioural design of the software units shall be described by means of techniques using automata and scenarios.

**Expected Output**: Software behavioural design model techniques [DDF, SDD; CDR]

**QDP Status (Ye):** See *Software Design Document (SDD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.8.3.13c (Behaviour modelling verification)*

### 9.1.82 Determination of design method consistency for real-time software (5.5.2.7a)

**ECSS-E-ST-40C Clause 5.5.2.7a**

It shall be ensured that all the methods utilized for different item of the same software are, from a dynamic stand-point, consistent among themselves and consistent with the selected computational model.

**Expected Output**: Compatibility of real-time design methods with the computational model [DDF, SDD; CDR]

**QDP Status (N):** See *No Logical and Computational Model*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.83 Development and documentation of the software user manual (5.5.2.8a)

**ECSS-E-ST-40C Clause 5.5.2.8a**

The supplier shall develop and document the software user manual.

**Expected Output**: Software user manual [DDF, SUM; CDR]

**QDP Status (Ye):** See *Software User Manual (SUM)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.84 Definition and documentation of the software unit test requirements and plan (5.5.2.9a)

**ECSS-E-ST-40C Clause 5.5.2.9a**

The supplier shall define and document responsibility and schedule, control procedures, testing approach, test design and test case specification for testing software units.

**Expected Output**: Software unit test plan [DJF, SUITP; CDR]

**QDP Status (Ye):** See *Software Unit and Integration Test Plan (SUITP)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.85 Conducting a detailed design review (5.5.2.10a)

**ECSS-E-ST-40C Clause 5.5.2.10a**

The supplier shall conduct a detailed design review (DDR) as anticipation of the CDR, in conformance with *5.3.4.3b*.

**QDP Status (N/A):** The software detailed design is not baselined before the start of the coding.

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.3.4.3b (Critical design review)*

### 9.1.86 Development and documentation of the software units (5.5.3.1a)

**ECSS-E-ST-40C Clause 5.5.3.1a**

The supplier shall develop and document the following: 1. the coding of each software unit; 2. the build procedures to compile and link software units;

**Expected Output**: a. Software component design documents and code (update) [DDF, SDD, source code; CDR]; b. Software configuration file - build procedures [DDF, SCF; CDR].

**QDP Status (Ye):** See *Software Design Document (SDD)*, *Software Configuration File (SCF)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.6a (Verification of software unit testing (plan and results))*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.87 Software unit testing (5.5.3.2a)

**ECSS-E-ST-40C Clause 5.5.3.2a**

The supplier shall develop and document the test procedures and data for testing each software unit.

**Expected Output**: a. Software component design document and code (update) [DDF, SDD, source code; CDR]; b. Software unit test plan (update) [DJF, SUITP; CDR]

**QDP Status (Ye):** See *On Demand Unit and Integration Testing*, *Software Unit and Integration Test Plan (SUITP)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.6a (Verification of software unit testing (plan and results))*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.88 Software unit testing (5.5.3.2b)

**ECSS-E-ST-40C Clause 5.5.3.2b**

The supplier shall test each software unit ensuring that it satisfies its requirements and document the test results.

**Expected Output**: a. Software component design document and code (update) [DDF, SDD, source code; CDR]; b. Software unit test reports [DJF, - ; CDR]

**QDP Status (Ye):** See *On Demand Unit and Integration Testing*, *Software Unit and Integration Test Plan (SUITP)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.6a (Verification of software unit testing (plan and results))*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.89 Software unit testing (5.5.3.2c)

**ECSS-E-ST-40C Clause 5.5.3.2c**

The unit test shall exercise: 1. code using boundaries at n-1, n, n+1 including looping instructions, while, for and tests that use comparisons; 2. all the messages and error cases defined in the design document; 3. the access of all global variables as specified in the design document; 4. out of range values for input data, including values that can cause erroneous results in mathematical functions; 5. the software at the limits of its requirements (stress testing).

**Expected Output**: Software unit test reports [DJF, - ; CDR]

**QDP Status (Ye):** See *On Demand Unit and Integration Testing*, *Software Unit and Integration Test Plan (SUITP)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.6a (Verification of software unit testing (plan and results))*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.90 Software integration test plan development (5.5.4.1a)

**ECSS-E-ST-40C Clause 5.5.4.1a**

The supplier shall complement the software integration test plan to define the integration of the software units and software components into the software item, providing the following data: 1. test design; 2. test case specification; 3. test procedures; 4. test data.

**Expected Output**: Software integration test plan (update) [DJF, SUITP; CDR]

**QDP Status (Ye):** See *On Demand Unit and Integration Testing*, *Software Unit and Integration Test Plan (SUITP)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.91 Software units and software component integration and testing (5.5.4.2a)

**ECSS-E-ST-40C Clause 5.5.4.2a**

The supplier shall integrate the software units and software components, and test them, as the aggregates are developed, in accordance with the integration plan, ensuring that each aggregate satisfies the requirements of the software item and that the software item is integrated at the conclusion of the integration activity.

**Expected Output**: Software integration test report [DJF, - ; CDR]

**QDP Status (Ye):** See *On Demand Unit and Integration Testing*, *Software Unit and Integration Test Plan (SUITP)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.92 Establishment of a software validation process (5.6.2.1a)

**ECSS-E-ST-40C Clause 5.6.2.1a**

The validation process shall be established to validate the software product.

**Expected Output**: Software validation plan - validation process identification [DJF, SValP; PDR]

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.93 Establishment of a software validation process (5.6.2.1b)

**ECSS-E-ST-40C Clause 5.6.2.1b**

Validation tasks defined in clauses *5.6.3* and *5.6.4* including associated methods, techniques, and tools for performing the tasks, shall be selected and the regression test strategy specified.

**Expected Output**: Software validation plan - methods and tools [DJF, SValP; PDR]

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.6.3.1a (Development and documentation of a software validation specification with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.1b (Development and documentation of a software validation specification with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.1c (Development and documentation of a software validation specification with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.2a (Conducting the validation with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.3a (Updating the software user manual)*
- *ECSS-E-ST-40C 5.6.3.4a (Conducting a critical design review)*
- *ECSS-E-ST-40C 5.6.4.1a (Development and documentation of a software validation specification with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.1b (Development and documentation of a software validation specification with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.1c (Development and documentation of a software validation specification with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.2a (Conducting the validation with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.2b (Conducting the validation with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.3a (Updating the software user manual)*
- *ECSS-E-ST-40C 5.6.4.4a (Conducting a qualification review)*

This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.94 Establishment of a software validation process (5.6.2.1c)

**ECSS-E-ST-40C Clause 5.6.2.1c**

The validation effort and the degree of organizational independence of that effort shall be determined, coherent with ECSS-Q-ST-80 clause *6.3.5.19*.

**Expected Output**: Software validation plan - effort and independence [DJF, SValP; PDR]

**QDP Status (Ye):** No organizational independence shall be established, see [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-Q-ST-80C-R1 6.3.5.19a (Testing and validation)*

This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.95 Selection of an ISVV organization (5.6.2.2a)

**ECSS-E-ST-40C Clause 5.6.2.2a**

If the project warrants an independent validation effort, a qualified organization responsible for conducting the effort shall be selected.

**Expected Output**: Independent software validation plan - organization selection [DJF, - ; PDR]

**QDP Status (N):** See *No Independent Software Verification and Validation*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.96 Selection of an ISVV organization (5.6.2.2b)

**ECSS-E-ST-40C Clause 5.6.2.2b**

The conductor shall be assured of the independence and authority to perform the validation tasks. {NOTE 1: This clause is applied with ECSS-M-ST-10 and ECSS-Q-ST-80, clause *6.3.5.28*.} {NOTE 2: The conductor is the person or the entity that takes in charge the validation tasks (e.g. test cases specification, design, execution and management).}

**Expected Output**: Independent software validation plan - level of independence [DJF, - ; PDR]

**QDP Status (N):** See *No Independent Software Verification and Validation*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-Q-ST-80C-R1 6.3.5.28a (Testing and validation)*

This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.97 Development and documentation of a software validation specification with respect to the technical specification (5.6.3.1a)

**ECSS-E-ST-40C Clause 5.6.3.1a**

The supplier shall develop and document, for each requirement of the software item in TS (including ICD), a set of tests, test cases (inputs, outputs, test criteria) and test procedures including: 1. testing with stress, boundary, and singular inputs; 2. testing the software product for its ability to isolate and reduce the effect of errors; {NOTE: For example: This reduction is done by graceful degradation upon failure, request for operator assistance upon stress, boundary and singular conditions.} 3. testing that the software product can perform successfully in a representative operational environment; 4. external interface testing including boundaries, protocols and timing test; 5. testing HMI applications as per ECSS-E-ST-10-11.

**Expected Output**: Software validation specification with respect to the technical specification [DJF, SVS; CDR]

**QDP Status (Ye):** *HMI* is not present in software product. See *Software Validation Specification (SVS) with Respect to TS*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-E-ST-40C 5.6.3.2a (Conducting the validation with respect to the technical specification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.98 Development and documentation of a software validation specification with respect to the technical specification (5.6.3.1b)

**ECSS-E-ST-40C Clause 5.6.3.1b**

Validation shall be performed by test.

**Expected Output**: Software validation specification with respect to the technical specification [DJF, SVS; CDR]

**QDP Status (Y):** See *Requirement Validation* and *Software Validation Specification (SVS) with Respect to TS*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-E-ST-40C 5.6.3.2a (Conducting the validation with respect to the technical specification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.99 Development and documentation of a software validation specification with respect to the technical specification (5.6.3.1c)

**ECSS-E-ST-40C Clause 5.6.3.1c**

If it can be justified that validation by test cannot be performed, validation shall be performed by either analysis, inspection or review of design.

**Expected Output**: Software validation specification with respect to the technical specification [DJF, SVS; CDR]

**QDP Status (Y):** See *Requirement Validation* and *Software Validation Specification (SVS) with Respect to TS*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-E-ST-40C 5.6.3.2a (Conducting the validation with respect to the technical specification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.100 Conducting the validation with respect to the technical specification (5.6.3.2a)

**ECSS-E-ST-40C Clause 5.6.3.2a**

The validation tests shall be conducted as specified in the output of clause *5.6.3.1*.

**Expected Output**: Software validation report with respect to the technical specification [DJF, - ; CDR]

**QDP Status (Y):** -

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.6.3.1a (Development and documentation of a software validation specification with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.1b (Development and documentation of a software validation specification with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.1c (Development and documentation of a software validation specification with respect to the technical specification)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.101 Updating the software user manual (5.6.3.3a)

**ECSS-E-ST-40C Clause 5.6.3.3a**

The supplier shall update the software user manual in accordance with the results of the validation activities with respect to the technical specification.

**Expected Output**: Software user manual (update) [DDF, SUM; CDR]

**QDP Status (Y):** -

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.102 Conducting a critical design review (5.6.3.4a)

**ECSS-E-ST-40C Clause 5.6.3.4a**

The supplier shall conduct a critical design review (CDR) in accordance with clause *5.3.4.3*.

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.3.4.3a (Critical design review)*
- *ECSS-E-ST-40C 5.3.4.3b (Critical design review)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.103 Development and documentation of a software validation specification with respect to the requirements baseline (5.6.4.1a)

**ECSS-E-ST-40C Clause 5.6.4.1a**

The supplier shall develop and document, for each requirement of the software item in RB (including IRD) , a set of tests, test cases (inputs, outputs, test criteria) and test procedures including: 1. testing against the mission data and scenario specified by the customer in *5.2.3.1* 2. testing with stress, boundary, and singular inputs; 3. testing the software product for its ability to isolate and reduce the effect of errors; {NOTE: For example: This reduction is done by graceful degradation upon failure, request for operator assistance upon stress, boundary and singular conditions.} 4. testing that the software product can perform successfully in a representative operational and non-intrusive environment. 5. external interface testing including boundaries, protocols and timing test; 6. testing HMI applications as per ECSS-E-ST-10-11.

**Expected Output**: Software validation specification with respect to the requirements baseline [DJF, SVS; QR, AR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.2.3.1a (Verification and validation process requirements)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-E-ST-40C 5.6.4.2a (Conducting the validation with respect to the requirements baseline)*

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.104 Development and documentation of a software validation specification with respect to the requirements baseline (5.6.4.1b)

**ECSS-E-ST-40C Clause 5.6.4.1b**

Validation shall be performed by test.

**Expected Output**: Software validation specification with respect to the requirements baseline [DJF, SVS; QR, AR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-E-ST-40C 5.6.4.2a (Conducting the validation with respect to the requirements baseline)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.105 Development and documentation of a software validation specification with respect to the requirements baseline (5.6.4.1c)

**ECSS-E-ST-40C Clause 5.6.4.1c**

If it can be justified that validation by test cannot be performed, validation shall be performed by either analysis, inspection or review of design.

**Expected Output**: Software validation specification with respect to the requirements baseline [DJF, SVS; QR, AR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-E-ST-40C 5.6.4.2a (Conducting the validation with respect to the requirements baseline)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.106 Conducting the validation with respect to the requirements baseline (5.6.4.2a)

**ECSS-E-ST-40C Clause 5.6.4.2a**

The validation tests shall be conducted as specified in the output of clause *5.6.4.1*.

**Expected Output**: Software validation report with respect to the requirements baseline [DJF, - ; QR, AR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.6.4.1a (Development and documentation of a software validation specification with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.1b (Development and documentation of a software validation specification with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.1c (Development and documentation of a software validation specification with respect to the requirements baseline)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.107 Conducting the validation with respect to the requirements baseline (5.6.4.2b)

**ECSS-E-ST-40C Clause 5.6.4.2b**

The validation tests shall be “black box”,i.e. performed on the final software product to be delivered , without any modification of the code or of the data. {NOTE: In particular, this is essential when an mission database is used to customize the final product, and when late versions of the database are used to update the software.}

**Expected Output**: Software validation report with respect to the requirements baseline [DJF, - ; QR, AR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.108 Updating the software user manual (5.6.4.3a)

**ECSS-E-ST-40C Clause 5.6.4.3a**

The supplier shall update the software user manual in accordance with the results of the validation activities with respect to the requirements baseline.

**Expected Output**: Software user manual (update) [DDF, SUM; QR, AR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.109 Conducting a qualification review (5.6.4.4a)

**ECSS-E-ST-40C Clause 5.6.4.4a**

The qualification review (QR) shall be conducted in accordance with clause *5.3.4.4*.

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.3.4.4a (Qualification review)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.110 Preparation of the software product (5.7.2.1a)

**ECSS-E-ST-40C Clause 5.7.2.1a**

The supplier shall prepare the deliverable software product for its installation in the target platform.

**Expected Output**: a. Software product [DDF, - ; QR, AR]; b. Software release document [DDF, SRelD; QR, AR]

**QDP Status (Ye):** See *Software Release Document (SRelD)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.111 Supplier's provision of training and support (5.7.2.2a)

**ECSS-E-ST-40C Clause 5.7.2.2a**

The supplier shall provide initial and continuing training and support to the customer if specified in the requirements baseline.

**Expected Output**: Training material [DDF, - ; QR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.112 Installation procedures (5.7.2.3a)

**ECSS-E-ST-40C Clause 5.7.2.3a**

The supplier shall develop procedures to install the software product in the target environment.

**Expected Output**: Installation procedures [DDF, SCF ; AR]

**QDP Status (Ye):** See *Software Configuration File (SCF)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.1.113 Installation activities reporting (5.7.2.4a)

**ECSS-E-ST-40C Clause 5.7.2.4a**

The resources and information to install the software product shall be determined and be available.

**Expected Output**: Installation report [DJF, - ; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.114 Installation activities reporting (5.7.2.4b)

**ECSS-E-ST-40C Clause 5.7.2.4b**

The supplier shall assist the customer with the set-up activities.

**Expected Output**: Installation report [DJF, - ; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.115 Installation activities reporting (5.7.2.4c)

**ECSS-E-ST-40C Clause 5.7.2.4c**

It shall be ensured that the software code and databases initialize, execute and terminate as specified in the installation plan.

**Expected Output**: Installation report [DJF, - ; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.116 Installation activities reporting (5.7.2.4d)

**ECSS-E-ST-40C Clause 5.7.2.4d**

The installation events and results shall be documented.

**Expected Output**: Installation report [DJF, - ; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.117 Acceptance test planning (5.7.3.1a)

**ECSS-E-ST-40C Clause 5.7.3.1a**

The customer shall establish an acceptance test plan specifying the intended acceptance tests with tests suited to the target environment.

**Expected Output**: Acceptance test plan [DJF, - ; QR, AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.6.3a (Software delivery and acceptance)*

### 9.1.118 Acceptance test execution (5.7.3.2a)

**ECSS-E-ST-40C Clause 5.7.3.2a**

The customer shall perform the acceptance testing.

**Expected Output**: Acceptance test report [DJF, - ; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.3.6.4a (Software delivery and acceptance)*

### 9.1.119 Executable code generation and installation (5.7.3.3a)

**ECSS-E-ST-40C Clause 5.7.3.3a**

The acceptance shall include generation of the executable code from configuration managed source code components and its installation on the target environment.

**Expected Output**: Software product [DDF, - ; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.120 Supplier's support to customer's acceptance (5.7.3.4a)

**ECSS-E-ST-40C Clause 5.7.3.4a**

The supplier shall support the customer's acceptance reviews and testing of the software product in preparation of the AR.

**Expected Output**: Joint review reports [DJF, -; AR]. {NOTE: Acceptance reviews and testing considers the results of the joint reviews (see 5.3.3), audits , testing and validation (see ECSS-Q-ST-80 clauses 5.2.3 and 6.3.5), and system validation testing (if performed)}

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.121 Supplier's support to customer's acceptance (5.7.3.4b)

**ECSS-E-ST-40C Clause 5.7.3.4b**

The results of the acceptance reviews and testing shall be documented.

**Expected Output**: Joint review reports [DJF, -; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.122 Evaluation of acceptance testing (5.7.3.5a)

**ECSS-E-ST-40C Clause 5.7.3.5a**

The acceptance tests shall be traced to the requirements baseline.

**Expected Output**: Traceability of acceptance tests to the requirements baseline [DJF, SVR; AR]

**QDP Status (US):** See *No Requirements Baseline (RB)* and *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.123 Conducting an acceptance review (5.7.3.6a)

**ECSS-E-ST-40C Clause 5.7.3.6a**

The acceptance review (AR) shall be conducted in accordance with clause *5.3.4.5*.

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.3.4.5a (Acceptance review)*

### 9.1.124 Establishment of the software verification process (5.8.2.1a)

**ECSS-E-ST-40C Clause 5.8.2.1a**

The verification process shall be established by the supplier to verify the software products.

**Expected Output**: Software verification plan - verification process identification [DJF, SVerP; PDR]

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.125 Establishment of the software verification process (5.8.2.1b)

**ECSS-E-ST-40C Clause 5.8.2.1b**

Life cycle activities and software products needing verification shall be determined based upon the scope, magnitude, complexity, and criticality analysis.

**Expected Output**: Software verification plan - software products identification [DJF, SVerP; PDR]

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.126 Establishment of the software verification process (5.8.2.1c)

**ECSS-E-ST-40C Clause 5.8.2.1c**

Verification activities and tasks defined in clause *5.8.3*, including associated methods, techniques, and tools for performing the tasks, shall be selected for the life cycle activities and software products.

**Expected Output**: Software verification plan - activities, methods and tools [DJF, SVerP; PDR]

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.8.3.1a (Verification of requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.2a (Verification of the technical specification)*
- *ECSS-E-ST-40C 5.8.3.3a (Verification of the software architectural design)*
- *ECSS-E-ST-40C 5.8.3.4a (Verification of the software detailed design)*
- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5b (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5c (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5d (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5e (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5f (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.6a (Verification of software unit testing (plan and results))*
- *ECSS-E-ST-40C 5.8.3.7a (Verification of software integration)*
- *ECSS-E-ST-40C 5.8.3.8a (Verification of software validation with respect to the technical specifications and the requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.8b (Verification of software validation with respect to the technical specifications and the requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.9a (Evaluation of validation: complementary system level validation)*
- *ECSS-E-ST-40C 5.8.3.10a (Verification of software documentation)*
- *ECSS-E-ST-40C 5.8.3.11a (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.11b (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.11c (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.12a (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.12b (Technical budgets management)*

- *ECSS-E-ST-40C 5.8.3.12c (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.13a (Behaviour modelling verification)*
- *ECSS-E-ST-40C 5.8.3.13b (Behaviour modelling verification)*
- *ECSS-E-ST-40C 5.8.3.13c (Behaviour modelling verification)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.127 Establishment of the software verification process (5.8.2.1d)

**ECSS-E-ST-40C Clause 5.8.2.1d**

A determination shall be made concerning the verification effort, the identification of risks and the degree of organizational independence.

**Expected Output**: Software verification plan - organizational independence, risk and effort identification [DJF, SVerP; PDR]

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.128 Selection of the organization responsible for conducting the verification (5.8.2.2a)

**ECSS-E-ST-40C Clause 5.8.2.2a**

If the project warrants an independent verification effort, a qualified organization shall be selected for conducting the verification.

**Expected Output**: Independent software verification plan - organization selection [DJF, - ; PDR]

**QDP Status (N):** See *No Independent Software Verification and Validation*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.129 Selection of the organization responsible for conducting the verification (5.8.2.2b)

**ECSS-E-ST-40C Clause 5.8.2.2b**

This organization shall have the independence and authority needed to perform the verification activities. {NOTE: ECSS-Q-ST-80 clause *6.2.6.13* (independent software verification) and ECSS-M-ST-10 (project planning and implementation) contain further requirements relevant for this clause.} {AIM: A coherent and consistent approach to project organization within each project.}

**Expected Output**: Independent software verification plan - level of independence [DJF, - ; PDR]

**QDP Status (N):** See *No Independent Software Verification and Validation*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.6.13a (Verification)*
- *ECSS-Q-ST-80C-R1 6.2.6.13b (Verification)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.130 Verification of requirements baseline (5.8.3.1a)

**ECSS-E-ST-40C Clause 5.8.3.1a**

The customer shall verify that the requirements baseline, including the interface requirements document: 1. specifies a clear description of the environment in which the software operates; 2. specifies the characteristics of all external systems (e.g. bus, computer, ground interface) in interaction with the software product; 3. specifies the controllability and observability points

for each application; 4. specifies the fault detection, identification, and recovery strategy to be implemented, and that the strategy is coherent with the dependability and safety level of the software under consideration; 5. specifies the modes/submodes and transition between modes (modes automaton); 6. specifies telemetries date management occurrences; 7. identifies the configuration data of the software; 8. identifies and justifies the margins policy in terms of memory and CPU allocation; 9. defines operational scenario; 10. includes consistent and verifiable requirements.

