 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N°332830 AND FRO M SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Enginieering **A**cce**L**eration

Space Use Case Requirements

D205.010

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Space Use Case Requirements
Deliverable No.	D205.010
Dissemination Level	CO
Nature	R
Document Version	V5.0
Date	2014-03-10
Contact	Ricardo Moreno Ruano
Organization	TAS-E
Phone	+34 918 07 79 00
E-Mail	Ricardo.MorenoRuano@external.thalesaleniaspace.com

AUTHORS TABLE

Name	Company	E-Mail
Ricardo Moreno Ruano	TAS-E	Ricardo.MorenoRuano@external.thalesaleniaspace.com
Rubén de Juan	ITI	rjuan@iti.es
Ismael Ripoll	ITI	iripoll@iti.es
Elena Alaña	GMV	ealana@gmv.es
Mª Carmen Lomba	GMV	mclomba@gmv.es
Carlos Zubieta	Orbital Aerospace	carlos.zubieta@orbital-aerospace.com
Susana Pérez	Tecnalia	susana.perezsanchez@tecnalia.com

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.01	2014/01/14	Creation of document	All
2.0	2014/01/28	Interoperability Challenges and Use Case sections added	7-11; 13-15
3.0	2014/02/05	Engineering Methods and general format	All
4.0	2014/02/14	Internal review changes and detailed engineering method spreadsheet	24-27
5.0	2014/03/10	Included external reviewers recommendations	7, 9, 10, 21-23, 26-27

CONTENT

Content of Figures

Content of Tables

1 Introduction

1.1 Role of deliverable and relationship to other CRYSTAL documents

This document has the following major purposes:

- Definition of the overall use case, including a detailed description of the underlying development processes and the set of involved process activities and engineering methods.
- Provide input to WP601 (IOS Development) required to derive specific IOS-related requirements.
- Provide input to WP602 (Platform Builder) required to derive adequate meta-models.
- Establish the technology baseline with respect to the use-case, and the expected progress beyond (existing functionalities vs. functionalities that are expected to be developed in CRYSTAL).

1.2 Structure of this document

The document is structured as follows:

- Section 2 presents the use case SW application, including its context into space domain, the HW platform where it will run and the standards that need to be compliant with.
- Section 3 makes a detailed presentation of the SW development phases required in space domain to be space qualified, identifying the engineering methods.
- In Section 4 is summarized the requirements for the CRYSTAL Space Toolset applied to Avionics Control Unit Software in form of selected engineering methods to be applied in it.

2 High-level description of use case and context

2.1 Use case description context

In the aerospace domain, the hardware manufacturer remains responsible in front of the customer (typically ESA) of the quality and performances of the software embedded in the units, even when the SW is procured from an external SW supplier. To assure the quality of this SW, ESA has developed standards for software engineering and software quality assurance. Through the application of these standards the safety and reliability of the code is assumed to be a consequence of the quality of the process used during the code development; this process is based on a series of customer-client meetings where abundant and exhaustive documentation relative to design, analysis and test of the SW product are reviewed. In addition to this, the SW has to undergo an Independent Software Verification and Validation process (ISVV) by a third party company, thus increasing certification efforts.

Apart from an intensive certification process, the limited resources and harsh operating environments (high radiation doses) mean that processor boards for space instruments are quite limited in scope compared to those in modern PCs. However, in space as well as in the consumer market, as the shrinking of the electronic components enables it, the trend is towards more dense, integrated and reconfigurable systems. Potential reconfiguration of the system (either by upload of SW components or new VHDL design) entails a new scenario where requirements may change during any moment of the project lifetime, covering from design phase to even on-flight operation and maintenance.

This use case will serve to assess CRYSTAL bricks and technologies (RTPs and IOS) as per the ECSS standards applicability in order to accelerate the development and certification processes of reconfigurable space-qualified systems, thus reducing time and costs efforts. The application to be implemented for the Space domain is the Low Level Software for an Avionics Control Unit which application software could include autonomous navigation features based on GPS, inertial and/or image acquisition inputs. This unit will be based in a LEON microprocessor architecture running in multicore configuration inside an FPGA exploiting state-of-the-art fault tolerant techniques.

2.1.1 Technical description

The word avionics being a contraction of 'aviation electronics', carries out activities related to the command and data handling (C&DH) sub-system, guidance navigation and control devices and associated software flown aboard a satellite. These items cannot be bought as off the shelf equipment based on conventional components because they must be able to carry on operating for years at a time while surviving the harsh space environment. Satellite avionics must be specifically designed and built instead.

Comprising computers, data bus, sensors and actuators and on-board software and algorithms, the avionics subsystem contributes a huge amount to a given mission's functionality but is complex

and expensive - corresponding to around 60% of overall development and verification costs of a typical satellite platform only.

Figure 2-1: Flight Model of an Avionics Unit

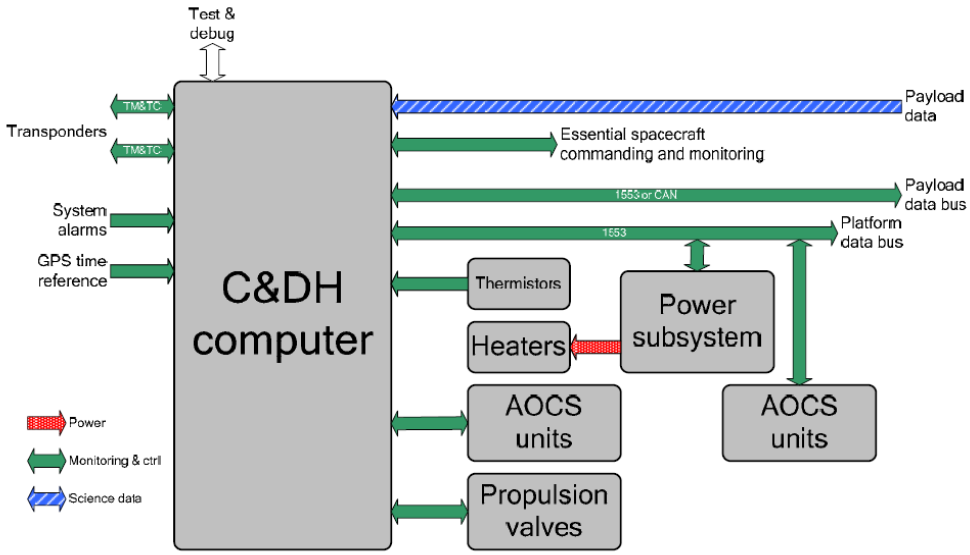

Figure 2-2: Block diagram of an Avionics Unit

The On Board Software implements satellite's vital functions such as: attitude and orbit control in both nominal and non-nominal cases, telecommands execution or dispatching, housekeeping telemetry gathering and formatting, on board time synchronisation and distribution, failure detection, isolation and recovery, etc.

