



BSI Standards Publication

Space engineering — Software engineering handbook

National foreword

This Published Document is the UK implementation of CEN/TR 17603-40:2022.

The UK participation in its preparation was entrusted to Technical Committee ACE/68, Space systems and operations.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

This publication is not to be regarded as a British Standard.

© The British Standards Institution 2022
Published by BSI Standards Limited 2022

ISBN 978 0 539 20383 7

ICS 35.080; 49.140

Compliance with a Published Document cannot confer immunity from legal obligations.

This Published Document was published under the authority of the Standards Policy and Strategy Committee on 31 August 2022.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

English version

Space engineering - Software engineering handbook

Ingénierie spatiale - Guide d'ingénierie logiciel

Raumfahrttechnik - Handbuch zur
Softwareentwicklung

This Technical Report was approved by CEN on 20 April 2022. It has been drawn up by the Technical Committee CEN/CLC/JTC 5.

CEN and CENELEC members are the national standards bodies and national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.



CEN-CENELEC Management Centre:
Rue de la Science 23, B-1040 Brussels

Table of contents

European Foreword	6
Introduction	7
1 Scope	8
2 References	10
3 Terms, definitions and abbreviated terms	12
3.1 Terms from other documents	12
3.2 Terms specific to the present document	12
3.3 Abbreviated terms	12
4 Introduction to space software	15
4.1 Getting started	15
4.1.1 Space projects	15
4.1.2 Space standards: The ECSS System	15
4.1.3 Key characteristics of the ECSS System	16
4.1.4 Establishing ECSS Standards for a space project	16
4.1.5 Software / ECSS Standards relevant for Software	17
4.1.6 Why are standards a MUST for the software development process?	19
4.1.7 Executing a space software project	20
4.1.8 Disciplines in Space Software Projects	21
4.2 Getting compliant	22
4.2.1 The ECSS-E-ST-40C roles	22
4.2.2 Compliance with the ECSS-E-ST-40C	26
4.2.3 Characterization of space software leading to various interpretations/applications of the standard	28
4.2.4 Software criticality categories	31
4.2.5 Tailoring	33
4.2.6 Contractual and Organizational Special Arrangements	35
5 Guidelines	41
5.1 Introduction	41
5.2 Software related system requirement process	41
5.2.1 Overview	41
5.2.2 Software related system requirements analysis	44
5.2.3 Software related system verification	45
5.2.4 Software related system integration and control	46
5.2.5 System requirement review	53
5.3 Software management process	54
5.3.1 Overview	54
5.3.2 Software life cycle management	54
5.3.3 Software project and technical reviews	55
5.3.4 Software project reviews description	55
5.3.5 Software technical reviews description	56
5.3.6 Review phasing	63
5.3.7 Interface management	64
5.3.8 Technical budget and margin management	65
5.3.9 Compliance to this Standard	65
5.4 Software requirements and architecture engineering process	65
5.4.1 Overview	65
5.4.2 Software requirement analysis	65

5.4.3	Software architectural design	67
5.4.4	Conducting a preliminary design review	76
5.5	Software design and implementation engineering process	76
5.5.1	Overview	76
5.5.2	Design of software items	76
5.5.3	Coding and testing	78
5.5.4	Integration	81
5.6	Software validation process	83
5.6.1	Overview	83
5.6.2	Validation process implementation	84
5.6.3	Validation activities with respect to the technical specification	91
5.6.4	Validation activities with respect to the requirement baseline	91
5.7	Software delivery and acceptance process	93
5.7.1	Overview	93
5.7.2	Software delivery and installation	93
5.7.3	Software acceptance	94
5.8	Software verification process	97
5.8.1	Overview	97
5.8.2	Verification process implementation	97
5.8.3	Verification activities	100
5.9	Software operation process	104
5.9.1	Overview	104
5.9.2	Process implementation	107
5.9.3	Operational testing	107
5.9.4	Software operation support	107
5.9.5	User support	107
5.10	Software maintenance process	108
5.10.1	Overview	108
5.10.2	Process implementation	108
5.10.3	Problem and modification analysis	115
5.10.4	Modification implementation	115
5.10.5	Conducting maintenance review	115
5.10.6	Software migration	115
5.10.7	Software retirement	115
6	Selected topics	116
6.1	Use Cases and Scenarios	116
6.1.1	Relation to the Standard	116
6.1.2	Introduction to use cases	116
6.1.3	Identification of use cases	117
6.1.4	Formalization of each use case	117
6.1.5	Definition and guidelines	119
6.2	Life cycle	120
6.2.1	Relation to the Standard	120
6.2.2	Introduction	120
6.2.3	Existing life-cycle models	122
6.2.4	Choosing a Software life-cycle	131
6.3	Model based Engineering	134
6.3.1	Relation to the Standard	134
6.3.2	Definition and guidelines	135
6.4	Testing Methods and Techniques	138
6.4.1	Relation to the Standard	138
6.4.2	Introduction	138
6.4.3	Definitions	138
6.4.4	Test objectives	138
6.4.5	Testing strategies and approaches	141
6.4.6	Real Time Testing	145
6.5	Autocode	146
6.5.1	Relation to the Standard	146

6.5.2	Introduction	146
6.5.3	Subsystem and software relationship around autocode.....	147
6.5.4	From subsystem model to autocoded model.....	150
7	Real-time software.....	152
7.1	Relation to the Standard	152
7.2	Software technical budget and margin philosophy definition	152
7.2.1	Introduction	152
7.2.2	Load and real-time	153
7.2.3	Memory capacity.....	156
7.2.4	Numerical Accuracy.....	156
7.2.5	Interface timing budget	156
7.3	Technical budget and margins computation.....	157
7.3.1	Load and real-time	157
7.3.2	Memory margins	158
7.3.3	Numerical accuracy budget management	158
7.3.4	Interface timing budget management	159
7.4	Selection of a computational model for real-time software	159
7.4.1	Introduction	159
7.4.2	Recommended Terminology	160
7.4.3	Computational model.....	164
7.5	Schedulability analysis for real-time software.....	169
7.5.1	Overview	169
7.5.2	Schedulability Analysis	169
Annex A	Documentation Requirement List.....	179
Annex B	Generic Techniques.....	183
B.1	Formal Methods.....	183
B.2	Functional Decomposition and Structured Analysis	184
B.2.1	Introduction	184
B.2.2	Data flow	185
B.2.3	Control Flow.....	186
B.3	Object-Oriented Analysis (OOA)	187
B.4	Architecture and Design.....	187
B.5	Data Description Languages.....	188
B.6	State machine modelling languages	189
B.7	ITIL ®	190
Annex C (normative)	"Software Maintenance Plan (SMP) – DRD"	192
C.1	DRD identification.....	192
C.1.1	Requirement identification and source document	192
C.1.2	Purpose and objective	192
C.2	Expected response.....	192
C.2.1	Scope and content.....	192
Figures		
Figure 4-1:	ECSS relations for discipline software	17
Figure 4-2:	Role relationships.....	26
Figure 4-3 :	Delivery of warranty and support between companies.....	36
Figure 4-4 :	Closely Coupled Build and Service Support Contract	36
Figure 5-1:	System database.....	48
Figure 5-2:	Constraints between life cycles	50

Figure 5-3: Software requirement reviews	53
Figure 5-4: Phasing between system reviews and flight software reviews	64
Figure 5-5: Phasing between ground segment reviews and ground software reviews	64
Figure 5-6 : Reuse in case of reference architecture	75
Figure 5-7 : Example ITIL processes	106
Figure 6-1: The autocoding process	149
Figure 7-1: Mitigation of theoretical worst case with operational scenarios.	154
Figure 7-2: An example of a complete task table with all timing figures	175
Figure 7-3 : Maintenance Cycle.....	191

Tables

Table 5-1: Possible review setup.....	59
Table 5-2: Example of review objectives and their applicability to each version.....	60
Table 6-1: Choosing a Software life-cycle	131
Table 6-2: Relation between the testing objectives and the testing strategies	145
Table 7-1: Schedulability Analysis Checklists	176

European Foreword

This document (CEN/TR 17603-40:2022) has been prepared by Technical Committee CEN/CLC/JTC 5 “Space”, the secretariat of which is held by DIN.

It is highlighted that this technical report does not contain any requirement but only collection of data or descriptions and guidelines about how to organize and perform the work in support of EN 16603-40.

This Technical report (CEN/TR 17603-40:2022) originates from ECSS-E-HB-40A.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN [and/or CENELEC] shall not be held responsible for identifying any or all such patent rights.

This document has been prepared under a mandate given to CEN by the European Commission and the European Free Trade Association.

This document has been developed to cover specifically space systems and has therefore precedence over any TR covering the same scope but with a wider domain of applicability (e.g.: aerospace).

Introduction

The ECSS-E-ST-40C Standard defines the principles and requirements applicable to space software engineering. This ECSS-E-HB-40A handbook provides guidance on the use of the ECSS-E-ST-40C.

History of the ECSS-E-40

At the beginning was ESA PSS-05. It was a prescriptive list of requirements ordered all along a waterfall lifecycle. It was necessary to improve it because it was too prescriptive and not flexible enough to apply new technologies such as UML.

ECSS was created in the 90's, and ECSS-E-40A was published in 1999. It was derived from ISO 12207, which is a process model. A process model proposes a set of abstract processes, and the software developer defines its own lifecycle that enters and leaves and re-enters the various processes. The process model was very abstract, with sort of meta-processes that were "invoking" other sub-processes. The new ECSS-E-40A was not prescriptive and very flexible to any kind of lifecycle.

ECSS-E-40A was improved because it was too abstract and it was not clear what had to be done. ECSS-E-40B was worked out in order to downsize the abstraction. The invocation was simplified, some processes were grouped. ECSS-E-40B was sent for public review.

The public review recommended improving further the pragmatic aspects of the standard. Therefore another ECSS-E-40B version was produced where the process model was streamlined.

At a workshop in 2004 on the use of ECSS-E-40B, it was recognised that some of the requirements left room for interpretation, which in turn lead to many discussions in the project reviews (especially when they were overlooked during the Software Development Plan review). Therefore the version C of ECSS-E-ST-40 was produced to improve the usability of the standard, refining and streamlining the open requirements, and somehow coming closer to the ESA PSS-05 spirit.

1

Scope

This Handbook provides advice, interpretations, elaborations and software engineering best practices for the implementation of the requirements specified in ECSS-E-ST-40C. The handbook is intended to be applicable to both flight and ground. It has been produced to complement the ECSS-E-ST-40C Standard, in the area where space project experience has reported issues related to the applicability, the interpretation or the feasibility of the Standard. It should be read to clarify the spirit of the Standard, the intention of the authors or the industrial best practices when applying the Standard to a space project.

The Handbook is not a software engineering book addressing the technical description and respective merits of software engineering methods and tools.

ECSS-E-HB-40A covers, in particular, the following:

- a. In section 4.1, the description of the context in which the software engineering standard operates, together with the explanation of the importance of following standards to get proper engineering.
- b. In section 4.2, elaboration on key concepts that are essential to get compliance with the Standard, such as the roles, the software characteristics, the criticality, the tailoring and the contractual aspects.
- c. In section 5, following the table of content of the ECSS-E-ST-40C Standard, discussion on the topics addressed in the Standard, with the view of addressing the issues that have been reported in projects about the interpretation, the application or the feasibility of the requirements. This includes in particular:
 1. Requirement engineering and the relationship between system and software
 2. Implementation of the requirements of ECSS-E-ST-40 when different life-cycle paradigms are applied (e.g., waterfall, incremental, evolutionary, agile) and at different levels of the Customer-Supplier Network
 3. Architecture, design and implementation, including real-time aspects
 4. Unit and integration testing considerations, testing coverage
 5. Validation and acceptance, including software validation facility and ISVV implementation
 6. Verification techniques, requirements and plan
 7. Software operation and maintenance considerations.
- d. In section 6 and 7, more information about selected topics addressed in section 5 such as (in section 6) use cases, life cycle, model based engineering, testing, automatic code generation, and (in section 7) technical budget and margin, computational model and schedule analysis.

NOTE In order to improve the readability of the Handbook, the following logic has been selected for sections 5, 6, and 7:

- section 5 follows the table of content of ECSS-E-ST-40C at least up to level 3 and generally up to level 4. For each sub clause of ECSS-E-ST-40C:
 - + either information is given fully in section 5,
 - + or there is a pointer into section 6 or section 7
 - + or the paragraph has been left intentionally empty for consistency with the ECSS-E-ST-40C table of content, in this case, only “ – ” is mentioned.
 - section 6 expands selected parts of section 5 when:
 - + either the volume of information was considered too large to stay in section 5,
 - + or the topic is addressed in several places of section 5

In any case, there is a pointer from section 5 to section 6, and section 6 mentions the various places in ECSS-E-ST-40C where the topic is addressed.
 - section 7 follows the same principles as section 6, but gathers the topics related to margins and to real-time.
- e. In Annex A, as a complement to the ECSS-E-ST-40C Annex A called Document Requirement List [DRL], the documents expected at the Technical Reviews such as SWRR, DDR, TRR and TRB.
- f. In Annex B, software engineering techniques appropriate for the implementation of specific ECSS-E-ST-40C clauses and their selection criteria, covering most of the software lifecycle.
- g. In Annex C, an example of the Document Requirement Definition of the Software Maintenance Plan.

