Master Thesis
Software Engineering
Thesis no: MSE-2009-02
February2009

Towards Optimization of Software V &V Activities in the Space Industry

[Two Industrial Case Studies]

Ehsan Ahmad
Bilal Raza

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 2 x 20 weeks of full time studies.

Contact Information:

Author(s):
Ehsan Ahmad
Address: Folkparksvagen 18, Ronneby 37240, Sweden
E-mail: ehsanahmad@gmail.com

Bilal Raza
Address: Folkparksvagen 18, Ronneby 37240, Sweden
E-mail: bilalraza001@gmail.com

External advisor(s):
Tanja Nordebäck
Swedish Space Corporation AB
Address: P.O Box 4207, SE-171 04 Solna, Sweden
E-Mail: tanja.nordeback@ssc.se

University advisor(s):
Dr. Robert Feldt
School of Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet: www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

ABSTRACT

Developing software for high-dependable space applications and systems is a formidable task. With new political and market pressures on the space industry to deliver more software at a lower cost, optimization of their methods and standards need to be investigated. The industry has to follow standards that strictly sets quality goals and prescribes engineering processes and methods to fulfill them. The overall goal of this study is to evaluate if current use of ECSS standards is cost efficient and if there are ways to make the process leaner while still maintaining the quality and to analyze if their V&V activities can be optimized. This paper presents results from two industrial case studies of companies in the European space industry that are following ECSS standards and have various V&V activities. The case studies reported here focused on how the ECSS standards were used by the companies and how that affected their processes and how their V&V activities can be optimized.

Keywords: Optimization, Verification and Validation, Space Industry, Dependable Software, ECSS Standards

CONTENTS

Paper

Appendix A

Appendix B

Appendix C

Appendix D

Appendix E

Appendix F

LIST OF TABLES

LIST OF FIGURES

ABBREVIATED TERMS

PDR……Preliminary Design Review
CDR……Critical Design Review
DDR……Detailed Design Review
SAT……..Site Acceptance Test
IDD…….Integration-driven Development
TDD……Test-driven Development
PSS……..Procedures, Standards and Specifications
ECSS……...European Cooperation for Space Standardization
FST……..Faults-slip Through

ACKNOWLEDGEMENTS

First of all we would like to thank Dr. Robert Feldt, our thesis advisor, for his constructive feedback and ideas throughout this study. We are also grateful to Tanja Nordebäck of Space division at Swedish Space Corporation (SSC), our industrial supervisor, and Annalena Johansson of RUAG Aerospace Sweden AB (RUAG) for their continuous support during our visits to their facilities. Also, many thanks to the employees of SSC and RUAG for taking the time out from their busy schedule and providing us with very valuable information during the interviews. We would like to thank Swedish National Space Board for supporting this work.

Finally, we would also like to thank our parents and siblings for always being there for us.

HMT- FORMAT

This thesis is structured according to 'Hybrid Master Thesis' (HMT) format, proposed at BTH in 2007. The main idea is to have a hybrid of ACM/IEEE paper and traditional master thesis. The main motivation behind this format is to increase the number of thesis which can be published and to reduce the difficulty of over viewing large reports. According to this format the thesis is divided into two main parts, the first part follows the ACM/IEEE paper format which focuses on relevant areas and key findings of the thesis and is comprised of 10-15 pages. The latter part consists of number of appendices which cover the different aspects in detail, such as methodology and results.

Towards Optimization of Software V&V Activities in the Space Industry

Ehsan Ahmad, Bilal Raza

Dept. of Systems and Software Engineering
Blekinge Institute of Technology
SE – 372 25 Ronneby, Sweden

Abstract— Developing software for high-dependable space applications and systems is a formidable task. With new political and market pressures on the space industry to deliver more software at a lower cost, optimization of their methods and standards need to be investigated. The industry has to follow standards that strictly sets quality goals and prescribes engineering processes and methods to fulfill them. The overall goal of this study is to evaluate if current use of ECSS standards is cost efficient and if there are ways to make the process leaner while still maintaining the quality and to analyze if their V&V activities can be optimized. This paper presents results from two industrial case studies of companies in the European space industry that are following ECSS standards and have various V&V activities. The case studies reported on here focused on how the ECSS standards were used by the companies and how that affected their processes and how their V&V activities can be optimized.

I. INTRODUCTION

Software development projects for space applications and systems tend to have different dynamics than software projects in other domains. Development of software for space applications poses additional challenges due to its inherent requirement that the end product must be highly dependable. It is a formidable task to develop such systems because of the specific space operations and the focus on reliability and dependability, in particular reliable protocols also differentiate them from other systems [1].

The industry has a long tradition of developing standards that strictly sets quality goals and prescribes engineering processes and methods to fulfill them. The European Cooperation for Space Standardization (ECSS) has developed a single set of standards for the European space projects [2, 3]. These standards are derived from PSS-05, an earlier space standard which were more prescriptive, demanded heavy documentation and favored Waterfall and incremental development models [4]. Since PSS-05 was a primary input for ECSS, activities at space industry have the legacy of PSS-05.

The quality of software is very much dependent upon Verification and Validation Activities (VAs) [20]. Both state-of-the-art and state-of-the-practice has proved that using combination of different VAs is more effective than compared to a single VA [21, 22, 23]. According to [5], verification and validation of critical systems like satellites in a cost effective manner is a challenging task and optimal VAs are necessary to ensure quality by maximizing success in a limited budget. There is a need to optimize VAs by understanding the overlapping and variability between them, without losing quality. Defect detection completeness and successful integration of different autonomous subsystems are the major requirements for ensuring the quality of such systems. Each autonomous subsystem can be verified by different VAs. These inter and intra subsystem verification processes results in complex and multifaceted quality assurance process for the whole system.

The space industry like any other industry is currently evolving and constantly has to face new political and market pressures. The trend has been that the traditionally high quality requirements remain but with increasing demands for lower development costs and faster delivery times. This requires an in depth analysis about their VAs and their approaches towards ECSS standards.

This paper presents various challenges which the space industry is facing due to the demands of ECSS, especially in regards to the VAs. There are three main contributions of this paper. First, this case study presents the identified and prioritized challenges in VAs of different space organizations. Secondly, it presents the effects of ECCS standards on VAs and finally discusses proposed solutions.

The next includes the introduction about the two companies, while section III explains the design of the study. Section IV describes the results and analysis and section V discusses main challenges and issues.. Section VI explains challenge-cause analysis of each company, section VII outlines the solutions based upon identified challenges and section VIII concludes the study.

II. CASE COMPANIES

The research was conducted at two Swedish space companies which are developing software and hardware

for the space industry. Below is the brief introduction about them.

A. RUAG Aerospace Sweden AB

RUAG Aerospace Sweden AB (RUAG) was formerly known as SAAB Space AB but was recently acquired by RUAG Aerospace, and thus changed their name. RUAG has a very long and vast experience concerning design, development and delivery of both hardware and software for computer and data handling related products in space programs. The main product areas are Data management systems, Fault-tolerant computers and processor products, Payload control computers, Data processing and Small mass memories. The software developed by RUAG for these computers is in the range from small boot software to full application software, but the main focus is on embedded and real-time software. The software development process used is based on the ECSS standards, mixed with an integration driven development approach.

B. Space Division at Swedish Space Corporation

The Space Division at the Swedish Space Corporation (SSC) develops software and hardware for space applications, such as for example the satellites PRISMA, Small-GEO and SMART OLEV. They are a system integrator and supplier for small and micro-satellites. They are also specialized in developing attitude orbit and control systems, on board data handling units etc. In recent years they have changed their software processes to be more agile, by using Scrum as a project management model and Test-driven development as an engineering model [6, 7, 8]. In a recent report we have reported on the match between Agile software development methodologies and ECSS [9].

III. Design of the Study

A. Research Questions

In this study, we aim to answer following questions:

RQ1: What is efficiency of current V&V activities used in space industry?

By answering this question, we will be able to identify current VAs used in space industry and their efficiency. Defect logs and related documents will be analyzed for the purpose.

RQ2: What are the effects of ECSS standards on V&V process?

Companies developing software for European Space Agency (ESA) have to follow ECSS standards. Answer to this question will help us to understand the requirements of ECSS standards and how it affects quality of the software and efficiency of the software development team.

RQ3: What are the challenges in the V&V process of space industry?

RQ4: Is it possible to propose solutions for challenges identified in RQ3?

RQ5: Are the proposed solutions applicable?