Based on the above, the very essence of an Avionics Control Unit is the microprocessor board, consisting of microprocessor, non-volatile memories, volatile memories and the companion chip that connects the microprocessor to different peripherals.

A modern Avionics Control Unit includes functions such as:

- DC/DC Power conversion and regulation
- Ground Telecommand Decoding
- Packet Telemetry Formatting
- On Board time management
- Autonomous Reconfiguration
- Local Mass Memory function
- Housekeeping telemetry
- Interfacing with other Avionics subsystems

2.1.2 HW platform

TAS-E is currently working in the design and development of a dual-FPGA board, which block diagram can be observed in Figure 2-3.

This dual-FPGA architecture allows evaluating multiprocessor systems where a main (multi)processor embedded within one of the FPGAs distributes the processing load to a second device; it is the Low Level SW of this multiprocessor whose development is the main driver of the Aerospace Demonstrator.

In this case, the second FPGA can embed another processor, a DSP or just implement some hardware (VHDL) algorithms.

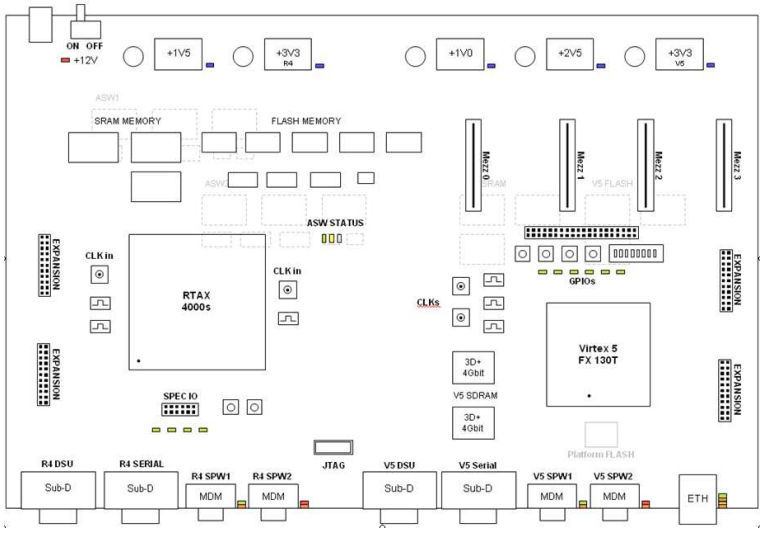

Figure 2-3: Dual-FPGA board potential lay-out

2.1.3 Low Level Software

On every Avionics Control Unit there are two different SW products:

- Low Level SW (LLSW): highly critical as stored in PROM/EEPROM and not modifiable in flight, supporting HW platform initialisation and minimum core of data handling function in order to enable diagnostic and load of ASW.
- Application SW (ASW): traditionally SW running in RAM (possibly stored in EEPROM) and completely reloadable in flight, supporting full data handling and processing functions.

The Low Level Software, in turn, implements Boot, Drivers and Test SW.

1. Boot SW: It is responsibility of the Boot SW to initialize the board, perform built-in tests, provide health status and launch Application Software from EEPROM memory area to RAM (see Figure 2-4).
2. Drivers: The drivers' library provides an abstraction layer between HW and other SW components. They are linked as a library to Low Level and Application SW.
3. Test SW: a Test SW will be implemented as dummy Application SW; the purpose of this Test SW is to validate the HDSW, check correct boot process, check correct communication with the RTOS and characterize the final Applicative SW behaviour and CPU load. Test Software is not included traditionally as flight SW.

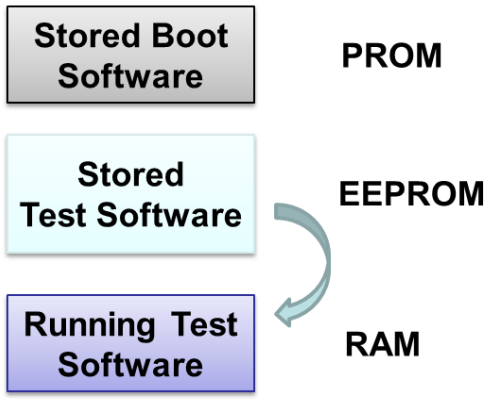

Figure 2-4: LLSW boot sequence

As example of potential components of Test SW, standard and specific benchmark applications are useful to characterize the architecture, identify bottlenecks and perform trade-off studies:

Mimicking benchmarks:

- I/O Bandwidth
- Digital filters (FIR, various numbers of taps)
- FFT (1024pt, 2048pt, 4096pt, 1920pt)
- CCSDS compliant data compression

Micro-benchmarks:

- CoreMark:
- EEMBC AutoBench

Apart from these benchmark applications, Test SW may also include mission specific functionalities:

- Specific thermal control
- FDIR surveillance of specific events
- Specific TM/TC protocol communication

The final set of functionalities to be implemented in the LLSW will be detailed in further deliverables.

Like any other space domain technology, SW is also regulated and must be compliant with its associated standards within ECSS series.