2 References

EN Reference	Reference in text	Title
EN 16601-00-01	ECSS-S-ST-00-01C	ECSS system - Glossary of terms
EN 16603-10	ECSS-E-ST-10C	Space engineering – System engineering general requirements
EN 16603-40	ECSS-E-ST-40C	Space engineering - Software
EN 16603-70-01	ECSS-E-ST-70-01C	Space engineering - On board control procedures
	ECSS-E-TM-40-07A	Space engineering - Simulation modelling platform
EN 16601-40	ECSS-M-ST-40C	Space project management - Configuration and information management
EN 16602-80	ECSS-Q-ST-80C	Space product assurance - Software product assurance
	Def Stan 00-54 (March 1999)	Requirements for Safety Related Electronic Hardware in Defence Equipment" Part No: 1: Requirements: Issue 1
	ESA ISVV Guide	ESA Guide for Independent Software Verification and Validation, Version 2.0, December 29, 2008
	IEEE 610.12-1990 (Jan 1990)	IEEE Standard Glossary of Software Engineering Terminology
	IEEE 754-1985 (Aug 1985)	IEEE Standard for Binary Floating-Point Arithmetic
	ISO/IEC 12207:2008	Systems and software engineering – Software life cycle processes
	NASA Study on Flight Software Complexity (May 2009)	Final Report. NASA Study on Flight Software Complexity. Commissioned by the NASA Office of Chief Engineer. Technical Excellence Program Adam West, Program Manager
	RNC-CNES-E-HB-70-501 (September 2008)	Space engineering monitoring and control specification guide, Version 2, September 16, 2008
	NASA Lessons Learned	http://llis.nasa.gov/llis/search/home.jsp
	NASA System Engineering Handbook	NASA/SP-2007-6105 Rev1 December 2007

EN Reference	Reference in text	Title
	ESA PSS-05-0 Issue 2 (February 1991)	ESA software engineering standards Issue 2
	Flight Computer Initialisation Sequence	Space Avionics Open Interface initiative document: SAVOIR/12-007/FT http://savoir.estec.esa.int

Terms, definitions and abbreviated terms

3.1 Terms from other documents

For the purpose of this document, the terms and definitions from ECSS-S-ST-00-01C and ECSS-E-ST-40C apply.

3.2 Terms specific to the present document

3.2.1 software product:

set of computer programs, procedures, documentation and their associated data

3.2.2 acceptance test

test of a system or functional unit usually performed by the customer on his premises after installation, with the participation of the supplier to ensure that the contractual requirements are met

[adapted from ISO/IEC 2382--20:1990]

NOTE ECSS-E-ST-40C relies on ECSS-E-ST-00-01 for the definition of acceptance test and software product (and consequently of its synonyms “software” and “software item”). However, these two terms have disappeared in its last revision C. The definitions of ECSS-E-ST-40B (and therefore ECSS-Q-ST-80B) are restored here.

3.3 Abbreviated terms

For the purpose of this document, the abbreviated terms from ECSS-S-ST-00-01C and ECSS-E-ST-40C apply.

Abbreviation	Meaning
AR	acceptance review
	NOTE The term SW-AR can be used for clarity to denote ARs that solely involve software products.
CDR	critical design review
	NOTE The term SW-CDR can be used for clarity to denote CDRs that solely involve software products.
CMMI	capability maturity model integration
COTS	commercial-off-the-shelf

Abbreviation	Meaning
CPU	central processing unit
DDF	design definition file
DDR	detailed design review
DJF	design justification file
DRD	document requirements definition
ECSS	European Cooperation for Space Standardization
eo	expected output
GS	ground segment
HMI	human machine interface
HSIA	hardware-software interaction analysis
HW	hardware
ICD	interface control document
INTRSA	international registration scheme for assessors
IRD	interface requirements document
ISO	International Organization for Standardization
ISV	independent software validation
ISVV	independent software verification and validation
MGT	management file
MF	maintenance file
MOTS	modified off-the-shelf
OBCP	on-board control procedure
OP	operational plan
ORR	operational readiness review
OTS	off-the-shelf
PAF	product assurance file
PDR	preliminary design review
	NOTE The term SW-PDR can be used for clarity to denote PDRs that solely involve software products.
PRR	preliminary requirement review
QR	qualification review
	NOTE The term SW-QR can be used for clarity to denote QRs that solely involve software products.
RB	requirements baseline
SCAMPI	standard CMMI appraisal method for process improvement
SDE	software development environment
SOS	software operation support

Abbreviation	Meaning
SPA	software product assurance
SPAMR	software product assurance milestone report
SPAP	software product assurance plan
SPR	software problem report
SRB	software review board
SRR	system requirements review
	NOTE The term SW-SRR can be used for clarity to denote SRRs that solely involve software products.
SW	software
SWE	software engineering
TRR	test readiness review
TS	technical specification

Introduction to space software

4.1 Getting started

4.1.1 Space projects

Space projects vary widely in purpose, size, complexity and availability of resources. Depending on the intended scenarios, the space projects will cover from small up to very complex developments which may consist of a space segment (e.g. manned spacecraft's like COLUMBUS, unmanned space vehicles like ATV or ARIANE rockets, all kinds of satellites, and payloads within a spacecraft or externally attached residing directly in space), of a launch service segment (e.g. ARIANE Launch complex at Kourou) and of a ground segment (e.g. operations and control centres as well as data evaluation sites).

The production of space systems calls for the cooperation of several organizations that share the common objective of providing a product that satisfies the customer's needs (performance within cost and schedule constraints). This setup requires a strong project management for all involved parties. To base business agreements (e.g. contracts) which define all kinds of applicable standards that are binding for the space project.

4.1.2 Space standards: The ECSS System

In order to enable all experts to "speak the same language" in a space project, a comprehensive set of coherent European Space Standards has been developed as a cooperative effort between the European space agencies and space industries under the heading of **European Cooperation for Space Standardization** (abbreviated: ECSS). This set of standards is addressing all essential aspects of the three major domains for the successful implementation of space programmes and projects and is covering all aspects of interest that might appear for the procurement of a generic space product.

Besides general objectives and standardization policies as in the top level documents entitled "ECSS System - Description, implementation and general requirements" (ECSS-S-ST-00C) and terminology "ECSS System - Glossary of terms" (ECSS-S-ST-00-01C) for a space project, the three major domains:

- Space project management,
- Space engineering, and
- Space product assurance

are addressed and their activities are described in a necessary level of detail for application / execution in the form of processes.

4.1.3 Key characteristics of the ECSS System

The ECSS System has four prominent characteristics, which are to provide:

- a. a comprehensive and coherent set of standards offering a complete and stable framework within which customers and suppliers at all levels can implement a project in an efficient and cost effective manner;
- b. a system of standards constructed in such a way that it can be applied throughout the life cycle of space projects and programmes;
- c. a system that can be utilized in activities ranging from small, individual, less complex projects to very large programmes comprising several projects involving many products and interfaces;
- d. a controlled method for tailoring the standards such that they can be applied selectively depending on the type, or phase of the project for which they are being used.

As a consequence, the ECSS System focuses primarily on what is required to comply with each standard, rather than how to achieve this. This approach provides the flexibility for different customers and suppliers to use established “in-house” procedures, or processes, to comply with these standards. The ECSS System also includes supporting standards, which identify specific procedures or processes, and Handbooks, which provide technical data and guidelines for procedures and processes. Handbooks are documents providing orientation, advice or recommendations on non-normative matters.

4.1.4 Establishing ECSS Standards for a space project

There is a 7-step process for the a) preparation and b) application of tailoring to establish the ECSS Standards and their requirements for a space project.

a) Preparatory activities:

- 1) Identification of project characteristics;
means to identify strategic and technical aspects of the space project.
- 2) Analysis of project characteristics and identification of risks;
enforces a close analytical look to significant cost, risk and technical drivers as well as critical issues and specific constraints. These are used to identify and evaluate inherent and induced risks.

b) Tailoring activities:

- 3) Selection of applicable ECSS Standards;
comprises to evaluate the ECSS Standards for relevance to the overall space project's needs. Those standards found to be relevant are identified as applicable standards for the implementation of the project.
- 4) Selection of applicable ECSS requirements;
Having established the list of applicable ECSS Standards for a project, the extent to which the requirements (contained within these standards) are made applicable is to be assessed against cost, schedule, and technical drivers, as well as against the identified risks and their mitigation strategies.
- 5) Addition of new requirements;
Where the applicable ECSS Standards do not include a specific requirement needed for the space project, a new requirement needs to be generated or adopted from an international standard.

6) Harmonization of applicable requirements;

Having completed the selection of applicable ECSS Standards and requirements and the addition of any new requirements, the coherence and consistency of the overall set of requirements to be applied to the space project is reviewed to eliminate the risk of conflict, duplication, or lack of necessary requirements.

7) Documenting ECSS Standards and requirements applicability

The process steps 1 to 6 intentionally do not define a specific format for generating and recording the results of the process. This approach is in line with the ECSS principle of identifying what is required, but not how this is to be achieved, and provides a degree of freedom for customers to select the most appropriate way to present the data within their particular environment.

One method of recording the applicability data in an efficient and structured manner is to consolidate it into an “**Applicable Requirements Matrix**”.

Further details on each of these activities are contained in document “**ECSS System - Description and implementation**” (ECSS-S-ST-00C).

4.1.5 Software / ECSS Standards relevant for Software

Within a space project “software” plays a key-role with respect to system functionality and operations. It is found at all levels, ranging from system functions down to firmware, including safety and mission critical functions. The Figure 4-1 identifies the main relations of the ECSS-E-ST-40C with the other ECSS Standards.

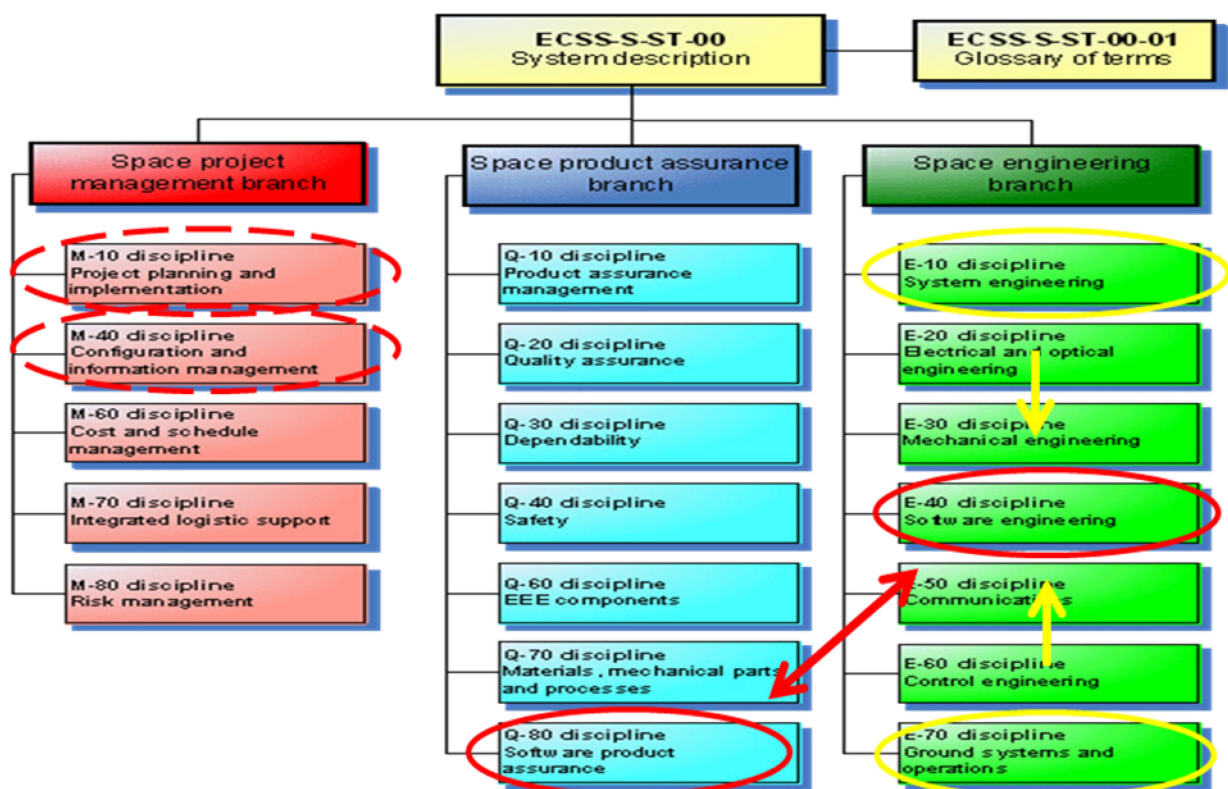


Figure 4-1: ECSS relations for discipline software

The ECSS-S-ST-00C document provides a general introduction to the ECSS System and to the use of the ECSS documents in all the space projects and gives therefore background information also for

software projects. The M standards for project management define phases, processes, reviews and rules for space project organization that apply also for software projects. In particular the **ECSS-M-ST-40C Rev1** defines the space configuration and information management requirements also for software projects. In addition, the **ECSS-M-ST-10C Rev1** requirements are tailored for software in the ECSS-S-ST-40 Software Management Process.