B. Research Design

This study is part of a project lunched at RUAG Aerospace Sweden AB (RUAG) and Swedish Space Corporation (SSC) to create more efficient VAs, in general, and within ECSS projects, in particular. We have focused on the experiences the companies have had in their ECSS projects and VAs. To answer the above questions the study is organized in three steps; Preliminary Investigation, Analysis and Solutions and Solutions Evaluation. Figure 1 further explains the organization of the study in relation with research questions.

To increase the validity of the results we have used triangulation, i.e. a variety of research methods. We combined a questionnaire, document analysis and semi-structured interviews.

Web-based Questionnaire

A web-based questionnaire was administered to relevant personnel at the two case companies. The questions determined their role and activities, their knowledge and views on ECSS in particular and on VAs in general. A total of 37 respondents (18 at SSC and 19 at RUAG) answered the questionnaire. The answer frequency was 32.73% at SSC and 59.38% at RUAG, total of 42.53%. Partly this is explained by the fact that at SSC it was distributed more widely; some of the receivers might not have been the right target group.

Semi-structured Interviews

Semi-structured interviews were conducted with a total number of 17 interviewees (9 at SSC and 8 at RUAG). The interviews were between 45 and 80 minutes in length. One researcher posed questions from a prepared list and the other researcher recorded the interviews. The interviews were transcribed and individually summarized by the two researchers. They summarized the transcriptions independently and then discussed their results until consensus was reached. The criticality level for each of the issues and challenges uncovered, were judged on a scale from General, Important to Critical, based on how frequently it was mentioned by different respondents and how important they judged it to be.

Document Analysis

Documents like Software Development Plan (SDP), Software Verification and Validation Plan (SVVP) and Software Quality Assurance Plan (SQAP) of both the companies were analyzed. Initially, these documents

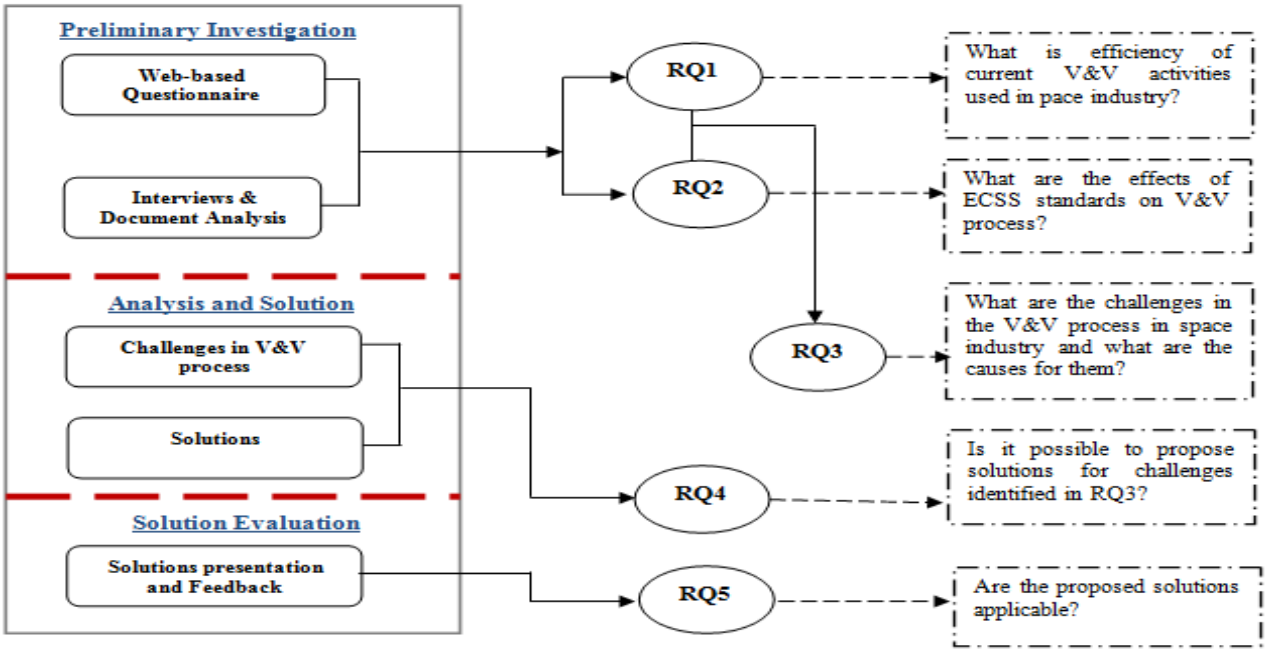

Fig 1: Research flow in relation to research questions

provided the basis for interviews and later they were complemented with the data of questionnaire and interviews.

IV. RESULTS AND ANALYSIS

A. *Web-based Questionnaire*

To compare questionnaire results of both companies, weighted average for each theme is calculated. Rest of the section presents this comparison in tabular form.

Theme 1: ECSS Standards

The theme1 of the survey is related to ECSS standards. The responses were given weightage from 1 to 5, where 1 being the lowest and 5 being the highest. Table I summarizes the results about theme1

TABLE I
THEME 1: ECSS STANDARDS

ECSS Issues	SSC	RUAG
Knowledge	2.1 (I know roughly what it is about – 2)	2.9 (I know its contents and how it affects the SWD activities – 3)
Effect on SWD	1.8 (Low – 2)	2.9 (High – 3)
Effect on SW quality	2.9 (Somewhat Positive – 3)	3.4 (Somewhat Positive – 3)
Effect on efficiency of SWD	2.4 (Somewhat Negative – 2)	2.0 (Somewhat Negative – 2)

It shows that there is a small difference in the knowledge distribution of ECSS among both companies, for SSC the average is more towards 'they know roughly what it is about' and for RUAG it is more towards 'they know its contents and how it affects the SWD activities'. SSC has an opinion that the effect of ECSS on SWD is low but RUAG thinks it is high. It also shows that both the companies agree that ECSS has 'positive effects' on the SWQ but the effects on the efficiency of SWD is 'somewhat negative'.

Theme 2: Effectiveness of VA's

The theme2 of survey is related to the effectiveness of VAs. The responses were given weightage from 1 to 4, where 1 being very ineffective and 4 being very effective. Table II summarizes the results about theme2.

TABLE II
THEME 2: EFFECTIVENESS OF VAS

VAs	SSC	RUAG
Requirement Review	3.0	3.1
Design Review	3.0	2.8
Code Review	2.7	3.4
Unit Testing	3.5	3.1
Integration Testing	3.5	2.7
System Testing	3.4	3.1
Validation Testing	3.0	3.3
Acceptance Testing	2.9	2.2

For RUAG the most effective VA's are 'code review' and 'validation testing' whereas for SSC 'unit testing' and 'integration testing' are more effective than other VAs. The least cost effective according to RUAG is acceptance testing whereas for SSC it is code review.

Theme 3: Effort required for VAs

The theme3 of survey is related to the effort required for VAs. The responses were given weightage from 1 to 4, where 1 being 'very low effort' and 4 being very high effort. Table III summarizes the results about theme3.

TABLE III
THEME 3: EFFORT REQUIRED FOR VAS

VAs	SSC	RUAG
Requirement Review	2.6	2.6
Design Review	2.6	2.4
Code Review	2.6	2.8
Unit Testing	3.1	3.2
Integration Testing	2.8	2.9
System Testing	3.3	3.6
Validation Testing	3	3.8
Acceptance Testing	2.8	2.7

For RUAG 'validation testing' and 'system testing' requires more effort compared to other VA's whereas for SSC it is system testing. Both the companies think that 'requirements review' and 'design review' requires less effort compared to other activities.

Theme 4: Change in effort for VAs if ECSS is not relevant

The theme4 of survey is related to their opinion about the change in effort for VA's, if ECSS is not relevant. The responses were given weightage from 1 to 4, where 1 being 'wouldn't do activity at all' and 4 being 'would put more effort on'. Table IV summarizes the results about theme4.

RUAG would like to put effort on 'integration testing' and less effort on 'acceptance testing' whereas SSC would like to put more effort on 'unit testing' and 'requirements review' and less effort on 'integration testing'.

B. *Semi-structured Interviews & Document analysis*

The results from the semi-structured interviews and document analysis are presented in section V as the main challenges and issues.