2.2 ECSS-Series

The European Cooperation for Space Standardization (ECSS) is a cooperative effort of the European Space Agency, national space agencies and European industry associations for the development of a coherent, single set of user-friendly standards for use in all European space activities. The result of this effort is the ECSS series of standards (ST), handbooks (HB) and Technical Memoranda (TM) which are organized in four branches:

- M: Space project management;
- Q: Space product assurance;
- E: Space engineering;
- U: Space sustainability

The main software standard is ECSS-E-ST-40, part of the ECSS engineering branch (E). It covers all aspects of space software engineering, from requirements definition to retirement. It defines the scope of the space software engineering processes, including details of the verification and validation processes, and their interfaces with management and product assurance, which are addressed in the management (M) and product assurance (Q) branches of the ECSS system.

ECSS-E-ST-40 refers the ECSS-Q-ST-80 for the Software Product Assurance requirements related to the development and maintenance of software for Space Systems. Both two apply to any software project. The ECSS-E-ST-40 provides a process model for the SW development activities, without prescribing a particular software life cycle.

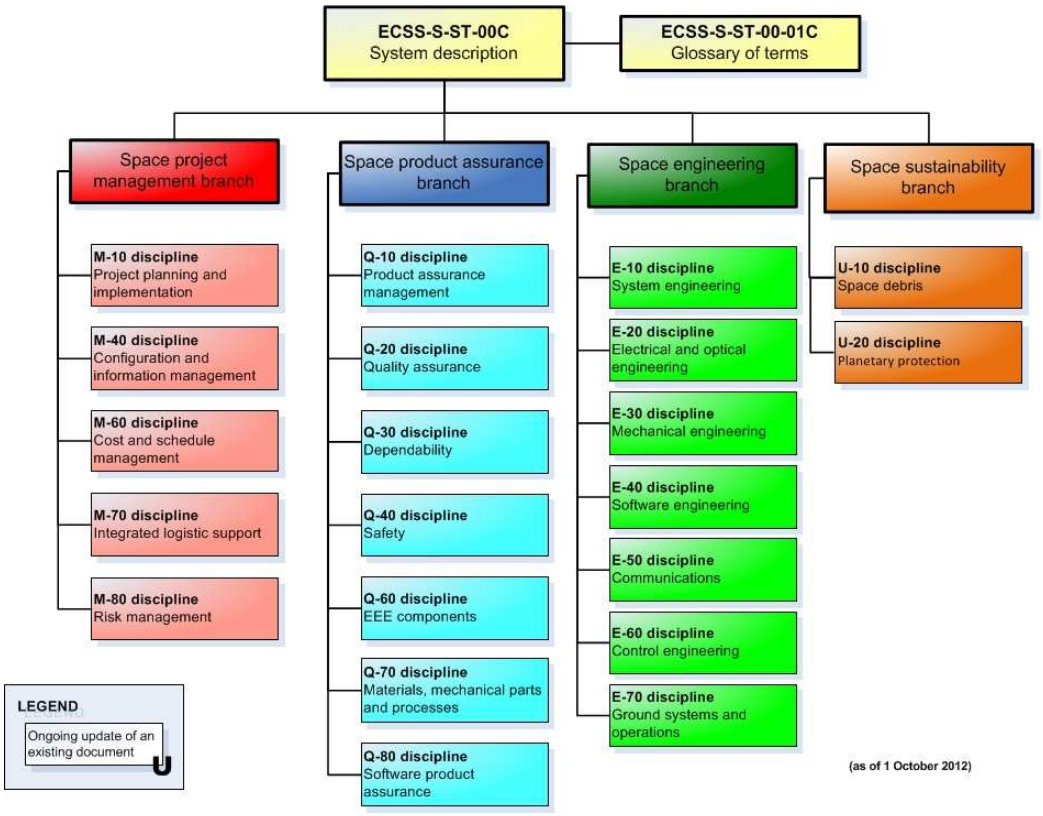

Figure 2-5: ECSS Architecture (from ECSS website)

2.2.1 High level process

One of the fundamental principles of the ECSS standard series, and distinctive difference compared to the standards from the other domains, is the explicit customer-supplier relationship, assumed for all system and software developments, where the supplier demonstrates compliance with the customer requirements and provides the specified evidence of compliance. How and which parts of the ECSS must be applied is specified through contract, in a way that depends on the given mission.

In the ECSS the development of the software is always related to a complete space project and its different phases, with a strong focus on the integration with system level activities.

ECSS is both a process-based and product- based framework, in fact, ECSS is based on “processes”, and lets at user’s choice an own life-cycle and approach, with appropriate methods and tools.

ECSS has the peculiarity of allowing the tailoring, on project-basis. Tailoring means that on a project-basis, and hierarchically in the chain, each Customer may determine the applicable ECSS standards and requirements therein, for his own Suppliers. This tailoring must be justified, coherent, and consistent throughout the Customer-Supplier chain, and always be visible to the higher level Customer.

3 Engineering Methods

3.1 Process activities in ECSS SW development

As stated in previous section the ECSS-E-ST-40 provides a process model for the SW development activities, without prescribing a particular software life cycle. It also assets the need of specifying SW RAMS (Reliability, Availability, Maintainability and Safety) requirements based on the System RAMS analysis result. At the same time the software is developed, a criticality analysis is carried out to assure the dependability and safety issues.

The ECSS-E-ST-40 standard defines a set of requirements for developing software in the scope of a space system project. But because these requirements cover a wide range of applications, some of them may not be applicable and the requirements must be tailored for each project. ECSS-E-ST-40 standard, states that there are several drivers for tailoring, such as dependability and safety aspects, software development constraints, product quality objectives and business objectives. For every software development a software development plan must be defined in order to instantiate the particular implementation of this standard in the project.

Software dependability and safety are part of the system dependability and safety programmes, including regular control meeting, technical reviews and documentation that forma part of the Product Assurance File. The different software safety and dependability assessment activities are represented in the figure below, in relation to the software development, verification and validation activities defined in ECSS-E-ST-40.