The context of the space software engineering activities is the overall Space System Engineering process. System engineering is defined as an interdisciplinary approach governing the total technical effort to transform requirements into a system solution. Its framework is defined by the "**Space engineering - System Engineering General Requirements**" (ECSS-E-ST-10C) **Standard** which is intended to apply to all space systems and products, at any level of the system decomposition, including hardware, software, procedures, man-in-the-loop, facilities and services.

The "**Space Engineering Software**" **Standard** (ECSS-E-ST-40C) covers the software development of the "Space system product software", i.e. all software that is part of a space system product tree and that is developed as part of a space project needs to apply this standard (for software deliverables and non-deliverables). It focuses on space software engineering processes requirements and their expected outputs. A special emphasis is put in the standard on the system-to-software relationship and on the verification and validation of software items.

This Standard is, to the extent made applicable by project business agreements, to be applied by all the elements of a space system, including the space segment, the launch service segment and the ground segment. It covers all aspects of space software engineering including requirements definition, design, production, verification and validation, transfer, operations and maintenance.

The Standard defines the scope of the space software engineering processes and its interfaces with management and product assurance, which are addressed in the Management (–M) and Product assurance (–Q) branches of the ECSS System, and explains how they apply in the software engineering processes. Together with the requirements found in the other branches of the ECSS Standards, this set provides a coherent and complete framework for software engineering in a space project. Software may be either a subsystem of a more complex system or it may also be an independent system.

In case of developing ground systems the "Space engineering - System Engineering General Requirements" (ECSS-E-ST-10C) Standard is extended by the "Space Engineering - Ground Systems and Operations" Standard (ECSS-E-ST-70C). Ground systems and operations are key elements of a space system and play an essential role in achieving mission success. Mission success is defined as the achievement of the target mission objectives as expressed in terms of the quantity, quality and availability of delivered mission products and services within a given cost envelope. Mission success requires successful completion of a long and complex process covering the definition, design, production, verification, validation, post-launch operations and post operational activities, involving both ground segment and space segment elements. It involves technical activities, as well as human and financial resources, and encompasses the full range of space engineering disciplines. Moreover it necessitates a close link with the design of the space segment in order to ensure proper compatibility between these elements of the complete space system. - Another specific link is made between ECSS-E-ST-70-01C and ECSS-S-ST-40C for On-Board Control Procedures (OBCP) / On-Board Application Programs (OBAP) development and validation processes.

The Space Engineering Software Standard is always complemented by the **Space Product Assurance Standard** (ECSS-Q-ST-80C), which specifies the product assurance aspects and is the entry point for ECSS-E-ST-40C into the Q-series of standards. All the requirements of the ECSS-E-ST-40C and ECSS-Q-ST-80C are applicable to the software development projects as a principle, unless criticality driven tailoring (or additional tailoring as per Annex R of ECSS-E-ST-40C for details) or other specific characteristics and constraints driven tailoring is applied. - Requirements for space configuration and information management are in **ECSS-M-ST-40C Rev 1**, which includes the software Document

Requirements Definition (DRD) for the software configuration file. However, the DRDs in the annexes are provided to help customers and suppliers with the project's setup. Together, both these standards either define or refer to the definition of all relevant processes for space software projects. ECSS-Q-ST-20C is the reference, through ECSS-Q-ST-80C, for the software acquisition process, and the software management process tailors ECSS-M-ST-10C for software.

4.1.6 Why are standards a **MUST** for the software development process?

Different groups of experts (disciplines) contribute to the development of a space project. A fundamental ECSS principle is the so-called "**customer-supplier**" relationship. All space project actors are either a customer or a supplier, or both. In its simplest form, a project can comprise one customer with just one supplier; however, most space projects comprise a number of hierarchical levels, where:

- a. the actor at the top level of the hierarchy is the top level customer,
- b. the actors at intermediate levels of the hierarchy are both supplier and customer,
- c. the actors at the lowest level of the hierarchy are suppliers only.

The customer is, in the general case, the procurer of two associated products: the hardware and the software for a system, subsystem, set, equipment or assembly. The concept of the "customer-supplier" relationship is applied recursively, i.e. the customer can be a supplier to a higher level in the space system. Examples for customers are therefore: European space agencies (ESA, CNES, DLR ...) and prime contractor space companies (e.g. for major space projects like COLUMBUS, ATV, ARIANE rockets ...) and for suppliers: companies or contractors from industries and from universities contributing by developing parts of the space project. **The fundamental principle of the "customer-supplier" relationship is assumed for all software developments.**

Within the space project, exchanges of products and services are governed by **business agreements**, used as a generic term throughout the ECSS Standards when referring to a legally binding agreement between two or more actors in the customer-supplier chain. These agreements include the terms and conditions agreed between the parties, the rules by which business is conducted, the actors' commitments and obligations for the provision of goods and services, the methods of acceptance and compensation, monetary, or otherwise. Business agreements serve as a framework prescribing the activities throughout the execution of work, and as a reference to verify compliance. Business agreements are recorded in a variety of forms, such as:

- a. Contracts
- b. Memoranda of understanding,
- c. Inter-governmental agreements,
- d. Inter-agency agreements,
- e. Partnerships,
- f. Bartering agreements, and
- g. Purchase orders.

The Space Engineering Software Standard is intended to help the customers to formulate their requirements and the suppliers to prepare their responses and to implement the work. Due to its nature it will help to get a complete view of the software from management point-of-view, to clarify the software activities, to focus the software effort according to the space project's context, and to verify the completeness of the intended statement of work. Furthermore the customer will be able to evaluate different proposals by checking the bidder's detailed compliance with the requirements.

The Space Engineering Software Standard may be adapted, i.e. **tailored** for the specific characteristics and constraints of a space project in conformance with ECSS-S-ST-00C. There are several drivers for tailoring, such as dependability and safety aspects, software development constraints, product quality objectives and business objectives. Tailoring for dependability and safety aspects is based on the selection of requirements related to the verification, validation and levels of proof demanded by the criticality of the software. So a number of factors have more or less influence on the tailoring: project characteristics, costs for the development and the operations & maintenance phase, number and skills of people required to develop, operate and maintain the software, criticality and complexity of the software, risk assessments. The type of software development (e.g. database or real-time) and the target system (e.g. embedded processor, host system, programmable device, or application specific integrated circuits) needs also to be taken into account. The tailoring of ECSS standards has direct influences to the business agreement of the space project (due to the binding clauses).

4.1.7 Executing a space software project

4.1.7.1 Overview

After successful negotiations a business agreement will be awarded from a customer to a supplier. The supplier then needs to implement his specific in-house organization for the space software project by assigning appropriate resources for the development of the "Space system product software" in the Project Management, Engineering and Product Assurance.

4.1.7.2 Lessons learned

Lessons learned show that spending the right level of effort already in the Development Phase will pay off at the end, e.g. in the mid- / long-term operations. If this concept is not implemented adequately, the efforts to be spent for corrective actions will be much higher when a problem is detected in a later phase, and are thereby becoming less cost-effective.

4.1.7.3 Project Management

The space software project is managed as a set of processes and will involve the customer (representatives) at main interaction points. These points are called: **reviews**. Their purpose is to synchronize software engineering processes.

Software management consists of supervising and monitoring the development of the software. It ensures that the correct procedures are followed, that the software is delivered on schedule and to budget, and that the final products are of the expected quality, e.g. they are correct, safe, efficient and easy to maintain or modify as required.

The main activities are the production of the **software plans** by defining and instantiating the particular implementation of the applicable software standards in the space software project. The plans include the life cycle description, activities description, milestones and outputs, the techniques, methods and tools to be used, and the risks identification (*not exhaustive list*).

4.1.7.4 Reviews

The reviews relevant to the software engineering processes are defined in detail in the "**Space Project Management - Project Planning and Implementation**" Standard (ECSS-M-ST-10C Rev1) as:

- a. System Requirements Review (SRR)
- b. Preliminary Design Review (PDR)
- c. Critical Design Review (CDR)
- d. Qualification Review (QR)

- e. Acceptance Review (AR)
- f. Operational Readiness Review (ORR).

The Space Engineering Software Standard offers in addition the possibility to anticipate the PDR in one *Software Requirements Review (SWRR)* or more, and the CDR into one *Detailed Design Review (DDR)* or more. The reviews occur at different levels in the customer–supplier hierarchy.

When the software development is included in a complete space project, the software engineering processes have also a relationship with the project phases (0, A, B, C, D, E, F) as the system level will induce driving constraints, such as availability of inputs (specifications, interface definitions), potential changes, delivery needs, schedule constraints, planning for system level reviews, etc. For software developments which are performed out of the context of a single space mission because they are developed for reuse, e.g. cross missions ground segment infrastructure, there is no particular link to phases of a space mission project. However, when the software is reused for a project, only mission specific elements are reviewed. The reused process is described in ECSS-Q-ST-80C.

The management process also includes the organization and handling of additional **joint reviews** (project and technical reviews for software), the definition of procedures for the management of the system interface, and the technical budget and margin management.

4.1.7.5 Processes

The notion of engineering processes is fundamental in the Space Engineering Software Standard (ECSS-E-ST-40C). These processes provide the means to describe the overall constraints and interfaces to the engineering processes at system level. At the same time, they provide the possibility to the supplier to implement the individual activities and tasks implied by the processes in accordance with a selected software life cycle.

The Standard is a process model, and does not prescribe a particular software life cycle. Although quite some space project reviews suggest a waterfall model, the software development plan can implement any life cycle, in particular by the use of the technical reviews whose formalism is setup less formal than the project reviews.

ECSS-E-ST-40C **Figure 4-2: Overview of the software life cycle processes** identifies the main concurrent software engineering processes and shows how and when they are synchronized by the customer-supplier reviews.

The relations between the software and the system reviews are addressed in chapter 5.2.5 in more detail.

4.1.8 Disciplines in Space Software Projects

4.1.8.1 Overview

The term “Discipline” is defined in ECSS-M-ST-10C, as “a specific area of expertise within a general subject”. The name of the discipline normally indicates the type of expertise in order to conduct subsequent software processes (for management, engineering and product assurance).

A discipline is associated with a set of related tasks to be carried out by an individual. A software expert of a discipline may carry out one, or several, development activities. Traditionally, these software experts correspond very closely to activities, so that, for example, an analyst prepares the software requirements, a designer prepares the architecture, and a programmer makes detailed design and coding. Nowadays one single expert may carry out all these three activities or even more, e.g. by using methodologies and toolsets.

The descriptions of the disciplines in the next (sub)sections are not meant to be exhaustive.

4.1.8.2 Software System Engineering (Co-engineering)

The Software System Engineering discipline contributes to the Space System Engineering process and covers tasks such as requirements engineering, system analyses and consolidation of requirements (by various analyses and trade-offs), preparing functional and physical architectures, system design and configurations, and defining (contractual) interfaces, system verification (to be conformant to requirements) and finally system engineering integration and control (supervising the construction of the computer programmes to meet all pre-conditions precisely). Due to the combination and the complementary work this process is regarded as "co-engineering".

4.1.8.3 Software Design and Implementation

In the Software Design and Implementation the detailed structure of the software functions is thought out. The software ultimately consists of components such as programs, processes and procedures, organized to provide functions to the space system. Designs very often consist of hierarchies of components. The software design and implementation include the architectural and detailed design, its documentation (in DDF and code) and justification (in DJF), and the coding, unit testing and integration testing of the components. In software integration and testing, software components, assemblies, and systems are fitted together and tested in a disciplined and rigorous way to ensure that the finished system is as nearly correct as possible.