TABLE IV
THEME 4: CHANGE IN EFFORT FOR VAS, IF ECSS IS NOT RELEVANT

VAs	SSC	RUAG
Requirement Review	3.4	3.3
Design Review	3.1	3
Code Review	3	2.6
Unit Testing	3.4	2.5
Integration Testing	3.3	3.6
System Testing	3.1	2.7
Validation Testing	3	2.7
Acceptance Testing	3.1	2.1

V. CHALLENGES AND ISSUES

A. *ECSS Standards*

Table V summarizes the main issues and challenges about ECSS which were discovered during the case studies. They are sorted from most critical to less critical. The empty cells indicate it was not an issue or challenge for that particular company.

Reusability and ECSS standards

RUAG reuses document templates for ECSS between projects but have issues in reusing source code. ECSS allows for reusability but other requirements of the standards make it hard to actually reuse. This is because the reused part of the software has to be fully verified and validated in the new context. Sometimes this generates more work than actual implementation from the beginning. The European space industry have traditionally been skeptical towards the reuse of source code since that was one of the main causes of the Ariane 5 mid-air explosion [10]. However, there have been recent results in clarifying the situation and proposing updates to the standards [11]. It is not clear if and how these proposed updates affect cost effectiveness; high costs of compliance when reusing software defeats the purpose of reuse.

Both companies agree that since space projects are similar to their previous projects they can benefit a lot from reusing artifacts from the previous projects, but they are behind and need improvements in this regard. In case of COTS it is not justifiable to verify them as per the requirements of ECSS Q80 because they have been used by other companies and continuously verified and validated.

Documenting compliance consumes QA resources

A primary problem is that the high requirement on detailed documentation and proofs of standard compliance takes resources away from actually performing quality assurance and verification and

TABLE V
CHANGES / ISSUES RELATED TO ECSS STANDARDS

Ids	CHALLENGE / ISSUE	SSC	RUAG
Reusability	Documenting quality when reusing development artefacts	Critical	Critical
Resource-intensive	Showing compliance takes resources from increasing quality	Critical	Important
Interpretation	Difference in interpretation of ECSS	Important	Critical
Increments	Limited support for (integration-driven) development in increments		Critical
Galileo standard	Differences between ECSS and Galileo		Critical
Knowledge	Distribution of ECSS knowledge in organization	Critical	
Innovation	ECSS limits innovation in processes, methods and tools	General	Important
Inflexibility	Hard to make changes / introduce new requirements during project		Important
Requirements	Documenting requirements for compliance proof	Important	
Tailoring	Unclear how to tailor ECSS	General	

validation activities. This does not seem to be addressed by the ECSS standards update that is in progress. Rather the latest ECSS update, version C, is more detailed and specific on which processes and methods must be followed, how things are to be performed and requires more detailed documentation. It seems to be inspired by the Galileo Software Standard (GSWS) which, like ECSS, has extensive backing from European Space Agency (ESA). RUAG considers the GSWS to be more formal and prescriptive than the ECSS, thus having a potentially higher cost for compliance.

ECSS can be misused as a marketing tool when the developing organizations focus on certain activities just to show off they are fully compliant with it. This adds to the unnecessary cost as some of these activities don't affect the quality.

Interpretation Differences

Another problem with ECSS is that it can be interpreted differently by different people and organizations. Even though it creates a common understanding between customers and suppliers, this understanding is too dependent on the actual persons involved. For example, at RUAG, there have been problems when the different reviewers from the customer have different interpretations of what the ECSS standard requires. This has been a problem when the reviewers are changed during a project or when different reviewers reviews different parts of the project; much rework has been the result. This takes away resources that could have been put into increasing the quality of the software instead. The problem is even larger if we consider multiple projects. The interpretations and expectations on the ECSS compliance can vary a lot between projects even if the same prime customer is involved.

Incremental development and ECSS

It is difficult to work in increments when following ECSS. This is because of a legacy in the standards of a traditional, linear, waterfall process and because of the requirement on external reviews. As long as they follow the external reviews the customers allow an incremental development process. However, the external reviews limits the extent to which the increment-driven development can be used. For example, requirements have to be assembled early in the project for the external PDR, so the incremental approach can only be used for detailed design, implementation and testing, not for the requirements. Also, if the customers require a separate detailed design review, the same limitation applies to detailed design and further constrains the use of increments. So there is a mismatch with ECSS if they go away from a waterfall kind of process.

Differences with Galileo Software Standards

In some projects RUAG have been following the Galileo Software Standards instead of ECSS. GSWS are based on ECSS and can be considered a tailored version of ECSS. But it tailored parts of ECSS that was open to interpretation; thus it is less flexible. At RUAG, they consider GSWS to be stricter but clearer than ECSS. They are not as open to alternative approaches. By following GSWS RUAG has changed their internal processes so that they are now more ECSS compliant by default. One important difference between the standards is that GSWS requires independent module/unit testing; there is no such requirement in ECSS.

Knowledge distribution

At SSC the ECSS knowledge is unevenly distributed and concentrated on fewer individuals. The explanation for this is that SSC have more projects where ECSS compliance is not a requirement. Thus, developers are not always in ECSS projects and get less experience with it.

Even so, this can create tension between different projects and add to costs for showing compliance.

Effects on innovation

ECSS helps the companies making sure they do not miss important aspects, but the standards make it hard to introduce new processes, methods and tools. Primarily this is because a lot of activities done to show compliance do not affect the quality of the software, while still requiring lots of resources. Thus there is less time to consider and implement improvement. A standard, by its very nature, also restricts the introduction of unknown methods and tools. As an example from SSC, they are introducing model-driven development, with automated code creation from models, to increase productivity and quality. However, it is not an efficient use of resources to have to prove code coverage and verify automatically generated code.

Tailoring of ECSS

Tailoring of ECSS, according to project needs, is very important but sometimes it is already done for RUAG by their customers. They are at the lower level of recursive customer-supplier cycle. This is a drawback because their competitors can use it in other projects to their advantage. Sometimes they are allowed to deviate a little from ECSS, if the prime is confident in their work and has worked with them before and they have a good relationship. In that case they are able to focus on a lot of technical issues which further improves the quality.

Inability to do tailoring of ECSS generates a lot of work and extra costs. In case if customers are less technical and have less knowledge about ECSS they may ask to implement them as it is.

Inflexibility and documentation of requirements

At RUAG they find it harder to make changes to requirements during an ECSS project. SSC has successfully introduced more agile processes, also in their ECSS projects, and does not seem to have the same problems. However, a related problem at SSC is that they are not clear on how to document requirements and requirement changes in a way that compliance can be shown.

ECSS favors water fall development methodology. It has strict toll gates like preliminary design review (PDR) and detailed design review (DDR) and they don't allow implementation before these reviews. RUAG wants to have DDR in small steps focusing on parts which are to be implemented and in some projects they have successfully been able to do DDR and implementation in parallel, depending upon the customers.

B. VAs in practice

Table VI summarizes the challenges and issues regarding VAs in Practice

Unstable and non-testable requirements

Both companies have issues in writing good requirements; they have certain guidelines to follow at that level. Functional and non functional requirements are more or less mixed when they get them from the customers who often tend to forget some of the nonfunctional requirements as well. There are three levels of requirements at both companies. Mission level or equipment level comes from the customer, which are then broken down to system level requirements. The system level requirements are then forwarded by the systems manager to the developers who further break them into implementation or unit level requirements. It is always an issue to pass information from one level to another due to communication gaps.

Defects in testing environment and tools

Both companies have had problems with in-house tools. Sometimes it requires too much effort to improve them according to project requirements. They use the same environment and tools for other projects as well, but with certain changes. The main components are the same but these changes require a lot of effort. They also find defects in their testing environment and tools and do not generate software problem reports (SPRs) about them.

During the verification and validation they find defects in the software due to these environments and tools, sometimes they develop the tools and software in parallel.

Limited focus on integration testing of software components

At RUAG they are not very good at integration testing of software modules. They combine them with the validation testing of hardware. One of the reasons is they are mostly working with hardware drivers and in these cases it is more efficient to test the integration of software modules at the validation. But at the same time they also built full application software, which requires testing them separately.

Inadequate internal formal reviews and inspection

RUAG is very good at reviews and inspection and they would even like to put more effort. It is the most cost effective activity for them. The downside is the dependence upon the person doing it and the list of issues they check keeps on updating. SSC doesn't focus too much on reviews or inspection. They have very informal checklists and have very limited reviews and inspection.

SSC values dynamic VA's more. This is one of the main differences in the approaches of the two case companies. But they don't have figures or measurements to back this up that which activity is more effective in finding defects in a cost efficient way.