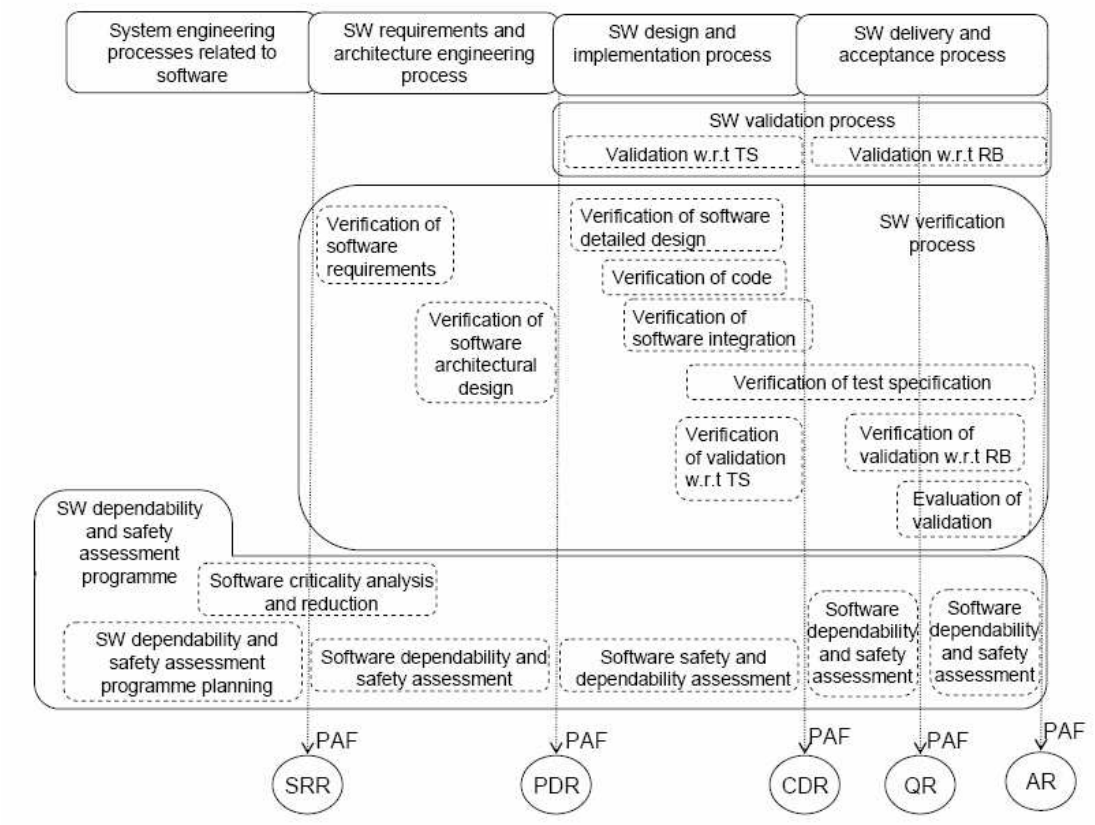

Figure 3-1: Software RAMS activities (ECSS-Q-HB-80)

ECSS-E-ST-40 describes the software processes and activities breakdown as described in Figure 3-2. Each process includes activities which are themselves decomposed into a list of one single or several tasks in the shape of process requirements (clauses), producing expected outputs.

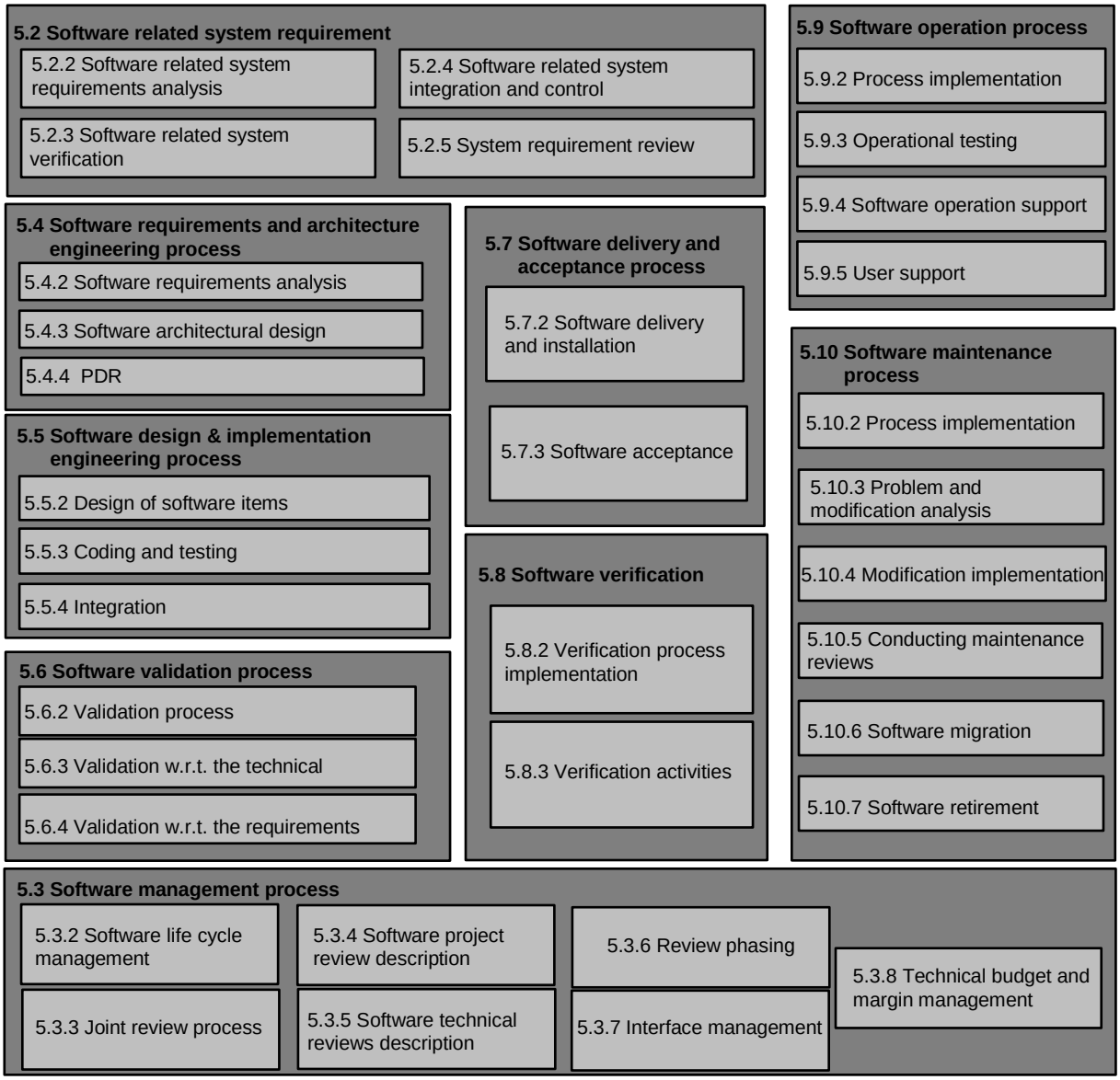

Figure 3-2: Structure of ECSS-E-ST-40C

Please note that, the software management process, software operational process, software maintenance process and software delivery and acceptance processes are out of the scope of the case study.