4.1.8.4 Software Configuration Management

Software Configuration Management is the process of ensuring that the delivered software is complete and forms a set of consistent elements. This is achieved by restricted access to the set of deliverable items, such as programs and documents that constitute each issue or version of the system. A disciplined sequence of activities is enforced when a change is required to a configuration item.

4.1.8.5 Software Product Assurance

Software Product Assurance is a set of processes of ensuring that the software is fit for use and is built in accordance with applicable project requirements. This includes the software characteristics of safety, reliability and maintainability. Fitness for use is achieved by rigorous control (inspections, audits and reviews), by analyses, by witnessing tests, and by ensuring that correct procedures are followed.

4.2 Getting compliant

4.2.1 The ECSS-E-ST-40C roles

4.2.1.1 Customer – Supplier definition

In accordance with the ECSS theoretical concept, a fundamental principle of the ECSS-E-ST-40C standard is the 'customer-supplier' relationship, assumed for all software developments. The project organization is defined in ECSS-M-ST-10C. The customer is, in the general case, the procurer of two associated products: the hardware and the software for a system, subsystem, set, equipment or assembly. The concept of the "customer-supplier" relationship is applied recursively, i.e. the customer can be a supplier to a higher level in the space system. The software customer therefore has two important interfaces:

- a. the software customer interfaces with his software and hardware suppliers in order to adequately allocate to them functional and performance requirements, through the functional analysis.

- b. the software customer assumes a supplier role to interface in turn with his customer at the next higher level, ensuring that higher level system requirements are adequately taken into account.

The customer main activities are recalled hereafter as a synthesis of the ECSS-E-ST-40C standard, the supplier role is fully defined in the standard and it is not summarised in this introduction

The customer derives the functional, performance, interface and quality requirements for the software, based on system engineering principles and methods. Software items are defined in the system breakdown at different levels. . The customer's requirements are specified by this process, and they provide the starting point for the software engineering.

As the customer supplier relationship is a formal relationship (e.g. contractual when the customer supplier are not in the same organisation), the customer requirements baseline (RB) should be completed by a SOW , to precise the commitment between the two parties and the scope of the activities to be performed by the supplier such as phases to be covered (development, operation , maintenance) , Customer Furnished Item (CFI) , schedule constraints , etc. ...

The customer activities are defined in the ECSS-E-ST-40C clause 5.2

When the first level SW supplier subcontracts SW activities to a third party, only those requirements of the ECSS-E-ST-40C standard that pertains to the subcontracted activity are applicable to the third party. The first level SW supplier nevertheless remains responsible of the ECSS-E-ST-40C conformance towards the Customer.

If the customer considers that it is necessary, the customer has to tailor the ECSS standard in the scope of its project.

Non-compliances to the standard or the tailored standard are agreed before the kick-off meeting.

In additional to the formal reviews , the customer and supplier may agree on additional technical reviews to address specific concerns as anticipation of formal reviews, technical focus , etc. (ECSS-E-ST-40C clause 5.3.3)

The customer has in charge to perform the acceptance activities (section 5.7.3)

The customer specifies which part of operation and maintenance activities are to be covered by the SW supplier in the SOW. Even if not explicitly covered by the C/D phase SOW, maintenance process is considered by both customer and supplier in order to ensure that the prerequisites for maintenance phase have been correctly taken into account.

4.2.1.2 User

The ECSS-E-ST-40C introduces the term "user".

The term "user" in ECSS is used in section 5.9 with the introduction of the SOS entity in charge to support the user during the operations. The user is defined in this case as the "final user" of the SW product, and so it is not the customer in the development phase, in the major cases. The final SW user is the operator of the system in which the software is embedded

The ECSS-E-ST-40C mentions in others requirements the notion of use case, user need, etc., which are related to all SW users during the different system integration phases.

It is the Customer responsibility to collect all the user's needs, and in particular the Use Case definitions, which define early the operational constraints. The customer must include the verification of the user's needs in the acceptance process.

4.2.1.3 The SOS Entity

The SOS (Software Operation Support) entity is responsible to support the Software Operation process as introduced in ECSS-E-ST-40C §4.2.9 and as defined in ECSS-E-ST-40C §5.9

The SOS entity main objective is to keep the software operational for the final user. The SOS entity plays a role only after the system Acceptance Review, and acts as a relay between the final user and maintainers, in charge of testing the operational software behaviour in the operational context, tracking user requests, deploying new releases when necessary, ensuring administration activities.

4.2.1.4 Maintainer

The maintenance process contains the activities and tasks of the maintainer. The objective is to modify an existing software product while preserving its original properties. This process includes the migration and retirement of the software product. The process ends with the retirement of the software product.

The activities provided in section 5.9.1.2 are specific to the maintenance process; however, the process utilize other processes in ECSS-E-ST-40C and the term supplier is interpreted as maintainer in this case.

The maintainer manages the maintenance process applying the management process.

4.2.1.5 Operator

The term “operator” appears inside the ECSS-E-ST-40C only:

- a. in the clause 5.10.6.2 for the establishment of the migration plan in collaboration with the maintainer
- b. in the clause 5.10.7.1 for the establishment of the retirement plan in collaboration with the maintainer

NOTE The responsible for these two plans are the maintainer and not the operator.

The operator is used to express the final user not only of the software product but also of the system which include the software product.

4.2.1.6 Conductor

The term “conductor” is introduced for the ISVV activities. The conductor is the person or the entity that takes in charge the verification and validation tasks of the ISVV. It is the ISVV supplier.

4.2.1.7 Roles relationship

All the roles introduced in the ECSS-E-ST-40C standard are not strictly complementary, but can have some overlaps. This paragraph present the relationship between these roles , that are classified at different level (SW level , system level, user level) and which are introduced in perspective with the different program phases

According to the different space project phases, the role is dispatched on these phases as follows:

- a. during SW development phase until the SW AR :
 1. the SW customer is in charge to specify the requirement baseline , to procure the SW and to accept the SW product
 2. the SW supplier is in charge to develop, to validate the product. The SW supplier is in direct relationship with the SW customer
 3. the ISVV conductor is present when an ISVV is required. The ISVV conductor is a third party w.r.t. the software customer and the software supplier. It can be in direct relationship with either of them, provided that the independence and authority are guaranteed according to ECSS-E-ST-40C requirement 5.6.2.2.

4. the SW final user is in fact the system operator. As a major concern, the customer is in charge to make sure that the needs of the final user are properly captured in the requirements baseline, to avoid late introduction of operational constraints.
- b. between the SW AR and the system AR :
1. the SW customer is in charge to integrate the SW product inside the system. It will be supported in this phase by the SW maintainer for expertise , analysis and correction or modification if it is necessary
 2. the SW maintainer is in charge to support the SW customer. The SW maintainer performs expertise and analysis on SW customer request, to correct the SW product, to implement modification or evolution on SW customer request. In general the SW maintainer is the SW supplier of the previous phase, but this responsibility can be translated to a third party.
 3. ISVV activities can be required according to modification or evolution requests and their impact on the critical parts of the software.
 4. the SW final user is in fact the system operator. There is no change with regard to the previous phase.
- c. After the system AR :
1. the SW final user who is in fact the system operator, is in charge to operate the system
 2. The SOS entity role appears in this phase and covers in fact two roles: assure the administration and the maintenance of the operations infrastructure, and assure the role of system maintainer. The SOS entity is in direct relationship with the SW final user (system operator)
 3. The SW maintainer is in charge to support the SOS entity. The SW maintainer performs expertise and analysis on SOS entity request, to correct the SW product, to implement modification or evolution on SOS entity request.

The role relationship are synthesised in Figure 4-2.

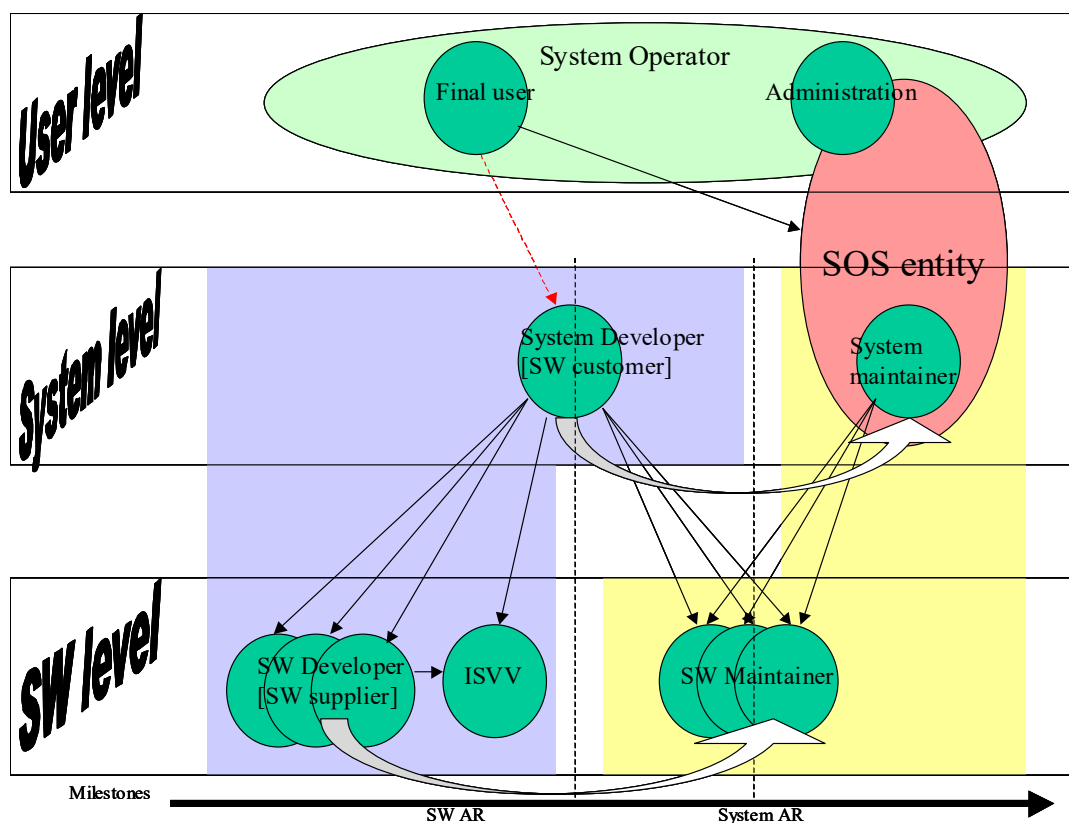


Figure 4-2: Role relationships

4.2.2 Compliance with the ECSS-E-ST-40C

The following text is extracted from ECSS-S-ST-00C:

The ECSS documents themselves do not have legal standing and they do not constitute business agreements: they are made applicable by invoking them in business agreements, most commonly in contracts. The applicability of standards and requirements is specified in the project requirements documents (PRDs), which are included in business agreements, which are agreed by the parties and binding them.

NOTE Description of PRD is provided in ECSS-M-ST-10, Clause 4.1.10.

The top-level customer's PRD forms the basis for the generation of all lower level customer PRDs. An integral part of a PRD, at any level, is the set of ECSS Standards tailored as necessary and documented in an "ECSS Applicability Requirements Matrix" (EARM), as described in clause 7.3.5.

A supplier, at any level, is responsible for demonstrating compliance with the project requirements contained in his customer's PRD, through, for example, the elaboration of a compliance matrix, and ultimately for supplying a conforming product. The compliance to the PRD is presented in an Implementation Documents (IDs), for example project plans (e.g. management, engineering and product assurance) and compliance matrix.

a. The supplier shall demonstrate compliance with the PRD requirements.

NOTE An "ECSS compliance matrix" (ECM) is a recommended method to document the demonstration of this compliance. The ECM is part of the project compliance matrix, addressing compliance to the applicable ECSS requirements (e.g. EARM).

b. The documentation identifying the compliance to ECSS requirements applicable to the project (e.g. the ECM) shall include following data:

1. The complete list of ECSS requirements applicable to the project (e.g. in the EARM).
2. For each requirement, the actual indication of compliance. When deviation is identified the justification is provided.

An example of template for an EARM for the requirements of ECSS-S-ST-00C is the following table:

Identifier	Requirement	Applicable (A/M/D/N)	Modified requirement

where:

applicable without change (A)

applicable with modification (M)

not applicable (D)

new generated requirements (N) including identification of origin if existing

According to ECSS-S-ST-00C, the customer provides an ECSS Applicability Requirements Matrix and the Supplier replies with an ECSS Compliance Matrix. It is important that the ECSS Compliance Matrix is done at the level of each requirements (as opposed to a global statement of compliance), in order to allow the Customer to detect early enough in the project the Non or Partial Compliance. It is essential to discuss them at the beginning of the project rather than discovering them at the end.

The attention of the Supplier is drawn to the risk taken by declaring systematically (global) compliance in the proposal. Although this might look like a way to increase the chances of being selected, it will likely backfire on the Supplier and the project when it will actually be discovered that this global compliance cannot be achieved.