More focus on structural coverage than Black box testing

In both companies, mostly the unit testing is done by the developer himself. This also depends upon the requirements from their customers. At RUAG, they focus

TABLE VI
CHANGES / ISSUES RELATED TO VAS IN PRACTICE

Ids	Challenge / Issue	SSC	RUAG
Requirements	Unstable and non-testable requirements	Critical	Critical
Testing Environment	Defects in testing environment and tools	Important	Critical
Integration Testing	Limited focus on integration testing of software components		Critical
Reviews and Inspection	Inadequate internal formal reviews and inspection	Critical	General
Unit Testing	More focus on structural coverage than Black box testing		Important
Independent V&V	Test cases are not reviewed by independent developer/ tester	Critical	

highly on coverage statistics. The focus of tests is more towards structural coverage and white box testing. Another problem is they are looking at what code does instead of looking at what the code should do and test that. SSC is now using more of test driven development so that the person writing the code will start by writing the unit tests and will focus more on black box perspective.

Test cases are not reviewed independently

At SSC, the developers are responsible for the development and testing of units and there is no independent verification of code or test cases at this stage. There are chances that defects may be missed and slipped onto later stages. RUAG has an independent review on the code and sometimes they also have reviews for test cases, depending upon the requirements from customers. There are pros and cons of having complete independent verification. There can have communication issues and loss of information between the people.

C. *Efficiency of VAs*

Table VII summarizes Issues/challenges regarding efficiency of VAs

Insufficient measurements

Both SSC and RUAG have insufficient measurements in their VAs. They don't measure the efficiency of different activities and don't even calculate the number of defects found at different levels. They are not good at measuring the results of what they are doing and why they are doing it. They don't have a formal list which says this category of faults should be detected at this stage and so on. But they have more or less an implicit list for doing unit testing.

Although, both companies do not have any measurements, but RUAG have an opinion that more defects are found in inspection and reviews than in module testing. They spent a lot of time during unit testing and its efficiency is low. However, SSC thinks that unit testing brings more value. These companies are interested in having metrics which are easy to use and follow up but the problem is the lack of statistics to be used in them.

Faults-slip-through among different stages

At SSC they find defects which are not local at the specific stage because the testing environment is not fully representative of the target environment and it takes extra time and effort to complete the activity. At RUAG, they focus too much on coverage figures during unit testing. Other reasons for faults to slip through are tight schedules during the project. In such circumstances they move away from optimal ways of doing things and rush through them.

They don't estimate the cost of finding defects in different phases. A metric could be obtained through reporting system or by expert judgments. If they evaluate the costs of finding defects in different phases then an improvement potential can be determined by calculating the cost of finding defects in specific phases to the cost of finding the similar defects in other phases. Hence, a total improvement potential can be calculated.

Vague classification of defects

The classification of defects in both companies is very vague. It is based upon the severity which is dependent upon the person classifying the defects. They use different tools and reporting systems for this. In large projects they cover the same things again and again and their processes check the similar things at different stages, which increases the cost and don't improve quality.

There is no clearly defined strategy about what kind of defects should be captured at which stage. There is no mapping between what type of tests a phase should cover and which faults should be found when executing those tests.

V&V experts are involved to the later stages only

V&V experts are not involved in early stages of the project in both companies. This may result in unstable requirements and cause problems later in the project as the developers at unit level don't have the full picture and often the project managers don't have the clear idea about the technical constraints. If the validation team is involved in early stages it would have been easier to validate and even reuse things. There are different levels of requirements involving different teams and there are communication issues between them.

V&V experts are considered second class engineers and are not involved actively in the early stages of the software development.

Inappropriate time for initial framework

At SSC, they don't focus too much on requirements review and spend less time during the requirements phase. Sometimes they implement things which don't connect all the way up to the requirements. They also make experienced guesses while implementation.

VI. Challenge-Cause Analysis

To better understand the dependencies among challenges and to find their causes a challenge-cause analysis is performed using Current-Reality Tree (CRT). CRT considers multiple challenges at the same time and is very helpful in improving system by identifying the root causes of those challenges. The identified challenges are called Un-desirable Effects (UDEs) are then traced to root causes. Figure 2 represents the CRT diagram for SSC while Figure 3 represents the CRT diagram of RUAG. The boxes with yellow backgrounds are identified as UDEs.

VII. Recommendations Based On Identified Challenges

This section discusses the recommendations for how the identified challenges can be addressed. By the help of challenge-cause analysis we concluded that both the organizations are facing problems due to three main causes; ECSS Standards, Faults-slip-through among different stages and Inappropriate selection of cost-effective VAs. Rest of this section discusses the recommendation made by authors to cope with the identified challenges.

A. ECSS Standards

Table VIII summarizes the recommended solutions for different stakeholders in the ECSS standards. For each challenge we list the solutions in three different categories based upon their relevance for: Development organization (RUAG and SSC in this case), Customer (ESA or other organization stating the requirements for a development project) and ECSS (the standards body).

B. Faults-slip-through measurement

Early involvement of V&V experts at both companies can improve their requirements phase. SSC should focus more on reviews and inspections and having appropriate time for initial framework will make the requirements more stable. Faults-slip-through among different stages is one of the biggest challenge for both the case companies and the main reasons are defects in the testing environment and inappropriate selection of VA's. In RUAG they focus more on coverage figures, by following test-driven development they can have focus on the black-box perspective and there will be less chances than faults will slip through to the later stages. ECSS on one hand improves the quality and at the same time it takes away many of the resources from quality by prescribing certain activities which doesn't have any positive impact such as documents and proofs that certain VAs are performed accordingly. Insufficient measurements and vague classification of defects also have an impact on faults to slip through. Both the companies are looking for: simple measurements so they may evaluate the efficiency of their VAs, the improvement potential they can have and a method by which they can have a combination of VAs to ensure that defects are covered. Lars-Ola et al. [19] proposed a method at Ericsson for faults-slip-through measurements which has three steps, in the first step a strategy is developed about what should be tested at which phase. This will have a direct mapping about what type of faults a certain phase should cover. In the second step, average cost of finding defects in various stages is determined; this can be obtained through the reporting system or by expert judgments. In the third step, an improvement potential is determined by calculating the difference between the costs of finding defects at the stage they were found to the cost of finding defects from the stage where they slipped through. The approach described the definitions and instructions about how to apply and follow up on the measurements. But the

TABLE VII
Challenges / Issues related to Efficiency of VAs

Ids	Challenge / Issue	SSC	RUAG
Measurements	Insufficient measurements	Critical	Critical
Faults-slip- through	Faults-slip-through among different stages	Critical	Critical
Defect Classification	Vague classification of defects	Important	Critical
Involvement V&V experts	V&V experts are involved to the later stages only	Critical	Important
Initial framework	Inappropriate time for initial framework	General	