**Expected Output**: Requirements baseline verification report [DJF, SVR; SRR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.131 Verification of the technical specification (5.8.3.2a)

**ECSS-E-ST-40C Clause 5.8.3.2a**

The supplier shall verify the technical specification including the interface control document ensuring that: 1. software requirements and interface are externally and internally consistent (not implying formal proof consistency); 2. the traceability between system requirements and software requirements is complete; 3. the software requirements that are not traced to the system requirements allocated to software are justified; 4. software requirements are verifiable; 5. software design is feasible; 6. operations and maintenance are feasible; 7. the software requirements related to safety, security, and criticality are correct; 8. the hardware environment constraints are identified; 9. the implementation constraints are identified; 10. the requirement verification method as specified in ECSS-Q-ST-80 clause *7.2.1.3* is feasible. 11. the logical model has been checked;

**Expected Output**: a. Requirements traceability matrices [DJF, SVR or (SRS and ICD); PDR]; b. Requirements verification report [DJF, SVR; PDR]

**QDP Status (Ye):** See *No Requirements Baseline (RB)*, *No Maintenance (MF)*, *No Operational Phase (OP)*, *No Logical and Computational Model*, *Software Verification Report (SVR)*, *Software Requirements Specification (SRS)*, *Software Interface Control Document (ICD)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-Q-ST-80C-R1 7.2.1.3a (Requirements baseline and technical specification)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.132 Verification of the software architectural design (5.8.3.3a)

**ECSS-E-ST-40C Clause 5.8.3.3a**

The supplier shall verify the architecture of the software item and the interface design ensuring that: 1. architecture and interface are externally consistent with the requirements of the software item; 2. there is internal consistency between the software components; 3. the traceability between the requirements and the software components is complete; 4. the software components that are not traced to the software requirements are justified; 5. producing a detailed design is feasible; 6. operations and maintenance are feasible; 7. the design is correct with respect to the requirements and the interfaces, including safety, security and other critical requirements; 8. the design implements proper sequence of events, inputs, outputs, interfaces, logic flow, allocation of timing and sizing budgets, and error handling; 9. the hierarchical breakdown from high level components to terminal ones is provided; 10. the dynamic features (tasks definition and priorities, synchronization mechanisms, shared resources management) are provided and the real-time choices are justified; 11. the synchronisation between external interface and internal timing is achieved.

**Expected Output**: a. Software architectural design to requirements traceability matrices [DJF, SVR or SDD; PDR]; b. Software architectural design and interface verification report [DJF, SVR; PDR]

**QDP Status (Ye):** See *Software Verification Report (SVR)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.133 Verification of the software detailed design (5.8.3.4a)

**ECSS-E-ST-40C Clause 5.8.3.4a**

The supplier shall verify the software detailed design ensuring that: 1. detailed design is externally consistent with the architecture; 2. there is internal consistency between software components and software units; 3. the traceability between the architecture and the detailed design is complete; 4. the software units that are not traced to the components are justified; 5. testing is feasible, by assessing that: (a) commandability and observability features are identified and included in the detailed design in order to prepare the effective testing of the performance requirements; (b) computational invariant properties and temporal properties are added within the design; (c) fault injection is possible. 6. operation and maintenance are feasible; 7. the design is correct with respect to requirements and interfaces, including safety, security, and other critical requirements; 8. the design implements proper sequence of events, inputs, outputs, interfaces, logic flow, allocation of timing and sizing budgets, and error handling; 9. the design model has been checked; 10. 1the hierarchical breakdown from high level components to terminal ones is provided; 11. 1the dynamic features (tasks definition and priorities, synchronization mechanisms, shared resources management) are provided and the real-time choices are justified; 12. 1the synchronisation between external interface and internal timing is achieved;

**Expected Output**: a. Detailed design traceability matrices [DJF, SVR or SDD; CDR]; b. Detailed design verification report [DJF, SVR; CDR]

**QDP Status (Ye):** See *Software Verification Report (SVR)*, *Software Design Document (SDD)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.134 Verification of code (5.8.3.5a)

**ECSS-E-ST-40C Clause 5.8.3.5a**

The supplier shall verify the software code ensuring that: 1. the code is externally consistent with the requirements and design of the software item; 2. there is internal consistency between software units; 3. the code is traceable to design and requirements, testable, correct, and in conformity to software requirements and coding standards; 4. the code that is not traced to the units is justified; 5. the code implements proper events sequences, consistent interfaces, correct data and control flow, completeness, appropriate allocation of timing and

sizing budgets, and error handling; 6. the code implements safety, security, and other critical requirements correctly as shown by appropriate methods; 7. the effects of run-time errors are controlled; 8. there are no memory leaks; 9. numerical protection mechanisms are implemented. {NOTE: "AM" means that the value is agreed with the customer and measured as per ECSS-Q-ST-80 clause *6.3.5.2*.}

**Expected Output**: a. Software code traceability matrices [DJF, SVR; CDR]; b. Software code verification report [DJF, SVR; CDR]

**QDP Status (Ye):** See *Software Verification Report (SVR)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.3.5.2a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.20a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.21a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.22a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.23a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.24a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.25a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.26a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.27a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.28a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.29a (Testing and validation)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.135 Verification of code (5.8.3.5b)

**ECSS-E-ST-40C Clause 5.8.3.5b**

The supplier shall verify that the following code coverage is achieved (category=coverage). Source code statement coverage: A=100% B=100% C=AM D=AM. Source code decision coverage: A=100% B=100% C=AM D=AM. Source code modified condition and decision coverage: A=100% B=AM C=AM D=AM. {NOTE: This requirement is met by running unit,

integration and validation tests, measuring the code coverage, and achieving the code coverage by additional (requirement based) tests, inspection or analysis.}

**Expected Output**: Code coverage verification report [DJF, SVR; CDR, QR, AR]

**QDP Status (Ye):** The QDP will aim at 100% source code statement coverage and 100% source code decision coverage.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.136 Verification of code (5.8.3.5c)

**ECSS-E-ST-40C Clause 5.8.3.5c**

Code coverage shall be measured by analysis of the results of the execution of tests.

**Expected Output**: Code coverage verification report [DJF, SVR; CDR, QR, AR]

**QDP Status (Y):** See *Software Verification Report (SVR)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.137 Verification of code (5.8.3.5d)

**ECSS-E-ST-40C Clause 5.8.3.5d**

If it can be justified that the required percentage cannot be achieved by test execution, then analysis, inspection or review of design shall be applied to the non covered code. {AIM: The goal of the complementary analysis is to assess that the non covered code behaviour is as expected.}

**Expected Output**: Code coverage verification report [DJF, SVR; CDR, QR, AR]

**QDP Status (Y):** See *Software Verification Report (SVR)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.138 Verification of code (5.8.3.5e)

**ECSS-E-ST-40C Clause 5.8.3.5e**

In case the traceability between source code and object code cannot be verified (e.g. use of compiler optimization), the supplier shall perform additional code coverage analysis on object code level as follows (category=coverage). Object code coverage: A=100% B=N/A C=N/A D=N/A. {NOTE: N/A means not applicable.}

**Expected Output**: Code coverage verification report [DJF, SVR; CDR, QR, AR]

**QDP Status (N/A):** No traceability between source code and object code required for criticality B, C, and D.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.139 Verification of code (5.8.3.5f)

**ECSS-E-ST-40C Clause 5.8.3.5f**

The supplier shall verify source code robustness (e.g. resource sharing, division by zero, pointers, run-time errors). {AIM: use static analysis for the errors that are difficult to detect at run-time.}

**Expected Output**: Robustness verification report [ DJF, SVR; CDR]

**QDP Status (Y):** Static analysis tools will be used to verify the robustness of the source code. See *Software Verification Report (SVR)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.140 Verification of software unit testing (plan and results) (5.8.3.6a)

**ECSS-E-ST-40C Clause 5.8.3.6a**

The supplier shall verify the unit tests results ensuring that: 1. the unit tests are consistent with detailed design and requirements; 2. the unit tests are traceable to software requirements, design and code; {NOTE: The trace to requirements is used to design the unit test cases in order to predict meaningful expected results.} 3. software integration and testing are feasible; 4. operation and maintenance are feasible; 5. all activities defined in clause *5.5.3* are performed; 6. test results conform to expected results; 7. test results, test logs, test data, test cases and procedures, and test documentation are maintained under configuration management; 8. normal termination (i.e. the test end criteria defined in the unit test plan) is achieved; 9. abnormal termination of testing process (e.g. incorrect major fault, out of time) is reported; 10. 1abnormal termination condition is documented in summary section of the unit test report, together with the unfinished testing and any uncorrected faults.

**Expected Output**: a. Software unit tests traceability matrices [DJF, SVR; CDR]; b. Software unit testing verification report [DJF, SVR; CDR]

**QDP Status (Ye):** See *Software Verification Report (SVR)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.5.3.1a (Development and documentation of the software units)*
- *ECSS-E-ST-40C 5.5.3.2a (Software unit testing)*
- *ECSS-E-ST-40C 5.5.3.2b (Software unit testing)*
- *ECSS-E-ST-40C 5.5.3.2c (Software unit testing)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.141 Verification of software integration (5.8.3.7a)

**ECSS-E-ST-40C Clause 5.8.3.7a**

The supplier shall verify that the integration has been performed according to the strategy specified in the software integration test plan, and the integration activities ensuring: 1. traceability to software architectural design; 2. internal consistency; 3. interface testing goals; 4. conformance to expected results.

**Expected Output**: Software integration verification report [DJF, SVR; CDR]

**QDP Status (Ye):** See *Software Verification Report (SVR)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.142 Verification of software validation with respect to the technical specifications and the requirements baseline (5.8.3.8a)

**ECSS-E-ST-40C Clause 5.8.3.8a**

The supplier shall verify the software validation results ensuring that the test requirements, test cases, test specifications, analysis, inspection and review of design cover all software requirements of the technical specification or the requirements baseline.

**Expected Output**: a. Traceability of the requirements baseline to the validation specification [DJF, SVR or SVS; QR, AR]; b. Traceability of the technical specification to the validation specification [DJF, SVR or SVS; CDR]

**QDP Status (Ye):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.143 Verification of software validation with respect to the technical specifications and the requirements baseline (5.8.3.8b)

**ECSS-E-ST-40C Clause 5.8.3.8b**

The supplier shall verify the software validation results ensuring conformance to expected results.

**Expected Output**: a. Validation report evaluation with respect to the technical specification [DJF, SVR; CDR]; b. Validation report evaluation with respect to the requirements baseline [DJF, SVR; QR]

**QDP Status (Ye):** See *No Requirements Baseline (RB)*, *Software Verification Report (SVR)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.144 Evaluation of validation: complementary system level validation (5.8.3.9a)

**ECSS-E-ST-40C Clause 5.8.3.9a**

The supplier shall identify the requirements of the technical specification and the requirements baseline that cannot be tested in its own environment, and shall forward to the customer a request to validate them at system level. {NOTE: For example: Some of the requirements cannot be verified because the test environment used for the validation does not allow it. These requirements can only be tested when the software is integrated within the system (e.g. satellite and launcher).}

**Expected Output**: Complement of validation at system level [DJF, SValP; PDR]

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.145 Verification of software documentation (5.8.3.10a)

**ECSS-E-ST-40C Clause 5.8.3.10a**

The supplier shall verify the software documentation ensuring that: 1. the documentation is adequate, complete, and consistent; 2. documentation preparation is timely; 3. configuration management of documents follows specified procedures.

**Expected Output**: Software documentation verification report [DJF, SVR; PDR, CDR, QR]

**QDP Status (Ye):** See *Software Verification Report (SVR)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.146 Schedulability analysis for real-time software (5.8.3.11a)

**ECSS-E-ST-40C Clause 5.8.3.11a**

As part of the verification of the software requirements and architectural design , the supplier shall use an analytical model (or use modelling and simulation if it can be demonstrated that no analytical model exists) to perform a schedulability analysis and prove that the design is feasible. {NOTE: The schedulability analysis proves that the real-time behaviour is predictable,i.e. that all the tasks complete before their deadline in the worst case condition.}

**Expected Output**: Schedulability analysis [DJF, SVR; PDR]

**QDP Status (N/A):** See *No Schedulability Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.147 Schedulability analysis for real-time software (5.8.3.11b)

**ECSS-E-ST-40C Clause 5.8.3.11b**

As part of the verification of the software detailed design , the supplier shall refine the schedulability analysis performed during the software architectural design on the basis of the software detailed design documentation.

**Expected Output**: Schedulability analysis (update) [DJF, SVR; CDR]

**QDP Status (N/A):** See *No Schedulability Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.148 Schedulability analysis for real-time software (5.8.3.11c)

**ECSS-E-ST-40C Clause 5.8.3.11c**

As part of the verification of the software coding and testing , the supplier shall update the schedulability analysis performed during the software detailed design with the actual information extracted from the code.

**Expected Output**: Schedulability analysis (update) [DJF, SVR; CDR]

**QDP Status (N/A):** See *No Schedulability Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*

- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.149 Technical budgets management (5.8.3.12a)

**ECSS-E-ST-40C Clause 5.8.3.12a**

As part of the verification of the software requirements and architectural design, the supplier shall estimate the technical budgets including memory size, CPU utilization and the way the deadline are met.

**Expected Output**: Technical budgets - memory and CPU estimation [DJF, SVR; PDR]

**QDP Status (Ye):** See *No Requirements Baseline (RB)* and *Resources and Performance*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.150 Technical budgets management (5.8.3.12b)

**ECSS-E-ST-40C Clause 5.8.3.12b**

As part of the verification of the software detailed design, the supplier shall update the estimation of the technical budgets.

**Expected Output**: Technical budgets (update) - memory and CPU estimation [DJF, SVR; CDR]

**QDP Status (Ye):** See *No Requirements Baseline (RB)* and *Resources and Performance*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.151 Technical budgets management (5.8.3.12c)

**ECSS-E-ST-40C Clause 5.8.3.12c**

As part of the verification of the coding, testing and validation, the technical budgets shall be updated with the measured values and shall be compared to the margins.

**Expected Output**: Technical budgets (update) - memory and CPU calculation [DJF, SVR; CDR, QR, AR]

**QDP Status (Ye):** See *No Requirements Baseline (RB)* and *Resources and Performance*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.152 Behaviour modelling verification (5.8.3.13a)

**ECSS-E-ST-40C Clause 5.8.3.13a**

As support to the verification of the software requirements, the supplier shall verify the software behaviour using the behavioural view of the logical model produced in *5.4.2.3c*.

**Expected Output**: Software behaviour verification [DJF, SVR; PDR]

**QDP Status (N):** See *No Logical and Computational Model*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.4.2.3c (Construction of a software logical model)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.153 Behaviour modelling verification (5.8.3.13b)

**ECSS-E-ST-40C Clause 5.8.3.13b**

As support to the verification of the software architectural design, the supplier shall verify the software behaviour using the behavioural view of the architecture produced in clause *5.4.3.4*

**Expected Output**: Software behaviour verification [DJF, SVR; PDR]

**QDP Status (N):** See *No Logical and Computational Model*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.4.3.4a (Description of software behaviour)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.154 Behaviour modelling verification (5.8.3.13c)

**ECSS-E-ST-40C Clause 5.8.3.13c**

As support to the verification of the software detailed design, the supplier shall verify the software behaviour using the software behavioural design model produced in *5.5.2.3a*. eoc., by means of the techniques defined in *5.5.2.6*.

**Expected Output**: Software behaviour verification [DJF, SVR;CDR]

**QDP Status (N):** See *No Logical and Computational Model*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.5.2.3a (Production of the detailed design model)*
- *ECSS-E-ST-40C 5.5.2.6a (Utilization of description techniques for the software behaviour)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-Q-ST-80C-R1 6.2.6.12a (Verification)*

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.1.155 Operational testing definition (5.9.2.1a)

**ECSS-E-ST-40C Clause 5.9.2.1a**

The SOS entity shall establish procedures for: 1. testing the software product in its operational environment; 2. entering problem reports and modification requests to the maintenance process (see clause *5.10*), and; 3. releasing the software product for operational use in accordance with the change control established and maintained in conformance with ECSS-M-ST-40.

**Expected Output**: Software operation support plan - operational testing specifications [OP, - ; ORR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.10.2.1a (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.1b (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.1c (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.1d (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.2a (Long term maintenance for flight software)*
- *ECSS-E-ST-40C 5.10.3.1a (Problem analysis)*
- *ECSS-E-ST-40C 5.10.3.1b (Problem analysis)*
- *ECSS-E-ST-40C 5.10.3.1c (Problem analysis)*
- *ECSS-E-ST-40C 5.10.3.1d (Problem analysis)*
- *ECSS-E-ST-40C 5.10.3.1e (Problem analysis)*
- *ECSS-E-ST-40C 5.10.4.1a (Analysis and documentation of product modification)*
- *ECSS-E-ST-40C 5.10.4.2a (Documentation of software product changes)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-E-ST-40C 5.10.4.3b (Invoking of software engineering processes for modification implementation)*
- *ECSS-E-ST-40C 5.10.4.3c (Invoking of software engineering processes for modification implementation)*
- *ECSS-E-ST-40C 5.10.4.3d (Invoking of software engineering processes for modification implementation)*

- *ECSS-E-ST-40C 5.10.4.3e (Invoking of software engineering processes for modification implementation)*
- *ECSS-E-ST-40C 5.10.5.1a (Maintenance reviews)*
- *ECSS-E-ST-40C 5.10.5.2a (Baseline for change)*
- *ECSS-E-ST-40C 5.10.6.1a (Applicability of this Standard to software migration)*
- *ECSS-E-ST-40C 5.10.6.2a (Migration planning and execution)*
- *ECSS-E-ST-40C 5.10.6.3a (Contribution to the migration plan)*
- *ECSS-E-ST-40C 5.10.6.4a (Preparation for migration)*
- *ECSS-E-ST-40C 5.10.6.5a (Notification of transition to migrated system)*
- *ECSS-E-ST-40C 5.10.6.5b (Notification of transition to migrated system)*
- *ECSS-E-ST-40C 5.10.6.6a (Post-operation review)*
- *ECSS-E-ST-40C 5.10.6.6b (Post-operation review)*
- *ECSS-E-ST-40C 5.10.6.7a (Maintenance and accessibility of data of former system)*
- *ECSS-E-ST-40C 5.10.7.1a (Retirement planning)*
- *ECSS-E-ST-40C 5.10.7.2a (Notification of retirement plan)*
- *ECSS-E-ST-40C 5.10.7.3a (Identification of requirements for software retirement)*
- *ECSS-E-ST-40C 5.10.7.4a (Maintenance and accessibility to data of the retired product)*

### 9.1.156 Software operation support plans and procedures development (5.9.2.2a)

**ECSS-E-ST-40C Clause 5.9.2.2a**

The SOS entity shall complement the software user manual with the additional plans and procedures necessary to support the operation of the software and to perform the user support.

**Expected Output**: Software operation support plan - plans and procedures [OP, - ; ORR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.157 Problem handling procedures definition (5.9.2.3a)

**ECSS-E-ST-40C Clause 5.9.2.3a**

The SOS entity shall establish procedures for receiving, recording, resolving, tracking problems, and providing feedback. {NOTE: ECSS-Q-ST-80 clause *5.2.6* (nonconformances) and clause *5.2.5* (software problems) contain further requirements relevant for this clause.}

**Expected Output**: Software operation support plan - procedures for problem handling [OP, - ; ORR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 5.2.5.1a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.5.2a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.5.3a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.5.4a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.6.1a (Nonconformances)*
- *ECSS-Q-ST-80C-R1 5.2.6.1b (Nonconformances)*
- *ECSS-Q-ST-80C-R1 5.2.6.1c (Nonconformances)*
- *ECSS-Q-ST-80C-R1 5.2.6.2a (Nonconformances)*

### 9.1.158 Operational testing execution (5.9.3.1a)

**ECSS-E-ST-40C Clause 5.9.3.1a**

For each release of the software product, the SOS entity shall perform operational testing in accordance with the applicable procedures.

**Expected Output**: Operational testing results [OP, - ; ORR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.159 Software operational requirements demonstration (5.9.3.2a)

**ECSS-E-ST-40C Clause 5.9.3.2a**

The customer shall ensure that, prior to the first operations, the software is capable of implementing the operational requirements, testing the software in the following conditions: 1. the operating hardware environment, 2. the cases in which the software is designed to be fault tolerant, 3. the system configuration, 4. the sequence of operations, and; 5. the SOS entity interventions. {NOTE: This demonstration can be part of the acceptance tests of the system.}

**Expected Output**: Operational testing results [OP, - ; ORR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.160 Software release (5.9.3.3a)

**ECSS-E-ST-40C Clause 5.9.3.3a**

The software product shall be released for operational use.

**Expected Output**: Software product [DDF, - ; ORR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.161 Software operation support performance (5.9.4.1a)

**ECSS-E-ST-40C Clause 5.9.4.1a**

The software operation support plan shall be executed.

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.162 Problem handling (5.9.4.2a)

**ECSS-E-ST-40C Clause 5.9.4.2a**

Encountered problems shall be recorded and handled in accordance with the applicable procedures.

**Expected Output**: Problem and nonconformance report [OP, - ; - ]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.163 Assistance to the user (5.9.5.1a)

**ECSS-E-ST-40C Clause 5.9.5.1a**

The SOS entity shall provide assistance and consultation to the users.

**Expected Output**: User's request record - user's request and subsequent actions [OP, - ; - ]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.164 Assistance to the user (5.9.5.1b)

**ECSS-E-ST-40C Clause 5.9.5.1b**

The SOS entity shall record and monitor user's requests and subsequent actions.

**Expected Output**: User's request record - user's request and subsequent actions [OP, - ; - ]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.165 Handling of user's requests (5.9.5.2a)

**ECSS-E-ST-40C Clause 5.9.5.2a**

The SOS entity shall forward user requests to the maintenance process for resolution.

**Expected Output**: User's request record - actions [OP, - ; - ]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.166 Handling of user's requests (5.9.5.2b)

**ECSS-E-ST-40C Clause 5.9.5.2b**

The SOS entity shall address user's requests.

**Expected Output**: User's request record - actions [OP, - ; - ]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.167 Handling of user's requests (5.9.5.2c)

**ECSS-E-ST-40C Clause 5.9.5.2c**

The SOS entity shall report to the originators of the requests the actions that are planned and taken.

**Expected Output**: User's request record - actions [OP, - ; - ]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.168 Provisions of work-around solutions (5.9.5.3a)

**ECSS-E-ST-40C Clause 5.9.5.3a**

If a reported problem has a temporary work-around solution before a permanent solution can be released, the SOS entity shall give to the originator of the problem report the option to use it.

**Expected Output**: User's request record - work around solution [OP, - ; - ]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.1.169 Provisions of work-around solutions (5.9.5.3b)

**ECSS-E-ST-40C Clause 5.9.5.3b**

Permanent corrections, releases that include previously omitted functions or features, and system improvements shall be applied to the operational software product using the maintenance process as specified in clause *5.10*.

**Expected Output**: User's request record - work around solution [OP, - ; - ]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.10.2.1a (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.1b (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.1c (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.1d (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*
- *ECSS-E-ST-40C 5.10.2.2a (Long term maintenance for flight software)*
- *ECSS-E-ST-40C 5.10.3.1a (Problem analysis)*
- *ECSS-E-ST-40C 5.10.3.1b (Problem analysis)*
- *ECSS-E-ST-40C 5.10.3.1c (Problem analysis)*
- *ECSS-E-ST-40C 5.10.3.1d (Problem analysis)*
- *ECSS-E-ST-40C 5.10.3.1e (Problem analysis)*
- *ECSS-E-ST-40C 5.10.4.1a (Analysis and documentation of product modification)*
- *ECSS-E-ST-40C 5.10.4.2a (Documentation of software product changes)*
- *ECSS-E-ST-40C 5.10.4.3a (Invoking of software engineering processes for modification implementation)*
- *ECSS-E-ST-40C 5.10.4.3b (Invoking of software engineering processes for modification implementation)*
- *ECSS-E-ST-40C 5.10.4.3c (Invoking of software engineering processes for modification implementation)*
- *ECSS-E-ST-40C 5.10.4.3d (Invoking of software engineering processes for modification implementation)*
- *ECSS-E-ST-40C 5.10.4.3e (Invoking of software engineering processes for modification implementation)*
- *ECSS-E-ST-40C 5.10.5.1a (Maintenance reviews)*
- *ECSS-E-ST-40C 5.10.5.2a (Baseline for change)*

- *ECSS-E-ST-40C 5.10.6.1a (Applicability of this Standard to software migration)*
- *ECSS-E-ST-40C 5.10.6.2a (Migration planning and execution)*
- *ECSS-E-ST-40C 5.10.6.3a (Contribution to the migration plan)*
- *ECSS-E-ST-40C 5.10.6.4a (Preparation for migration)*
- *ECSS-E-ST-40C 5.10.6.5a (Notification of transition to migrated system)*
- *ECSS-E-ST-40C 5.10.6.5b (Notification of transition to migrated system)*
- *ECSS-E-ST-40C 5.10.6.6a (Post-operation review)*
- *ECSS-E-ST-40C 5.10.6.6b (Post-operation review)*
- *ECSS-E-ST-40C 5.10.6.7a (Maintenance and accessibility of data of former system)*
- *ECSS-E-ST-40C 5.10.7.1a (Retirement planning)*
- *ECSS-E-ST-40C 5.10.7.2a (Notification of retirement plan)*
- *ECSS-E-ST-40C 5.10.7.3a (Identification of requirements for software retirement)*
- *ECSS-E-ST-40C 5.10.7.4a (Maintenance and accessibility to data of the retired product)*

### 9.1.170 Establishment of the software maintenance process (5.10.2.1a)

**ECSS-E-ST-40C Clause 5.10.2.1a**

The maintainer shall develop, document, and execute plans and procedures for conducting the activities and tasks of the maintenance process.