3.2 SW Life V-Cycle

The Use case life cycle will be based on a typical V-Cycle (Design and Verification) life cycle. The following figure describes this life cycle:

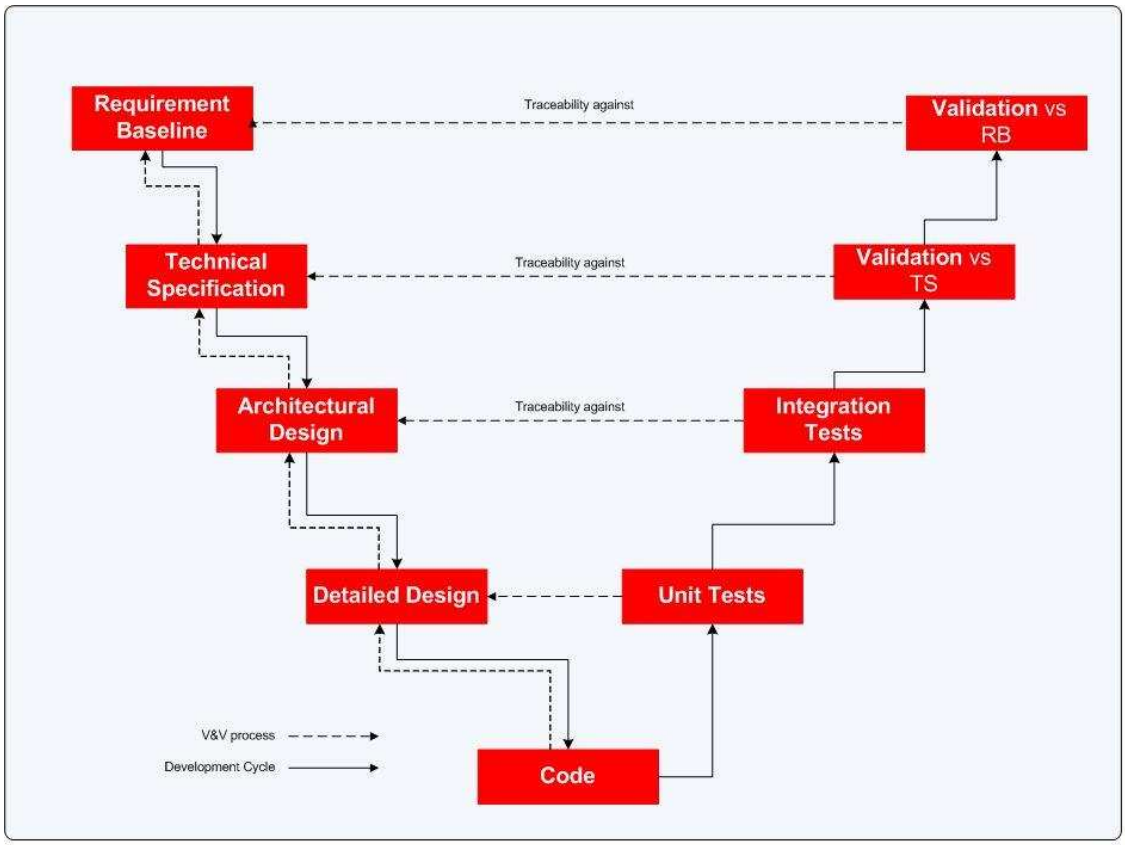

Figure 3-3: Software V cycle

The following sections describe the phases performed during the SW project life development cycle. This is divided in a combination of several phases with different objectives. Steps from one phase to the next one are interfaced by a project formal review. For each project phase, the following information is described:

- Phase start.
- Phase activities.
- Phase end.

Stakeholders/actors:

- Final customer (typically ESA, provides User Requirements Documentation).
- HW manufacturer (i.e TAS-E: derives Requirements Baseline from User Requirements and pass it to SW supplier).
- SW supplier: external company to HW manufacturer (or different product line within the same organization in case of big companies).

3.2.1 Requirements baseline specification phase (RB)

➢ RB phase start

The phase formally starts at Kick Off project meeting.

➢ RB phase activities

The main activities in this phase are:

- To analyse the User Requirements (UR) in order to provide a detailed version of the URs.
- To establish the Requirements Baseline (RB) that will form the basis for all activities carried out in the project.
- To analyse the critical functions identified in the Requirements Baseline.

➢ RB phase end

The RB phase concludes with a System Requirement Review (SRR) with the aim of verifying the contents of the Requirements Baseline.

3.2.2 Technical specification definition phase (TS)

➢ TS phase start

This phase formally starts on approval of the Requirements Baseline (RB) in the System Requirements Review (SRR).

➢ TS phase activities

The software engineering activities in this phase include:

- Establishing a functional breakdown & data flow schema (logical model) of the software product; finalising all requirements to obtain an approved baseline for the development. The logical model is an abstract description of what the system should and should not do, and should not contain specific implementation terms.
- Establish the Technical Specification (i.e., Software Requirements Specification and Interface Control Document).
- Define the software verification and validation planning. The validation test plan (VVP) aims to conduct all validation testing and it will be based on the Software Requirements Specification (SRS). All intended functionality must be tested and checked if all desired behaviour is met according to the expectation. The Software Test Plan is defined in the following manner:
- Items subject to validation.
- Validation tasks to be performed.
- Resources, responsibilities, and schedule for validation.
- Procedures for forwarding validation reports to the customer and other parties.
- Software Critically Analysis Report (SCAR) related activities.