A Partial Compliance needs to be detailed such that the Customer can assess the extent to which the objective of the ECSS requirement is covered, and whether a different way to achieve the objective might be acceptable.

A Non-Compliance needs also to be investigated in terms of feasibility and acceptability in the scope of the project.

There are historical records of PC/NC that ended up being acknowledged in an update of the ECSS-E-ST-40C Standard, such as the structural coverage of code achieved only by unit tests (version B) or through all the test campaign (version C, Note in 5.8.3.5b) or delivery of detailed design, not as a document (version B), but as an electronic file (version C, Ye in Annex R; accounting of models in 5.3.2.4).

When a Supplier has his own internal software engineering standard (e.g. as part of a company quality model), he can negotiate with the Customer the approval of a compliance matrix between the ECSS-E-ST-40C standard and the company standard, provided that all the information required by ECSS is actually delivered. This is in particular the objective of ECSS-E-ST-40C clause 5.3.9.2 (Documentation compliance) to allow a mapping of the ECSS DRDs on company documentation.

This is done today at each project level. However, it would be beneficial to put in place a customer/supplier company level agreement, although there is no today formalized organisational structure to do so.

4.2.3 Characterization of space software leading to various interpretations/applications of the standard

4.2.3.1 Introduction

Software is found in the space system at all levels, ranging from system functions down to firmware, including safety and mission critical functions.

ECSS-E-ST-40C is applicable for the development of any kind of software in the space domain, including software developed in R&D projects, in order to prove new technologies and novel concepts, or algorithms, prototype software that can be used in performing mission trade-offs, in designing the HCIs. or with the sole purpose of debugging an operational implementation.

Usually the space software is distinguished between flight and ground segment software.

In addition to the criticality levels, the following software characterisation is typically used for further tailoring of ECSS-E-ST-40C.

4.2.3.2 Flight software characterization

The overall aspects that may characterize flight software projects are:

- a. Nature of carrier: Spacecraft, Launcher, Micro Gravity, Space station,
- b. Nature of application: Command & Control, Data Processing, MMI, Real time, Security
- c. Nature of development process: From scratch, Reuse, Prototyping or R&D
- d. Criticality of Software: category A, B, C, D,
- e. Programmatic: Governance, schedule, Risk, Cost

Various application families have been identified in the space flight software domain:

- a. command and control software (on board of satellites, probes, spacecraft, but also command and control for payloads, control software of vehicles, hardware command and control, equipment, sensor, actuator software, up to the control software of robots and the central computers of space stations). They generally take care of control and data handling, and feature a high level of criticality. The focus is put on the integrity of data, which impacts on the design constraints and the technology used.
- b. data processing software (on board satellites, probes, spacecraft and robots). The software processes the payload data and can be less critical. The focus is put on the performance (e.g. image processing).
- c. micro-gravity facility and experiment software. It is generally a self-contained (software) system which includes command, control and data handling software for the check-out, error, FDIR, time, mode, TM/TC and resources management and local data processing. Its failure, by design, does not affect the whole space infrastructure.
- d. control MMI software is generally running on a laptop for developing and running the experiment procedures and the various sub systems controllers, for managing concurrency and resources access. Although operating in space, its characteristics are closed to the commercial desktop software, and similar technologies and criteria can be used.

Launcher software (flight control software and equipment software) has a short operational life combined with high availability and reliability. Furthermore it runs once and cannot be tested in the real environment.

Flight software includes low level software such as boot, initialisation, board support package, that often cannot be modified during operation.

Flight software runs in a hostile environment. A failure of the flight software that results in the non-acceptance of commands can lead to the loss of the spacecraft. Software maintenance is more difficult on-board as it is difficult to access and is usually required continuing operation. In case also of a remaining fault within flight software, investigations are difficult to conduct due to less observability than on ground. This leads to specific dependability requirements.

Furthermore during operation, the spacecraft visibility can be limited and the spacecraft can be hidden from the ground control (by the earth itself or by a planet). This limits the access time and can lead to specific requirements for the autonomy of the spacecraft.

The operational lifetime of a spacecraft is generally long (in the range from 2 to 15 years). This puts specific constraints on the maintenance of the flight software development environment, as in most of the cases the on board software is the only spacecraft component that can be modified during in-orbit life. This allows correcting malfunctions in the software, but also to circumvent HW problems or to adapt to not-predicted situations in-orbit. The software development environment needs also to be working during all the operational lifetime of the spacecraft, this may require specific measures such as hardware support provisions or emulation on new hardware.

4.2.3.3 Ground segment software characterization

The overall aspects that may characterize ground software projects are:

- a. Deployment environment: Dedicated systems, Cloud, Web-based system, Mobile & Embedded,
- b. Nature of application: Command & Control, Data Processing, MMI, Real time, Security
- c. Nature of development process: From scratch, Reuse, Prototyping or R&D
- d. Criticality of Software: category A, B, C, D,
- e. Programmatic: Governance, Schedule, Risk, Cost

The ground segment software is software that is part of the ground system and is therefore used for operating and exploiting the space segment. The ground segment software can be distributed across various subsystems depending on the ground segment architecture:

- a. Mission operations system, including (not extensive list) the Mission Control System, Flight Dynamics Systems, Mission Planning, Operational Simulators and Data Distribution Systems.
- b. Payload operations and data system;
- c. Ground station system;
- d. Ground communications system.

Ground segment software may be composed of critical, real-time, near real time (data driven payload data segment) and non-real-time (e.g. batch processing, mission planning, data processing and archiving) components. Some elements of the ground segment may stay operational even 5 to 10 years after the end of the satellite lifetime (e.g. the data archive systems), so it may have a long lifetime, including maintenance.

Many elements of ground segment software such as the Mission Control System and the operational simulator are often developed based on an existing, generic and mission independent infrastructure. Most projects build on the out-of-the-box functionality provided by the infrastructure and customise it for addressing mission specific needs. New software development projects (from scratch) are therefore quite unusual in the traditional ground segment domain. The heavy reuse of generic infrastructure in the ground segment, resolves many system engineering design aspects at a generic and mission

independent level (for a group of similar space missions with common characteristics). In fact the utilisation of the infrastructure software is not limited to reuse of software functionality but more importantly to re-use of a system-design at ground segment level. In other words the subject infrastructure can be seen in many cases as a reference ground segment system architecture, which defines the composing components of the ground segment software and establishes standard interfaces among them. Hence when developing the components of the ground segment for a particular mission (software systems such as the Mission Control System, Flight Dynamics software systems, operational simulator or the ground station software), the software development does not need to start from the system related software requirements engineering phase (the system context is established through the reuse of system design provided by the infrastructure).

The ground segment infrastructure is generic and covers the common requirements of many missions. For specific families of missions such as EO or deep-space missions, which share additional common requirements, special profiles of the infrastructure may be developed to increase the reuse even more. From this infrastructure perspective, many developments are not driven by the requirements of a single space mission and are not implemented as part of a particular space programme, but have their more or less independent and parallel lifecycle. These infrastructure developments are planned ahead of the missions (e.g. new versions of the mission control system or simulator infrastructure). Therefore some system requirements are not fully stable. Performance requirements, in particular, need to be forecasted or maximized;

The lifecycle of ground segment software elements is often more complicated than that of the flight software. It is not unusual that the different components of the ground infrastructure are developed by different Suppliers, maintained by yet other Suppliers and provided as Customer Furnished Items to the Suppliers who are responsible for the development of the mission specific software components (which again may be maintained by other Suppliers). The fact that more than one mission use at same time components of common infrastructure software, adds additional complexity to the governance of the infrastructure software lifecycle. To give an example the management of software defects on the infrastructure software can only be governed by configuration change boards, involving representatives of all software projects, reusing the subject software. These characteristics impact directly many aspects of software development life cycle, including the design, implementation, validation, maintenance and operation.

Components of ground software undergo an operational validation, which may be performed, depending on the schedule, before or after the individual software development project is finally accepted (AR), e.g. after the launch of the satellite or in the context of simulation campaigns;

The overall Ground Segment composes typically significantly larger number of complex systems in comparison to flight software (from the ground station to the mission control system to data distribution system and payload operations planning system, internet connectivity units such as routers and firewalls, server and PC operating systems, databases, ...). Each of these systems is typically deployed in a geographically different location and operated by different entities. The process of end-to-end validation of such a distributed system can therefore usually be achieved only at system level and scenario based.

The Ground Segment (GS) often relies on commercial software such as operating systems and COTS. More-over with emerging software development technologies such as web-based applications, Java EE, .NET, SMP-2 based simulator model developments or the principles of Service Oriented Architectures, the GS software becomes more and more developed based on **frameworks** related to these technologies. The validation of these frameworks and COTS is often not a realistic approach. In particular as the COTS and framework are often not available at source code level and limited information regarding their test coverage is available.

It is worth noting that it is not unusual for ground software components developed based on such frameworks, that a significant percentage of the overall functionality of the “system” is provided by the framework (this can reach in some cases 90%). The flight software is in contrary often a custom development fully available at source code level.

The development of parts of the ground segment infrastructure (e.g. routers and gateways) is not directly managed by the ground data system developer and the operating entity (e.g. network) the level of quality of these units can only be assessed at contractual level (service level agreements SLA) or via system testing.

The human factor is an important factor in Ground Segment software, since it is very often manually or semi-automatically operated. A typical characteristic of the Ground Segment software is the provision of Man-Machine-Interfaces. Hence validating the software against all possible combinations of human interactions with it through the Man-Machine-Interfaces (flow of all possible inputs) is often unfeasible.

The Ground Segment software Error control mechanisms are above the level of software only. Examples: critical commands must be sent twice. CRC codes ensure that commands are not corrupted. In case of major problems the spacecraft will enter in safe mode, alerts are verified by humans conducting flight operation procedures;

Most ground segment software problems impact in the worst case the availability of the system. Ground segment software can however usually be corrected and redeployed much easier than flight software. This is for instance often related to much easier access to the system for debugging and redeployment (no need for satellite pass, change of hardware components, extend the resources, e.g. memory, storage). If the maintainability is high (i.e. short time is required to correct failures), there are margins to increase availability of the software i.e. the probability that the software system is available.

Suitability of Ground Segment software for a specific project may be claimed based on product service history, provided that conditions for product service history application are complied with. Most software routines are taken from previous versions of the system. It is particularly the case for complex systems such as flight dynamic software, which may have been developed over the years and are validated operationally (at system level) for a number of missions but not at unit and integration test level and there may provide limited test coverage information.

Ground software is normally

- a. data driven (e.g. the behaviour depends on the specific TM/TC data being received/generated and the input entered by the user via the MMI);
- b. configuration driven (IP addresses and ports, database adapters, WS endpoints, all options existing in the TM/TC database definitions);
- c. human driven (with multiple users, multiple applications, multiple operational modes, etc.).

4.2.4 Software criticality categories

4.2.4.1 Software criticality analysis

The criticality of a software product is determined (ECSS-Q-ST-80C Annex D) at system level, based on the severity of the consequences of the system failures that the software can cause.

The software criticality category is the only criterion used to define a pre-tailoring of ECSS-E-ST-40C in its annex R.

Other characteristics of software, are not used to select the criticality for pre-tailoring. For example:

- a. functional requirements for classes of missions or
- b. non-functional requirements such as target platform (real-time or embedded vs. workstation)
- c. role in a system (flight, operation, testing)

The process leading to the determination of the software criticality is described in ECSS-Q-HB-80-03A. Based on software-level analyses, the individual components of the software product, may be classified at a lower criticality category, provided that no failure propagation is possible from lower-criticality components to higher-criticality components.

The consideration of typical software failure modes (see ECSS-Q-HB-80-03A section 6.2.2.2: functionality not performed, wrongly performed, performed with wrong timing (sporadic or persistent)) suggests that the most critical components are found in the ones allowing, for example:

- a. to start the software (boot, initialization)
- b. to command and observe the software
- c. to put the system in a safe state in case of failure
- d. to remotely modify the software.

or components whose design or role inside the architecture increases the risk of fault propagation (e.g. operating system, drivers, interrupt handlers, board support package)

To avoid failure propagation, techniques based on fault containment such as Time and Space Partitioning or On Board Control Procedure interpreter, can allow the coexistence of different criticality level components in the same software.

4.2.4.2 Critical software

Critical software is defined in both standards as software of criticality category A, B or C.

Therefore category D represents non-critical software.

Any software subject to ECSS-E-ST-40C is at least category D. Any further tailoring applied to the Category D pre-tailoring cannot be justified by criticality. It may be justified by other criteria, such as the risk that the project is prepared to take.