**Expected Output**: Maintenance plan - plans and procedures [MF, - ; QR, AR, ORR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.171 Establishment of the software maintenance process (5.10.2.1b)

**ECSS-E-ST-40C Clause 5.10.2.1b**

Software maintenance shall be performed using the same procedures, methods, tools and standards as used for the development.

**Expected Output**: Maintenance plan - applicability of development process procedures, methods, tools and standards [MF, - ; QR, AR, ORR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.172 Establishment of the software maintenance process (5.10.2.1c)

**ECSS-E-ST-40C Clause 5.10.2.1c**

The maintainer shall implement (or establish the organizational interface with) the configuration management process (ECSS-M-ST-40) for managing modifications.

**Expected Output**: Maintenance plan - configuration management process [MF, - ; QR, AR, ORR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.173 Establishment of the software maintenance process (5.10.2.1d)

**ECSS-E-ST-40C Clause 5.10.2.1d**

The maintainer shall establish procedures for receiving, recording and tracking problem reports and modification requests, providing feedback to the requester.

**Expected Output**: Maintenance plan - problem reporting and handling [MF, - ; QR, AR, ORR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.174 Establishment of the software maintenance process (5.10.2.1e)

**ECSS-E-ST-40C Clause 5.10.2.1e**

Whenever problems are encountered, they shall be recorded and entered in accordance with the change control established and maintained in conformance with ECSS-M-ST-40. {NOTE: ECSS-Q-ST-80 clause *5.2.6* (nonconformances) and clause *5.2.5* (software problems) contain further requirements relevant for this clause.}

**Expected Output**: Problem and nonconformance report [MF, - ; QR ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 5.2.5.1a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.5.2a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.5.3a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.5.4a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.6.1a (Nonconformances)*
- *ECSS-Q-ST-80C-R1 5.2.6.1b (Nonconformances)*
- *ECSS-Q-ST-80C-R1 5.2.6.1c (Nonconformances)*
- *ECSS-Q-ST-80C-R1 5.2.6.2a (Nonconformances)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.175 Long term maintenance for flight software (5.10.2.2a)

**ECSS-E-ST-40C Clause 5.10.2.2a**

If the spacecraft lifetime goes after the expected obsolescence date of the software engineering environment, then the maintainer shall propose solutions to be able to produce and upload modifications to the spacecraft up to its end of life.

**Expected Output**: Maintenance plan - long term maintenance solutions [MF, - ; QR, AR, ORR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.176 Problem analysis (5.10.3.1a)

**ECSS-E-ST-40C Clause 5.10.3.1a**

The maintainer shall analyse the problem report or modification requests for its impact on the organization, the existing system, and the interfacing systems for the following: 1. type (e.g. corrective, improvement, preventive, or adaptive to new environment); 2. scope (e.g. size of modification, cost involved, and time to modify); 3. criticality (e.g. impact on performance, safety, or security).

**Expected Output**: Modification analysis report and problem analysis report [MF, - ; -]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.177 Problem analysis (5.10.3.1b)

**ECSS-E-ST-40C Clause 5.10.3.1b**

The maintainer shall reproduce or verify the problem.

**Expected Output**: Modification analysis report and problem analysis report [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.178 Problem analysis (5.10.3.1c)

**ECSS-E-ST-40C Clause 5.10.3.1c**

Based upon the analysis, the maintainer shall develop options for implementing the modification.

**Expected Output**: Modification analysis report and problem analysis report [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.179 Problem analysis (5.10.3.1d)

**ECSS-E-ST-40C Clause 5.10.3.1d**

The maintainer shall document the problem or the modification request, the analysis results, the priorities (in terms of operation needs, risk, effort) and implementation options in the problem analysis report or in the modification analysis report, respectively.

**Expected Output**: Modification analysis report and problem analysis report [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.180 Problem analysis (5.10.3.1e)

**ECSS-E-ST-40C Clause 5.10.3.1e**

The maintainer shall obtain approval for the selected modification option in accordance with procedures agreed with the customer.

**Expected Output**: Modification approval [MF; -]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.181 Analysis and documentation of product modification (5.10.4.1a)

**ECSS-E-ST-40C Clause 5.10.4.1a**

The maintainer shall conduct and document an analysis to determine which documentation, models, software units, and their versions shall be modified.

**Expected Output**: Modification documentation [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.182 Documentation of software product changes (5.10.4.2a)

**ECSS-E-ST-40C Clause 5.10.4.2a**

All changes to the software product shall be documented in accordance with the procedures for document control and configuration management.

**Expected Output**: Modification documentation [MF, - ;-]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.183 Invoking of software engineering processes for modification implementation (5.10.4.3a)

**ECSS-E-ST-40C Clause 5.10.4.3a**

The maintainer shall apply the software engineering processes as specified in clauses *5.3* to *5.8* while implementing the modifications:

**Expected Output**: Modification documentation [MF, - ;-]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.3.2.1a (Software life cycle identification)*
- *ECSS-E-ST-40C 5.3.2.1b (Software life cycle identification)*
- *ECSS-E-ST-40C 5.3.2.1c (Software life cycle identification)*
- *ECSS-E-ST-40C 5.3.2.1d (Software life cycle identification)*
- *ECSS-E-ST-40C 5.3.2.2a (Identification of interfaces between development and maintenance)*
- *ECSS-E-ST-40C 5.3.2.3a (Software procurement process implementation)*
- *ECSS-E-ST-40C 5.3.2.4a (Automatic code generation)*
- *ECSS-E-ST-40C 5.3.2.4b (Automatic code generation)*
- *ECSS-E-ST-40C 5.3.2.4c (Automatic code generation)*
- *ECSS-E-ST-40C 5.3.2.4d (Automatic code generation)*
- *ECSS-E-ST-40C 5.3.2.4e (Automatic code generation)*
- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*
- *ECSS-E-ST-40C 5.3.3.1a (Joint reviews)*
- *ECSS-E-ST-40C 5.3.3.2a (Software project reviews)*
- *ECSS-E-ST-40C 5.3.3.2b (Software project reviews)*
- *ECSS-E-ST-40C 5.3.3.3a (Software technical reviews)*
- *ECSS-E-ST-40C 5.3.3.3b (Software technical reviews)*
- *ECSS-E-ST-40C 5.3.3.3c (Software technical reviews)*
- *ECSS-E-ST-40C 5.3.4.1a (System requirement review)*
- *ECSS-E-ST-40C 5.3.4.2a (Preliminary design review)*
- *ECSS-E-ST-40C 5.3.4.2b (Preliminary design review)*
- *ECSS-E-ST-40C 5.3.4.3a (Critical design review)*
- *ECSS-E-ST-40C 5.3.4.3b (Critical design review)*

- *ECSS-E-ST-40C 5.3.4.4a (Qualification review)*
- *ECSS-E-ST-40C 5.3.4.5a (Acceptance review)*
- *ECSS-E-ST-40C 5.3.5.1a (Test readiness reviews)*
- *ECSS-E-ST-40C 5.3.5.2a (Test review board)*
- *ECSS-E-ST-40C 5.3.6.1a (Review phasing for flight software)*
- *ECSS-E-ST-40C 5.3.6.1b (Review phasing for flight software)*
- *ECSS-E-ST-40C 5.3.6.2a (Review phasing for ground software)*
- *ECSS-E-ST-40C 5.3.7.1a (Interface management procedures)*
- *ECSS-E-ST-40C 5.3.8.1a (Software technical budget and margin philosophy definition)*
- *ECSS-E-ST-40C 5.3.8.2a (Technical budget and margin computation)*
- *ECSS-E-ST-40C 5.3.9.1a (Compliance matrix)*
- *ECSS-E-ST-40C 5.3.9.2a (Documentation compliance)*
- *ECSS-E-ST-40C 5.8.2.1a (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.1b (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.1d (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.2a (Selection of the organization responsible for conducting the verification)*
- *ECSS-E-ST-40C 5.8.2.2b (Selection of the organization responsible for conducting the verification)*
- *ECSS-E-ST-40C 5.8.3.1a (Verification of requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.2a (Verification of the technical specification)*
- *ECSS-E-ST-40C 5.8.3.3a (Verification of the software architectural design)*
- *ECSS-E-ST-40C 5.8.3.4a (Verification of the software detailed design)*
- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5b (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5c (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5d (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5e (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5f (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.6a (Verification of software unit testing (plan and results))*
- *ECSS-E-ST-40C 5.8.3.7a (Verification of software integration)*
- *ECSS-E-ST-40C 5.8.3.8a (Verification of software validation with respect to the technical specifications and the requirements baseline)*

- *ECSS-E-ST-40C 5.8.3.8b (Verification of software validation with respect to the technical specifications and the requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.9a (Evaluation of validation: complementary system level validation)*
- *ECSS-E-ST-40C 5.8.3.10a (Verification of software documentation)*
- *ECSS-E-ST-40C 5.8.3.11a (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.11b (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.11c (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.12a (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.12b (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.12c (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.13a (Behaviour modelling verification)*
- *ECSS-E-ST-40C 5.8.3.13b (Behaviour modelling verification)*
- *ECSS-E-ST-40C 5.8.3.13c (Behaviour modelling verification)*

This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.184 Invoking of software engineering processes for modification implementation (5.10.4.3b)

**ECSS-E-ST-40C Clause 5.10.4.3b**

Test and evaluation criteria for testing and evaluating the modified and the unmodified parts (models, software units, components, and configuration items) of the system shall be defined and documented.

**Expected Output**: Modification documentation [MF, - ;-]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.185 Invoking of software engineering processes for modification implementation (5.10.4.3c)

**ECSS-E-ST-40C Clause 5.10.4.3c**

The complete and correct implementation of the new and modified requirements shall be ensured.

**Expected Output**: Modification documentation [MF, - ;-]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.186 Invoking of software engineering processes for modification implementation (5.10.4.3d)

**ECSS-E-ST-40C Clause 5.10.4.3d**

It also shall be ensured that the original, unmodified requirements have not been affected.

**Expected Output**: Modification documentation [MF, - ;-]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.187 Invoking of software engineering processes for modification implementation (5.10.4.3e)

**ECSS-E-ST-40C Clause 5.10.4.3e**

The test results shall be documented.

**Expected Output**: Modification documentation [MF, - ;-]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.188 Maintenance reviews (5.10.5.1a)

**ECSS-E-ST-40C Clause 5.10.5.1a**

The maintainer shall conduct joint reviews with the organization authorizing the modification to determine the integrity of the modified system.

**Expected Output**: Joint review reports [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.189 Baseline for change (5.10.5.2a)

**ECSS-E-ST-40C Clause 5.10.5.2a**

Upon successful completion of the reviews, a baseline for the change shall be established.

**Expected Output**: Baseline for changes [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.190 Applicability of this Standard to software migration (5.10.6.1a)

**ECSS-E-ST-40C Clause 5.10.6.1a**

If a system or software product (including data) is migrated from an old to a new operational environment, it shall be ensured that any software product or data produced or modified during migration conform to this Standard.

**Expected Output**: Migration plan [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.191 Migration planning and execution (5.10.6.2a)

**ECSS-E-ST-40C Clause 5.10.6.2a**

A migration plan shall be developed, documented, and executed, including the following items: 1. requirements analysis and definition of migration; 2. development of migration tools; 3. conversion of software product and data; 4. migration execution; 5. migration verification; 6. support for the old environment in the future; 7. operator involvement in the activities.

**Expected Output**: Migration plan [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.192 Contribution to the migration plan (5.10.6.3a)

**ECSS-E-ST-40C Clause 5.10.6.3a**

The maintainer shall contribute to the migration plan and justification including the following items: 1. statement of why the old environment is no longer to be supported; 2. description of the new environment with its date of availability; 3. description of other support options

available, once support for the old environment has been removed; 4. the date as of which the transition takes place.

**Expected Output**: Migration plan [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.193 Preparation for migration (5.10.6.4a)

**ECSS-E-ST-40C Clause 5.10.6.4a**

If parallel operations of the old and new environments are conducted for transition to the new environment, training shall be provided and specified in the operational plan.

**Expected Output**: Migration plan [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.194 Notification of transition to migrated system (5.10.6.5a)

**ECSS-E-ST-40C Clause 5.10.6.5a**

When the scheduled migration takes place, notification shall be sent to all parties involved.

**Expected Output**: Migration notification [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.195 Notification of transition to migrated system (5.10.6.5b)

**ECSS-E-ST-40C Clause 5.10.6.5b**

All associated old environment's documentation, logs, and code shall be placed in archives.

**Expected Output**: Migration notification [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.196 Post-operation review (5.10.6.6a)

**ECSS-E-ST-40C Clause 5.10.6.6a**

A post-operation review shall be performed to assess the impact of changing to the new environment.

**Expected Output**: Post operation review report [OP, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.197 Post-operation review (5.10.6.6b)

**ECSS-E-ST-40C Clause 5.10.6.6b**

The results of the review shall be sent to the appropriate authorities for information, guidance, and action.

**Expected Output**: Post operation review report [OP, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.198 Maintenance and accessibility of data of former system (5.10.6.7a)

**ECSS-E-ST-40C Clause 5.10.6.7a**

Data used by or associated with the old environment shall be accessible in accordance with the requirements for data protection and audit applicable to the data.

**Expected Output**: Migration plan [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.199 Retirement planning (5.10.7.1a)

**ECSS-E-ST-40C Clause 5.10.7.1a**

Upon customer's request to retire a software product, a retirement plan to remove active support by the operator and maintainer shall be developed, documented and executed, ensuring: 1. cessation of full or partial support after the period of time specified by the customer; 2. archiving of the software product and its associated documentation; 3. responsibility for any future residual support issues; 4. transition to the new software product; 5. accessibility of archive copies of data.

**Expected Output**: Retirement plan [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.200 Notification of retirement plan (5.10.7.2a)

**ECSS-E-ST-40C Clause 5.10.7.2a**

The maintainer shall notify the retirement plan and related activities, including the following items: 1. description of the replacement or upgrade with its date of availability; 2. statement of why the software product is no longer to be supported; 3. description of other support options available, once support is removed.

**Expected Output**: Retirement notification [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.201 Identification of requirements for software retirement (5.10.7.3a)

**ECSS-E-ST-40C Clause 5.10.7.3a**

If parallel operations of the retiring and the new software product are conducted for transition to the new system, user training shall be provided as specified in the business agreement.

**Expected Output**: Retirement plan [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

### 9.1.202 Maintenance and accessibility to data of the retired product (5.10.7.4a)

**ECSS-E-ST-40C Clause 5.10.7.4a**

Data used by or associated with the retired software product shall be accessible in accordance with the business agreement requirements for data protection and audit applicable to the data.

**Expected Output**: Retirement plan [MF, - ; - ]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.1a (Operational testing definition)*
- *ECSS-E-ST-40C 5.9.5.3b (Provisions of work-around solutions)*

## 9.2 Tailoring of ECSS-Q-ST-80C Rev.1

Table 2: Compliance Matrix of ECSS-Q-ST-80C Rev.1 to QDP

| Clause | QDP Status | Clause | QDP Status | Clause | QDP Status |
|---|---|---|---|---|---|
| 5.1.1a | *Y* | 5.1.2.1a | *Y* | 5.1.2.2a | *Y* |
| 5.1.2.3a | *Y* | 5.1.3.1a | *Y* | 5.1.3.2a | *Y* |
| 5.1.4.1a | *Y* | 5.1.4.2a | *Y* | 5.1.5.1a | *Y* |
| 5.1.5.2a | *Y* | 5.1.5.3a | *Y* | 5.1.5.4a | *Y* |
| 5.2.1.1a | *Y* | 5.2.1.1b | *Y* | 5.2.1.2a | *Y* |
| 5.2.1.3a | *Y* | 5.2.1.4a | *N* | 5.2.1.5a | *Y* |
| 5.2.1.5b | *Y* | 5.2.2.1a | *Y* | 5.2.2.2a | *Y* |
| 5.2.2.3a | *Y* | 5.2.3a | *Y* | 5.2.4a | *Y* |
| 5.2.5.1a | *Y* | 5.2.5.2a | *Y* | 5.2.5.3a | *Y* |
| 5.2.5.4a | *Y* | 5.2.6.1a | *Y* | 5.2.6.1b | *Y* |
| 5.2.6.1c | *Y* | 5.2.6.2a | *Y* | 5.2.7.1a | *Y* |
| 5.2.7.2a | *Ye* | 5.3.1a | *Y* | 5.3.2.1a | *N/A* |
| 5.3.2.2a | *N/A* | 5.4.1.1a | *N/A* | 5.4.1.2a | *N/A* |
| 5.4.2.1a | *Y* | 5.4.2.2a | *Y* | 5.4.3.1a | *Y* |
| 5.4.3.2a | *Ye* | 5.4.3.3a | *Y* | 5.4.3.4a | *N/A* |
| 5.4.4a | *N* | 5.5.1a | *N/A* | 5.5.2a | *N/A* |
| 5.5.3a | *N/A* | 5.5.4a | *N/A* | 5.5.5a | *N/A* |
| 5.5.6a | *Y* | 5.6.1.1a | *Y* | 5.6.1.2a | *Y* |
| 5.6.1.3a | *Y* | 5.6.2.1a | *Y* | 5.6.2.2a | *Y* |
| 5.6.2.3a | *Y* | 5.7.1a | *Y* | 5.7.2.1a | *N* |
| 5.7.2.2a | *N* | 5.7.2.3a | *N* | 5.7.2.4a | *N* |
| 5.7.3.1a | *N* | 5.7.3.1b | *N* | 5.7.3.2a | *N* |
| 5.7.3.3a | *N* | 6.1.1a | *Y* | 6.1.1b | *Ye* |
| 6.1.2a | *Y* | 6.1.3a | *Y* | 6.1.4a | *Y* |
| 6.1.5a | *N* | 6.2.1.1a | *Ye* | 6.2.1.2a | *Y* |
| 6.2.1.3a | *Y* | 6.2.1.4a | *Y* | 6.2.1.5a | *Y* |
| 6.2.1.6a | *Y* | 6.2.1.7a | *Y* | 6.2.1.8a | *Y* |
| 6.2.1.9a | *Y* | 6.2.2.1a | *N* | 6.2.2.2a | *N* |
| 6.2.2.3a | *N* | 6.2.2.3b | *N* | 6.2.2.4a | *N* |
| 6.2.2.5a | *N* | 6.2.2.6a | *N* | 6.2.2.7a | *N* |
| 6.2.2.8a | *N* | 6.2.2.9a | *N* | 6.2.2.10a | *Y* |
| 6.2.3.1a | *N/A* | 6.2.3.1b | *N/A* | 6.2.3.2a | *N* |
| 6.2.3.3a | *N* | 6.2.3.4a | *Y* | 6.2.3.5a | *Y* |

continues on next page

Table 2 – continued from previous page

| Clause | QDP Status | Clause | QDP Status | Clause | QDP Status |
|---|---|---|---|---|---|
| 6.2.3.6a | *Y* | 6.2.3.7a | *Y* | 6.2.3.8a | *Y* |
| 6.2.4.1a | *Y* | 6.2.4.2a | *Y* | 6.2.4.3a | *Y* |
| 6.2.4.4a | *Y* | 6.2.4.5a | *Y* | 6.2.4.5b | *Y* |
| 6.2.4.6a | *Y* | 6.2.4.7a | *Ye* | 6.2.4.8a | *Y* |
| 6.2.4.9a | *Y* | 6.2.4.10a | *Y* | 6.2.4.11a | *Y* |
| 6.2.5.1a | *Y* | 6.2.5.2a | *Y* | 6.2.5.3a | *Y* |
| 6.2.5.4a | *Y* | 6.2.5.5a | *Y* | 6.2.6.1a | *Ye* |
| 6.2.6.2a | *Y* | 6.2.6.2b | *Y* | 6.2.6.3a | *Y* |
| 6.2.6.4a | *Y* | 6.2.6.5a | *Y* | 6.2.6.6a | *Y* |
| 6.2.6.7a | *Y* | 6.2.6.8a | *Ye* | 6.2.6.9a | *Y* |
| 6.2.6.10a | *Y* | 6.2.6.11a | *Y* | 6.2.6.12a | *Y* |
| 6.2.6.13a | *N* | 6.2.6.13b | *N* | 6.2.7.2a | *N/A* |
| 6.2.7.3a | *Y* | 6.2.7.4a | *Y* | 6.2.7.5a | *Y* |
| 6.2.7.6a | *Y* | 6.2.7.7a | *Y* | 6.2.7.8a | *Y* |
| 6.2.7.8b | *Y* | 6.2.7.9a | *Y* | 6.2.7.10a | *Y* |
| 6.2.7.11a | *Y* | 6.2.8.1a | *Y* | 6.2.8.2a | *Y* |
| 6.2.8.3a | *Y* | 6.2.8.4a | *Y* | 6.2.8.5a | *Y* |
| 6.2.8.6a | *Y* | 6.2.8.7a | *Y* | 6.3.1.1a | *US* |
| 6.3.1.2a | *US* | 6.3.1.3a | *US* | 6.3.2.1a | *US* |
| 6.3.2.2a | *Y* | 6.3.2.3a | *N* | 6.3.2.4a | *Ye* |
| 6.3.2.5a | *Y* | 6.3.3.1a | *Y* | 6.3.3.2a | *Y* |
| 6.3.3.3a | *N/A* | 6.3.3.4a | *Y* | 6.3.3.5a | *Y* |
| 6.3.3.5b | *Y* | 6.3.3.6a | *Y* | 6.3.3.7a | *US* |
| 6.3.4.1a | *Y* | 6.3.4.2a | *Y* | 6.3.4.3a | *Ye* |
| 6.3.4.4a | *Y* | 6.3.4.5a | *Y* | 6.3.4.6a | *Y* |
| 6.3.4.6b | *Ye* | 6.3.4.7a | *Y* | 6.3.4.8a | *Y* |
| 6.3.5.1a | *Y* | 6.3.5.2a | *Y* | 6.3.5.3a | *Y* |
| 6.3.5.4a | *N/A* | 6.3.5.5a | *Y* | 6.3.5.5b | *Y* |
| 6.3.5.6a | *Y* | 6.3.5.7a | *Y* | 6.3.5.8a | *Y* |
| 6.3.5.9a | *Y* | 6.3.5.10a | *Y* | 6.3.5.11a | *Y* |
| 6.3.5.12a | *Y* | 6.3.5.13a | *N* | 6.3.5.14a | *Y* |
| 6.3.5.15a | *Y* | 6.3.5.16a | *Y* | 6.3.5.17a | *Y* |
| 6.3.5.18a | *Y* | 6.3.5.19a | *Ye* | 6.3.5.20a | *US* |
| 6.3.5.21a | *Y* | 6.3.5.22a | *Ye* | 6.3.5.23a | *Ye* |
| 6.3.5.24a | *N/A* | 6.3.5.25a | *Ye* | 6.3.5.26a | *US* |
| 6.3.5.27a | *US* | 6.3.5.28a | *N* | 6.3.5.29a | *Ye* |
| 6.3.5.30a | *Y* | 6.3.5.31a | *Y* | 6.3.5.32a | *Y* |
| 6.3.6.1a | *Y* | 6.3.6.2a | *US* | 6.3.6.3a | *US* |
| 6.3.6.4a | *US* | 6.3.6.5a | *Y* | 6.3.6.6a | *US* |
| 6.3.6.7a | *US* | 6.3.6.8a | *US* | 6.3.6.9a | *US* |
| 6.3.7.1a | *US* | 6.3.7.2a | *US* | 6.3.7.3a | *US* |
| 6.3.8.1a | *US* | 6.3.8.2a | *US* | 6.3.8.3a | *US* |
| 6.3.8.4a | *US* | 6.3.8.5a | *US* | 6.3.8.6a | *US* |
| 6.3.8.7a | *US* | 7.1.1a | *US* | 7.1.2a | *Ye* |
| 7.1.3a | *Y* | 7.1.4a | *Y* | 7.1.5a | *Ye* |

continues on next page

Table 2 – continued from previous page

| Clause | QDP Status | Clause | QDP Status | Clause | QDP Status |
|---|---|---|---|---|---|
| 7.1.6a | *Y* | 7.1.7a | *N/A* | 7.1.8a | *Y* |
| 7.2.1.1a | *Ye* | 7.2.1.2a | *Y* | 7.2.1.3a | *Ye* |
| 7.2.2.1a | *Y* | 7.2.2.2a | *Y* | 7.2.2.3a | *Y* |
| 7.2.3.1a | *Y* | 7.2.3.2a | *Y* | 7.2.3.3a | *Y* |
| 7.2.3.4a | *Y* | 7.2.3.5a | *Y* | 7.2.3.6a | *Y* |
| 7.3.1a | *Y* | 7.3.2a | *Y* | 7.3.3a | *Ye* |
| 7.3.4a | *Y* | 7.3.5a | *Y* | 7.3.6a | *Y* |
| 7.3.7a | *Y* | 7.4.1a | *N/A* | 7.4.2a | *US* |
| 7.4.3a | *US* | 7.4.4a | *N/A* | 7.4.5a | *US* |
| 7.5.1a | *N/A* | 7.5.2a | *N/A* | 7.5.3a | *N/A* |

### 9.2.1 Organization (5.1.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.1a**

The supplier shall ensure that an organizational structure is defined for software development, and that individuals have defined tasks and responsibilities.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.2 Responsibility and authority (5.1.2.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.2.1a**

The responsibility, the authority and the interrelation of personnel who manage, perform and verify work affecting software quality shall be defined and documented.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.3 Responsibility and authority (5.1.2.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.2.2a**

The responsibilities and the interfaces of each organisation, either external or internal, involved in a project shall be defined and documented.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.4 Responsibility and authority (5.1.2.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.2.3a**

The delegation of software product assurance tasks by a supplier to a lower level supplier shall be done in a documented and controlled way, with the supplier retaining the responsibility towards the customer.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.5 Resources (5.1.3.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.3.1a**

The supplier shall provide adequate resources to perform the required software product assurance tasks.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.6 Resources (5.1.3.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.3.2a**

Reviews and audits of processes and of products shall be carried out by personnel not directly involved in the work being performed.