➢ TS phase end

The Technical Specification phase concludes with a SWRR (Software Requirements Review). The objective of the SWRR is to baseline requirements.

3.2.3 Architectural Design phase (AD)

➢ Architectural Design phase start

This phase formally starts on approval of the RB in the SWRR (Software Requirements Review).

➢ Architectural Design phase activities

The software engineering activities in this phase include:

- Create the software top-level architecture in compliance with the SRS. The process followed to create the architectural design is top down. A root class representing the

overall system according to the SRS requirements is created that it is further decomposed into smaller pieces and SRS requirements are distributed.

- Prepare the Performance and Schedulability Analysis Report, including budget information. This document provides estimations for the SW memory and CPU budgets based on available technical specifications.
- Software Critically Analysis Report (SCAR) related activities.

➢ Architectural Design phase activities end

The Architectural Design phase concludes with a PDR Preliminary Design Review (PDR) The objective of the PDR is to baseline high level architecture and to give formal approval to start the Detailed Design project phase.

3.2.4 Detailed Design phase (DD)

➢ Detailed Design phase start

The phase starts after the Preliminary Design Review (PDR). The Detailed Design for each component of the SW shall be delivered at DDR, when the Detailed Design phase is finished.

➢ Detailed Design phase activities

In DD phase, lower-level components of the architectural design are decomposed until they can be expressed as modules in the selected programming language. Starting from the bottom-level components in the ADD, the design proceeds to lower levels via stepwise refinement of each module specification.

Although design should normally proceed downwards, some of the lowest level components may need to be designed (and coded) first (e.g., utility libraries).

Software Critically Analysis Report (SCAR) related activities must be performed.

➢ Detailed Design phase end

The DD phase culminates with the Detailed Design Review (DDR). When the design of each module is completed, reviewed and approved, it can be coded.

3.2.5 Coding & unit test phase (CO)

➢ Coding & unit test phase start

This phase starts when the detailed design is approved at Detailed Design Review (DDR).

➢ Coding & unit test phase activities

This phase consists of the coding and unit level testing of all units in the software. Both static analysis and dynamic analysis would be performed.

Static source code analysis is performed by measuring several software metrics parameters and comparing the obtained values with acceptable limits.

Results of source code static and dynamic analyses will be included in the software metrics report. The coverage objectives for unit test will be 100% statement coverage for modules classified as criticality category C (major consequences) and 100% decision coverage for category B (critical).

Software Critically Analysis Report (SCAR) related activities must be performed.

➢ Coding & unit test phase end

The coding phase culminates with a review, ITRR (Integration Test Readiness Review). The meeting verifies that the source code meets the design goals approved in the Detailed Design, to ensure that the software is ready to be integrated.

3.2.6 Integration test phase (IT)

- IT phase start

This phase starts when the code has been tested at unit level. This phase will be overlapped with the previous one. The integration strategy will be to ensure that all SW components of Use case SW are integrated and ready to be validated in the Use Case SW Validation environment.

- IT phase activities

Activities during the IT phase include:

- Integrating the software modules into the software product and testing them.
- Preparing the test cases and procedures against TS/RB.

- IT phase end

The IT Phase culminates with a TRR (Test Readiness Review). The objective of this meeting is to determine whether the integration of the SW product and the software test cases against TS are sufficient for the SW validation to begin. In particular, the status of each of the following will be assessed:

- Software test cases against TS.
- SW to be tested.
- Test software: Testing environment to support the System Software validation testing.

3.2.7 Validation of TS phase (VT)

- Validation of TS phase start

This phase formally starts when the code is successfully integrated, and with the approval from TRR (Test Readiness Review). TRR assess whether the integration of the SW product and the software test cases against TS/RB are sufficient for Use Case SW validation tests to begin.

- Validation of TS phase activities

Activities during the VT phase include:

- Performing the Validation Tests against the TS/RB.
- Documenting the results in the Test Report.
- Software Critically Analysis Report (SCAR) related activities.

- Validation of TS phase end

This phase will conclude with a CDR (Critical Design Review). The major purpose of the CDR is to ensure that system tests have been completed to a level at which the software can be integrated in the target platform, and that the delivery data pack is acceptable and accords with project requirements. The CDR will assess the completeness of the system tests and of all delivery documentation.

The scope of CDR will include the review of the current status and results of the validation against the TS/RB.

3.2.8 Acceptance phase (AT)

In this phase the SW is delivered, installed and accepted by the customer after acceptance tests performance, i.e: RB tests in final HW flight model and including equipment environmental tests (vacuum, thermal and electromagnetic tests).

- AT phase start

This phase shall start on approval of the Preliminary V0 version CDR to proceed with formal validation of the user requirements established in the SSS.

- AT phase activities

During this phase SW validation continues. During this phase a QR (Qualification Review) is planned on the SW V0 version to verify that the software product meets all of its specified requirements in the requirements baseline.

- AT phase end

This phase shall conclude with an Acceptance Review (AR). The major purpose of the AR is to ensure that the acceptance testing has been completed and that the delivery data pack is in an acceptable form and complies with project requirements. The acceptance of the software at this review is preliminary.

The review shall assess the completeness of acceptance testing and of all delivered documents.

3.2.9 Dependability and Safety process

Dependability and Safety Activities are performed during the entire SW life cycle. The following activities are performed and reported in the Software Criticality Analysis Report (SCAR):

- Functional analysis

  The functional analysis is a common basic task necessary to perform subsequent Dependability and Safety activities. Its purpose is to identify software critical functions. This task is primary based on the Use Case functional description. Later, it will be refined according to the Software Requirements Specification. Finally, when the software architecture is available, functions previously identified are mapped to software components.

- Analysis of failure modes

  The potential failure modes associated to each software function are identified.