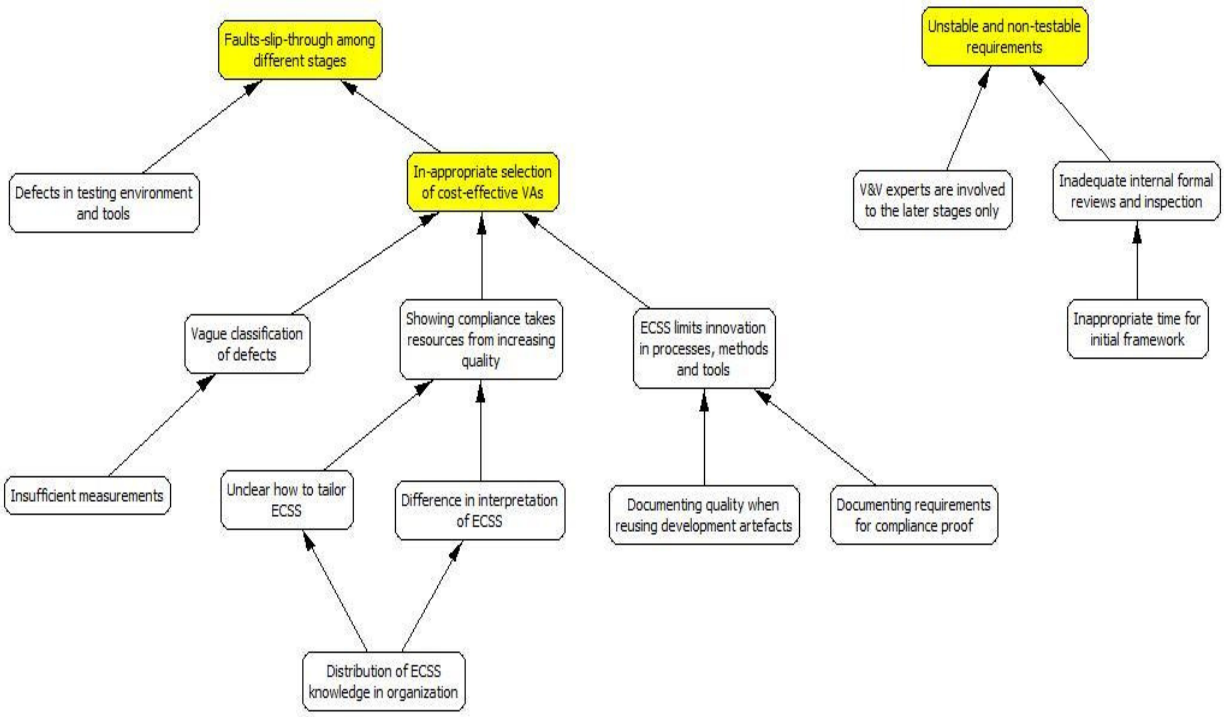

Fig. 2: Challenge-cause analysis for SSC

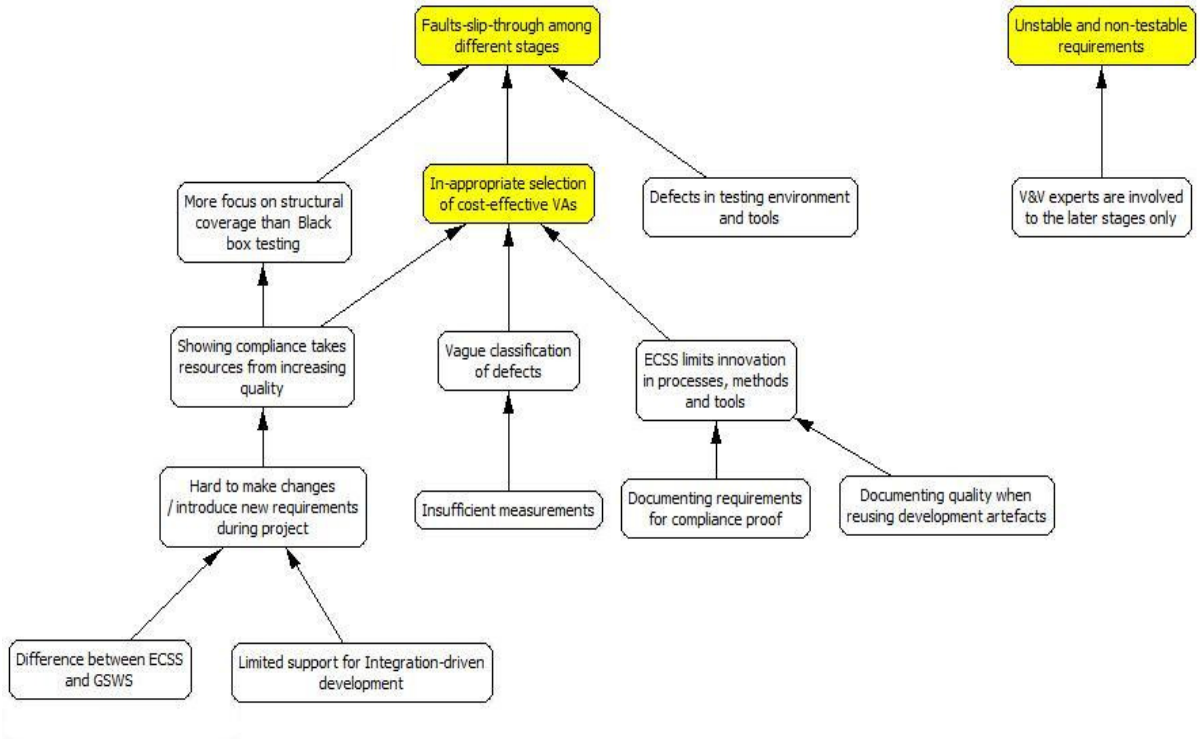

Fig. 3: Challenge-cause analysis for RUAG

Pre-requisite for applying this method is a strategy and classification of defects and measurements about the cost of finding defects at various stages.

C. Strategy for selection of Cost-effective VAs

Current Reality Tree (CRT) also shows that both companies are facing problems due to inappropriate selection of VAs at different stages of software development life cycle. For example, at SSC, the developer himself performs unit testing of the code which can cause the faults to slip through to the next stage. On the other hand at RUAG, code inspection is performed by the independent person at unit level to ensure full structural coverage, but they are lacking in integration testing. Both the companies are facing problems in identifying the appropriate VAs at different stages and there is a need for a strategy to select appropriate VAs.

There are some strategies focusing on the selection of combination of VAs. Baret el al. [12] used the idea of mapping matrix for optimizing the testing process. The matrix is filled by placing VAs and defect types in rows and columns, respectively. If any VA has the ability to detect a specific defect type, then the cell representing that VA (row) and defect type (column) is marked with "X". Wagner's model of quality economics presents cost versus benefit analysis by using more detailed metrics and equations [13, 14]. This model requires a lot of data initially and cannot be a candidate strategy for the selection of cost-effective VAs because of lack in data at both companies. Murnane et al in [15], presented a method for the selection of test cases by tailoring Black box testing. The limitation of this method is its only focus on one aspect of V&V process. Combination framework by Bradbury et al [16] focuses on mutation for defects and automated VAs and does not provide any guideline for the selection of VAs.

A significant number of interviewees also mentioned that defect detection completeness is one of the main requirement for the space industry and optimal VAs are required to achieve quality in a limited budget.

Strooper et al Model [17]

The strategy presented by Strooper et al [17] for the selection of cost-effective VAs is a candidate solution. It aims at selecting and evaluating different combinations of VAs by focusing on maximizing completeness and minimizing effort thus reducing cost and enhancing the efficiency, in four steps. Systematic way of applying empirical information makes this strategy a competitive approach. The strategy analyzes different combination of VAs by exploring effort and defect detection effectiveness of the VAs, iteratively. Each iteration determines whether the particular combination produces expected results or adjustments should be made. This model has four steps;

- Step 1: Pre-selection: Collect cost-effect information
- Step 2: Argument 1, Maximize completeness
- Step 3: Argument 2, Minimize effort
- Step 4: Post-Selection, Updating Cost-effective information

Following are some reasons for the selection of this strategy

Maximizes completeness and minimizes effort

Step 2 of this model requires the combinational selection of VAs to ensure that all the defects are covered by VAs. Step 3 ensures to minimize effort by selecting the combination of VAs which requires minimum effort from the combinations selected in step 2.

Supports decision by empirical information

Both SSC and RUAG don't have enough initial data in terms of defect logging and efficiency of VAs. This model is flexible enough as it can be initiated by educated guesses of V&V experts but they will have empirical data right after the first iteration. This data can also be used to determine whether the selection produced expected results or not.

Scalability

The model is flexible to be used at any stage of V&V process i.e. unit, integration or system level.

VIII. Validity threats

To increase the validity of results we have used triangulation, i.e. a variety of research methods. The results are based upon the combination of questionnaire, document analysis and semi structured interviews. There were two researchers involved in it who worked independently and discussed the results. External Validity is a valid threat for this research because the case companies are based in Sweden and have relatively small software divisions. They may have different perception about ECSS and have different issues in their V&V activities, from other companies. A small survey at other companies can improve the external validity of this research. At SSC, the survey questionnaire was sent to broad set of persons this might be one of the reason that they showed less knowledge of ECSS standards.