**QDP Status (Y):** Patches for RTEMS are reviewed by members of the RTEMS mailing list. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.7 Software product assurance manager/engineer (5.1.4.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.4.1a**

The supplier shall identify the personnel responsible for software product assurance for the project (SW PA manager/engineer).

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.8 Software product assurance manager/engineer (5.1.4.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.4.2a**

The software product assurance manager/engineer shall 1. report to the project manager (through the project product assurance manager, if any); 2. have organisational authority and independence to propose and maintain a software product assurance programme in accordance with the project software product assurance requirements; 3. have unimpeded access to higher management as necessary to fulfil his/her duties.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.9 Training (5.1.5.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.5.1a**

The supplier shall review the project requirements to establish and make timely provision for acquiring or developing the resources and skills for the management and technical staff.

**Expected Output**: Training plan [MGT, -; SRR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.10 Training (5.1.5.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.5.2a**

The supplier shall maintain training records.

**Expected Output**: Records of training and experience [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.11 Training (5.1.5.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.5.3a**

The supplier shall ensure that the right composition and categories of appropriately trained personnel are available for the planned activities and tasks in a timely manner.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.12 Training (5.1.5.4a)

**ECSS-Q-ST-80C Rev.1 Clause 5.1.5.4a**

The supplier shall determine the training subjects based on the specific tools, techniques, methodologies and computer resources to be used in the development and management of the software product. {NOTE: Personnel can undergo training to acquire skills and knowledge relevant to the specific field with which the software is to deal.}

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.13 Software product assurance planning and control (5.2.1.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.1.1a**

The supplier shall develop a software product assurance plan in response to the software product assurance requirements in conformance with DRD in annex B.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.14 Software product assurance planning and control (5.2.1.1b)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.1.1b**

The software product assurance plan shall be either a standalone document or a section of the supplier overall product assurance plan.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.15 Software product assurance planning and control (5.2.1.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.1.2a**

Any internal manuals, standards or procedures referred to by the software product assurance plan shall become an integral part of the supplier's software product assurance programme.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.16 Software product assurance planning and control (5.2.1.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.1.3a**

The software product assurance plan shall be revisited and updated as needed at each milestone to ensure that the activities to be undertaken in the following phase are fully defined.

**Expected Output**: Software product assurance plan [PAF, SPAP; CDR, QR, AR, ORR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.17 Software product assurance planning and control (5.2.1.4a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.1.4a**

Before acceptance review, the supplier shall either supplement the software product assurance plan to specify the quality measures related to the operations and maintenance processes, or issue a specific software product assurance plan.

**Expected Output**: Software product assurance plan [PAF, SPAP; AR]

**QDP Status (N):** See *No Maintenance (MF)* and *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.18 Software product assurance planning and control (5.2.1.5a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.1.5a**

The supplier shall provide with the software product assurance plan a compliance matrix documenting conformance with the individual software product assurance requirements applicable for the project or business agreement.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.19 Software product assurance planning and control (5.2.1.5b)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.1.5b**

For each software product assurance requirement, the compliance matrix shall provide a reference to the document where the expected output of that requirement is located. {NOTE: For compliance with the required DRDs a general statement of compliance is acceptable.}

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.20 Software product assurance reporting (5.2.2.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.2.1a**

The supplier shall report on a regular basis on the status of the software product assurance programme implementation, if appropriate as part of the overall product assurance reporting of the project.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.21 Software product assurance reporting (5.2.2.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.2.2a**

The software product assurance report shall include: 1. an assessment of the current quality of the product and processes, based on measured properties, with reference to the metrication as defined in the software product assurance plan; 2. verifications undertaken; 3. problems detected; 4. problems resolved.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.22 Software product assurance reporting (5.2.2.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.2.3a**

The supplier shall deliver at each milestone review a software product assurance milestone report, covering the software product assurance activities performed during the past project phases.

**Expected Output**: Software product assurance milestone report [PAF, SPAMR; SRR, PDR, CDR, QR, AR, ORR]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.23 Audits (5.2.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.3a**

For software audits, ECSS-Q-ST-10 clause 5.2.3 shall apply.

**Expected Output**: Audit plan and schedule [PAF, -; SRR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.24 Alerts (5.2.4a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.4a**

For software alerts, ECSS-Q-ST-10 clause 5.2.9 shall apply.

**Expected Output**: a. Preliminary alert information [PAF, -; -]; b. Alert information [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.25 Software problems (5.2.5.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.5.1a**

The supplier shall define and implement procedures for the logging, analysis and correction of all software problems encountered during software development.

**Expected Output**: Software problem reporting procedures [PAF, -; PDR]

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.3a (Problem handling procedures definition)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*
- *ECSS-Q-ST-80C-R1 6.2.5.4a (Process metrics)*

### 9.2.26 Software problems (5.2.5.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.5.2a**

The software problem report shall contain the following information: 1. identification of the software item; 2. description of the problem; 3. recommended solution; 4. final disposition; 5. modifications implemented (e.g. documents, code, and tools); 6. tests re-executed.

**Expected Output**: Software problem reporting procedures [PAF, -; PDR]

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.3a (Problem handling procedures definition)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*
- *ECSS-Q-ST-80C-R1 6.2.5.4a (Process metrics)*

### 9.2.27 Software problems (5.2.5.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.5.3a**

The procedures for software problems shall define the interface with the nonconformance system (i.e. the circumstances under which a problem qualifies as a nonconformance).

**Expected Output**: Software problem reporting procedures [PAF, -; PDR]

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.3a (Problem handling procedures definition)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*
- *ECSS-Q-ST-80C-R1 6.2.5.4a (Process metrics)*

### 9.2.28 Software problems (5.2.5.4a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.5.4a**

The supplier shall ensure the correct application of problem reporting procedures.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.3a (Problem handling procedures definition)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*
- *ECSS-Q-ST-80C-R1 6.2.5.4a (Process metrics)*

### 9.2.29 Nonconformances (5.2.6.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.6.1a**

For software nonconformance handling, ECSS-Q-ST-10-09 shall apply

**Expected Output**: a. NCR SW procedure as part of the Software product assurance plan [PAF, SPAP; SRR]; b. Nonconformance reports [DJF, -; -]

**QDP Status (Y):** See [EDI19b] and [EDI19d].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.3a (Problem handling procedures definition)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*

### 9.2.30 Nonconformances (5.2.6.1b)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.6.1b**

When dealing with software nonconformance, the NRB shall include, at least, a representative from the software product assurance and the software engineering organizations.

**Expected Output**: Identification of SW experts in NRB [MGT, -; SRR]

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.3a (Problem handling procedures definition)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*

### 9.2.31 Nonconformances (5.2.6.1c)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.6.1c**

The NRB shall dispose software nonconformances according to the following criteria: 1. use “as-is”, when the software is found to be usable without eliminating the nonconformance; 2. fix, when the software product can be made fully in conformance with all specified requirements, by: (a) correction of the software, (b) addition of software patches, or (c) re-design. 3. return to supplier, for procured software products (e.g. COTS).

**Expected Output**: Nonconformance reports [DJF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.3a (Problem handling procedures definition)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*

### 9.2.32 Nonconformances (5.2.6.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.6.2a**

The software product assurance plan shall specify the point in the software life cycle from which the nonconformance procedures apply.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.9.2.3a (Problem handling procedures definition)*
- *ECSS-E-ST-40C 5.10.2.1e (Establishment of the software maintenance process)*

### 9.2.33 Quality requirements and quality models (5.2.7.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.7.1a**

Quality models shall be used to specify the software quality requirements.

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.5.1a (Process metrics)*
- *ECSS-Q-ST-80C-R1 6.3.4.2a (Coding)*
- *ECSS-Q-ST-80C-R1 7.1.4a (Product metrics)*

### 9.2.34 Quality requirements and quality models (5.2.7.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.2.7.2a**

The following characteristics shall be used to specify the quality model: 1. functionality; 2. reliability; 3. maintainability; 4. reusability; 5. suitability for safety; 6. security; 7. usability; 8. efficiency; 9. portability; 10. software development effectiveness. {NOTE 1: Quality models are the basis for the identification of process metrics (see clause *6.2.5*) and product metrics (see clause *7.1.4*).} {NOTE 2: quality models are also addressed by ISO/IEC 9126 or ECSS-Q-HB-80-04.}

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR]

**QDP Status (Ye):** Quality models for security are excluded for the QDP. See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.5.1a (Process metrics)*
- *ECSS-Q-ST-80C-R1 6.3.4.2a (Coding)*
- *ECSS-Q-ST-80C-R1 7.1.4a (Product metrics)*

### 9.2.35 Risk management (5.3.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.3.1a**

Risk management for software shall be performed by cross-reference to the project risk policy, as specified in ECSS-M-ST-80.

**QDP Status (Y):** See [EDI19c] and [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.36 Critical item control (5.3.2.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.3.2.1a**

For critical item control, ECSS-Q-ST-10-04 shall apply.

**QDP Status (N/A):** No critical items are foreseen.

For an overview of all clauses, see the *tailoring table*.

### 9.2.37 Critical item control (5.3.2.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.3.2.2a**

The supplier shall identify the characteristics of the software items that qualify them for inclusion in the Critical Item List.

**QDP Status (N/A):** No critical items are foreseen.

For an overview of all clauses, see the *tailoring table*.

### 9.2.38 Supplier selection (5.4.1.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.4.1.1a**

For supplier selection ECSS-Q-ST-20 clause 5.4.1 shall apply.

**Expected Output**: a. Results of pre-award audits and assessments [PAF, -; -]; b. Records of procurement sources [PAF, -; -]

**QDP Status (N/A):** No supplier selection is foreseen.

For an overview of all clauses, see the *tailoring table*.

### 9.2.39 Supplier selection (5.4.1.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.4.1.2a**

For the selection of suppliers of existing software, including software contained in OTS equipments and units, the expected output of clauses *6.2.7.2* to *6.2.7.6* shall be made available.

**Expected Output**: Software reuse file [DJF, SRF; -]

**QDP Status (N/A):** No supplier selection is foreseen.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.7.2a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.3a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.4a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.5a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.6a (Reuse of existing software)*

### 9.2.40 Supplier requirements (5.4.2.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.4.2.1a**

The supplier shall establish software product assurance requirements for the next level suppliers, tailored to their role in the project, including a requirement to produce a software product assurance plan.

**Expected Output**: Software product assurance requirements for suppliers [PAF, -; SRR]

**QDP Status (Y):** The SPAP produced by EDISOFT will be used in the project. All the product assurance requirements are placed in the EDISOFT's SPAP. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.41 Supplier requirements (5.4.2.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.4.2.2a**

The supplier shall provide the software product assurance requirements applicable to the next level suppliers for customer's acceptance.

**Expected Output**: Software product assurance requirements for suppliers [PAF, -; SRR]

**QDP Status (Y):** The SPAP produced by EDISOFT will be used in the project. All the product assurance requirements are placed in the EDISOFT's SPAP. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.42 Supplier monitoring (5.4.3.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.4.3.1a**

The supplier shall monitor the next lower level suppliers' conformance to the product assurance requirements.

**QDP Status (Y):** The SPAP produced by EDISOFT will be used in the project. All the product assurance requirements are placed in the EDISOFT's SPAP. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.43 Supplier monitoring (5.4.3.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.4.3.2a**

The monitoring process shall include the review and approval of the next lower level suppliers' product assurance plans, the continuous verification of processes and products, and the monitoring of the final validation of the product.

**QDP Status (Ye):** The SPAP produced by EDISOFT will be used in the project. All the product assurance requirements are placed in the EDISOFT's SPAP. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.44 Supplier monitoring (5.4.3.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.4.3.3a**

The supplier shall ensure that software development processes are defined and applied by the next lower level suppliers in conformance with the software product assurance requirements for suppliers.

**Expected Output**: Next level suppliers' software product assurance plan [PAF, SPAP; PDR]

**QDP Status (Y):** The SPAP produced by EDISOFT will be used in the project. All the product assurance requirements are placed in the EDISOFT's SPAP. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.45 Supplier monitoring (5.4.3.4a)

**ECSS-Q-ST-80C Rev.1 Clause 5.4.3.4a**

The supplier shall provide the next lower level suppliers' software product assurance plan for customer's acceptance.

**Expected Output**: Next level suppliers' software product assurance plan [PAF, SPAP; PDR]

**QDP Status (N/A):** The SPAP produced by EDISOFT will be used in the project. All the product assurance requirements are placed in the EDISOFT's SPAP. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.46 Criticality classification (5.4.4a)

**ECSS-Q-ST-80C Rev.1 Clause 5.4.4a**

The supplier shall provide the lower level suppliers with the relevant results of the safety and dependability analyses performed at higher and his level (ref. clauses *6.2.2.1* and *6.2.2.2*), including: 1. the criticality classification of the software products to be developed; 2. information about the failures that can be caused at higher level by the software products to be developed.

**Expected Output**: Safety and dependability analyses results for lower level suppliers [RB, -; SRR]

**QDP Status (N):** Safety and Dependability analysis will not be performed. Criticality classified by contract.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.2.1a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.2a (Software dependability and safety)*
- *ECSS-Q-ST-80C-R1 6.2.2.10a (Software dependability and safety)*

This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*

### 9.2.47 Procurement documents (5.5.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.5.1a**

For procurement documents, ECSS-Q-ST-20 clause 5.4.2 shall apply.

**QDP Status (N/A):** No procurement is planned.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.3a (Software procurement process implementation)*

### 9.2.48 Review of procured software component list (5.5.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.5.2a**

The choice of procured software shall be described and submitted for customer review.

**Expected Output**: Software development plan [MGT, SDP; SRR, PDR]

**QDP Status (N/A):** No procurement is planned.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.3a (Software procurement process implementation)*

### 9.2.49 Procurement details (5.5.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.5.3a**

For each of the software items the following data shall be provided: 1. ordering criteria; 2. receiving inspection criteria; 3. back-up solutions if the product becomes unavailable; 4. contractual arrangements with the supplier for the development, maintenance and upgrades to new releases. {NOTE: Examples of ordering criteria are: versions, options and extensions.}

**Expected Output**: Procurement data [MGT, -; SRR, PDR]

**QDP Status (N/A):** No procurement is planned.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.3a (Software procurement process implementation)*

### 9.2.50 Identification (5.5.4a)

**ECSS-Q-ST-80C Rev.1 Clause 5.5.4a**

All the procured software shall be identified and registered by configuration management.

**QDP Status (N/A):** No procurement is planned.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.3a (Software procurement process implementation)*

### 9.2.51 Inspection (5.5.5a)

**ECSS-Q-ST-80C Rev.1 Clause 5.5.5a**

The supplier shall subject the procured software to a planned receiving inspection, in accordance with ECSS-Q-ST-20 clause 5.4.4, and the receiving inspection criteria as required by clause 5.5.3.

**Expected Output**: Receiving inspection report [PAF, -; PDR, CDR, QR]

**QDP Status (N/A):** No procurement is planned.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.3a (Software procurement process implementation)*

### 9.2.52 Exportability (5.5.6a)

**ECSS-Q-ST-80C Rev.1 Clause 5.5.6a**

Exportability constraints shall be identified.

**QDP Status (Y):** -

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.3a (Software procurement process implementation)*

### 9.2.53 Methods and tools (5.6.1.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.6.1.1a**

Methods and tools to be used for all the activities of the development cycle, (including requirements analysis, software specification, modelling, design, coding, validation, testing, configuration management, verification and product assurance) shall be identified by the supplier and agreed by the customer.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.54 Methods and tools (5.6.1.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.6.1.2a**

The choice of development methods and tools shall be justified by demonstrating through testing or documented assessment that: 1. the development team has appropriate experience or training to apply them, 2. the tools and methods are appropriate for the functional and operational characteristics of the product, and 3. the tools are available (in an appropriate hardware environment) throughout the development and maintenance lifetime of the product.

**Expected Output**: Software product assurance milestone report [PAF, SPAMR; SRR, PDR]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.55 Methods and tools (5.6.1.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.6.1.3a**

The correct use of methods and tools shall be verified and reported.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.56 Development environment selection (5.6.2.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.6.2.1a**

The software development environment shall be selected according to the following criteria: 1. availability; 2. compatibility; 3. performance; 4. maintenance; 5. durability and technical consistency with the operational equipment; 6. the assessment of the product with respect to requirements, including the criticality category; 7. the available support documentation; 8. the acceptance and warranty conditions; 9. the conditions of installation, preparation, training and use; 10. the maintenance conditions, including the possibilities of evolutions; 11. copyright and intellectual property rights constraints; 12. dependence on one specific supplier.

**Expected Output**: Software development plan [MGT, SDP; SRR, PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.57 Development environment selection (5.6.2.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.6.2.2a**

The suitability of the software development environment shall be justified.

**Expected Output**: Software development plan [MGT, SDP; SRR, PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.58 Development environment selection (5.6.2.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.6.2.3a**

The availability of the software development environment to developers and other users shall be verified before the start of each development phase.

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.59 Process assessment (5.7.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.7.1a**

The supplier shall monitor and control the effectiveness of the processes used during the development of the software, including the relevant processes corresponding to the services called from other organizational entities outside the project team. {NOTE: The process assessment and improvement performed at organization level can be used to provide evidence of compliance for the project.}

**Expected Output**: Software process assessment records: Overall assessments and improvement programme plan [PAF, -; -]

**QDP Status (Y):** See [EDI19e] and [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.60 Assessment process (5.7.2.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.7.2.1a**

The process assessment model and method to be used when performing any software process assessment shall be documented.

**Expected Output**: a. Software process assessment record: assessment model [PAF, -; -]; b. Software process assessment record: assessment method [PAF, -; -]

**QDP Status (N):** Process assessment is not foreseen to be performed in this project.

For an overview of all clauses, see the *tailoring table*.

### 9.2.61 Assessment process (5.7.2.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.7.2.2a**

Assessments performed and process assessment models used shall be in conformance with ISO/IEC 15504 (Part 2). {NOTE 1: The model and method documented in ECSS-Q- HB-80-02 are conformant to ISO/IEC 15504 (Part 2).} {NOTE 2: Currently the CMMI model is not fully conformant to ISO/IEC 15504, however it can be used, provided that the SCAMPI A method is applied.}

**Expected Output**: a. Software process assessment record: evidence of conformance of the process assessment model [PAF, -; -]; b. Software process assessment record: assessment method [PAF, -; -].

**QDP Status (N):** Process assessment is not foreseen to be performed in this project.

For an overview of all clauses, see the *tailoring table*.

### 9.2.62 Assessment process (5.7.2.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.7.2.3a**

The process assessment model, the method, the assessment scope, the results and the assessors shall be verified as complying with the project requirements. {NOTE 1: Examples of assessment scopes are: organizational unit evaluated, and processes evaluated.} {NOTE 2: ECSS-Q-HB-80-02 provides space specific process reference model and their indicators.}

**Expected Output**: Software process assessment record: Software process assessment recognition evidence [PAF, -; -]

**QDP Status (N):** Process assessment is not foreseen to be performed in this project.

For an overview of all clauses, see the *tailoring table*.

### 9.2.63 Assessment process (5.7.2.4a)

**ECSS-Q-ST-80C Rev.1 Clause 5.7.2.4a**

Assessments, carried out in accordance with ECSS-Q-HB-80-02, shall be performed by a competent assessor, whereas the other assessment team members can be either competent assessor or provisional assessor. {NOTE 1: For other assessment schemes conformant to ISO/IEC 15504 (Part 2), assessors certified under INTRSA are competent assessors.} {NOTE 2: When using CMMI/SCAMPI A, SEI authorized lead appraisers are competent assessors.}

**Expected Output**: Software process assessment record: competent assessor justification [PAF, -; -]

**QDP Status (N):** Process assessment is not foreseen to be performed in this project.

For an overview of all clauses, see the *tailoring table*.

### 9.2.64 Process improvement (5.7.3.1a)

**ECSS-Q-ST-80C Rev.1 Clause 5.7.3.1a**

The results of the assessment shall be used as feedback to improve as necessary the performed processes, to recommend changes in the direction of the project, and to determine technology advancement needs.

**QDP Status (N):** Process assessment is not foreseen to be performed in this project.

For an overview of all clauses, see the *tailoring table*.

### 9.2.65 Process improvement (5.7.3.1b)

**ECSS-Q-ST-80C Rev.1 Clause 5.7.3.1b**

The suppliers shall ensure that the results of previous assessments are used in its project activity

**Expected Output**: Software process assessment records: improvement plan [PAF, -; -]

**QDP Status (N):** Process assessment is not foreseen to be performed in this project.

For an overview of all clauses, see the *tailoring table*.