- Criticality assessment

  A criticality category is assigned to each software component based on the effects of the associated failure modes. The criticality of the software component corresponds to the highest severity of the potential failure modes of that component.
  Compensation and recovery actions are extracted from FMEA (Failure Modes and Effects Analysis) analysis. They are evaluated to decide their implementation or, if the final decision is for no actuation, a documented rationale has to be added.
  The set of critical software components is listed but it shall be verified and reviewed at each software cycle review.

- Verification of the implementation of compensation provisions

  Recommendations/compensation provisions to the overall software life cycle are provided in order to fulfill the required measures and assure the required reliability.
  The implementation of approved compensation provisions must be checked. A document containing the traceability matrix that traces compensation provisions to those requirements and software components that implement them has to be produced.

3.3 Engineering methods identification

An engineering method describes how an activity can be conducted using guidelines, tools and languages that interoperate with each other.

The following table identifies the engineering methods extracted for the activities contained within the previous defined ECSS processes.

Engineering Method	Description
RB Requirements Analysis	The customer must specify the Requirements Baseline (RB) in a complete, correct, consistent, precise and unambiguous mode. Every requirement contained in the Requirements Baseline must be traced to a TS requirement (and viceversa) to assure the completion of the SSS specification. A verification method has to be assigned to every requirement (e.g., analysis, inspection, test).
TS Requirements Analysis	TS requirements must be specified by the provider in a complete, correct, consistent, precise and unambiguous mode. A logical model is used to produce a structured set of software requirements that is consistent, coherent and complete. These software requirements specify the functionality, performance, quality, interfaces reliability, etc., of the system to be developed. TS requirements must be traced to RB requirements and a verification method must be assigned.
Architectural & Detailed Design	To create the architectural and detailed design of the SW. Design components must be traced to TS requirements (and vice versa).
Performance analysis	To perform the performance report results.
Schedulability analysis	To perform the schedulability report results (e.g., WCET).
Coding	To perform the SW coding.
Static code analysis	To perform Source code files static analysis by measuring several software metrics parameters and comparing the obtained values with acceptable limits.
Dynamic code analysis	Test Coverage Analysis is the evaluation of the adequacy of testing by collecting information about how much of the software was executed during the test.
Unit testing	Unit testing aim is to check the conformance of each software operation with its detailed design.
Integration testing	Integration Testing aims to demonstrate that the implementation matches the architectural design.
Validation testing wrt TS	This validation Testing aims to demonstrate that the implementation matches the TS requirements.
Validation testing wrt RB	This validation Testing aims to demonstrate that the implementation matches the RB requirements.
Design Verification Matrix	The Design Verification Matrix (DVM) will show the manner (i.e. analysis, inspection, test, etc.) and on which model(s) each individual specification parameter is to be verified. A DVM will be elaborated and maintained: • To trace the evidence of verification of each requirement. • Each terminal object documented in the ADD shall be traceable from (forwards traceability) and to (backwards traceability) the requirements of

Engineering Method	Description
	the SRS. • Each software module identified and described in the DDD shall be traceable from (forwards traceability) and to (backwards traceability) the object defined in the architectural design.
RAMS analysis.	To conduct the Dependability and Safety analysis according to the software criticality (e.g., FMEA method).

Table 3-1: Engineering methods and associated description

4 Use case requirements

The previous section described a full SW development process as per [ESA-E-40, 2009] and [ESA-Q-80, 2009], including details about all phases, reviews and documentation to be provided. For the CRYSTAL Space Toolset applied to Avionics Control Unit Software a reduced set of these methods has been selected and its implementation will be required at a lower level of exigency with respect to an official “ESA project”.

4.1 Engineering methods required

Engineering method	Description	CRYSTAL Associated Brick	Rationale
TS Requirements Analysis	Technical requirements must be specified in a complete, correct, consistent, precise and unambiguous mode.	AUGE - B2.51	Traditionally performed with Word/Excel
Architectural & Detailed Design	To create the architectural and detailed design of the SW.	AFTS-DM – B2.54 Scheduling Requirement Analysis – B2.55	No experience with SoC <-> FPGA reconfigurable systems
Schedulability analysis	To perform the schedulability report results (e.g., WCET).	Scheduling Requirement Analysis – B2.55	Traditionally performed with Excel
Coding	To perform the SW coding.	Not required	In-house solution available
Unit testing	Unit testing aim is to check the conformance of each software operation with its detailed design.	Not required	In-house solution available
Integration testing	Integration Testing aims to demonstrate that the implementation matches the architectural design.	Not required	In-house solution available
Validation testing wrt TS	This validation Testing aims to demonstrate that the implementation matches the TS requirements.	AUGE – B2.51	Currently no automatic relationship between Requirements Documentation and Test Description
RAMS analysis.	To assess conformity with respect to a Dependability and Safety analysis.	Safety Analysis for Aerospace – B2.53	No experience in SW RAMS analysis by TAS-E

Table 4-1: Engineering methods and tools implemented in the use case

A first pass alignment with respect to the Engineering Methods included in the Public Aero Use Case defined in D6.11.51 allows identifying the following potential matches and requirements:

- Verify requirements / verify design against requirements:
  - A test procedure shall exist covering each requirement susceptible to be verified by test.
  - Percentage coverage of a full set or partial list of requirements shall be provided.
- Provide specification document
  - Documentation in standard format (i.e: .doc / .pdf) shall be produced.

- Traceability
    - Architectural design / Schedulability analysis artefacts shall be traced against Technical Specification Requirements bidirectionally (through traceability matrix).

This list will be better explained and detailed in future deliverables, when both Public Aero and UC2.5 use cases are better consolidated.

4.2 Interoperability challenges

Besides the classic interactions between the different phases of the product life cycle, some specific requirements for space domain not present in other sectors are: the complexity of the system, the scarce computational resources available in the spacecraft and the radiation doses that on-board electronics receive during each mission lifetime. These facts push the requirements and the selection of the engineering tools, which relationships and artefacts should be perfectly traceable at each point of the development life cycle in order to be compliant with the highly demanding ECSS standards for Product Quality Assurance.