IX. Conclusions

This paper describes the results of two industrial case studies of companies in the European space industry. Based on a triangulated research method using three sources of data it presented the issues and challenges that were identified. It described the possible ways that the main stakeholders, the developing organizations, the customers and the ECSS standards organization, can work together to address these challenges. It also discussed the

TABLE VIII
RECOMMENDED SOLUTIONS FOR DIFFERENT STAKEHOLDERS AND CHALLENGES

Challenge Id	Developing Organization	Customer	ECSS
Reusability			Clarify reuse of artefacts and VAs, Evaluate cost effectiveness
Resource-intensive	Document inefficiency & overlap between activities		
Interpretation		Designate ECSS authority for each project	
Increments	Describe alternative processes that works & point out ECSS mismatches	Allow process experimentation	Allow alternative Processes, Consider simpler/quicker evolution process
Galileo standard			Clarify relationship between ECSS & other common standards & motivate differences
Knowledge	Ensure broad ECSS knowledge even when non-ECSS projects are run in parallel	Ensure coherent ECSS knowledge among reviewers and among projects	Develop lightweight ECSS teaching materials
Innovation	Try alternatives in non-ECSS projects first & consider ECSS when evaluating them		Ensure the standard is primarily goal-driven & only secondary method prescriptive, Clarify explicitly for big trends like model driven how they can be incorporated
Inflexibility			Extend to agile and alternative processes
Requirements	Show alternative ways to document requirements for compliance		Evaluate alternatives for documentation
Tailoring		Describe requirements on tailoring documentation	Clarify how to tailor and document tailoring

possible ways forward to reach the goal of creating more cost-effective verification and validation activities framework for the space industry.

ACKNOWLEDGMENT

This work has been supported by the Swedish National Space Board.

REFERENCES

[1] C. Mazza, “Standards: the foundation for Space IT,” in Workshop on Space Information Technology in the 21st Century. Darmstadt, Germany: European Space Operations, September 2000.

[2] European Cooperation for Space Standardization, “ECSS-S-ST-00C - Description, implementation and general requirements.” ESAESTEC, Requirements & Standards Division, July 2008.

[3] L. Balestra, “European Cooperation for Space Standardization (ECSS),” in Trilateral Safety & Mission Assurance Conference (TRISMAC 2008), April 14–16 2008

[4] M. Jones, Mortensen, and J. Fairclough, “The ESA Software Engineering Standards: Past, Present and Future,” in Proceedings of the 3rd International Software Engineering Standards Symposium (ISESS ’97). Washington, DC, USA: IEEE Computer Society, 1997, p.119.

[5] Brat. G, Denney. E, Giannakopoulou. D, Frank. J, Jonsson. A, “Verification of Autonomous Systems for Space Applications”, in Proceedings of IEEE Aerospace Conference (NASA Ames Res. Centre, Moffett Field,CA, USA, 04 - 11 March 2007), IEEE Computer Society, Washington, DC. USA.

[6] A. Cockburn, Agile Software Development. Addison-Wesley Professional, 2002.

[7] D. Astels, Test Driven development: A Practical Guide. Prentice Hall Professional Technical Reference, 2003.

[8] K. Schwaber, Agile Software Development with Scrum. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[9] B. Raza, E. Ahmad, R. Feldt, and T. Nordeb¨ack, “ECSS Standard Compliant Agile Development for Dependable Space Software – an Industrial Case Study,” 2008, in submission.

[10] J. Lions, “Ariane 5 Flight 501 Failure – Report by the Inquiry Board,” Tech. Rep., 1996.

[11] M. Rodriguez, J. G. Silva, P. Rodriguez-Dapena, H. van Loon, and F. Aldea-Montero, Reuse of Existing Software in Space Projects Proposed Approach and Extensions to Product Assurance and Software Engineering Standards, 2005, pp. 258–267

[12] N. Barret, S. Martin, and C. Dislis, "Test Process Optimization: Closing the Gap in the Defect Spectrum," In Proceedings of the International test conference, 1999, pp. 124-129

[13] S. Wagner, "Modelling the Quality Economics of Defect-Detection Techniques," In Proceedings of the International Conference on Software Engineering, 2006, pp. 69-74

[14] S. Wagner, "Software Quality Economics for Combining Defect-Detection Techniques," In Proceedings of the Net.Object Days 2005 Workshop on Software Quality (WOSQ'06), 2006, pp. 69-74. (Node'05), 2006, pp. 559-574.

[15] T. Murnane, K. Reed, and R. Hall, "Tailoring of Black-Box Testing Methods," In Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06), 2006, pp. 292-299.

[16] J. S. Bradbury, J. R. Cordy, and J. Dingel, "An Empirical Framework for Comparing Effectiveness of Testing and Property-Based Formal Analysis," In Proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, 2005, pp. 2-5

[17] Margaret A. Wojcicki, Paul Strooper, “An Iterative Empirical Strategy for the Systematic Selection of a Combination of Verification and Validation Technologies”, in Proceedings of 29th International Conference on Software Engineering (Minneapols, MN, USA, 20 - 26 May 2007), ICSE 2007, IEEE Computer Society, Washington, DC. USA.

[18] Brat. G, Denney. E, Giannakopoulou. D, Frank. J, Jonsson. A, “Verification of Autonomous Systems for Space Applications”, in

Proceedings of IEEE Aerospace Conference (NASA Ames Res. Centre, Moffett Field,CA, USA, 04 - 11 March 2007), IEEE Computer Society, Washington, DC. USA.

[19] L.-O. Damm, L. Lundberg, and C. Wohlin, "Faults-slip-through - a concept for measuring the efficiency of the test process," Software Process: Improvement and Practice, vol. 11, no. 1, pp. 47–59, 2006.

[20] Rakitin, S, (2001) Software Verification and Validation for Practitioners and Managers (Second Edition), Artech House , London, UK.

[21] J. Myers. "A controlled experiment in program, testing and code walkthroughs inspections". Communications of the ACM, vol. 21/9, September 1978, ACM, New York, NY, USA.

[22] R. W. Selby. "Combining software testing strategies: An empirical evaluation", in *Proceedings of the Workshop on Software Testing (Banff, Canada, July 1986)*, IEEE Computer Society, Washington, DC. USA.

[23] M. Wood, M. Roper, A. Brooks, J. Miller. "Comparing and combining software defect detection techniques: a replicated empirical study", in *Proceedings of 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering (Zurich, Switzerland 22-25 September 1997),* ESEC '97/FSE-5, Springer-Verlag, New York, USA.

Appendix A

Approach used in design of the study, is discussed in this appendix. It explains the techniques used for data collection and motivation behind the selection these techniques

A.1 Design of Study

This study is part of a project at RUAG Aerospace Sweden AB (RUAG) and Swedish Space Corporation (SSC) to create more efficient verification activities (VAs) in general, and within ECSS projects in particular. The research was conducted using mixed methodology for inquiring problems and verifying results. Different strategies (document analysis, semi-structured interviews and web-based questionnaire) used in this research complemented each other and each technique provided input and feedback for the next strategy to further explore the area of study. To answer the research questions the study was organized in three steps; Preliminary Investigation, Analysis and Solution Identification and Solutions Evaluation. Figure A-1 further explains the research flow of the study organization of the study in relation with artefacts of each phase. Triangulation i.e. a variety of research methods also increase the validity of the results we have used triangulation, i.e. a variety of research methods.