### 9.2.66 Process improvement (5.7.3.2a)

**ECSS-Q-ST-80C Rev.1 Clause 5.7.3.2a**

The process improvement shall be conducted according to a documented process improvement process. {NOTE 1: For the definition of the process improvement process, see ECSS-Q-HB-80-02.} {NOTE 2: For CMMI, the process improvement is described in the OPF (Organizational Process Focus) process area.}

**Expected Output**: Software process assessment records: improvement process [PAF, -; -]

**QDP Status (N):** Process assessment is not foreseen to be performed in this project.

For an overview of all clauses, see the *tailoring table*.

### 9.2.67 Process improvement (5.7.3.3a)

**ECSS-Q-ST-80C Rev.1 Clause 5.7.3.3a**

Evidence of the improvement in performed processes or in project documentation shall be provided. {NOTE: See ECSS-Q-HB-80-02.}

**Expected Output**: Software process assessment records: evidence of improvements [PAF, -; -]

**QDP Status (N):** Process assessment is not foreseen to be performed in this project.

For an overview of all clauses, see the *tailoring table*.

### 9.2.68 Life cycle definition (6.1.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.1.1a**

The software development life cycle shall be defined or referenced in the software product assurance plan.

**QDP Status (Y):** See [EDI19e] and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.69 Life cycle definition (6.1.1b)

**ECSS-Q-ST-80C Rev.1 Clause 6.1.1b**

The following characteristics of the software life cycle shall be defined: 1. phases; 2. input and output of each phase; 3. status of completion of phase output; 4. milestones; 5. dependencies; 6. responsibilities; 7. role of the customer at each milestone review, in conformance with ECSS-M-ST-10 and ECSS-M-ST-10-01.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.70 Process quality objectives (6.1.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.1.2a**

In the definition of the life cycle and associated milestones and documents, the quality objectives shall be used.

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.71 Life cycle definition review (6.1.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.1.3a**

The software life cycle shall be reviewed against the contractual software engineering and product assurance requirements.

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.72 Life cycle resources (6.1.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.1.4a**

The software life cycle shall be reviewed for suitability and for the availability of resources to implement it by all functions involved in its application.

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.73 Software validation process schedule (6.1.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.1.5a**

A milestone (SW TRR as defined in ECSS-E-ST-40 clause *5.3.5.1*) shall be scheduled immediately before the software validation process starts, to check that: 1. the software status is compatible with the commencement of validation activities; 2. the necessary resources, software product assurance plans, test and validation documentation, simulators or other technical means are available and ready for use.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (N):** TRR is not foreseen in this project.

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.3.5.1a (Test readiness reviews)*

### 9.2.74 Documentation of processes (6.2.1.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.1.1a**

The following activities shall be covered either in software-specific plans or in project general plans: 1. development; 2. specification, design and customer documents to be produced; 3. configuration and documentation management; 4. verification, testing and validation activities; 5. maintenance.

**Expected Output**: Software project plans [MGT, MF, DJF]

**QDP Status (Ye):** See *No Maintenance (MF)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.75 Documentation of processes (6.2.1.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.1.2a**

All plans shall be finalized before the start of the related activities.

**Expected Output**: Software project plans [MGT, MF, DJF]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.76 Documentation of processes (6.2.1.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.1.3a**

All plans shall be updated for each milestone to reflect any changes during development.

**Expected Output**: Software project plans [MGT, MF, DJF]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.77 Documentation of processes (6.2.1.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.1.4a**

The software product assurance plan shall identify all plans to be produced and used, the relationship between them and the time-scales for their preparation and update.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.78 Documentation of processes (6.2.1.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.1.5a**

Each plan shall be reviewed against the relevant contractual requirements.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.79 Documentation of processes (6.2.1.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.1.6a**

Procedures and project standards shall address all types of software products included in the project.

**Expected Output**: Procedures and standards [PAF, -; PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.80 Documentation of processes (6.2.1.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.1.7a**

All procedures and project standards shall be finalized before starting the related activities.

**Expected Output**: Procedures and standards [PAF, -; PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.81 Documentation of processes (6.2.1.8a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.1.8a**

Each procedure or standard shall be reviewed against the relevant plans and contractual requirements.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.82 Documentation of processes (6.2.1.9a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.1.9a**

Before any activity is started, each procedure or standard for that activity shall be reviewed by all functions involved in its application, for suitability and for the availability of resources to implement it.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.83 Software dependability and safety (6.2.2.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.1a**

For the system-level analyses leading to the criticality classification of software products based on the severity of failures consequences, ECSS-Q-ST-40 clause 6.5.6.3, and ECSS-Q-ST-30 clause 5.4, shall apply.

**Expected Output**: Criticality classification of software products [PAF, -; SRR, PDR]

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 5.4.4a (Criticality classification)*

### 9.2.84 Software dependability and safety (6.2.2.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.2a**

The supplier shall perform a software dependability and safety analysis of the software products, using the results of system-level safety and dependability analyses, in order to determine the criticality of the individual software components.

**Expected Output**: Software dependability and safety analysis report [PAF, -; PDR]

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 5.4.4a (Criticality classification)*

### 9.2.85 Software dependability and safety (6.2.2.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.3a**

The supplier shall identify the methods and techniques for the software dependability and safety analysis to be performed at technical specification and design level.

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*

### 9.2.86 Software dependability and safety (6.2.2.3b)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.3b**

Methods and techniques for software dependability and safety analysis shall be agreed between the supplier and customer. {NOTE: ECSS-Q-HB-80-03 provides indication on methods and techniques that can be applied such as: • software failure modes, effects and criticality analysis (for the performing of this analysis, see also ECSS-Q-ST-30-02); • software fault tree analysis; • software common cause failure analysis.}

**Expected Output**: Criticality classification of software components [PAF, -; PDR]

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*

### 9.2.87 Software dependability and safety (6.2.2.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.4a**

Based on the results of the software criticality analysis, the supplier shall apply engineering measures to reduce the number of critical software components and mitigate the risks associated with the critical software (ref. clause *6.2.3*).

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.3.1a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.1b (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.2a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.3a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.4a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.5a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.6a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.7a (Handling of criticality software)*
- *ECSS-Q-ST-80C-R1 6.2.3.8a (Handling of criticality software)*

This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*

### 9.2.88 Software dependability and safety (6.2.2.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.5a**

The supplier shall report on the status of the implementation and verification of the SW dependability and safety analysis recommendations.

**Expected Output**: Software dependability and safety analysis report [PAF, -; CDR, QR, AR]

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*

### 9.2.89 Software dependability and safety (6.2.2.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.6a**

The supplier shall update the software dependability and safety analysis at each software development milestone, to confirm the criticality category of software components.

**Expected Output**: Software dependability and safety analysis report [PAF, -; CDR, QR, AR]

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*

### 9.2.90 Software dependability and safety (6.2.2.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.7a**

The supplier shall provide the results of the software dependability and safety analysis for integration into the system-level dependability and safety analyses, addressing in particular: 1. additional failure modes identified at software design level; 2. recommendations for system-level activities. {NOTE: For example: introduction of hardware inhibits, and modifications of the system architecture.}

**Expected Output**: Software dependability and safety analysis report [PAF, -; PDR, CDR]

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*

### 9.2.91 Software dependability and safety (6.2.2.8a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.8a**

As part of the software requirements analysis activities (ref. clause *6.3.2*), the supplier shall contribute to the Hardware-Software Interaction Analysis (HSIA) by identifying, for each hardware failure included in the HSIA, the requirements that specify the software behaviour in the event of that hardware failure.

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.3.2.1a (Software requirements analysis)*
- *ECSS-Q-ST-80C-R1 6.3.2.2a (Software requirements analysis)*
- *ECSS-Q-ST-80C-R1 6.3.2.3a (Software requirements analysis)*
- *ECSS-Q-ST-80C-R1 6.3.2.4a (Software requirements analysis)*
- *ECSS-Q-ST-80C-R1 6.3.2.5a (Software requirements analysis)*

This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*

### 9.2.92 Software dependability and safety (6.2.2.9a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.9a**

During the verification and validation of the software requirements resulting from the Hardware-Software Interaction Analysis, the supplier shall verify that the software reacts correctly to hardware failures, and no undesired software behaviour occurs that lead to system failures.

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*

### 9.2.93 Software dependability and safety (6.2.2.10a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.2.10a**

If it cannot be prevented that software components cause failures of higher criticality components, due to failure propagation or use of shared resources, then all the involved components shall be classified at the highest criticality category among them. {NOTE: Failures of higher-criticality software components caused by lower-criticality components can be prevented by design measures such as separate hardware platforms, isolation of software processes or prohibition of shared memory (segregation and partitioning).}

**Expected Output**: The following outputs are expected: a. Software product assurance plan [PAF, SPAP; PDR, CDR]; b. Software dependability and safety analysis report [PAF, -; PDR, CDR, QR, AR]

**QDP Status (Y):** Criticality category classified by contract.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 5.4.4a (Criticality classification)*

### 9.2.94 Handling of criticality software (6.2.3.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.3.1a**

<<deleted>>

**QDP Status (N/A):** All components have the same criticality.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*

### 9.2.95 Handling of criticality software (6.2.3.1b)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.3.1b**

<<deleted>>

**QDP Status (N/A):** All components have the same criticality.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*

### 9.2.96 Handling of criticality software (6.2.3.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.3.2a**

The supplier shall define, justify and apply measures to assure the dependability and safety of critical software. {NOTE:These measures can include: • use of software design or methods that have performed successfully in a similar application; • insertion of features for failure isolation and handling (ref. ECSS-Q-HB-80-03, software failure modes and effects analysis); • defensive programming techniques, such as input verification and consistency checks; • use of a “safe subset” of programming language; • use of formal design language for formal

proof; • 100 % code branch coverage at unit testing level; • full inspection of source code; • witnessed or independent testing; • gathering and analysis of failure statistics; • removing deactivated code or showing through a combination of analysis and testing that the means by which such code can be inadvertently executed are prevented, isolated, or eliminated. }

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR, CDR]

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*

### 9.2.97 Handling of criticality software (6.2.3.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.3.3a**

The application of the chosen measures to handle the critical software shall be verified.

**Expected Output**: Software product assurance milestone report [PAF, SPAMR; PDR, CDR, QR, AR]

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*

### 9.2.98 Handling of criticality software (6.2.3.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.3.4a**

Critical software shall be subject to regression testing after: 1. any change of functionality of the underlying platform hardware; 2. any change of the tools that affect directly or indirectly the generation of the executable code. {NOTE 1: In case of minor changes in tools that affect the generation of the executable code, a binary comparison of the executable code generated by the different tools can be used to verify that no modifications are introduced. NOTE 2: Example for item 1: instruction set of a processor.}

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR, CDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*

### 9.2.99 Handling of criticality software (6.2.3.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.3.5a**

The need for additional verification and validation of critical software shall be analysed after: 1. any change of functionality or performance of the underlying platform hardware; 2. any change in the environment in which the software or the platform hardware operate.

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR, CDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*

### 9.2.100 Handling of criticality software (6.2.3.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.3.6a**

Identified unreachable code shall be removed and the need for re-verification and re-validation shall be analysed.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*

### 9.2.101 Handling of criticality software (6.2.3.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.3.7a**

Unit and integration testing shall be (re-)executed on non-instrumented code.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*

### 9.2.102 Handling of criticality software (6.2.3.8a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.3.8a**

Validation testing shall be (re-)executed on non-instrumented code.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-Q-ST-80C-R1 6.2.2.4a (Software dependability and safety)*

### 9.2.103 Software configuration management (6.2.4.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.1a**

ECSS-M-ST-40 shall be applied for software configuration management, complemented by the following requirements.

**QDP Status (Y):** The “Following requirements” are not specified. See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.104 Software configuration management (6.2.4.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.2a**

The software configuration management system shall allow any reference version to be re-generated from backups.

**Expected Output**: Software configuration management plan [MGT, SCMP; SRR, PDR]

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.105 Software configuration management (6.2.4.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.3a**

The software configuration file and the software release document shall be provided with each software delivery.

**Expected Output**: a. Software configuration file [DDF, SCF; -]; b. Software release document [DDF, SRelD; -]

**QDP Status (Y):** See *Software Configuration File (SCF)* and *Software Release Document (SRelD)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.106 Software configuration management (6.2.4.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.4a**

The software configuration file shall be available and up to date for each project milestone.

**Expected Output**: Software configuration file [DDF, SCF; CDR, QR, AR, ORR]

**QDP Status (Y):** See *Software Configuration File (SCF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.107 Software configuration management (6.2.4.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.5a**

Any components of the code generation tool that are customizable by the user shall be put under configuration control.

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.108 Software configuration management (6.2.4.5b)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.5b**

The change control procedures defined for the project shall address the specific aspects of these components.

**Expected Output**: a. Software configuration file [DDF, SCF; CDR, QR, AR, ORR]; b. Software configuration management plan [MGT, SCMP; SRR, PDR].

**QDP Status (Y):** See *Software Configuration File (SCF)* and [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.109 Software configuration management (6.2.4.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.6a**

The supplier shall ensure that all authorized changes are implemented in accordance with the software configuration management plan.

**Expected Output**: Authorized changes - Software configuration file [DDF, SCF; CDR, QR, AR, ORR]

**QDP Status (Y):** See *Software Configuration File (SCF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.110 Software configuration management (6.2.4.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.7a**

The following documents shall be controlled (see ECSS-Q-ST-10 clause 5.2.5): 1. procedural documents describing the quality system to be applied during the software life cycle; 2. planning documents describing the planning and progress of the activities; 3. documents describing a particular software product, including: (a) development phase inputs, (b) development phase outputs, (c) verification and validation plans and results, (d) test case specifications, test procedures and test reports, (e) traceability matrices, (f) documentation for the software and system operators and users, and (g) maintenance documentation.

**QDP Status (Ye):** See *No Maintenance (MF)* and [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.111 Software configuration management (6.2.4.8a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.8a**

The supplier shall identify a method and tool to protect the supplied software against corruption. {NOTE: For example: source, executable and data.}

**Expected Output**: a. Software product assurance plan [PAF, SPAP; SRR, PDR]; b. Software configuration file [DDF, SCF; CDR, QR, AR, ORR]

**QDP Status (Y):** See *Software Configuration File (SCF)* and See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.112 Software configuration management (6.2.4.9a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.9a**

The supplier shall define a checksum-type key calculation for the delivered operational software. {NOTE: For example: executable binary, database.}

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.113 Software configuration management (6.2.4.10a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.10a**

The checksum value shall be provided in the software configuration file with each software delivery.

**Expected Output**: Software configuration file [DDF, SCF; -]

**QDP Status (Y):** See *Software Configuration File (SCF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.114 Software configuration management (6.2.4.11a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.4.11a**

The media through which the software is delivered to the customer shall be marked by the supplier indicating the following information as a minimum: 1. the software name; 2. the version number; 3. the reference to the software configuration file.

**Expected Output**: a. Software product assurance plan [PAF, SPAP; SRR, PDR]; b. Labels [DDF, -; -]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.5a (Changes to baselines)*

### 9.2.115 Process metrics (6.2.5.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.5.1a**

Metrics shall be used to manage the development and to assess the quality of the development processes. {NOTE: Process metrics are based on quality models (see clause *5.2.7*).}

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 5.2.7.1a (Quality requirements and quality models)*
- *ECSS-Q-ST-80C-R1 5.2.7.2a (Quality requirements and quality models)*

### 9.2.116 Process metrics (6.2.5.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.5.2a**

Process metrics shall be collected, stored and analysed on a regular basis by applying quality models and procedures.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.117 Process metrics (6.2.5.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.5.3a**

The following basic process metrics shall be used within the supplier's organization: 1. duration: how phases and tasks are being completed versus the planned schedule; 2. effort: how much effort is consumed by the various phases and tasks compared to the plan.

**Expected Output**: Internal metrics report

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.118 Process metrics (6.2.5.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.5.4a**

Process metrics shall be used within the supplier's organization and reported to the customer, including: 1. number of problems detected during verification; 2. number of problems detected during integration and validation testing and use. {NOTE: See also software problem reporting described in clause *5.2.5*.}

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 5.2.5.1a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.5.2a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.5.3a (Software problems)*
- *ECSS-Q-ST-80C-R1 5.2.5.4a (Software problems)*

### 9.2.119 Process metrics (6.2.5.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.5.5a**

Metrics reports shall be included in the software product assurance reports.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.120 Verification (6.2.6.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.1a**

Activities for the verification of the quality requirements shall be specified in the definition of the verification plan. {NOTE: Verification includes various techniques such as review, inspection, testing, walk-through, cross-reading, desk-checking, model simulation, and many types of analysis such as traceability analysis, formal proof or fault tree analysis.}

**Expected Output**: Software verification plan [DJF, SVerP; PDR]

**QDP Status (Ye):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.121 Verification (6.2.6.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.2a**

The outputs of each development activity shall be verified for conformance against pre-defined criteria.

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.122 Verification (6.2.6.2b)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.2b**

Only outputs which have been subjected to planned verifications shall be used as inputs for subsequent activities.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.123 Verification (6.2.6.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.3a**

A summary of the assurance activities concerning the verification process and their findings shall be included in software product assurance reports.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.124 Verification (6.2.6.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.4a**

The completion of actions related to software problem reports generated during verification shall be verified and recorded.

**Expected Output**: Software problem reports [DJF, -; SRR, PDR, CDR, QR, AR, ORR]

**QDP Status (Y):** Use of RTEMS project ticket system and project internal ticket system.

For an overview of all clauses, see the *tailoring table*.

### 9.2.125 Verification (6.2.6.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.5a**

Software containing deactivated code shall be verified specifically to ensure that the deactivated code cannot be activated or that its accidental activation cannot harm the operation of the system.

**Expected Output**: Software verification report [DJF, SVR; CDR, QR, AR]

**QDP Status (Y):** See *Software Verification Report (SVR)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.126 Verification (6.2.6.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.6a**

Software containing configurable code shall be verified specifically to ensure that any unintended configuration cannot be activated at run time or included during code generation.

**Expected Output**: Software verification report [DJF, SVR; CDR, QR, AR]

**QDP Status (Y):** See *Software Verification Report (SVR)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.127 Verification (6.2.6.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.7a**

The supplier shall ensure that: 1. the planned verification activities are adequate to confirm that the products of each phase are conformant to the applicable requirements; 2. the verification activities are performed according to the plan.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.128 Verification (6.2.6.8a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.8a**

Reviews and inspections shall be carried out according to defined criteria, and according to the defined level of independence of the reviewer from the author of the reviewed item.

**QDP Status (Ye):** Large parts of the source code was produced in a different project by the same staff that does now the reviews. Due to the elapse of more than one year between writing of the code and the reviews carried out in this project, sufficient independence is ensured by the pass of time. In general, changes in the RTEMS project are reviewed by persons on the RTEMS developer mailing list. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.129 Verification (6.2.6.9a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.9a**

Each review and inspection shall be based on a written plan or procedure. {NOTE: For projects reviews, ECSS-E-ST-40 clause *5.3.3.3*, bullet b and Annex P are applicable.}

**Expected Output**: Review and inspection plans or procedures [PAF, -; -]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.3.3.3c (Software technical reviews)*

### 9.2.130 Verification (6.2.6.10a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.10a**

The review or inspection plans or procedures shall specify: 1. the reviewed or inspected items; 2. the person in charge; 3. the participants; 4. the means of review or inspection (e.g. tools or check list); 5. the nature of the report.

**Expected Output**: Review and inspection plans or procedures [PAF, -; -]

**QDP Status (Y):** See *Requirement Validation* and see :ref:[EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.131 Verification (6.2.6.11a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.11a**

Review and inspection reports shall: 1. refer to the corresponding review/inspection procedure or plan; 2. identify the reviewed item, the author, the reviewer, the review criteria and the findings of the review.

**Expected Output**: Review and inspection reports [PAF, -; -]

**QDP Status (Y):** See *Requirement Validation* and see :ref:[EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.132 Verification (6.2.6.12a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.12a**

Traceability matrices (as defined in ECSS-E-ST-40 clause *5.8*) shall be verified at each milestone.

**Expected Output**: Software product assurance milestone report [PAF, SPAMR; SRR, PDR, CDR, QR, AR, ORR]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.8.2.1a (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.1b (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.1d (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.2a (Selection of the organization responsible for conducting the verification)*
- *ECSS-E-ST-40C 5.8.2.2b (Selection of the organization responsible for conducting the verification)*
- *ECSS-E-ST-40C 5.8.3.1a (Verification of requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.2a (Verification of the technical specification)*
- *ECSS-E-ST-40C 5.8.3.3a (Verification of the software architectural design)*
- *ECSS-E-ST-40C 5.8.3.4a (Verification of the software detailed design)*
- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5b (Verification of code)*

- *ECSS-E-ST-40C 5.8.3.5c (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5d (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5e (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5f (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.6a (Verification of software unit testing (plan and results))*
- *ECSS-E-ST-40C 5.8.3.7a (Verification of software integration)*
- *ECSS-E-ST-40C 5.8.3.8a (Verification of software validation with respect to the technical specifications and the requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.8b (Verification of software validation with respect to the technical specifications and the requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.9a (Evaluation of validation: complementary system level validation)*
- *ECSS-E-ST-40C 5.8.3.10a (Verification of software documentation)*
- *ECSS-E-ST-40C 5.8.3.11a (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.11b (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.11c (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.12a (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.12b (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.12c (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.13a (Behaviour modelling verification)*
- *ECSS-E-ST-40C 5.8.3.13b (Behaviour modelling verification)*
- *ECSS-E-ST-40C 5.8.3.13c (Behaviour modelling verification)*

### 9.2.133 Verification (6.2.6.13a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.13a**

Independent software verification shall be performed by a third party.

**QDP Status (N):** See *No Independent Software Verification and Validation*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.8.2.2b (Selection of the organization responsible for conducting the verification)*

### 9.2.134 Verification (6.2.6.13b)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.6.13b**

Independent software verification shall be a combination of reviews, inspections, analyses, simulations, testing and auditing. {NOTE: This requirement is applicable where the risks associated with the project justify the costs involved. The customer can consider a less rigorous level of independence, e.g. an independent team in the same organization.}

**Expected Output**: a. ISVV plan [DJF, -; SRR, PDR]; b. ISVV report [DJF, -; PDR, CDR, QR, AR]

**QDP Status (N):** See *No Independent Software Verification and Validation*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.8.2.2b (Selection of the organization responsible for conducting the verification)*

### 9.2.135 Reuse of existing software (6.2.7.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.2a**

Analyses of the advantages to be obtained with the selection of existing software (ref. 3.2.11) instead of new development shall be carried out.

**Expected Output**: a. Software reuse approach, including approach to delta qualification [PAF, SPAP; SRR, PDR]; b. Software reuse file [DJF, SRF; SRR, PDR]

**QDP Status (N/A):** No selection will be performed.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 5.4.1.2a (Supplier selection)*
- *ECSS-Q-ST-80C-R1 6.2.7.7a (Reuse of existing software)*

### 9.2.136 Reuse of existing software (6.2.7.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.3a**

The existing software shall be assessed with regards to the applicable functional, performance and quality requirements.

**Expected Output**: a. Software reuse approach, including approach to delta qualification [PAF, SPAP; SRR, PDR]; b. Software reuse file [DJF, SRF; SRR, PDR]

**QDP Status (Y):** See *Software Reuse File (SRF)* and see [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 5.4.1.2a (Supplier selection)*
- *ECSS-Q-ST-80C-R1 6.2.7.7a (Reuse of existing software)*

### 9.2.137 Reuse of existing software (6.2.7.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.4a**

The quality level of the existing software shall be analysed with respect to the project requirements, according to the criticality of the system function implemented, taking into account the following aspects: 1. software requirements documentation; 2. software architectural and detailed design documentation; 3. forward and backward traceability between system requirements, software requirements, design and code; 4. unit tests documentation and coverage; 5. integration tests documentation and coverage; 6. validation documentation and coverage; 7. verification reports; 8. performance; 9. operational performances; 10. residual nonconformances and waivers; 11. user operational documentation; {NOTE 1: Examples of performance are memory occupation, CPU load. NOTE 2: Example of user operation documentation is a user manual.).