For software not directly involved in the operation of the system, but rather in its testing (such as simulators or software validation facilities), it is important that its criticality is also determined based on the indirect consequence of its failures on the system. In particular, it is assessed whether

- a. a failure of a simulator can lead to a wrong system requirement having a system consequence, or
- b. a software testing environment failure can lead to the non-detection of system non-compliance having a system consequence, or
- c. these failures can be detected by other means in the system life cycle

For software developed iteratively in several contracts, with the aim of increasing progressively its maturity from prototype to a reusable product (e.g. R&D), the tailoring of the software engineering standard in each iteration depends both on its criticality (the expected criticality of the function for which the software will eventually be used, when the foreseen maturity is achieved) and on its level of Technology Readiness.

For example, early prototypes could be developed according to Cat D tailoring, but need to be re-engineered afterwards at the right criticality level of their intended use.

4.2.5 Tailoring

The contract perimeter is defining the extent to which the ECSS-E-ST-40C is applicable, because some activities such as operation, maintenance, retirement, or ISVV (for criticality category A and B), may not be included in the contract. But this is more related to "applicability" of the Standard rather than to its "tailoring".

In the same spirit, if activities of ECSS-E-ST-40C are done by another role than the one specified, this will also lead to tailoring, for example when system/customer tasks are delegated to software/supplier. i.e. not all of clause 5.2 may be done by the Customer, or the Supplier may be responsible for the acceptance test plan, or the Supplier may not be responsible for the definition of the external interfaces.

The ECSS-E-ST-40C introduces in Annex R a pre-tailoring based on software criticality.

These pre-tailoring may be further tailored according to criteria different from criticality, mainly according to the level of risk which is taken by not performing given engineering activities, or doing them differently than what is specified in the Standard.

It should be considered that taking a risk on high criticality software may have consequences much more important than taking a risk on non-critical software.

On the other hand, activities not required for non-critical software (e.g. higher code coverage, ISVV) may be performed for other reasons, such as intended reuse in a critical context. For example: (non-exhaustive)

- a. merging the requirement baseline and the technical specification increases the risk of losing the customer or supplier standpoint, i.e. to miss a use case or to miss an implementation requirement.
- b. skipping joint reviews, increases the risk to discover late disagreements between customer and supplier on the product capability or quality, causing substantial reengineering
- c. not managing technical budgets and margins, increases the risk to discover unfeasibility late in the project
- d. not using design methods, increases the risk to develop weak architectures and inconsistent designs
- e. not defining interfaces, increases the risk of integration issues
- f. not documenting the detailed design, increases the risk to lose control on the software development such as capability to anticipate implementation errors, to debug, to integrate, to maintain, to master safety and dependability, etc.
- g. skipping unit tests, increases the risk to discover faults late in the process and to jeopardize the schedule.
- h. not rerunning the full validation tests on the last version of the software, increases the risk to leave faults in the product
- i. not performing full verification activities, increases the risk to affect the quality of the product
- j. not complying with DRDs content, increases the risk of having non-complete documentation, and of missing information for maintenance
- k. non-compliance with the DRDs structure, increases the effort of the reviewers

It is emphasized that the risk taken does not affect only the project management, but also the dependability and safety of the product, i.e. adequacy to its criticality level.

Each risk taken by tailoring out a software engineering requirement is analysed in the scope of the project, and its consequences are assessed to decide if it is acceptable or not.

Any further tailoring is therefore associated to a risk management, which is specific of each project. For this reason, it is not possible to propose other generic tailoring tables.

However, there are activities that can never be tailored out, e.g.:

- a. agreeing on a development approach
- b. specifying software and reviewing the specification
- c. producing, validating and accepting software
- d. managing the configuration

Examples of tailoring:

- a. Prototypes and study software projects may come with minimal requirements for documentation and validation. These types of application are typically not re-used after having demonstrated the concept or their feasibility. Only a small proportion of prototypes are later used either for integration into other software products or as stand-alone applications. In the case where it is desirable to re-use a prototype, its reusability needs to be evaluated and the appropriate re-engineering will be performed.
- b. The incremental development of a technology - following an improvement of its Technology readiness level (TRL) - could also lead to tailoring of the software engineering activities. The full list of activities that should be performed for a software "product" [TRL 6] will be performed incrementally along several contracts and development steps. This takes into account the foreseen criticality of the product in its intended operational environment. The tailoring addresses the extent of functions implemented in the software, the extent of its validation, verification and documentation, and the criticality level of each increment (see also 4.2.4 on software criticality).

NOTE Interpretation of the technology readiness levels for software is given in the chapter 7 of the ESA document "Guidelines for the use of TRLs in ESA programmes".

- c. For criticality category D software, the level of granularity of the detailed design, its depth, or in other words the concept of unit (raised to the level of function or even application), can be tuned to a level which takes into account both the needs in maintainability, or traceability, but also a realistic engineering approach related to the type of software (e.g. ground software). The internal interfaces [5.5.2.2] are defined at the same level of granularity, as well as the unit tests strategy.
- d. For criticality category D software, the level of definition of the computational model can be adapted to the type of software and the criticality of its real-time aspects.
- e. When ECSS-E-ST-40C requirements are tailored out, the associated Expected Output in the DRD is also tailored out. The DRDs propose a way to deliver the requested information, however 5.3.9.2 allows a different documentation structure, provided that all the needed information is there. This can be seen as a sort of tailoring of the DRDs.
- f. Also, some documentation can remain available in Supplier premises without delivery, or can even be waived by the Customer.

4.2.6 Contractual and Organizational Special Arrangements

4.2.6.1 Contractual and Organisational Aspects of Software Development

Software development projects are usually realised in the frame of a contract between the supplier and the customer. The ECSS-E-ST-40C standard governs many aspects of the overall software development lifecycle and impacts the contractual agreement between the supplier and customer accordingly.

There are however some aspects such as the organisational structure of the customer and the supplier as well as the legal aspects of the contract, which may equally impact specific areas of software development lifecycle. Examples are software delivery and acceptance modalities or the agreed warranty periods between the customer and the supplier, copyright and Intellectual Property Rights (IPR) agreements, delivery schedule with associated definition of inputs and outputs.

It is neither in the scope of the ECSS-E-ST-40C nor its intention to address all possible scenarios of such aspects. This section gives an overview of some known issues, which are complementary to the specifications of the ECSS-E-ST-40C standard, and are typically specified in form of customised requirements in the Statement of the Work (SoW), or as part of the contract between the supplier and the Customer.

None of the following points in the sections below are mandatory and for inclusion in the contract between the customer and supplier. The objective is to raise awareness of the typical concerns which are not governed by the ECSS-E-ST-40C standard. For each of the mentioned aspects the customer/supplier analyses the need, and reflects the result in the contractual agreement if applicable.

4.2.6.2 Warranty, User support and Maintenance

Several activities happen towards the end of the development phase: warranty, user support, maintenance.

Warranty is a pure contractual issue included in the development contract, and is addressed in 4.2.6.6.2. User support belongs to the operation process section 5.9. The Software Maintenance phase is covered by section 5.10. The three activities may be performed within different contractual scope, and therefore may be performed by different actors.

Warranty is done by the software Supplier; User support is done by the SOS entity; the Maintenance can be done by the software Supplier or a third party.

An overview of the relationship between the different participants of the maintenance and support phases for a flight, or smaller ground system is shown in Figure 4-3. The Software supplier will be contracted by the System Integrator, or System Prime to develop, test and deliver the software in accordance with the Software Development Plan.

In some cases the System Supplier and Software Supplier will be the same company. In this case the arrangement may be similar but software warranty and support will be supplied by the software team rather than a separate company with a separate contract. For spacecraft there may not be system or software support, as authority may be transferred to a third party entity belonging to the operations organisation.

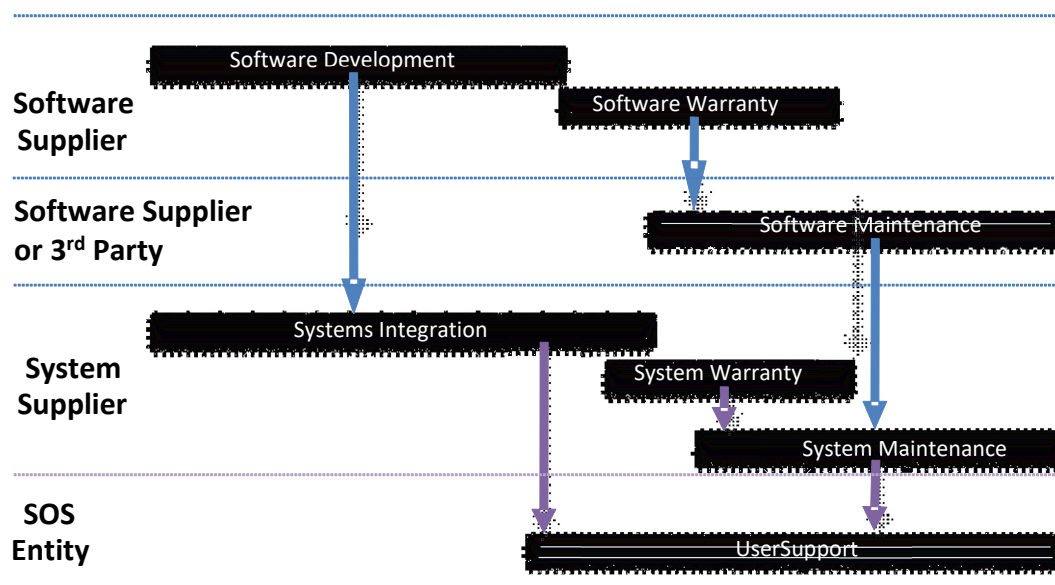


Figure 4-3 : Delivery of warranty and support between companies

Commonly now, larger ground system projects are being procured with the build and user support contracts linked very closely together (see Figure 4-4). This means that the contractor is asked to build a system, then maintain it and, in some cases, operate the system (including SOS entity role). In this case the service contract may imply that a team of people are employed during normal working hours, or on a 24/7 basis.

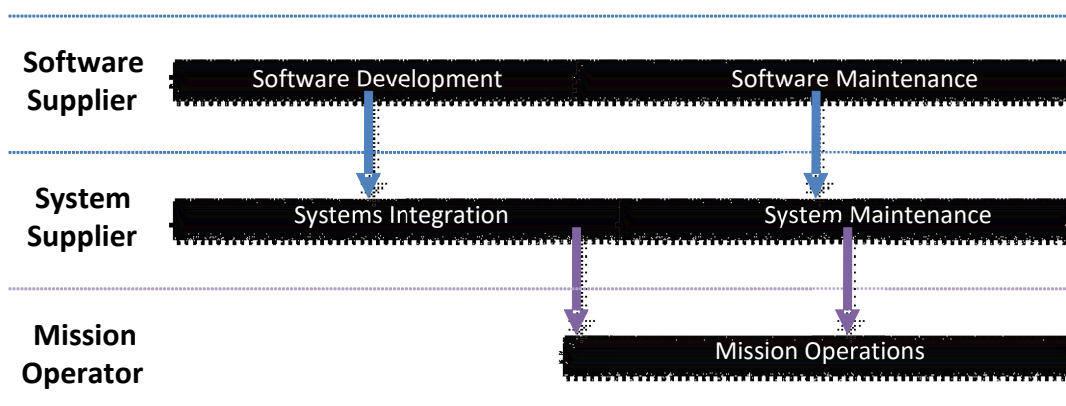


Figure 4-4 : Closely Coupled Build and Service Support Contract

For all types of contract it is usual to provision for the fix of defects, incidents or problems within an agreed period. Usually this period is different for the different criticality of defects. The contract may specify a Service Level Agreement (SLA) that defines the response time and couples them to contractual penalties or incentives.

4.2.6.3 Software Delivery Modalities

The **language**, **format** and **media** on which software and documentation is delivered is subject to agreement between the Customer and the Supplier, hence often specified through dedicated requirements in the Contractual agreement (e.g. often in the Statement of Work). Also the naming convention for the documents as well as the structure of the files and folders for software deliveries may be specified according to the Customers' requirements.

The Following requirements are provided as an example for such complementary specifications:

DELI-010	All the deliverables shall be written in British English.
DELI-020	All the deliverables documents shall be provided in Microsoft Word format in the version in use at the Agency and in PDF format. At the time of writing, the Agency uses Microsoft Office Suite 2003. . The latest version to be used shall be specified at KoM.
DELI-030	All the deliverables documents shall be produced based on the official document templates [RD1].
DELI-040	The deliverables for which the Customer owns the IPR shall not bear any information identifying the Contractor's Company or staff who produced the deliverable (e.g. Contractor's logo).
DELI-050	All deliverables shall be delivered in their original source format. Comment: For example in MS Word format for a document or in the source format of the UML tool for a UML document.
DELI-060	All deliverables documents shall in addition be delivered in PDF format.
DELI-070	All deliverables are subject to Customer review and approval.
DELI-080	All Documents shall be delivered in two original paper versions.
DELI-090	All documents delivered to the Customer for approval shall be signed electronically by the Contractor technical responsible.
DELI-100	All deliverable documents shall be named according to the rules defined in the Configuration Management Plan.
DELI-110	The deliverables shall be delivered to the Customer on duly labelled media as defined by the Configuration Management Plan.
DELI-120	The organisation of the files and directories on the media shall be approved by the Customer Technical representative.
DELI-130	Deliveries shall be done on DVD.
DELI-140	All documents shall be electronically uploaded on the document management portal, specified by the Customer.