4.2.1 Interaction between design, implementation and testing

For example, in order to perform the mandatory real-time schedulability analysis, an accurate measurement of the capabilities and performance of the platform is required. Among others, the temporal parameters of the platform (context switch, MIPS, memory contention, interrupt service, etc.) shall be measured at different phases of the project: during the architectural design phase, an estimation of these parameters will be used to select the main building elements (processors, amount of memory, buses, configuration, etc.); during the implementation, a more accurate values using more realistic workload would help to correct and reconfigure the design in case of deviations; and at the end of the project, the evidences required for the certification have to contain precise and up to date timing values of all the real-time components.

Another scenario where the tests shall be linked with the design tools is when the result of some benchmarks/tests are used to make a design decision. In order to motivate some design choices, the associated tests and benchmarks shall be bound with the corresponding elements of the design model.

The design tools may need to interact with verification tools. This interaction may be done directly or through the requirements phase by adding new requirements generated from the design.

4.2.2 Single/unified naming space domain

Considering the following claim “independently of the complexity of the project, the whole project must be understood by a small group of engineers” in order to have a fluid communication among the main designers, a single namespace shall the used. This way, both the tools and the people will be able to name and refer clearly and unequivocally each given object. Object naming is an important issue when several groups of engineers have to work together. It is not only necessary to agree on a common ontology, but also on the name of every major element of the system. Those names must be different, but also must be easy to remember in order to be easily used by the people. Once a name is assigned to an important element it shall be preserved along the whole live of the system, as long as the system is not deeply redesigned.

4.2.3 Version and modification tracking

Any object of the software live cycle (element of a model, a requisite, a test, a function, a library etc.) may change (i.e. modified, extended, fixed, etc.) long the live of the product. It shall be able to track the modification history of each element.

It is important to track those changes, and be able to determine to which extend each modification affects other elements. For example, a bug fix in a code function may have no impact on the system or may change the value of the WCET, and so it may invalidate the results of some tests or some scheduling analysis.

The simple solution is to maintain a single global project state. Each change is carried out in the project central database. In order to avoid inconsistencies, the database may be locked when multiple changes have to be committed. This is development model is robust, but not efficient.

4.2.4 Decoupled working teams

The development of a complex project involves the "concurrent" work of several teams of engineers (partners) located at different physical locations, may be at different time zones. Each group contributes to the project by creating new elements, changing existing ones or creating new relations between them.

The tools shall provide the mechanisms to work concurrently on the same project and be able to merge, combine or exchange the results in a consistent manner. The more complex is a project the more teams are involved in its development, and the more concurrency is expected to occur.

4.2.5 Parallel development of several solutions

Implicit in the V model is the idea that the output of phase is used as input to the next one. This is a convenient simplification for small projects, but may be limiting for large ones. For example, it may be possible to think or sketch two different high level designs given a single set of requirements. During the initial phases of a project, there may be a large number of choices and options that may yield in two or more alternative solutions/designs. Those solutions are then refined and analysed until one of them is finally selected. It may be necessary to develop some prototypes and perform several tests on them to gather the information necessary to select the best alternative.

Also, due to the strong deadlines of some missions (for example, interplanetary missions have typically a narrow launch time window) it may be necessary to work in parallel in several solutions in order to avoid delays due to major re-designs.

The tools shall provide the mechanism for developing several concurrent products and transfer or reuse information from one branch to another.

The tools shall provide the ability to create "branches" or "alternatives" models, and work with them in parallel.

4.2.6 Summarising

From the engineering methods and the V model of the use case are detected the following IOS interactions:

- There is an interaction between the requirement management tools and the design/modelling tools for linking requirements with the object models satisfying those requirements.

- There is a direct interaction between the requirement management tools and testing tools in order to link requirements with test validation results.
- There is necessary a link between the tools used for the architectural design and schedulability analysis.
- There shall be a common namespace along the whole project, at least for the most relevant/visible objects.
- There shall be a uniform and consistent management of the versions, milestones, and alternative designs.

5 Terms, Abbreviations and Definitions

Please add additional terms, abbreviations and definitions for your deliverable.

CRYSTAL	**CR**itical S**YST**em Enginieering **A**cce**L**eration
ADD	Architectural Design
AR	Acceptance Review
CCSDS	Consultative Committee for Space Data Systems
CO	Confidential, only for members of the consortium (including the JU).
CDR	Critical Design Review
D	Demonstrator
DD	Detailed Design
DDR	Detailed Design Review
DSP	Digital Signal Processor
DVM	Design Verification Matrix
ECSS	European Committee for Space Standardization
EEMBC	Embedded Microprocessor Benchmark Consortium
ESA	European Space Agency
FIR	Finite Impulse Response
FFT	Fast Fourier Transform
FMEA	Failure Modes and Effects Analysis
FPGA	Field Programmable Gate Array
HDSW	Hardware Dependent SoftWare
IT	Integration Tests
ITRR	Integration Test Readiness Review
O	Other
P	Prototype
PDR	Preliminary Design Review
PP	Restricted to other program participants (including the JU).
PU	Public
QR	Qualification Review
R	Report
RAMS	Reliability, Availability, Maintainability and Safety
RB	Requirements Baseline
RE	Restricted to a group specified by the consortium (including the JU).
SCAR	Software Criticality Analysis Report
SP	Subproject
SRR	System Requirement Review

SRS	Software Requirements Specification
SWRR	Software Requirements Review
TM/TC	Telemetry / Telecommand
TRR	Test Readiness Review
TS	Technical specification
VHDL	VHSIC Hardware Description Language
WP	Work Package

Table 5-1: Terms, Abbreviations and Definitions

6 References

[ESA-E-40, 2009]	ESA; *Space Engineering - Software*; ECSS-E-ST-40C(6March2009)
[ESA-Q-80, 2009]	ESA; *Space Product Assurance - Software Product Assurance*; ECSS-Q-ST-80C(6March2009)
[DOW Crystal, 2013]	Crystal JU; *Annex I – Description of Work*; DOW CRYSTAL (332830) Part_A_B 2013-02-28

7 Annex I: Detailed Description of Engineering Methods

Detailed_Engineering_Methods_WP2.5.xls