**Expected Output**: a. Software reuse approach, including approach to delta qualification [PAF, SPAP; SRR, PDR]; b. Software reuse file [DJF, SRF; SRR, PDR]

**QDP Status (Y):** See *Software Reuse File (SRF)* and see [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 5.4.1.2a (Supplier selection)*
- *ECSS-Q-ST-80C-R1 6.2.7.7a (Reuse of existing software)*

### 9.2.138 Reuse of existing software (6.2.7.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.5a**

The results of the reused software analysis shall be recorded in the software reuse file, together with an assessment of the possible level of reuse and a description of the assumptions and the methods applied when estimating the level of reuse. {NOTE: Results of the reused software analysis, such as detailed reference to requirement and design documents, test reports and coverage results.}

**Expected Output**: a. Software reuse approach, including approach to delta qualification [PAF, SPAP; SRR, PDR]; b. Software reuse file [DJF, SRF; SRR, PDR]

**QDP Status (Y):** See *Software Reuse File (SRF)* and see [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 5.4.1.2a (Supplier selection)*
- *ECSS-Q-ST-80C-R1 6.2.7.7a (Reuse of existing software)*

### 9.2.139 Reuse of existing software (6.2.7.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.6a**

The analysis of the suitability of existing software for reuse shall be complemented by an assessment of the following aspects: 1. the acceptance and warranty conditions; 2. the available support documentation; 3. the conditions of installation, preparation, training and use; 4. the identification and registration by configuration management; 5. maintenance responsibility and conditions, including the possibilities of changes; 6. the durability and validity of methods and tools used in the initial development, that are envisaged to be used again; 7. the copyright and intellectual property rights constraints (modification rights); 8. the licensing conditions; 9. exportability constraints.

**Expected Output**: Software reuse file [DJF, SRF; SRR, PDR]

**QDP Status (Y):** See *Software Reuse File (SRF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 5.4.1.2a (Supplier selection)*
- *ECSS-Q-ST-80C-R1 6.2.7.7a (Reuse of existing software)*

### 9.2.140 Reuse of existing software (6.2.7.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.7a**

Corrective actions shall be identified, documented in the reuse file and applied to the reused software not meeting the applicable requirements related to the aspects as specified in clauses *6.2.7.2* to *6.2.7.6*.

**Expected Output**: Software reuse file [DJF, SRF; SRR, PDR]

**QDP Status (Y):** See *Software Reuse File (SRF)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 6.2.7.2a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.3a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.4a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.5a (Reuse of existing software)*
- *ECSS-Q-ST-80C-R1 6.2.7.6a (Reuse of existing software)*

This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*

### 9.2.141 Reuse of existing software (6.2.7.8a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.8a**

Reverse engineering techniques shall be applied to generate missing documentation and to reach the required verification and validation coverage.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*

### 9.2.142 Reuse of existing software (6.2.7.8b)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.8b**

For software products whose life cycle data from previous development are not available and reverse engineering techniques are not fully applicable, the following methods shall be applied: 1. generation of validation and verification documents based on the available user documentation (e.g. user manual) and execution of tests in order to achieve the required level of test coverage; 2. use of the product service history to provide evidence of the product's suitability for the current application, including information about: (a) relevance of the product service history for the new operational environment; (b) configuration management and change control of the software product; (c) effectiveness of problem reporting; (d) actual error rates and maintenance records; (e) impact of modifications.

**Expected Output**: Software reuse file [DJF, SRF; SRR, PDR]

**QDP Status (Y):** See *Software Reuse File (SRF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*

### 9.2.143 Reuse of existing software (6.2.7.9a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.9a**

The software reuse file shall be updated at project milestones to reflect the results of the identified corrective actions for reused software not meeting the project requirements.

**Expected Output**: Software reuse file [DJF, SRF; CDR, QR, AR]

**QDP Status (Y):** See *Software Reuse File (SRF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*

### 9.2.144 Reuse of existing software (6.2.7.10a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.10a**

All the reused software shall be kept under configuration control.

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*

### 9.2.145 Reuse of existing software (6.2.7.11a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.7.11a**

The detailed configuration status of the reused software baseline shall be provided to the customer in the reuse file for acceptance.

**Expected Output**: Software reuse file [DJF, SRF; SRR, PDR, CDR, QR, AR]

**QDP Status (Y):** See *Software Reuse File (SRF)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.4.3.7a (Reuse of existing software)*

### 9.2.146 Automatic code generation (6.2.8.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.8.1a**

For the selection of tools for automatic code generation, the supplier shall evaluate the following aspects: 1. evolution of the tools in relation to the tools that use the generated code as an input; 2. customization of the tools to comply with project standards; 3. portability requirements for the generated code; 4. collection of the required design and code metrics; 5. verification of software components containing generated code; 6. configuration control of the tools including the parameters for customisation; 7. compliance with open standards. {NOTE: Examples for item 1: compilers or code management systems.}

**QDP Status (Y):** No use of automatically generated code is planned

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.4d (Automatic code generation)*

### 9.2.147 Automatic code generation (6.2.8.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.8.2a**

The requirements on testing applicable to the automatically generated code shall ensure the achievement of the same objectives as those for manually generated code.

**Expected Output**: Validation and testing documentation [DJF, SValP; PDR], [DJF, SVS; CDR, QR, AR], [DJF, SUITP; PDR, CDR]

**QDP Status (Y):** No use of automatically generated code is planned

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.4d (Automatic code generation)*

### 9.2.148 Automatic code generation (6.2.8.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.8.3a**

The required level of verification and validation of the automatic generation tool shall be at least the same as the one required for the generated code, if the tool is used to skip verification or testing activities on the target code.

**QDP Status (Y):** No use of automatically generated code is planned

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.4d (Automatic code generation)*

### 9.2.149 Automatic code generation (6.2.8.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.8.4a**

Modelling standards for automatic code generation tools shall be defined and applied.

**Expected Output**: Modelling standards [PAF, -; SRR, PDR]

**QDP Status (Y):** No use of automatically generated code is planned

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.4d (Automatic code generation)*

### 9.2.150 Automatic code generation (6.2.8.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.8.5a**

Adherence to modelling standards shall be verified.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** No use of automatically generated code is planned

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.4d (Automatic code generation)*

### 9.2.151 Automatic code generation (6.2.8.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.8.6a**

Clause 6.3.4 shall apply to automatically generated code, unless the supplier demonstrates that the automatically generated code does not need to be manually modified.

**QDP Status (Y):** No use of automatically generated code is planned

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.4d (Automatic code generation)*

### 9.2.152 Automatic code generation (6.2.8.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.2.8.7a**

The verification and validation documentation shall address separately the activities to be performed for manually and automatically generated code.

**Expected Output**: Validation and testing documentation [DJF, SValP; PDR], [DJF, SVS; CDR, QR, AR], [DJF, SUITP; PDR, CDR]

**QDP Status (Y):** No use of automatically generated code is planned

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.3.2.4d (Automatic code generation)*

### 9.2.153 Software related system requirements process (6.3.1.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.1.1a**

For the definition of the software related system requirements to be specified in the requirements baseline, ECSS-E-ST-40 clause *5.2* shall apply.

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.2.2.1a (Specification of system requirements allocated to software)*
- *ECSS-E-ST-40C 5.2.2.2a (Identification of observability requirements)*
- *ECSS-E-ST-40C 5.2.2.3a (Specification of HMI requirements)*
- *ECSS-E-ST-40C 5.2.3.1a (Verification and validation process requirements)*
- *ECSS-E-ST-40C 5.2.3.2a (System input for software validation)*
- *ECSS-E-ST-40C 5.2.3.3a (System input for software installation and acceptance)*
- *ECSS-E-ST-40C 5.2.4.1a (Identification of software versions for software integration into the system)*
- *ECSS-E-ST-40C 5.2.4.1b (Identification of software versions for software integration into the system)*
- *ECSS-E-ST-40C 5.2.4.2a (Supplier support to system integration)*
- *ECSS-E-ST-40C 5.2.4.3a (Interface requirement specification)*
- *ECSS-E-ST-40C 5.2.4.4a (System database)*
- *ECSS-E-ST-40C 5.2.4.5a (Development constraints)*
- *ECSS-E-ST-40C 5.2.4.6a (On board control procedures)*
- *ECSS-E-ST-40C 5.2.4.7a (Development of software to be reused)*
- *ECSS-E-ST-40C 5.2.4.8a (Software safety and dependability requirements)*
- *ECSS-E-ST-40C 5.2.4.9a (Format and data medium)*
- *ECSS-E-ST-40C 5.2.5a (System requirements review)*

### 9.2.154 Software related system requirements process (6.3.1.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.1.2a**

The requirements baseline shall be subject to documentation control and configuration management as part of the development documentation.

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.155 Software related system requirements process (6.3.1.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.1.3a**

For the definition of the requirements baseline, all results from the safety and dependability analyses (including results from the HSIA ECSS-Q-ST-30 clause 6.4.2.3) shall be used.

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.156 Software requirements analysis (6.3.2.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.2.1a**

The requirements baseline shall be analyzed to fully and unambiguously define the software requirements in the technical specification.

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.2.2.8a (Software dependability and safety)*

### 9.2.157 Software requirements analysis (6.3.2.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.2.2a**

The technical specification shall be subject to documentation control and configuration management as part of the development documentation.

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.2.2.8a (Software dependability and safety)*

### 9.2.158 Software requirements analysis (6.3.2.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.2.3a**

For the definition of the technical specification, all results from the safety and dependability analyses (including results from the HSIA ECSS-Q-ST-30 clause 6.4.2.3) shall be used.

**QDP Status (N):** See *No Software Dependability and Safety Analysis*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.2.2.8a (Software dependability and safety)*

### 9.2.159 Software requirements analysis (6.3.2.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.2.4a**

In addition to the functional requirements, the technical specification shall include all non-functional requirements necessary to satisfy the requirements baseline, including, as a minimum, the following: 1. performance, 2. safety, 3. reliability, 4. robustness, 5. quality, 6. maintainability, 7. configuration management, 8. security, 9. privacy, 10. metrication, and 11. verification and validation. {NOTE: Performance requirements include requirements on numerical accuracy.}

**Expected Output**: Software requirements specification [TS, SRS; PDR]

**QDP Status (Ye):** Security and privacy will not be considered. TS will be done by reverse engineering of RTEMS. See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.2.2.8a (Software dependability and safety)*

### 9.2.160 Software requirements analysis (6.3.2.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.2.5a**

Prior to the technical specification elaboration, customer and supplier shall agree on the following principles and rules as a minimum: 1. assignment of persons (on both sides) responsible for establishing the technical specification; 2. methods for agreeing on requirements and approving changes; 3. efforts to prevent misunderstandings such as definition of terms, explanations of background of requirements; 4. recording and reviewing discussion results on both sides.

**QDP Status (Y):** See [EDI19c] and [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 6.2.2.8a (Software dependability and safety)*

### 9.2.161 Software architectural design and design of software items (6.3.3.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.3.1a**

The design definition file shall be subject to documentation control and configuration management.

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*.

### 9.2.162 Software architectural design and design of software items (6.3.3.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.3.2a**

Mandatory and advisory design standards shall be defined and applied.

**Expected Output**: Design standards [PAF, -; SRR, PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.163 Software architectural design and design of software items (6.3.3.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.3.3a**

For software in which numerical accuracy is relevant to mission success specific rules on design and code shall be defined to ensure that the specified level of accuracy is obtained. {NOTE: For example: for an attitude and orbit control subsystem, scientific data generation components.}

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR]

**QDP Status (N/A):** Numerical accuracy is irrelevant to the RTEMS real-time operating system.

For an overview of all clauses, see the *tailoring table*.

### 9.2.164 Software architectural design and design of software items (6.3.3.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.3.4a**

Adherence to design standards shall be verified.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.165 Software architectural design and design of software items (6.3.3.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.3.5a**

The supplier shall define means, criteria and tools to ensure that the complexity and modularity of the design meet the quality requirements.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.166 Software architectural design and design of software items (6.3.3.5b)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.3.5b**

The design evaluation shall be performed in parallel with the design process, in order to provide feedback to the software design team.

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.167 Software architectural design and design of software items (6.3.3.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.3.6a**

Synthesis of the results obtained in the software complexity and modularity evaluation and corrective actions implemented shall be described in the software product assurance reports.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** Software product (RTEMS) already exits, modifications to design shall be constrained. See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.168 Software architectural design and design of software items (6.3.3.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.3.7a**

The supplier shall review the design documentation to ensure that it contains the appropriate level of information for maintenance activities.

**Expected Output**: a. Software product assurance plan [PAF, SPAP; PDR]; b. Software product assurance reports [PAF, -; -]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.169 Coding (6.3.4.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.4.1a**

Coding standards (including consistent naming conventions and adequate commentary rules) shall be specified and observed.

**Expected Output**: Coding standards [PAF, -; PDR]

**QDP Status (Y):** Coding standard implicitly defined by static analysis tools such as Coverity Scan (https://scan.coverity.com/projects/rtems), sonarcloud (https://sonarcloud.io/about), cppcheck (http://cppcheck.sourceforge.net/); coding conventions defined by RTEMS (https://devel.rtems.org/wiki/Developer/Coding/Conventions)

For an overview of all clauses, see the *tailoring table*.

### 9.2.170 Coding (6.3.4.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.4.2a**

The standards shall be consistent with the product quality requirements. {NOTE: Coding standards depend on the software quality objectives (see clause *5.2.7*).}

**Expected Output**: Coding standards [PAF, -; PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 5.2.7.1a (Quality requirements and quality models)*
- *ECSS-Q-ST-80C-R1 5.2.7.2a (Quality requirements and quality models)*

### 9.2.171 Coding (6.3.4.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.4.3a**

The tools to be used in implementing and checking conformance with coding standards shall be identified in the product assurance plan before coding activities start.

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR]

**QDP Status (Ye):** Source code already exists. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.172 Coding (6.3.4.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.4.4a**

Coding standards shall be reviewed with the customer to ensure that they reflect product quality requirements.

**Expected Output**: Coding standards and description of tools [PAF, -; PDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.173 Coding (6.3.4.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.4.5a**

Use of low-level programming languages shall be justified.

**Expected Output**: Software development plan [MGT, SDP; PDR]

**QDP Status (Y):** Part of RTEMS architecture support and BSPs are developed in assembly code.

For an overview of all clauses, see the *tailoring table*.

### 9.2.174 Coding (6.3.4.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.4.6a**

The supplier shall define measurements, criteria and tools to ensure that the software code meets the quality requirements.

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.175 Coding (6.3.4.6b)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.4.6b**

The code evaluation shall be performed in parallel with the coding process, in order to provide feedback to the software programmers.

**QDP Status (Ye):** Source code already exists. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.176 Coding (6.3.4.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.4.7a**

Synthesis of the code analysis results and corrective actions implemented shall be described in the software product assurance reports.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.177 Coding (6.3.4.8a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.4.8a**

The code shall be put under configuration control immediately after successful unit testing.

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*.

### 9.2.178 Testing and validation (6.3.5.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.1a**

Testing shall be performed in accordance with a strategy for each testing level (i.e. unit, integration, validation against the technical specification, validation against the requirements baseline, acceptance), which includes: 1. the types of tests to be performed; 2. the tests to be performed in accordance with the plans and procedures; 3. the means and organizations to perform assurance function for testing and validation. {NOTE: For examples for item 1 are: functional, boundary, performance, and usability tests.}

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR, CDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.179 Testing and validation (6.3.5.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.2a**

Based on the criticality of the software, test coverage goals for each testing level shall be agreed between the customer and the supplier and their achievement monitored by metrics: 1. for unit level testing; 2. for integration level testing; 3. for validation against the technical specification and validation against the requirements baseline.

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR, CDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.180 Testing and validation (6.3.5.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.3a**

The supplier shall ensure through internal review that the test procedures and data are adequate, feasible and traceable and that they satisfy the requirements.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.181 Testing and validation (6.3.5.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.4a**

Test readiness reviews shall be held before the commencement of test activities, as defined in the software development plan.

**Expected Output**: Test readiness review reports [DJF, -; TRR]

**QDP Status (N/A):** No TRR will be performed.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.182 Testing and validation (6.3.5.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.5a**

Test coverage shall be checked with respect to the stated goals.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.183 Testing and validation (6.3.5.5b)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.5b**

Feedback from the results of test coverage evaluation shall be continuously provided to the software developers.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.184 Testing and validation (6.3.5.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.6a**

The supplier shall ensure that nonconformances and software problem reports detected during testing are properly documented and reported to those concerned.

**Expected Output**: Nonconformance reports and software problem reports [DJF, -; CDR, QR, AR, ORR]

**QDP Status (Y):** Use of RTEMS project ticket system and project internal ticket system.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.185 Testing and validation (6.3.5.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.7a**

The test coverage of configurable code shall be checked to ensure that the stated requirements are met in each tested configuration.

**Expected Output**: Statement of compliance with test plans and procedures [PAF, -; CDR, QR, AR, ORR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.186 Testing and validation (6.3.5.8a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.8a**

The completion of actions related to software problem reports generated during testing and validation shall be verified and recorded.

**Expected Output**: Software problem reports [DJF, -; SRR, PDR, CDR, QR, AR, ORR]

**QDP Status (Y):** Use of RTEMS project ticket system and project internal ticket system.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.187 Testing and validation (6.3.5.9a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.9a**

Provisions shall be made to allow witnessing of tests by the customer.

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.188 Testing and validation (6.3.5.10a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.10a**

Provisions shall be made to allow witnessing of tests by supplier personnel independent of the development. {NOTE: For example: specialist software product assurance personnel.}

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.189 Testing and validation (6.3.5.11a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.11a**

The supplier shall ensure that: 1. tests are conducted in accordance with approved test procedures and data, 2. the configuration under test is correct, 3. the tests are properly documented, and 4. the test reports are up to date and valid.

**Expected Output**: Statement of compliance with test plans and procedures [PAF, -; CDR, QR, AR, ORR]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.190 Testing and validation (6.3.5.12a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.12a**

The supplier shall ensure that tests are repeatable by verifying the storage and recording of tested software, support software, test environment, supporting documents and problems found.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.191 Testing and validation (6.3.5.13a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.13a**

The supplier shall confirm in writing that the tests are successfully completed.

**Expected Output**: Testing and validation reports [DJF, -; CDR, QR, AR, ORR]

**QDP Status (N):** Not sure how “in writing” should work practically.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.192 Testing and validation (6.3.5.14a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.14a**

Review boards looking to engineering and product assurance aspects shall be convened after the completion of test phases, as defined in the software development plan.

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.193 Testing and validation (6.3.5.15a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.15a**

Areas affected by any modification shall be identified and re-tested (regression testing).

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.194 Testing and validation (6.3.5.16a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.16a**

In case of re-testing, all test related documentation (test procedures, data and reports) shall be updated accordingly.

**Expected Output**: Updated test documentation [DJF, -; CDR, QR, AR, ORR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.195 Testing and validation (6.3.5.17a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.17a**

The need for regression testing and additional verification of the software shall be analysed after any change of the platform hardware.

**Expected Output**: Updated test documentation [DJF, -; CDR, QR, AR, ORR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.196 Testing and validation (6.3.5.18a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.18a**

The need for regression testing and additional verification of the software shall be analysed after a change or update of any tool used to generate it. {NOTE: For example: source code or object code.}

**Expected Output**: Updated test documentation [DJF, -; CDR, QR, AR, ORR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.197 Testing and validation (6.3.5.19a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.19a**

Validation shall be carried out by staff who have not taken part in the design or coding of the software being validated. {NOTE: This can be achieved at the level of the whole software product, or on a component by component basis.}

**QDP Status (Ye):** Large parts of the source code was produced in a different project by the same staff that does now the validation tests. Due to the elapse of more than one year between writing of the code and the validation tests carried out in this project, sufficient independence is ensured by the pass of time. In general, changes (this includes test cases) in the RTEMS project are reviewed by persons on the RTEMS developer mailing list. See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.1c (Establishment of a software validation process)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.198 Testing and validation (6.3.5.20a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.20a**

Validation of the flight software against the requirement baseline on the flight equipment model shall be performed on a software version without any patch.

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.199 Testing and validation (6.3.5.21a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.21a**

The supplier shall review the test documentation to ensure that it is up to date and organized to facilitate its reuse for maintenance.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.200 Testing and validation (6.3.5.22a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.22a**

Tests shall be organized as activities in their own right in terms of planning, resources and team composition.

**Expected Output**: Test and validation documentation [DJF, SValP; PDR], [DJF, SUITP; PDR, CDR]

**QDP Status (Ye):** See *Software Unit and Integration Test Plan (SUITP)* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.201 Testing and validation (6.3.5.23a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.23a**

The necessary resources for testing shall be identified early in the life cycle, taking into account the operating and maintenance requirements.

**Expected Output**: Test and validation documentation [DJF, SValP; PDR], [DJF, SUITP; PDR, CDR]

**QDP Status (Ye):** See *No Maintenance (MF)*, *No Operational Phase (OP)*, *Software Unit and Integration Test Plan (SUITP)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.202 Testing and validation (6.3.5.24a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.24a**

Test tool development or acquisition (hardware and software) shall be planned for in the overall project plan.

**Expected Output**: Test and validation documentation [DJF, SValP; PDR], [DJF, SUITP; PDR, CDR]

**QDP Status (N/A):** No acquisition is foreseen.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.203 Testing and validation (6.3.5.25a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.25a**

The supplier shall establish and review the test procedures and data before starting testing activities and also document the constraints of the tests concerning physical, performance, functional, controllability and observability limitations.

**Expected Output**: Test and validation documentation [DJF, SValP; PDR], [DJF, SVS; CDR, QR, AR], [DJF, SUITP; PDR, CDR]

**QDP Status (Ye):** See *Software Verification Report (SVR)*, *Software Unit and Integration Test Plan (SUITP)*, and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.204 Testing and validation (6.3.5.26a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.26a**

Before offering the product for delivery and customer acceptance, the supplier shall validate its operation as a complete product, under conditions similar to the application environment as specified in the requirements baseline.

**QDP Status (US):** See *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.205 Testing and validation (6.3.5.27a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.27a**

When testing under the operational environment is performed, the following concerns shall be addressed: 1. the features to be tested in the operational environment; 2. the specific responsibilities of the supplier and customer for carrying out and evaluating the test; 3. restoration of the previous operational environment (after test).

**Expected Output**: Test and validation documentation [DJF, -; AR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.206 Testing and validation (6.3.5.28a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.28a**

Independent software validation shall be performed by a third party. {NOTE: This requirement is applicable where the risks associated with the project justify the costs involved. The customer can consider a less rigorous level of independence, e.g. an independent team in the same organization.}

**Expected Output**: a. ISVV plan [DJF, -; SRR, PDR]; b. ISVV report [DJF, -; PDR, CDR, QR, AR]

**QDP Status (N):** See *No Independent Software Verification and Validation*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.6.2.2b (Selection of an ISVV organization)*
- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.207 Testing and validation (6.3.5.29a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.29a**

The validation shall include testing in the different configurations possible or in a representative set of them when it is evident that the number of possible configurations is too high to allow validation in all of them.

**Expected Output**: Test and validation documentation [DJF, SValP; PDR], [DJF, SVS; CDR, QR, AR]

**QDP Status (Ye):** See *Software Validation Specification (SVS) with Respect to TS* and [EDI19c].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clauses:

- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.208 Testing and validation (6.3.5.30a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.30a**

Software containing deactivated code shall be validated specifically to ensure that the deactivated code cannot be activated or that its accidental activation cannot harm the operation of the system.