4.2.6.4 Software Development Environment

The Selection of the supporting tools for performing the tasks in all stages of the software development lifecycle could be subject to agreement between the Customer and Supplier, hence often specified through dedicated contractual statements in the Statement of the Work or in the RB.

In particular the following tasks are often performed in specialised tooling, for example (non-exhaustive list):

- a. Requirements Management
- b. Software modelling and design
- c. Software build and deployment management
- d. Software Configuration Management
- e. Software Testing, Verification and Validation
- f. Software Defect Management
- g. Software IPR and Asset Management

The application of ECSS-E-ST-40C mandates only the performance of the tasks as part of the Software Development Lifecycle activities. It does not mandate the usage of any tooling. However the utilisation of software development support tooling, facilitates appropriate execution of the corresponding tasks.

4.2.6.5 Software Development Methodologies

The ECSS-E-ST-40C does not mandate or exclude the adoption of a particular software development methodology, e.g. object oriented, service oriented, agile software development or adoption of model driven architecture paradigm.

The adoption of any software development methodology (and utilisation of related tooling) is therefore subject to contractual agreement between the Customer and the Supplier and is specified through dedicated requirements.

4.2.6.6 Software Acceptance Modalities and Warranty Period

4.2.6.6.1 Overview

The Acceptance testing period, also referred to as the check-out period starts with the delivery of the software by the Supplier to the Customer, and ends with the formal acceptance of the software by the Customer.

The duration of the acceptance period, i.e. the time granted to the Customer for performing its own validation of the delivered software is subject to contractual agreement and not specified by ECSS-E-ST-40C. It is therefore specified through dedicated requirements as part of the Contract (often in SoW).

Also the condition for acceptance is clearly specified as part of the contractual agreement between the Customer and Supplier.

The Customer may perform additional testing and validation at the top of the SVS specified by the Supplier during the acceptance period, and may report all identified software defects and non-compliances against the RB/TS to the Supplier. It is irrelevant how the Customer identifies software defects/non compliances, as long as they can be traced back to a clear non-compliance against the RB/TS. It is not subject to any contractual agreement between the Supplier and Contractor, which tests the Customer may execute during the acceptance period, since it's purely the Customers decision.

Since acceptances are often linked to payment milestones, the Customer and the Supplier may agree contractually on a stepwise acceptance approach, in which case the steps, their duration and the exact criteria for successful completion are contractually specified and agreed upon by both parties. This can for instance be realized through introduction of preliminary and final acceptance milestones.

In special cases, where the target operational environment of software is not easily accessible, e.g. in case of remote ground stations, the Customer and the Supplier may contractually arrange for additional acceptance milestones such as site and operational site acceptances.

Another case of additional contractually arranged acceptances is the so called Factory Acceptance, which is performed in the Supplier environment and aims at serving as a confident building step for the Customer before the software is delivered by the Supplier in the Customer's environment. (see section 5.7)

4.2.6.6.2 Warranty

The start and duration of the warranty is governed by contractual agreement between the Customer and the Supplier. It is also governed by the legal terms of the country/European/international law, applicable to the contract.

The following requirements are examples of contractual requirements for regulating the warranty start and duration:

The Contractor shall offer a rolling warranty starting with the first delivery and expiring 3 months after the Final Acceptance of the last delivery.

Comment: "Rolling warranty" means that, at any point in time inside the warranty period defined in this requirement, the warranty covers the whole system.

During the warranty period, the Contractor shall be responsible to investigate and fix all errors found in the system and deliver corresponding corrections.

The Contractor shall perform an End of Warranty delivery, which shall reflect the latest status of the System both in terms of software and in terms of documentation.

The original Warranty period, which is provisioned for in the contractual agreement between the supplier and customer, often ends before the end of operational usage of the software (e.g. end of the space mission).

Software maintenance is in such cases established via the contractual agreements (typically in form of service level agreements) between the customer and the supplier (which can be the same supplier who developed the software or a different party who provides only the maintenance) in accordance to the operational needs of the customer on the subject software. The difference between warranty and maintenance is sometimes a source of confusion. The nature of the work to be performed under warranty and maintenance is similar (software defect management and removal). See also 5.10.2.1.2 for more details.

4.2.6.7 Intellectual Property Rights management

The assignment of Intellectual Property Rights for the software and other software related artefacts, e.g. documents, build and test scripts, etc. is not governed by the ECSS-E-ST-40 and is subject to contractual agreement between the Customer and the Supplier within the applicable legal frame.

4.2.6.8 Other development constraints / Customer Furnished Items

Specific constraints may be imposed by the Customer, which are not governed by ECSS-S-ST-40C. They need to be defined in the contractual agreement between the Customer and the Supplier. Examples are subcontracting scheme, target cost, particular risk sharing dispositions, the mandated extent of reuse, use of existing Customer Furnished Items, use of reference architecture, protection of secure material, artefacts relating to export or government control (e.g. EAR/ITAR).

5

Guidelines

5.1 Introduction

This chapter includes guidelines for the application of ECSS-E-ST-40C to space projects.

For the convenience of the reader, the note of section 1 is repeated here:

- NOTE In order to improve the readability of the Handbook, the following logic has been selected for the sections 5, 6, and 7:
- section 5 follows the table of content of ECSS-E-ST-40C at least up to level 3 and generally up to level 4. For each sub clause of ECSS-E-ST-40C:
 - + either information is given fully in section 5,
 - + or there is a pointer into section 6 or section 7
 - + or the paragraph has been left intentionally empty for consistency with the ECSS-E-ST-40C table of content, in this case, only “ – ” is mentioned.
 - - section 6 expands selected parts of section 5 when:
 - + either the volume of information was considered too large to stay in section 5,
 - + or the topic is addressed in several places of section 5

In any case, there is a pointer from section 5 to section 6, and section 6 mentions the various places in ECSS-E-ST-40C where the topic is addressed.
 - section 7 follows the same principles as section 6, but gathers the topics related to margins and to real-time.

5.2 Software related system requirement process

5.2.1 Overview

5.2.1.1 Introduction

System-software level activities seem to be systematically underestimated or even misunderstood. Most software suppliers consider that these are not applicable to them, in particular in the case of avionics equipment or payloads. To illustrate the issue, some results are taken from a NASA study (*Final Report - NASA Study on Flight Software Complexity - Commissioned by the NASA Office of Chief Engineer, Technical Excellence Program, Adam West, Program Manager Editor: Daniel L. Dvorak, Systems and Software Division, Jet Propulsion Laboratory, California Institute of Technology*). The study analysed the flow of engineering activities related to flight software and impacting its complexity. It came with

recommendations, in particular on system engineering. Some of them are quoted below from the Executive Summary of the Final report:

- a. *"Engineers and scientists often do not realize the downstream complexity and cost-driving factors entailed by their local decisions. Overly stringent requirements and simplistic hardware interfaces can complicate software; flight software de-scoping decisions and ill-conceived autonomy can complicate operations; and a lack of consideration for testability can complicate verification efforts. It is therefore recommended to look at educational materials, such as a "complexity primer" and the addition of "complexity lessons" to NASA Lessons Learned.¹*
- b. Unsubstantiated requirements have caused unnecessary complexity in flight software, either because the requirement was unnecessary or overly stringent. Rationale statements have often been omitted or misused in spite of best practices that call for a rationale for every requirement. The NASA Systems Engineering Handbook² states that rationale is important, and it provides guidance on how to write a good rationale and check it. In situations where well-substantiated requirements entail significant software effort, software managers should proactively inform the project of the impact. In some cases this might stimulate new discussion to relax hard-to-achieve requirements.
- c. Engineering trade studies involving multiple stakeholders (flight, ground, hardware, software, testing, and operations) can reduce overall complexity, but we found that trade studies were often not done or only done superficially. Whether due to schedule pressure or unclear ownership, the result is lost opportunities to reduce complexity. Project managers need to understand the value of multi-disciplinary trade studies in reducing downstream complexity, and project engineers should raise complexity concerns as they become apparent."

5.2.1.2 System engineering

System engineering is an interdisciplinary approach governing the total technical effort to transform requirements into a system solution. The relevant framework is contained in the **"Space engineering - System Engineering General Requirements"** (ECSS-E-ST-10C) **Standard** describing the ECSS relations for the software discipline.

The system engineering activities of a project are conducted by an entity within the project team of a supplier. This entity is called "system engineering organization". The system engineering process consists of activities to be performed by the system engineering organization within each project phase. **It is recursively applied by each system engineering organization of each supplier of the elements of the product decomposition.**

The system engineering process is intrinsically iterative across the whole life of a project, in particular in the initial phases (i.e. 0, A, and B) of the development of a complex system (e.g. a spacecraft), procured through a multi-layered set of suppliers. During these phases, the system engineering organization derives design oriented technical solutions using as an input the design-independent customer requirements.

Through this process the system engineering organization performs a multidisciplinary functional decomposition to obtain lower level products (both hardware and software). At the same time the system engineering organization decides on balanced allocations, throughout the system, of resources allocated by the customer and respects agreed margin philosophies as a function of the relevant technology readiness levels. The functional decomposition defines, for each level of the system, the technical requirements for the procurement of subassemblies or lower level products as well as the requirements for the verification of the final characteristics of each product.

¹ <http://llis.nasa.gov/llis/search/home.jsp>

² NASA/SP-2007-6105 Rev1

The system engineering process uses the results of these lower level verification activities to build bottom-up multi-layered evidence that the customer requirements have been met.

The system engineering process is applied with various degrees of depth depending on the level of maturity of the product (e.g. new development or off-the-shelf).

The system engineering process can be applied with a different level of tailoring as agreed between customer and supplier in their business agreement.

The system engineering organization has interfaces with organizations in charge of management, product assurance, engineering disciplines, production, and operations and logistics.

5.2.1.3 System level framework relevant for Software

The **context** of the space software engineering activities is the overall Space System Engineering process. The software related system requirement process, produces the information for input to the system requirements review (SRR). This establishes the functional and the performance requirements baseline (RB) of the software development. It constitutes the link between the space system processes, as defined in ECSS-E-ST-10C for flight systems, and in case of ground systems in ECSS-E-ST-70C, and the software processes.

According to the recursive customer-supplier model of ECSS, at each level, the customer is responsible for the delivery of a system in which the developed software is integrated. According to the recursive system-subsystem model of ECSS-E-ST-10C, he is responsible for the specification of the system requirements at lower level and, in particular, for any software comprised in the system.

From these perspectives major contributions from the "Software" discipline are absolutely necessary for the system level in order to provide a space system / product that will finally satisfy the customer's needs. A Software System Engineering organization or (depending on the hierarchical level of the customer-supplier relationship) at least function is therefore fundamental for the project setup on all supplier levels - independent from the value of contribution (e.g. Subsystem, Equipment's, Payloads) to the final space system or product. This organization or function serves as focal point / interface for all software aspects.

5.2.1.4 Software System Engineering

Software System Engineering comprises the work of software specialists (like analysts, designers, programmers, quality engineers and others).

When the software development is included in a complete space project, the software engineering processes have also a relationship with the project phases (0, A, B, C, D, E, F) and in particular with the system level reviews (e.g. some system-software activities are executed in phase B).

5.2.1.5 Co-engineering by means of an Integrated System / Software team

The idea is to transfer knowledge and responsibility between the system team and the software teams and vice versa. This is an important aspect to be considered during the development processes since misunderstandings and disparate responsibilities will prompt for failures. Furthermore this will shorten the loop back from RB to TS even down to software design and code if felt necessary in order to assess the feasibility and completeness of the software requirements. Therefore trade-offs may be conducted more efficiently than the standard Waterfall model.

In case the software supplier belongs to the same organization as the system provider, some activities may be performed jointly or by the same person when different tasks are assigned to it. In other words: the same person could perform tasks wearing different organizational hats (for customer + supplier / for system + for software levels).

NOTE This might also work for a software supplier from a different organisation, if a dedicated business agreement can be established early enough, e.g. in phase B.

5.2.1.6 Preconditions / Tailoring

The **System Requirement process** is a system process and initially a customer process. However the software development is an integral part of an overall space system product. The purpose of this process is to generate explicit requirements for the system from the view of a system engineer and / or a customer. The output is the Requirement Baseline (RB).