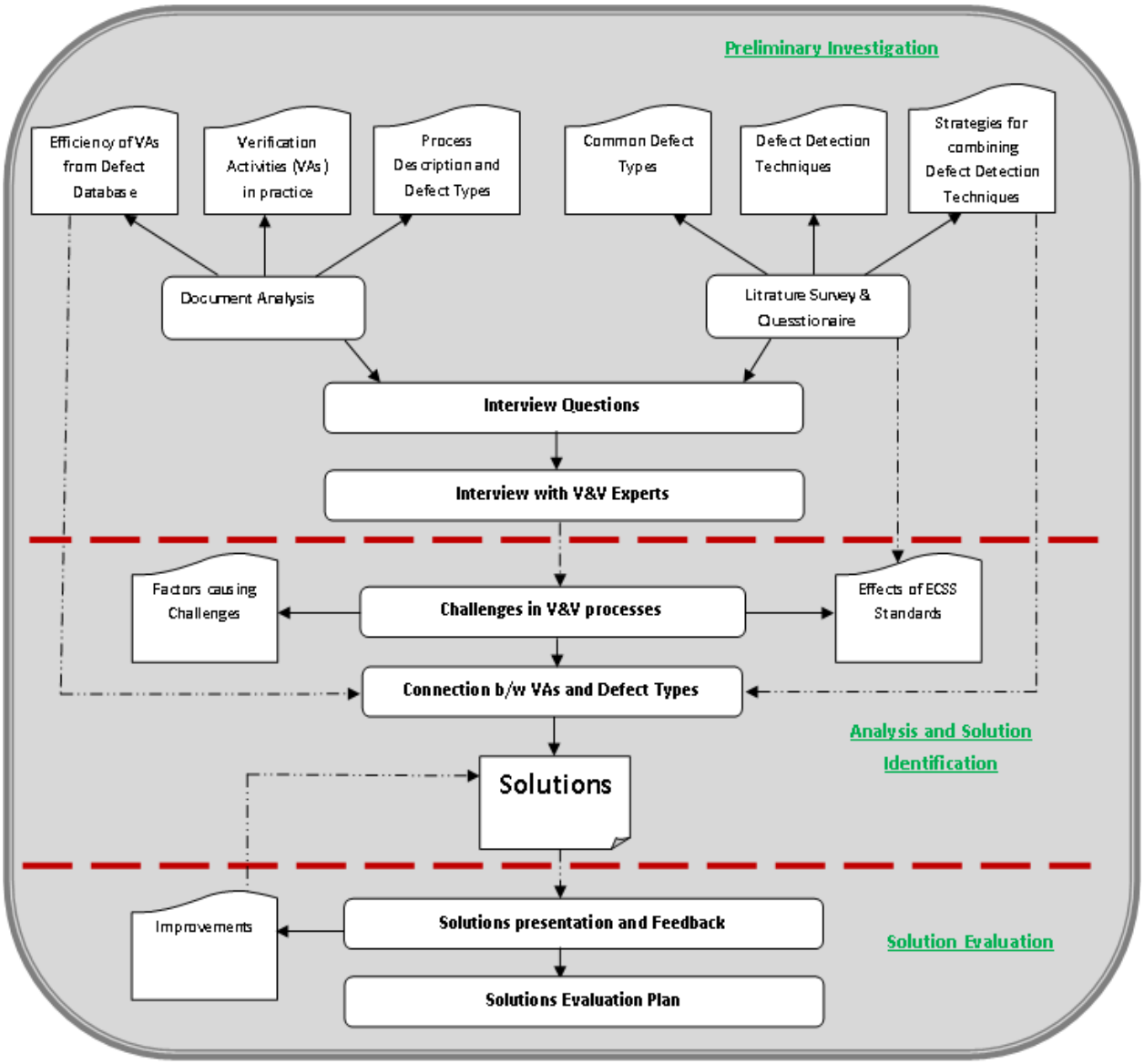

Figure A-1: Organization of Research

A.2 Web-based Questionnaire

A Web-based questionnaire was conducted for both the companies. The questionnaire contained both general and technical questions. The general questions were targeted for at personal information whereas the technical questions were focused on their experience in software development, V&V process, and knowledge about ECSS standards. For better understanding the technical questions were organized into following four themes:

- **Theme 1:** ECSS Standards
- **Theme 2:** Effectiveness of VAs
- **Theme 3:** Effort required for VAs
- **Theme 4:** Change in VAs, if ECSS is not relevant

Following Table [IX], contains the questions asked in web-based questionnaire

Table IX: Organization of web-based questionnaire

Themes	Questions
General	What is your name?
	What is your age?
	Which part of the company / project you are a part of?
	Which area of Software development are you involved in?
	How many years have you worked within software development?
	How many years have you worked software testing / verification and/ or validation
ECSS Standards	What is your knowledge about ECSS standards?
	To what degree ECSS affect how you develop software?
	Is the effect of ECSS on the quality of your software?
	Is the effect of ECSS on the efficiency of your software development
Effectiveness of VAs	For each of the verification activities that you have been involved in, please stated how effective in finding defects they are?
	How large a percentage of the total development costs do you judge that all verification/validation/testing activities take?
Effort required for VAs	For each of the verification activities that you have been involved in, please stated how much effort they require?
Change in VAs, if ECSS in not relevant	For each of the following activities if you only consider its effect on software quality and you are not required to follow the ECSS standards, what would you change?
	Do you have any general comments on Verification Activities (VAs) of your company? Any comments about the ECSS standards? What would be most important for your company to improve when it comes to VAs, software quality and/ or ECSS?

A.3 Semi-Structured Interviews

According to [11, 12], review of theoretical knowledge and published practices must be conducted along with industry observations to find out the commonalities of a specific problem. Therefore, semi-structured interviews were planned for both the case companies. The questions for the interviews were based upon the results of the already conducted web-based questionnaire for VAs experts at RUAG and SSC. The interviews helped in getting insight about the variations, artefacts, and complexities of the state of the practice at both companies. The interviews were transcribed and individually summarized by both the authors. By discussing the summaries while merging them a consensus and higher validity was planned to be achieved. like web-based questionnaire interviews questions were also divided into different themes:

- Theme 1: VAs in practice
- Theme 2: Effectiveness of VAs
- Theme 3: ECSS Standards

Following Table [X], contains the questions designed for the interviews

Table X: Organization of Interview Questions

Themes	Questions
General	What is your name?
	What is your age?
	Can you briefly describe your previous experience regarding testing, verification and/or validation?
	Which of the projects you have been involved in at SSC / RUAG?
VAs in Practice	Can you briefly describe verification and validation activities at SSC / RUAG?
	Have you been involved in VAs in any other company? If yes? How do you differentiate it from VAs activities at SSC/RUAG?
	Do you log all types of defects? Are there any which you don't log?
	How do you design test case and scripts? Which technique do you follow?
	Do you document anything about unit testing? If not why?
	Which verification method do you use in verification level/stage you are involved in?
	What technique do you use in each verification method that you are involved in? And Why?
	How do you perform Inspection?

	Which type of defects do you find by each technique?
	Why do you introduce Test-driven development at SSC?
	What are the effects of using Test-driven development on ECSS standards?
	Why do you introduce Integration-driven development at RUAG?
	What are the effects of using Integration-driven development on ECSS standards?
Effectiveness of VAs	Do you calculate the effectiveness of a particular verification method/technique? If yes how?
	Do you think more effort should be placed on metrics to measure the efficiency and effectiveness of different VAs?
	Since you are working with critical systems, how does it affect verification process?
	Do you reuse V&V plans? Especially those which are not in compliance with ECSS?
	Do you think your early involvement in SDLC can be helpful in improving your performance?
ECSS Standards	What is your knowledge about ECSS standards?
	When did you start following ECSS standards?
	How does ECSS affect your software development process?
	Which parts of the ECSS standards are you following? Why are you following only them? Why are you not following the other parts?
	Which parts of ECSS standards, you think can be avoided? And why?
	What are the positive effects of ECSS standards?
	What are the negative effects of ECSS standards? What can be done to reduce negative effects of ECSS standards?
	Which ECSS verification level you are working in? (Equipment, Subsystem, Element, System)
	Which ECSS verification stage is relevant to you? (Qualification, Acceptance, Pre-launch, In-Orbit, Post-launch)
	What are you future plans about ECSS at SSC / RUAG?

Appendix B

This appendix provides information about the European Space Agency (ESA) and European Corporation for Space of Standardization (ECSS)

B.1 Introduction

The European Cooperation for Space Standardization (ECSS) is an initiative to develop a coherent, single set of user-friendly standards for use in all European space activities [13].
In 1993 the European Space Agency (ESA) along with other national space agencies and industries realized the need of a single coherent, recognized and accepted system of standards to replace the practice-based PSS-05 standard [21]. The first document of this new system of standards, named European Cooperation for Space Standardization (ECSS), was introduced in 1996. The idea is that ECSS standards should be continuously created and updated to adapt to changing needs of the industry. A revision process started in 2006 and two batches of updates were released in 2008. Further batches will be published in 2009.

The revised system differentiates between standards, handbooks and technical memoranda. ECSS standards divide activities into three areas: management, engineering and product assurance and has four levels:

- Level 0 – discusses policy, architecture and objectives of ECSS
- Level 1 – describes the strategy within management, product assurance and engineering by highlighting the requirements and interfaces of level 2 elements
- Level 2 – explains objectives and functions of each domain. It is considered as branch-specific level
- Level 3 – lists methods, procedures and tools to achieve the requirement of the level 2 documents. It is also known as technical domain specific level.

The intention of ECSS presented in [21], can be summarized as:

- Requirements on project level instead of organizational level, as an organization can have multiple projects simultaneously
- Flexible to be used with other quality standards
- Can be applied as a whole or partially if the connection with other standards is possible
- Provides tailoring options

B.2 ECSS-E-40 (IB, 2B)

ECSS-E-40 (Part 1B and 2B) and ECSS-Q-80B are related to software. ECSS-E-40 is based on ISO 12207 and allows suppliers to define their own standards, which are in compliance with or tailored to it [18, 20].