**Expected Output**: Testing and validation reports [DJF, -; CDR, QR, AR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.209 Testing and validation (6.3.5.31a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.31a**

Software containing configurable code shall be validated specifically to ensure that unintended configuration cannot be activated at run time or included during code generation.

**Expected Output**: Testing and validation reports [DJF, -; CDR, QR, AR]

**QDP Status (Y):** See *Software Validation Specification (SVS) with Respect to TS*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.210 Testing and validation (6.3.5.32a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.5.32a**

Activities for the validation of the quality requirements shall be specified in the definition of the validation specification.

**Expected Output**: Software validation specification [DJF, SVS; CDR, QR, AR]

**QDP Status (Y):** See *Software Validation Specification (SVS) with Respect to TS*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-Q-ST-80C-R1 7.2.3.1a (Test and validation documentation)*

### 9.2.211 Software delivery and acceptance (6.3.6.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.6.1a**

The roles, responsibilities and obligations of the supplier and customer during installation shall be established.

**Expected Output**: Installation procedure [DDF, SCF; AR]

**QDP Status (Y):** See *Software User Manual (SUM)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.212 Software delivery and acceptance (6.3.6.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.6.2a**

The installation shall be performed in accordance with the installation procedure.

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.213 Software delivery and acceptance (6.3.6.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.6.3a**

The customer shall establish an acceptance test plan specifying the intended acceptance tests including specific tests suited to the target environment (see ECSS-E-ST-40 clause *5.7.3.1*). {NOTE 1: The acceptance tests can be partly made up of tests used during previous test activities.} {NOTE 2: The acceptance test plan takes into account the requirement for operational demonstration, either as part of acceptance or after acceptance.}

**Expected Output**: Acceptance test plan [DJF, -; QR, AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.7.3.1a (Acceptance test planning)*

### 9.2.214 Software delivery and acceptance (6.3.6.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.6.4a**

The customer shall ensure that the acceptance tests are performed in accordance with the approved acceptance test plan (see ECSS-E-ST-40 clause *5.7.3.2*).

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*. This clause references the following clause:

- *ECSS-E-ST-40C 5.7.3.2a (Acceptance test execution)*

### 9.2.215 Software delivery and acceptance (6.3.6.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.6.5a**

Before the software is presented for customer acceptance, the supplier shall ensure that: 1. the delivered software complies with the contractual requirements (including any specified content of the software acceptance data package); 2. the source and object code supplied correspond to each other; 3. all agreed changes are implemented; 4. all nonconformances are either resolved or declared.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.216 Software delivery and acceptance (6.3.6.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.6.6a**

The customer shall verify that the executable code was regenerated from configuration managed source code components and installed in accordance with predefined procedures on the target environment.

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.217 Software delivery and acceptance (6.3.6.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.6.7a**

Any discovered problems shall be documented in nonconformance reports.

**Expected Output**: Nonconformance reports [DJF, -; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.218 Software delivery and acceptance (6.3.6.8a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.6.8a**

On completion of the acceptance tests, a report shall be drawn up and be signed by the supplier's representatives, the customer's representatives, the software quality engineers of both parties and the representative of the organization charged with the maintenance of the software product.

**Expected Output**: Acceptance test report [DJF, -; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.219 Software delivery and acceptance (6.3.6.9a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.6.9a**

The customer shall certify conformance to the procedures and state the conclusion concerning the test result for the software product under test (accepted, conditionally accepted, rejected).

**Expected Output**: Acceptance test report [DJF, -; AR]

**QDP Status (US):** See *No Installation and Acceptance*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.220 Operations (6.3.7.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.7.1a**

During operations, the quality of the mission products related to software shall be agreed with the customer and users. {NOTE: Quality of mission products can include parameters such as: error-free data, availability of data and permissible outages; permissible information degradation}.

**Expected Output**: Software operation support plan [OP, -; ORR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.221 Operations (6.3.7.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.7.2a**

During the demonstration that the software conforms to the operational requirements, the following shall be covered as a minimum: 1. availability and maintainability of the host system (including reboot after maintenance interventions); 2. safety features; 3. human-computer interface; 4. operating procedures; 5. ability to meet the mission product quality requirements.

**Expected Output**: Validation of the operational requirements [PAF, -; ORR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.222 Operations (6.3.7.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.7.3a**

The product assurance plan for system operations shall include consideration of software.

**Expected Output**: Input to product assurance plan for systems operation [PAF, -; ORR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.223 Maintenance (6.3.8.1a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.8.1a**

The organization responsible for maintenance shall be identified to allow a smooth transition into the operations and maintenance. {NOTE: An organization, with representatives from both supplier and customer, can be set up to support the maintenance activities. Attention is drawn to the importance of the flexibility of this organization to cope with the unexpected occurrence of problems and the identification of facilities and resources to be used for the maintenance activities.}

**Expected Output**: Maintenance plan [MF, -; QR, AR, ORR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.224 Maintenance (6.3.8.2a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.8.2a**

The maintenance organization shall specify the assurance, verification and validation activities applicable to maintenance interventions.

**Expected Output**: Maintenance plan [MF, -; QR, AR, ORR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.225 Maintenance (6.3.8.3a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.8.3a**

The maintenance plans shall be verified against specified requirements for maintenance of the software product. {NOTE: The maintenance plans and procedures can address corrective, improving, adaptive and preventive maintenance, differentiating between “routine” and “emergency” maintenance activities.}

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.226 Maintenance (6.3.8.4a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.8.4a**

The maintenance plans and procedures shall include the following as a minimum: 1. scope of maintenance; 2. identification of the first version of the software product for which maintenance is to be done; 3. support organization; 4. maintenance life cycle; 5. maintenance activities; 6. quality measures to be applied during the maintenance; 7. maintenance records and reports.

**Expected Output**: Maintenance plan [MF, -; QR, AR, ORR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.227 Maintenance (6.3.8.5a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.8.5a**

Rules for the submission of maintenance reports shall be established and agreed as part of the maintenance plan.

**Expected Output**: Maintenance plan [MF, -; QR, AR, ORR]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.228 Maintenance (6.3.8.6a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.8.6a**

All maintenance activities shall be logged in predefined formats and retained.

**Expected Output**: Maintenance records [MF, -; -]

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.229 Maintenance (6.3.8.7a)

**ECSS-Q-ST-80C Rev.1 Clause 6.3.8.7a**

Maintenance records shall be established for each software product, including, as a minimum, the following information,: 1. list of requests for assistance or problem reports that have been received and the current status of each; 2. organization responsible for responding to requests for assistance or implementing the appropriate corrective actions; 3. priorities assigned to the corrective actions; 4. results of the corrective actions; 5. statistical data on failure occurrences and maintenance activities. {NOTE: The record of the maintenance activities can be utilized for evaluation and enhancement of the software product and for improvement of the quality system itself}.

**Expected Output**: Maintenance records [MF, -; -]. Software product quality assurance

**QDP Status (US):** See *No Maintenance (MF)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.230 Deriving of requirements (7.1.1a)

**ECSS-Q-ST-80C Rev.1 Clause 7.1.1a**

The software quality requirements (including safety and dependability requirements) shall be derived from the requirements defined at system level.

**Expected Output**: a. Requirement baseline [RB, SSS; SRR]; b. Technical specification [TS, SRS; PDR]

**QDP Status (US):** See *No Software Dependability and Safety Analysis* and *No Requirements Baseline (RB)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.231 Quantitative definition of quality requirements (7.1.2a)

**ECSS-Q-ST-80C Rev.1 Clause 7.1.2a**

Quality requirements shall be expressed in quantitative terms or constraints.

**Expected Output**: a. Requirement baseline [RB, SSS; SRR]; b. Technical specification [TS, SRS; PDR]

**QDP Status (Ye):** Only expected output b. See *Software Requirements Engineering* and *Software Requirements Specification (SRS)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.232 Assurance activities for product quality requirements (7.1.3a)

**ECSS-Q-ST-80C Rev.1 Clause 7.1.3a**

The supplier shall define assurance activities to ensure that the product meets the quality requirements as specified in the technical specification.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.233 Product metrics (7.1.4a)

**ECSS-Q-ST-80C Rev.1 Clause 7.1.4a**

In order to verify the implementation of the product quality requirements, the supplier shall define a metrication programme based on the identified quality model (see clause *5.2.7*), specifying: 1. the metrics to be collected and stored; 2. the means to collect metrics (measurements); 3. the target values, with reference to the product quality requirements; 4. the analyses to be performed on the collected metrics, including the ones to derive: (a) descriptive statistics; (b) trend analysis (such as trends in software problems). 5. how the results of the analyses performed on the collected metrics are fed back to the development team and used to identify corrective actions; 6. the schedule of metrics collection, storing, analysis and reporting, with reference to the whole software life cycle. {NOTE 1: Guidance for software metrication programme implementation can be found in ECSS-Q-HB-80-04. NOTE 2: Example to item 4(a): the number of units at each level of complexity.}

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-Q-ST-80C-R1 5.2.7.1a (Quality requirements and quality models)*
- *ECSS-Q-ST-80C-R1 5.2.7.2a (Quality requirements and quality models)*

### 9.2.234 Basic metrics (7.1.5a)

**ECSS-Q-ST-80C Rev.1 Clause 7.1.5a**

The following basic products metrics shall be used: 1. size (code); 2. complexity (design, code); 3. fault density and failure intensity; 4. test coverage; 5. number of failures.

**Expected Output**: Software product assurance plan [PAF, SPAP; SRR, PDR]

**QDP Status (Ye):** Only 1., 2. (code), and 4. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.235 Reporting of metrics (7.1.6a)

**ECSS-Q-ST-80C Rev.1 Clause 7.1.6a**

The results of metrics collection and analysis shall be included in the software product assurance reports, in order to provide the customer with an insight into the level of quality obtained.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.236 Numerical accuracy (7.1.7a)

**ECSS-Q-ST-80C Rev.1 Clause 7.1.7a**

Numerical accuracy shall be estimated and verified.

**Expected Output**: Numerical accuracy analysis [DJF, SVR; PDR, CDR, QR]

**QDP Status (N/A):** Numerical accuracy is irrelevant to the RTEMS real-time operating system.

For an overview of all clauses, see the *tailoring table*.

### 9.2.237 Analysis of software maturity (7.1.8a)

**ECSS-Q-ST-80C Rev.1 Clause 7.1.8a**

The supplier shall define the organization and means implemented to collect and analyse data required for the study of software maturity. {NOTE: For example: failures, corrections, duration of runs}.

**Expected Output**: Software product assurance reports [PAF, -; -]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.238 Requirements baseline and technical specification (7.2.1.1a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.1.1a**

The software quality requirements shall be documented in the requirements baseline and technical specification.

**Expected Output**: a. Requirement baseline [RB, SSS; SRR]; b. Technical specification [TS, SRS; PDR]

**QDP Status (Ye):** Only expected output b. See *Software Requirements Engineering* and *Software Requirements Specification (SRS)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.239 Requirements baseline and technical specification (7.2.1.2a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.1.2a**

The software requirements shall be: 1. correct; 2. unambiguous; 3. complete; 4. consistent; 5. verifiable; 6. traceable.

**QDP Status (Y):** See *Software Requirements Engineering* and *Software Requirements Specification (SRS)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.240 Requirements baseline and technical specification (7.2.1.3a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.1.3a**

For each requirement the method for verification and validation shall be specified.

**Expected Output**: a. Requirement baseline [RB, SSS; SRR]; b. Technical specification [TS, SRS; PDR]

**QDP Status (Ye):** Only expected output b. See *Software Requirements Engineering* and *Software Requirements Specification (SRS)*.

For an overview of all clauses, see the *tailoring table*. This clause is referenced by the following clause:

- *ECSS-E-ST-40C 5.8.3.2a (Verification of the technical specification)*

### 9.2.241 Design and related documentation (7.2.2.1a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.2.1a**

The software design shall meet the non-functional requirements as documented in the technical specification.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.242 Design and related documentation (7.2.2.2a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.2.2a**

The software shall be designed to facilitate testing.

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.243 Design and related documentation (7.2.2.3a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.2.3a**

Software with a long planned lifetime shall be designed with minimum dependency on the operating system and the hardware, in order to aid portability. {NOTE: This requirement is applicable to situations where the software lifetime can lead to the obsolescence and non-availability of the original operating system and/or hardware, thereby jeopardizing the maintainability the software.}

**Expected Output**: a. Software product assurance plan [PAF, SPAP; SRR, PDR]; b. Justification of design choices [DDF, SDD; PDR, CDR]

**QDP Status (Y):** See *Software Design Document (SDD)* and [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.244 Test and validation documentation (7.2.3.1a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.3.1a**

Detailed test and validation documentation (data, procedures and expected results) defined in the ECSS-E-ST-40 DJF shall be consistent with the defined test and validation strategy (see clause *6.3.5* and ECSS-E-ST-40 clauses *5.5.3*, *5.5.4*, *5.6* and *5.8*).

**QDP Status (Y):** See [EDI19e].

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.5.3.1a (Development and documentation of the software units)*
- *ECSS-E-ST-40C 5.5.3.2a (Software unit testing)*
- *ECSS-E-ST-40C 5.5.3.2b (Software unit testing)*
- *ECSS-E-ST-40C 5.5.3.2c (Software unit testing)*
- *ECSS-E-ST-40C 5.5.4.1a (Software integration test plan development)*
- *ECSS-E-ST-40C 5.5.4.2a (Software units and software component integration and testing)*
- *ECSS-E-ST-40C 5.6.2.1a (Establishment of a software validation process)*
- *ECSS-E-ST-40C 5.6.2.1b (Establishment of a software validation process)*
- *ECSS-E-ST-40C 5.6.2.1c (Establishment of a software validation process)*
- *ECSS-E-ST-40C 5.6.2.2a (Selection of an ISVV organization)*
- *ECSS-E-ST-40C 5.6.2.2b (Selection of an ISVV organization)*

- *ECSS-E-ST-40C 5.6.3.1a (Development and documentation of a software validation specification with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.1b (Development and documentation of a software validation specification with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.1c (Development and documentation of a software validation specification with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.2a (Conducting the validation with respect to the technical specification)*
- *ECSS-E-ST-40C 5.6.3.3a (Updating the software user manual)*
- *ECSS-E-ST-40C 5.6.3.4a (Conducting a critical design review)*
- *ECSS-E-ST-40C 5.6.4.1a (Development and documentation of a software validation specification with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.1b (Development and documentation of a software validation specification with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.1c (Development and documentation of a software validation specification with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.2a (Conducting the validation with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.2b (Conducting the validation with respect to the requirements baseline)*
- *ECSS-E-ST-40C 5.6.4.3a (Updating the software user manual)*
- *ECSS-E-ST-40C 5.6.4.4a (Conducting a qualification review)*
- *ECSS-E-ST-40C 5.8.2.1a (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.1b (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.1c (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.1d (Establishment of the software verification process)*
- *ECSS-E-ST-40C 5.8.2.2a (Selection of the organization responsible for conducting the verification)*
- *ECSS-E-ST-40C 5.8.2.2b (Selection of the organization responsible for conducting the verification)*
- *ECSS-E-ST-40C 5.8.3.1a (Verification of requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.2a (Verification of the technical specification)*
- *ECSS-E-ST-40C 5.8.3.3a (Verification of the software architectural design)*
- *ECSS-E-ST-40C 5.8.3.4a (Verification of the software detailed design)*
- *ECSS-E-ST-40C 5.8.3.5a (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5b (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5c (Verification of code)*

- *ECSS-E-ST-40C 5.8.3.5d (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5e (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.5f (Verification of code)*
- *ECSS-E-ST-40C 5.8.3.6a (Verification of software unit testing (plan and results))*
- *ECSS-E-ST-40C 5.8.3.7a (Verification of software integration)*
- *ECSS-E-ST-40C 5.8.3.8a (Verification of software validation with respect to the technical specifications and the requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.8b (Verification of software validation with respect to the technical specifications and the requirements baseline)*
- *ECSS-E-ST-40C 5.8.3.9a (Evaluation of validation: complementary system level validation)*
- *ECSS-E-ST-40C 5.8.3.10a (Verification of software documentation)*
- *ECSS-E-ST-40C 5.8.3.11a (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.11b (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.11c (Schedulability analysis for real-time software)*
- *ECSS-E-ST-40C 5.8.3.12a (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.12b (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.12c (Technical budgets management)*
- *ECSS-E-ST-40C 5.8.3.13a (Behaviour modelling verification)*
- *ECSS-E-ST-40C 5.8.3.13b (Behaviour modelling verification)*
- *ECSS-E-ST-40C 5.8.3.13c (Behaviour modelling verification)*
- *ECSS-Q-ST-80C-R1 6.3.5.1a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.2a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.3a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.4a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.5a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.5b (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.6a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.7a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.8a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.9a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.10a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.11a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.12a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.13a (Testing and validation)*

- *ECSS-Q-ST-80C-R1 6.3.5.14a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.15a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.16a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.17a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.18a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.19a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.20a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.21a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.22a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.23a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.24a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.25a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.26a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.27a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.28a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.29a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.30a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.31a (Testing and validation)*
- *ECSS-Q-ST-80C-R1 6.3.5.32a (Testing and validation)*

### 9.2.245 Test and validation documentation (7.2.3.2a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.3.2a**

The test documentation shall cover the test environment, tools and test software, personnel required and associated training requirements.

**QDP Status (Y):** See *Software Requirements Engineering*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.246 Test and validation documentation (7.2.3.3a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.3.3a**

The criteria for completion of each test and any contingency steps shall be specified.

**QDP Status (Y):** See *Software Requirements Engineering*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.247 Test and validation documentation (7.2.3.4a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.3.4a**

Test procedures, data and expected results shall be specified.

**QDP Status (Y):** See *Software Requirements Engineering*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.248 Test and validation documentation (7.2.3.5a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.3.5a**

The hardware and software configuration shall be identified and documented as part of the test documentation.

**QDP Status (Y):** See *Software Requirements Engineering*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.249 Test and validation documentation (7.2.3.6a)

**ECSS-Q-ST-80C Rev.1 Clause 7.2.3.6a**

For any requirements not covered by testing a verification report shall be drawn up documenting or referring to the verification activities performed.

**Expected Output**: Software verification report [DJF, SVR; CDR, QR, AR]

**QDP Status (Y):** See *Software Requirements Engineering* and *Software Verification Report (SVR)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.250 Software reuse/Customer requirements (7.3.1a)

**ECSS-Q-ST-80C Rev.1 Clause 7.3.1a**

For the development of software intended for reuse, ECSS-E-ST-40 clauses *5.2.4.7* and *5.4.3.6* shall apply.

**QDP Status (Y):** See ECSS-E-ST-40C tailoring.

For an overview of all clauses, see the *tailoring table*. This clause references the following clauses:

- *ECSS-E-ST-40C 5.2.4.7a (Development of software to be reused)*
- *ECSS-E-ST-40C 5.4.3.6a (Definition of methods and tools for software intended for reuse)*
- *ECSS-E-ST-40C 5.4.3.6b (Definition of methods and tools for software intended for reuse)*
- *ECSS-E-ST-40C 5.4.3.6c (Definition of methods and tools for software intended for reuse)*

### 9.2.251 Software reuse/Separate documentation (7.3.2a)

**ECSS-Q-ST-80C Rev.1 Clause 7.3.2a**

The information related to the components developed for reuse shall be separated from the others in the technical specification, design justification file, design definition file and product assurance file.

**QDP Status (Y):** The other components do not exist. The complete RTEMS real-time operating system is intended to be reused.

For an overview of all clauses, see the *tailoring table*.

### 9.2.252 Software reuse/Self-contained information (7.3.3a)

**ECSS-Q-ST-80C Rev.1 Clause 7.3.3a**

The information related to components developed for reuse in the technical specification, the design justification file, the design definition file and the product assurance file shall be self-contained.

**QDP Status (Ye):** The product assurance file contains documents used by the overall project. See [EDI19e].

For an overview of all clauses, see the *tailoring table*.

### 9.2.253 Software reuse/Requirements for intended reuse (7.3.4a)

**ECSS-Q-ST-80C Rev.1 Clause 7.3.4a**

The technical specification of components developed for reuse shall include requirements for maintainability, portability and verification of those components.

**Expected Output**: Technical specification for reusable components [TS, -; PDR]

**QDP Status (Y):** See *Software Requirements Engineering*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.254 Software reuse/Configuration management for intended reuse (7.3.5a)

**ECSS-Q-ST-80C Rev.1 Clause 7.3.5a**

The configuration management system shall include provisions for handling specific aspects of software developed for reuse, such as: 1. longer lifetime of the components developed for reuse compared to the other components of the project; 2. evolution or change of the development environment for the next project that intends to use the components; 3. transfer of the configuration and documentation management information to the next project reusing the software.

**Expected Output**: Software configuration management plan [MGT, SCMP; SRR, PDR]

**QDP Status (Y):** See [EDI19b].

For an overview of all clauses, see the *tailoring table*.

### 9.2.255 Software reuse/Testing on different platforms (7.3.6a)

**ECSS-Q-ST-80C Rev.1 Clause 7.3.6a**

Where the components developed for reuse are developed to be reusable on different platforms, the testing of the software shall be performed on all those platforms.

**Expected Output**: Verification and validation documentation for reusable components [DJF, -; CDR]

**QDP Status (Y):** See [EDI19c].

For an overview of all clauses, see the *tailoring table*.

### 9.2.256 Software reuse/Certificate of conformance (7.3.7a)

**ECSS-Q-ST-80C Rev.1 Clause 7.3.7a**

The supplier shall provide a certificate of conformance that the tests have been successfully completed on all the relevant platforms. {NOTE: In case not all platforms are available, the certificate of conformance states the limitations of the validation performed.}

**Expected Output**: Verification and validation documentation for reusable components [DJF, -; CDR]

**QDP Status (Y):** See [EDI19d].

For an overview of all clauses, see the *tailoring table*.

### 9.2.257 Operational system/Hardware procurement (7.4.1a)

**ECSS-Q-ST-80C Rev.1 Clause 7.4.1a**

The subcontracting and procurement of hardware shall be carried out according to the requirements of ECSS-Q-ST-20 clause 5.4.

**Expected Output**: a. Justification of selection of operational ground equipment [DJF, -; SRR, PDR]; b. Receiving inspection reports [PAF, -; SRR, PDR]

**QDP Status (N/A):** No hardware procurement or subcontracting are planned.

For an overview of all clauses, see the *tailoring table*.

### 9.2.258 Operational system/Service procurement (7.4.2a)

**ECSS-Q-ST-80C Rev.1 Clause 7.4.2a**

The procurement of support services to be used in operational phases shall be justified as covering service level agreements, quality of services and escalation procedures, as needed for system exploitation and maintenance.

**Expected Output**: Justification of selection of operational support services [DJF, -; SRR, PDR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.259 Operational system/Constraints (7.4.3a)

**ECSS-Q-ST-80C Rev.1 Clause 7.4.3a**

The choice of procured hardware and services shall address the constraints associated with both the development and the actual use of the software.

**Expected Output**: Justification of selection of operational ground equipment [DJF, -; SRR, PDR]

**QDP Status (US):** See *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.260 Operational system/Selection (7.4.4a)

**ECSS-Q-ST-80C Rev.1 Clause 7.4.4a**

The ground computer equipment and supporting services for implementing the final system shall be selected according to the project requirements regarding: 1. performance; 2. maintenance; 3. durability and technical consistency with the operational equipment; 4. the assessment of the product with respect to requirements, including the criticality category; 5. the available support documentation; 6. the acceptance and warranty conditions; 7. the conditions of installation, preparation, training and use; 8. the maintenance conditions, including the possibilities of evolutions; 9. copyright constraints; 10. availability; 11. compatibility; 12. site operational constraints.

**Expected Output**: Justification of selection of operational ground equipment [DJF, -; SRR, PDR]

**QDP Status (N/A):** No ground computer equipment.

For an overview of all clauses, see the *tailoring table*.