The System Requirement process is very often subject to tailoring, especially for Research & Development and Prototyping projects. For general tailoring refer to Appendix S of the ECSS-E-ST-40C or chapter 4.2.5 of this Handbook. See also the tailoring of the process in case of software or infrastructure reuse in 5.4.3.7.2 of this document.

- a. The following points aim at clarifying the border between RB and TS:
- b. A **SSS requirement** should define the software need in the system (the **WHAT**). This allows being independent (as far as possible) of the hardware and software implementation, i.e. several solutions may answer to the same need. Hence, for example for flight software, the RB includes IRDs or User Manuals for Space-to-Ground interfaces, Operations, Command / Control, Equipment's, HW/SW, OBC, HSIA, Electrical IRD describing the constraints of the system when there are several pieces of equipment, and the SSS explains what is the SW behaviour with respect to these interfaces.
- c. A **SRS requirement** should answer to the needs through a functional implementation taking into account software and hardware constraints in the system (closer to the **HOW**, at least from a logical or functional point of view). This allows for development of the software detailed design. Hence, for example for flight software, the Technical Specification is including the SW TM/TC ICD.

The quality of the requirements in general and of the Requirement Baseline in particular has proven to directly impact the success of the software project. The tailoring of this process, which in a way is a self-assessment of the customer, needs to be performed carefully. In case of doubt, the elaboration of the Requirement Baseline can be delegated to the Supplier, but the responsibility and the approval of it needs to stay at the customer level. Any (initially) missing system requirements can be included through the System Requirement Review being the synchronization point. The SRR also allows the supplier to verify the Requirement Baseline, to adopt it and to suggest improvements.

If there are critical requirements identified at a later stage in the project, their implementation may induce high extra costs and major schedule delays.

If applicable, the software related system requirement process consists of the following 3 activities:

- d. software related system requirements analysis
- e. software related system verification
- f. software related system integration and control

which are described in sufficient level of detail in the ECSS-E-ST-40C.

5.2.2 Software related system requirements analysis

5.2.2.1 Specification of system requirements allocated to software

The Use Case technique can support this activity and is described in 6.1

5.2.2.2 Identification of observability requirements

The operability of the Space Vehicle is a major objective of Monitoring & Control. It must meet not only the nominal operating needs, but also the anomaly processing and flight feedback information requirements. The induced requirements are directly applicable to the software (both ground and flight) and constraints its design.

Although belonging to the E70 “ground and operations” branch of the ECSS standards, the ECSS-E-ST-70-11C - “space segment operability” document defines in detail the requirements applicable to the spacecraft, covering both commandability and observability.

This document provides all the necessary inputs to the software for what relates to this discipline, in conjunction with ECSS-E-ST-70-41A “packet usage standard” and ECSS-E-ST-70-01C “OBCP”.

The Requirement Baseline should therefore make applicable explicitly these standards, and possibly refine all the included requirements down to the software level during the system / software engineering phase.

5.2.2.3 Specification of Human Machine Interface requirements

In principle the ECSS-E-ST-10-11C standard is applicable to ground segment as its introduction states “This Standard defines requirements for the integration of the human in the loop for space system products”. Many processes discussed in ECSS-E-ST-10-11C are general ones for any human system integration project.

However, it is clear that the ECSS-E-ST-10-11C standard puts emphasis on the Human Space Flight domain. The document could be tailored for Mission Control System needs, with the relevant stakeholder cooperation. Alternatively, upon approval by all stakeholders, another specific standard that meets or exceeds the ECSS-E-ST-10-11C standard, for this domain, could be made applicable.

5.2.3 Software related system verification

5.2.3.1 Verification and Validation process requirements

-

5.2.3.2 System input for software validation

The standard asserts that the customer specifies 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 (see 5.2.3.2 of ECSS-E-ST-40C). It also asserts that the supplier develops and documents, for each requirement of the software item in RB, a set of tests that validate the software against the mission data and scenario specified by the customer (see clause 5.6.4.1 of ECSS-E-ST-40C).

Scenarios are therefore essential to produce the software validation plan. It strengthens the suggestion that Use case technique (see 6.1) should be used for the definition of software requirements, at RB level and TS level. Thus the RB defines use cases that are refined in the TS. Once the RB and TS are written, the software validation plan is almost specified, because, if use cases are used all along the RB and TS elaboration, then they can be easily transformed into “test cases”. This facilitates the preparation of the Software Validation Specification (SVS). However, the validation is based on requirements, and the validation plan must cover all the requirements. Achieving coverage requires complementing or enriching the scenario.

5.2.3.3 System input for software installation and acceptance

-

5.2.4 Software related system integration and control

5.2.4.1 Identification of software versions for software integration into the system

See 5.3.5.2

5.2.4.2 Supplier support to system integration

-

5.2.4.3 Interface requirement specification

-

5.2.4.4 System database

5.2.4.4.1 Introduction

This part aims at providing some details about the System Data Base (SDB), its interface with the other space software (particularly the central on-board software) and some of its development constraints.

All engineering processes involved in the space system life cycle are based on data management processes and tools. The SDB is the reference repository of all the system level data, e.g.:

- a. System configuration data,
- b. System element physical properties,
- c. TM/TC data definitions,
- d. On-board software parameters / monitoring / Safe Guard Memory / OBCP definitions,
- e. On-board communication protocol data,
- f. AIT specific data,
- g. Ground calibration/decalibration functions,
- h. Operation / FDIR specific data,
- i. Electrical I/F data (at functional interface level),
- j. Simulator data (modelling input).

The System team is responsible to define, configure, distribute and validate these system level data (see ECSS-E-ST-40C clause 5.2.4.4), with the support of the SDB.

The SDB is thus in the heart of the industrial process (i.e. development, validation, operations) of a complete system (including ground and flight segments). It interfaces with almost all engineering space subsystems (named SDB users below), e. g. the ground segment (including the control centre), the validation facilities, the simulators, the on-board software's, the operation tools...

The objective is to configure all software(s) and subsystems in a consistent way (see ECSS-E-ST-40C clause 5.4.3.6c).

One key point in order to ensure a proper exchange of information is to define, when possible, a well-defined name for all the described entities. It is of high added value for the whole project to ensure a consistent naming of e.g. telecommands, telemetry parameters, system parameters, from the Requirement Baseline to the software implementation and the SDB.

The SDB may be distributed or centralized. In any case, the important point is to ensure the consistency of the structure and of the provided data.

Data structures are defined at the SDB user (e.g. on-board software, Ground segments) PDR. Final content and values are defined at CDR.

The SDB tool itself is software designed through a conceptual Data Model that defines all data structures and external interfaces needed by SDB users, for software generation (including documentation) and/or operations. The SDB tool needs to be available in accordance to subsystems schedule; this creates strong dependencies among life-cycles of these subsystems and SDB tool. Hence the clause 5.2.8.4 provides some recommendations to properly tailor the ECSS-E-ST-40C.

5.2.4.4.2 On-board SW SDB user

The development of the on-board software of a space segment is based in particular on data:

- a. provided by the system/operations team (e.g. numerical data, housekeeping packet definition, monitoring thresholds),
- b. or produced by the on-board software itself (e.g. detailed description of telecommands, telemetry reports, and software and telemetry parameters).

The configuration data values produced at system level are necessary to build the on-board software binary image. Preliminary values are necessary to perform on-board software validation (especially closed loop tests) and the final values can be delivered later for final acceptance tests (see also ECSS-E-ST-40C clause 5.6.4.2b).

Due to the huge volume of system level configuration data, it is highly recommended that the introduction of the system level data into the on-board software binary image build process is automated. This will provide the additional benefit of being able to introduce late modifications of configuration data with a minimal cost.

Most of the data produced by the on-board software are specified at the time of the Technical Specification production. Defined in the Software Interface Control Document or SRS, they are then introduced in SDB. Their values are then defined by the software or the System teams.

Here also, it is highly recommended to put in place a computer assisted and even automated process, based on a formalized description in an electronic format (e.g. XML Schema, Eclipse Modelling Framework) of the corresponding section of the Technical documentation.

The data base parameters, that are used to configure the software, are defined and verified by the system team. They become requirements for the software, and as such they are validated (to some extent for the system level data) by the software team. They are fully validated in context of the system tests.

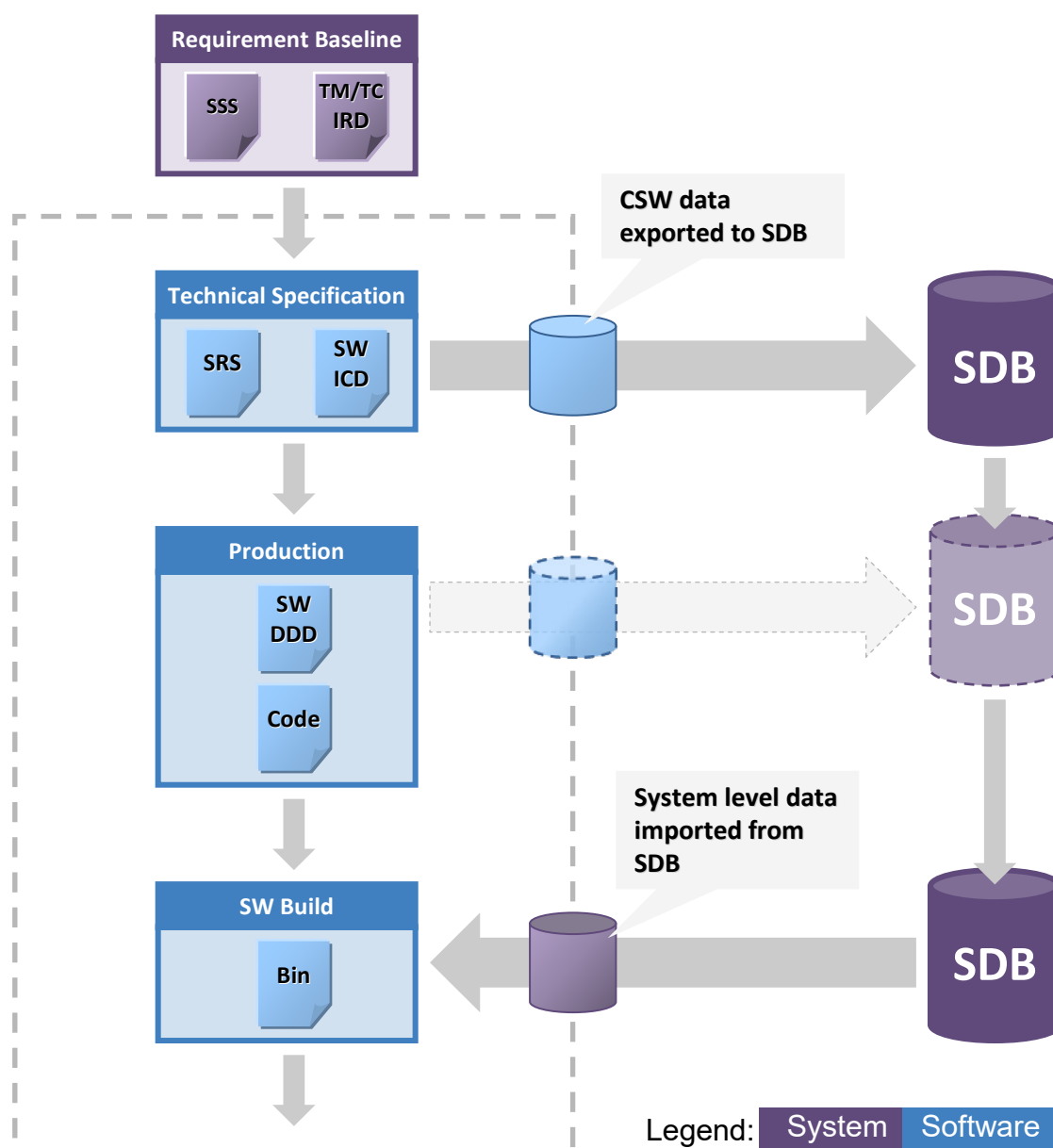


Figure 5-1: System database

NOTE CSW denote on-board software ("central software")

5.2.4.4.3 Ground segment SDB user

For the traditional ground segment software, only a part of the SDB is relevant which is limited to the information related to the TM/TC. For example, the "pure On Board information", like harness, are not used. Other data are specific for the ground segment, e.g. alarms, calibration/de-calibration functions, ground modes...

Since the ground segment is composed of many components, such as Mission Control System, Ground Station System, Data Distribution System, Network and Communication System, which are geographically distributed and developed, owned and operated by different entities, each subsystem may use a database of configurable items at subsystem level, and regularly synchronized from the SDB.