### 9.2.261 Operational system/Maintenance (7.4.5a)

**ECSS-Q-ST-80C Rev.1 Clause 7.4.5a**

Taking account of the provider's maintenance and product policy, it shall be ensured that the hardware and support services can be maintained throughout the specified life of the software product within the operational constraints.

**QDP Status (US):** See *No Maintenance (MF)* and *No Operational Phase (OP)*.

For an overview of all clauses, see the *tailoring table*.

### 9.2.262 Firmware/Device programming (7.5.1a)

**ECSS-Q-ST-80C Rev.1 Clause 7.5.1a**

The supplier shall establish procedures for firmware device programming and duplication of firmware devices.

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR]

**QDP Status (N/A):** No firmware support.

For an overview of all clauses, see the *tailoring table*.

### 9.2.263 Firmware/Marking (7.5.2a)

**ECSS-Q-ST-80C Rev.1 Clause 7.5.2a**

The firmware device shall be indelibly marked to allow the identification (by reference) of the hardware component and of the software component.

**Expected Output**: Software product assurance plan [PAF, SPAP; PDR]

**QDP Status (N/A):** No firmware support.

For an overview of all clauses, see the *tailoring table*.

### 9.2.264 Firmware/Calibration (7.5.3a)

**ECSS-Q-ST-80C Rev.1 Clause 7.5.3a**

The supplier shall ensure that the firmware programming equipment is calibrated.

**QDP Status (N/A):** No firmware support.

For an overview of all clauses, see the *tailoring table*.

## 9.3 Tailoring of SOW QDP Requirements

### 9.3.1 RS-1

**RS-1**

RTEMS shall be qualified to software criticality level “B” without performing ISVV.

Note: this effectively means qualification at criticality level “C”, which is sufficient for the typical RTEMS-SMP applications (i.e. data processing on instruments). However, by ensuring that all QA processes are in place for category-“B”, it would be possible to scale up to include ISVV, in case there is a future mission requirement to do so.

**QDP Status (Ye):** The pre-qualification of RTEMS will be done according to the *Tailoring of ECSS-E-ST-40C* and *Tailoring of ECSS-Q-ST-80C Rev.1* and only for the RTEMS components defined by the space profile proposal, see [eb19].

### 9.3.2 RS-2

**RS-2**

The qualification data pack shall be approved by ESA.

Note: The approved qualification data pack shall be digitally signed by ESA and released under the Creative Commons BY-NC-ND 4.0 license, with ESA as the copyright owner. Usage of the qualification data pack outside ESA missions or ESA member states, or usage outside the space domain, or in commercial projects needs to be agreed with ESA separately.

**QDP Status (Ye):** The ESA copyright can only cover an approved QDP as a container with items which have other copyright holders.

### 9.3.3 RS-3

**RS-3**

The qualification data pack shall be distributable through the European Space Software Repository.

Note: The qualification data pack shall be free of any proprietary information, or information with a restrictive use license.

**QDP Status (Y):** Yes.

### 9.3.4 RS-4

**RS-4**

The qualification of RTEMS with SMP disabled at compilation time (referred to as “classic RTEMS”) shall be demonstrated on the LEON3FT and LEON4FT processor architectures.

**QDP Status (Y):** Yes, see *Variants*.

### 9.3.5 RS-5

**RS-5**

The qualification environment shall be used to produce the qualification data pack for classic RTEMS on the GR712RC (using a single core).

**QDP Status (Y):** Yes, see *Variants*.

### 9.3.6 RS-6

**RS-6**

The qualification environment shall be used to produce the qualification data pack for classic RTEMS on the GR740 (using a single core).

**QDP Status (Y):** Yes, see *Variants*.

### 9.3.7 RS-7

**RS-7**

The qualification of RTEMS with SMP enabled at compilation time (referred to as "RTEMS-SMP") shall be demonstrated on the LEON3FT and LEON4FT processor architectures.

**QDP Status (Y):** Yes, see *Variants*.

### 9.3.8 RS-8

**RS-8**

The qualification environment shall be used to produce the qualification data pack for RTEMS-SMP on the GR712RC (using both cores).

**QDP Status (Ye):** The number of cores which are utilized by the RTEMS software product of the corresponding QDP variant will be determined by the application configuration. This can be one or two cores. See *Variants*.

### 9.3.9 RS-9

**RS-9**

The qualification environment shall be used to produce the qualification data pack for RTEMS-SMP on the GR740 (using two, three or all four cores).

**QDP Status (Ye):** The number of cores which are utilized by the RTEMS software product of the corresponding QDP variant will be determined by the application configuration. This can be one, two, three, or four cores. See *Variants*.

### 9.3.10 RS-10

**RS-10**

The qualification shall consider all on-chip peripherals and take into account all known System-on-Chip and board errata.

Note: during Task 2.1, a decision must be made in agreement with ESA, which peripherals are included in the scope of this qualification activity.

**QDP Status (Ye):** The supported on-chip peripherals are determined by the space profile proposal, see [eb19].

### 9.3.11 RS-11

**RS-11**

The baseline hardware target for the GR712RC shall be http://www.gaisler.com/index.php/products/boards/gr712rc-board.

**QDP Status (Y):** Yes.

### 9.3.12 RS-12

**RS-12**

The baseline hardware target for the GR740 shall be http://www.gaisler.com/index.php/products/boards/gr-cpci-gr740.

**QDP Status (Y):** Yes.

### 9.3.13 RS-13

**RS-13**

The qualification shall be performed with the following cross-compilers, or their future evolutions if they appear during the execution of this project:

- The GCC compiler built automatically via the RTEMS source builder
- GCC 7.2.0 as part of Gaisler RCC
- LLVM/Clang 4.0.0 as part of Gaisler RCC

**QDP Status (Ye):** The compiler of the Gaisler RCC will only be used if they work out of the box.

### 9.3.14 RS-14

**RS-14**

The qualified mathematical library shall be used as the baseline for all qualification configurations.

Note: The library will be delivered by ESA or made available on ESSR website.

**QDP Status (Y):** Yes.

### 9.3.15 RS-15

**RS-15**

Dedicated qualification configurations shall be created to qualify OpenMP and MTAPI on RTEMS-SMP.

**QDP Status (N):** OpenMP and MTAPI are not included in the space profile proposal, see [eb19].

## 9.4 Justifications of Tailoring Decisions

### 9.4.1 No Requirements Baseline (RB)

The *RB* (*SSS* and *IRD*) will not be produced, since the system requirements are to be defined by the end user of the *QDP*. RTEMS is designed as a reusable software product which can be utilized by application designers to ease the development of their applications. The requirements of the end system (system requirements) using RTEMS are only known to the application designer. RTEMS itself is developed by the RTEMS maintainers and they do not know the requirements of a particular end system in general. RTEMS is designed as a real-time operating system to meet typical system requirements for a wide range of applications. Its suitability for a particular application must be determined by the application designer based on the technical specification provided by RTEMS accompanied with performance data for a particular target platform.

### 9.4.2 No Installation and Acceptance

The actual installation and acceptance of the QDP is not included in the task to create the QDP. This is done by the end user of the QDP. In this project, this is a part of Task 4 carried out by Jena-Optronik.

### 9.4.3 No Maintenance (MF)

The maintenance software life cycle state is outside the scope of this project. The RTEMS development will continue independently. One goal of this project is to establish procedures in the RTEMS community so that the quality level achieved by this activity can be maintained in the future development of RTEMS.

### 9.4.4 No Operational Phase (OP)

The operational software life cycle state is outside the scope of this project. The end users of the QDP will have an operational phase if the software product is used in their applications.

### 9.4.5 On Demand Unit and Integration Testing

The RTEMS design and development started in the 1980s. The software industry discovered the value of unit tests after this decade. The initial RTEMS test suite contained not unit tests and currently there are only a couple of unit tests in the RTEMS test suite. The approach of the RTEMS Project was always to do the testing at the API level even for tests which target internal implementation functions, see for example this discussion on the RTEMS development mailing list: [PATCH 3/3] smpschedsem01: New test Test to verify task priority is inherited from a semaphore. The main reason for this is to find code which is not used by an API through code coverage analysis. The purpose of RTEMS is to provide an API. All code which is not needed by an API is superfluous and should be removed or deactivated.

The spirit of the RTEMS test suite is to do unit tests of internal functions through API level calls. This is also the benchmark for the validation tests developed by this activity. The *Action*

*Requirement Item Type* specification items allow a very detailed specification of functions. For directives, not only the function parameters are considered, but also states of the system relevant to the directive call. Examples are the state of threads which are affected by the directive, the execution context and mode of the caller, interrupts which happen during the directive call, activity on other processors during the directive call, and the state of the object accessed by the directive. The RTEMS Test Framework which was developed for this activity provides support for Interrupt Tests.

The goal of this activity is to reach 100% statement and branch coverage through validation tests. Once this is achieved, this will show that everything in the implementation serves a purpose defined at API level. One problem with this approach is that it is not immediately clear for a given software unit to which requirements it belongs. It would be very labour intensive to explicitly link each unit to a set of requirements. It would be also hard to maintain this information. For a given unit, the source code and SDD could be used to manually get this information. A verification activity could do this for a random sample set of units.

For an operating system, there are some difficulties if unit tests for all units should be carried out. Some units change the state of the processor at register level. This makes mocking difficult. The inputs to units are no only passed as parameters, there is a considerable amount of global state in the system. The test execution may be controlled by the system under test. For an operation system which is developed from scratch it would make sense to write a mocking framework to do unit tests right from the beginning of the development. However, RTEMS is a fully designed, developed, and tested software product. In this case, it is more efficient to avoid any mocking and instead set up the pre-conditions for tests through the already existing API level functions. This is what we do for the validation tests.

Unit and integration tests will only be performed on a subset of units in case validation tests are difficult to carry out or insufficient. For example, unit tests may be done for the chain and red-black tree data structures.

### 9.4.6 Combined Unit and Integration Testing

Unit and integration tests will be combined into a single *Software Unit and Integration Test Plan (SUITP)* and *Software Unit and Integration Test Report*. The boundaries between unit and integration tests for a software product which ships as a single library are vague. To avoid the trouble of giving a precise definition of what unit and integration tests are with respect to each other they are combined into one set of tests.

### 9.4.7 No Logical and Computational Model

As per negotiation item ESA-14, ESA agreed to downgrade in the logical and computational model requirement to criticality category C (not requiring the logical and computational modelling of RTEMS architecture and design). The formal verification task (Task 3) carried out by Lero in this activity should compensate for this. In addition, the RTEMS real-time operating-system itself has no active components (e.g. threads). It is entirely event driven (application, interrupts). It provides the means to instantiate active components by the application.

### 9.4.8 No Schedulability Analysis

Schedulability analysis can only be performed at application level.

### 9.4.9 No Software Dependability and Safety Analysis

Software dependability and safety activities and reports will not be produced. RTEMS Improvement SCAR report and findings will be used in this project as inputs.

### 9.4.10 No Independent Software Verification and Validation

The *ISVV* is excluded from the project by the *SOW*.

### 9.4.11 No Numerical Accuracy Analysis

As operating system RTEMS does not use floating-point numbers. Therefore, no numerical accuracy analysis can be performed.

# BIBLIOGRAPHY

[RTEa] RTEMS Classic API Guide. URL: https://docs.rtems.org/branches/master/c-user.pdf.

[RTEb] RTEMS Software Engineering. URL: https://docs.rtems.org/branches/master/eng.pdf.

[Bra97] Scott Bradner. Key words for use in RFCs to Indicate Requirement Levels. BCP 14, RFC Editor, March 1997. http://www.rfc-editor.org/rfc/rfc2119.txt. URL: http://www.rfc-editor.org/rfc/rfc2119.txt.

[BA14] Jace Browning and Robert Adams. Doorstop: Text-Based Requirements Management Using Version Control. *Journal of Software Engineering and Applications*, 7:187–194, 2014. URL: http://www.scirp.org/pdf/JSEA_2014032713545074.pdf.

[ECS08a] ECSS. *ECSS-E-ST-70-11C – Space segment operability*. European Cooperation for Space Standardization, 2008. URL: https://ecss.nl/standard/ecss-e-st-70-11c-space-segment-operability/.

[ECS08b] ECSS. *ECSS-E-ST-70C – Ground systems and operations*. European Cooperation for Space Standardization, 2008. URL: https://ecss.nl/standard/ecss-e-st-70c-ground-systems-and-operations/.

[ECS08c] ECSS. *ECSS-M-ST-10-01C – Organization and conduct of reviews*. European Cooperation for Space Standardization, 2008. URL: https://ecss.nl/standard/ecss-m-st-10-01c-organization-and-conduct-of-reviews/.

[ECS08d] ECSS. *ECSS-M-ST-80C – Risk management*. European Cooperation for Space Standardization, 2008. URL: https://ecss.nl/standard/ecss-m-st-80c-risk-management/.

[ECS08e] ECSS. *ECSS-S-ST-00C – Description, implementation and general requirements*. European Cooperation for Space Standardization, 2008. URL: https://ecss.nl/standard/ecss-s-st-00c-description-implementation-and-general-requirements-31-july-2008/.

[ECS09a] ECSS. *ECSS-E-ST-10-06C - Technical requirements specification*. European Cooperation for Space Standardization, 2009. URL: https://ecss.nl/standard/ecss-e-st-10-06c-technical-requirements-specification/.

[ECS09b] ECSS. *ECSS-E-ST-40C - Software general requirements*. European Cooperation for Space Standardization, 2009. URL: https://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/.

[ECS09c] ECSS. *ECSS-M-ST-10C Rev.1 – Project planning and implementation*. European Cooperation for Space Standardization, 2009. URL: https://ecss.nl/standard/ecss-m-st-10c-rev-1-project-planning-and-implementation/.

[ECS09d] ECSS. *ECSS-M-ST-40C Rev.1 - Configuration and information management*. European Cooperation for Space Standardization, 2009. URL: https://ecss.nl/standard/ecss-m-st-40c-rev-1-configuration-and-information-management/.

[ECS09e] ECSS. *ECSS-Q-ST-30-02C – Failure modes, effects (and criticality) analysis (FMEA/FMECA)*. European Cooperation for Space Standardization, 2009. URL: https://ecss.nl/standard/ecss-q-st-30-02c-failure-modes-effects-and-criticality-analysis-fmeafmeca/.

[ECS10] ECSS. *ECSS-Q-HB-80-02 Part 2A – Software process assessment and improvement – Part 2: Assessor instrument*. European Cooperation for Space Standardization, 2010. URL: https://ecss.nl/hbstms/ecss-q-hb-80-02-part-2a-software-process-assessment-and-improvement-part-2-assessor-instrume

[ECS11] ECSS. *ECSS-Q-HB-80-04A – Software metrication programme definition and implementation*. European Cooperation for Space Standardization, 2011. URL: http://ecss.nl/hbstms/ecss-q-hb-80-04a-software-metrication-handbook/.

[ECS12a] ECSS. *ECSS-E-ST-10-03C - Space engineering - Testing*. European Cooperation for Space Standardization, 2012. URL: https://ecss.nl/standard/ecss-e-st-10-03c-testing/.

[ECS12b] ECSS. *ECSS-S-ST-00-01C - Glossary of terms*. European Cooperation for Space Standardization, 2012. URL: https://ecss.nl/standard/ecss-s-st-00-01c-glossary-of-terms-1-october-2012/.

[ECS13] ECSS. *ECSS-E-HB-40A – Software engineering handbook*. European Cooperation for Space Standardization, 2013. URL: https://ecss.nl/hbstms/ecss-e-hb-40a-software-engineering-handbook-11-december-2013/.

[ECS14] ECSS. *ECSS-Q-ST-20-07C – Quality and safety assurance for space test centres*. European Cooperation for Space Standardization, 2014. URL: https://ecss.nl/standard/ecss-q-st-20-07c-quality-and-safety-assurance-for-space-test-centres-1-october-2014/.

[ECS15] ECSS. *ECSS-E-ST-10-24C - Space Engineering - Interface Management*. European Cooperation for Space Standardization, 2015. URL: https://ecss.nl/standard/ecss-e-st-10-24c-interface-management-1-june-2015/.

[ECS16] ECSS. *ECSS-Q-ST-10C Rev.1 – Product assurance management*. European Cooperation for Space Standardization, 2016. URL: https://ecss.nl/standard/ecss-q-st-10c-rev-1-product-assurance-management-15-march-2016/.

[ECS17a] ECSS. *ECSS-E-ST-10C Rev.1 - System engineering general requirements*. European Cooperation for Space Standardization, 2017. URL: https://ecss.nl/standard/ecss-e-st-10c-rev-1-system-engineering-general-requirements-15-february-2017/.

[ECS17b] ECSS. *ECSS-Q-ST-30C Rev.1 – Dependability*. European Cooperation for Space Standardization, 2017. URL: https://ecss.nl/standard/ecss-q-st-30c-rev-1-space-product-assurance-dependability-15-february-2017/.

[ECS17c] ECSS. *ECSS-Q-ST-40C Rev.1 – Safety*. European Cooperation for Space Standardization, 2017. URL: https://ecss.nl/standard/ecss-q-st-40c-rev-1-safety-15-february-2017/.

[ECS17d] ECSS. *ECSS-Q-ST-80C Rev.1 - Software product assurance*. European Cooperation for Space Standardization, 2017. URL: https://ecss.nl/standard/ecss-q-st-80c-rev-1-software-product-assurance-15-february-2017/.

[ECS18a] ECSS. *ECSS-E-ST-10-02C - Space engineering - Verification*. European Cooperation for Space Standardization, 2018. URL: https://ecss.nl/standard/ecss-e-st-10-02c-rev-1-verification-1-february-2018/.

[ECS18b] ECSS. *ECSS-Q-ST-20C Rev.2 – Quality assurance*. European Cooperation for Space Standardization, 2018. URL: https://ecss.nl/standard/ecss-q-st-20c-rev-2-quality-assurance-1-february-2018/.

[EDI19a] EDISOFT. *Qualification Toolchain Software Design Document, Release 1*. 2019.

[EDI19b] EDISOFT. *Software Configuration Management Plan, Release 2*. 2019.

[EDI19c] EDISOFT. *Software Development Plan, Release 2*. 2019.

[EDI19d] EDISOFT. *Software Product Assurance Milestone Report, Release 2*. 2019.

[EDI19e] EDISOFT. *Software Product Assurance Plan, Release 2*. 2019.

[EDI19f] EDISOFT. *Software Review Plan, Release 1*. 2019.

[EDI19g] EDISOFT. *Software Review Plan, Release 2*. 2019.

[EDI19h] EDISOFT. *Technical Note on Agile Process, Release 1*. 2019.

[EDI19i] EDISOFT. *Tools Identification, Release 2*. 2019.

[EDI20] EDISOFT. *Software Reuse File, Release 2*. 2020.

[eb19] embedded brains. *Technical Note: Space Profile, Release 2*. embedded brains GmbH, 2019.

[Gai18a] Gaisler. *GR712RC, Data Sheet, Version 2.4*. Cobham plc, 2018. URL: https://www.gaisler.com/doc/gr712rc-datasheet.pdf.

[Gai18b] Gaisler. *GR712RC, User's Manual, Version 2.12*. Cobham plc, 2018. URL: https://www.gaisler.com/doc/gr712rc-usermanual.pdf.

[Gai18c] Gaisler. *GR740, Data Sheet and User's Manual, Version 1.10*. Cobham plc, 2018. URL: https://www.gaisler.com/doc/gr740/GR740-UM-DS-1-10.pdf.

[Gai19] Gaisler. *GR740, Data Sheet and User's Manual, Version 2.3*. Cobham plc, 2019. URL: https://www.gaisler.com/doc/gr740/GR740-UM-DS-2-3.pdf.

[GSW04] GSWS. *GALILEO SOFTWARE STANDARD*. Galileo Industries, 2004.

[IEC10a] IEC. *Functional safety of electric/electronic/programmable electronic safety-related systems - Part 1: General requirements*. INTERNATIONAL ELECTROTECHNICAL COMMISSION, 2010. URL: www.iec.ch.

[IEC10b] IEC. *Functional safety of electric/electronic/programmable electronic safety-related systems - Part 3: Software requirements*. INTERNATIONAL ELECTROTECHNICAL COMMISSION, 2010. URL: www.iec.ch.

[ISO10] ISO. *ISO/IEC 9899:201x, Programming languages - C*. ISO, 2010. URL: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1548.pdf.

[ISO11] ISO. *ISO 26262 Road vehicles – Functional safety*. International Organization for Standardization, 2011. URL: https://www.iso.org/standard/43464.html.

[MW10] Alistair Mavin and Philip Wilkinson. Big Ears (The Return of Easy Approach to Requirements Engineering). In *18th Requirements Engineering Conference*, 277–282. 11 2010. URL: https://www.researchgate.net/profile/Alistair_Mavin/publication/224195362_Big_Ears_The_Return_of_Easy_Approach_to_Requirements_Engineering/links/568ce39808ae197e426a075e/Big-Ears-The-Return-of-Easy-Approach-to-Requirements-Engineering.pdf, doi:10.1109/RE.2010.39.

[MWGU16] Alistair Mavin, Philip Wilkinson, Sarah Gregory, and Eero Uusitalo. Listens Learned (8 Lessons Learned Applying EARS). In *24th International Requirements Engineering Conference*. September 2016. URL: https://www.researchgate.net/profile/Alistair_Mavin/publication/308970788_Listens_Learned_8_Lessons_Learned_Applying_EARS/links/5ab0d42caca2721710fe5017/Listens-Learned-8-Lessons-Learned-Applying-EARS.pdf, doi:10.1109/RE.2016.38.

[MWHN09] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. Easy approach to requirements syntax (EARS). In *17th Requirements Engineering Conference*, 317–322. 10 2009. URL: https://www.researchgate.net/profile/Alistair_Mavin/publication/224079416_Easy_approach_to_requirements_syntax_EARS/links/568ce3bf08aeb488ea311990/Easy-approach-to-requirements-syntax-EARS.pdf, doi:10.1109/RE.2009.9.

[Mot88] Motorola. *Real Time Executive Interface Definition*. Motorola Inc., Microcomputer Division and Software Components Group, Inc., January 1988. DRAFT 2.1. URL: https://ftp.rtems.org/pub/rtems/publications/RTEID-ORKID/RTEID-2.1/RTEID-2_1.pdf.

[S20511a] SC-205. *Formal Methods Supplement to DO-178C and DO-278A*. RTCA, Inc, 2011. URL: www.rtca.org.

[S20511b] SC-205. *Software Considerations in Airbone Systems and Equipment Certification*. RTCA, Inc, 2011. URL: www.rtca.org.

[S20511c] SC-205. *Software Tool Qualification Considerations*. RTCA, Inc, 2011. URL: www.rtca.org.

[SPA91] SPARC. *The SPARC Architecture Manual, Version 8*. SPARC International, Inc., 1991. URL: https://www.gaisler.com/doc/sparcv8.pdf.

[SPA96] SPARC. *System V Application Binary Interface, SPARC Processor Supplement*. The Santa Cruz Operation, Inc. and AT&T, 3 edition, 1996. URL: https://www.gaisler.com/doc/sparc-abi.pdf.

[SPA02] SPARC. *SPARC Assembly Language Reference Manual*. Sun Microsystems, Inc., 2002. URL: https://docs.oracle.com/cd/E18752_01/pdf/816-1681.pdf.

[VIT90] VITA. *Open Real-Time Kernel Interface Definition*. VITA, the VMEbus International Trade Association, August 1990. Draft 2.1. URL: https://ftp.rtems.org/pub/rtems/publications/RTEID-ORKID/ORKID-2.1/ORKID-2_1.pdf.

[WB13] Karl Wiegers and Joy Beatty. *Software Requirements*. Microsoft Press, 3 edition, 2013. ISBN 0735679665, 9780735679665.