- The requirements of ECSS-E-40, at a very high level, can be summarized as follows:
- Customer itself is responsible for system requirement engineering and review

- Supplier is responsible for software requirement analysis, architectural design and Preliminary Design Review (PDR)
- Supplier is responsible for software design, coding, unit testing, integration testing and Critical Design Review (CDR)
- Verification and validation of PDR, technical aspects of CDR is conducted by supplier in its own environment
- Software delivery and Site Acceptance Test (SAT) is carried out by supplier in the operational environment supplied by the customer
- Supplier is responsible for selecting software development process and required resources

This standard is based on a recursive concept of customer supplier relationship. A customer at one level can be a supplier for a higher level. According to clause 4 of ECSS-E-40 Part 1B [17], customer is responsible for defining both functional and performance requirements, interface between software components and interface between software and hardware while the supplier is supposed to maintain the interface with the customer to ensure the proper consideration of higher level system requirements.

B.3 ECSS-Q-80B

ECSS-Q-80B defines requirements for product quality assurance to ensure that the software development produces safe and reliable software [18, 19]. The requirements of this standard can be summarized as:

- Supplier has to develop and maintain a comprehensive product assurance plan with focus on continuous process assessment and improvement
- Selection of software development process according to ECSS-E-40 Part 1B, must be assured by fulfilling the requirements of each phase
- Requirements to assure the quality of final software product by indentifying quality attributes, measureable quality objectives and set of metrics to verify quality objectives

Appendix C

This appendix provides information about Swedish Space Corporation. It provides the summaries of interview, survey and document analysis conducted for data collection at Swedish Space Corporation.

C.1 Introduction

SSC was established in 1972 by the Swedish Government which is completely owned by the Swedish state and administered by the Ministry of Enterprise, Energy and Communications. In 1978, the Swedish Space Corporation launched its satellite operations, and currently they have expanded considerably. They have various satellite stations and more than a dozen parabolic antennae for communicating with orbiting satellites [13].

The group has 560 employees and it has five different facilities in Sweden

- Solna (head office and engineering center)
- Esrange Space Center, Kiruna (launch and test services and satellite communication)
- Vidsel (test services for aerospace systems)
- Ågesta (teleport services)
- Salmijärvi (satellite operations for ESA) (SSC website)

The company also has an office in Beijing, China. It is also the full-owner of four subsidiaries with facilities in Uppsala Sweden, Germany, France and Chile.

Currently, SSC is working on the following projects related to satellite systems which are at different stages: PRISMA, SMALLGEO, ODIN, PROBA-3, and SMART-OLEV. SSC has also been involved in various projects and activities related to MARITIME SURVEILLANCE, BALLOONS, ROCKETS and SATELLITE SERVICES [13]. Their future projects include personal sub orbital flights. Spaceport Sweden and Virgin Galactic plan to offer trips into near space from Kiruna Airport as early as 2012. The space travellers will, amongst other things, experience 4-5 minutes of weightlessness while enjoying the astonishing view of Earth [13].

The Space Division at the Swedish Space Corporation (SSC) develops software and hardware for space applications, such as for example the satellites PRISMA, Small-GEO and SMART OLEV. They are a system integrator and supplier for small and micro-satellites. They are also specialized in developing attitude orbit and control systems, on board data handling units etc. In recent years they have changed their software processes to be more agile, by using Scrum as a project management model and Test-driven development as an engineering model. The case study was conducted at the space division of SSC.

C.2 Data Collection

C.2.1 Document Analysis Results

Since OBSW and AOCS/GNC are developed by different teams and different V&V plans are developed by AOCS or OBSW teams. For ESA projects, V&V plans are then authorized by the primary contractors and are distributed to ESA as well. Change history is maintained for each V&V plan in terms of versions.

C.2.1.1 OBSW Verification and Validation Plan (PRISMA, SMART-OLEV)

OBSW is responsible for controlling the spacecraft throughout the mission. The software executes on SSC developed Leon-based Core board. The OBSW provides functionality for GNC, telecommand & telemetry, thermal & power control, payload control, autonomy, and redundancy & fault management. Different components are developed by different teams. For PRISMA the responsibility split can be summarized as under:

- GNC Team – responsible for GNC, ACS, and ORB cores.
- CNED – develops part of GNC core. GNC is responsible for integration and test of those parts into the GNC core.
- DLR – develops part of GNC core and GIF component contained in XU. GNC team is responsible of the parts in GNC core and OBSW team for the part in XU component.
- OBSW Team – responsible for all development and test of all parts of OBSW, except those mentioned above. And for integration and building of complete OBSW.

For SMART-OLEV only two teams are defined:

- GNC Team – Responsible for GNC Core.
- OBS Team – Responsible for all development and test of all parts of OBS, except GNC Core, and for integration and building of complete On-Board software.

C.2.1.1.1 OBSW Test Process and Methods

The incremental approach is used for the development and testing. All the requirements in project (except GNC unit requirements) are stored in Telelogic DOORS.

Requirements are written on three levels:

- System Level – behavior and requirement for the whole system.
- Subsystem Level – requirements for satellite subsystems.
- Unit Level – implementation requirements for different components that are derived from the subsystem level.

The division into subsystem is functional, which means that different parts are implemented in software. Only requirements on subsystem and unit levels are covered by test activities described in this document. System level requirements are verified by AIV, AIT and BTM. Test activities are based upon requirements and mission representatives to obtain coverage of implicit requirements and validate functionality from a real-world perspective. Verification is done on both unit and subsystem level in all development teams. This is important, for example since test activities on subsystem level assumes that unit level requirements are fulfilled and likewise for system tests. Validation is only needed on

subsystem and system level. Testing activities has carried in the following sequence in each increment:

1. New functionality and unit test is done.
2. Following three steps are repeated until sufficient quality has been delivered or deadline for release is reached
    a. OBSW team builds the complete software based upon version information from development teams
    b. System and integration tests are performed
    c. Defect corrections and unit test are done
3. Release documents are updated and software is released

C.2.1.1.2 Test Management Tools

- **Telelogic DOORS**

The DOORS requirement management system can also be used for test case specification and simple test case result documentation. The major pros for this approach are connection between requirement and test cases are easily specified and that the system is already in use and there is no use for installation or learning.

But if it is used more extensively the list of cons becomes further longer. For example there is no support for requirement coverage analysis, nice structuring of test cases and execution is almost impossible, all attributes for test cases/executions/ results must be created manually and this is not built in defect management system. This is due to the fact that DOORS in not really intended for test management.

- **Mercury Quality Centre**

Mercury Quality Centre (MQC) is a test management system with support for the following tasks:

- Requirement management
- Writing test case specifications
- Planning test case execution
- Reporting and analyzing test result and progress
- Managing defects

The requirements are managed in DOORS. But since it is necessary to connect requirements to test cases/test and analyze requirement coverage etc the requirements are copied from DOORS to MQC.

C.2.1.1.3 Test Environment

- **MATLAB/Simulink**

Most of the ASW software is developed in Simulink with the addition of Real-time Workshop. This is graphical environment that supplies possibility for simulations, interface to software written in other languages and generation of C code. Due to the possibility of simulations the system is also heavily used for test and prototyping. Input and output can be supplied / obtained either via the graphical interface or via special implementations of the interface to software components outside Simulink. The prior is typically used to test ASW cores (implementation is contained in Simulink) and the latter for test of other ASW components (have interface to BSW which is implemented in C).

- **PRISMA SATLAB**

SATLAB is a real-time test environment developed by SSC that contains the following parts:

- Onboard Core Boards – this can be seen as the test object. A core board can be configured to be either Main or Target
- Satellite Simulator (SATSIM) – A real-time simulator running models of the sensors, actuators and space environment for Target and Main
- RAMSES – part of the electric ground support system that is developed in parallel to OBSW and shall be used for PRISMA operations. Contains a PLUTO interpreter that shall be used for test automation
- Networks and Adapters – used to connect different parts of the system and substitute the RF link between and spacecraft

The usage of TM/TC unit on Target core board is workaround for better testability. This TM/TC unit is not used in flight. Instead it is replaced by the Inter Satellite Link (ISL) and all telemetry/telecommands is transmitted / received through the main satellites TM/TC unit. Three different SATLAB configurations are specified here as under:

- **PRISMA SATLAB configuration 1**

Only core board is connected and is alternated between Main or Target configuration. Whether the core board shall be in nominal or redundant mode not specified and switching between them is encouraged to increase target environment configuration coverage. This is test environment can be used for many OBSW system test cases and due to lack of core it is believed to be used for a majority of them.

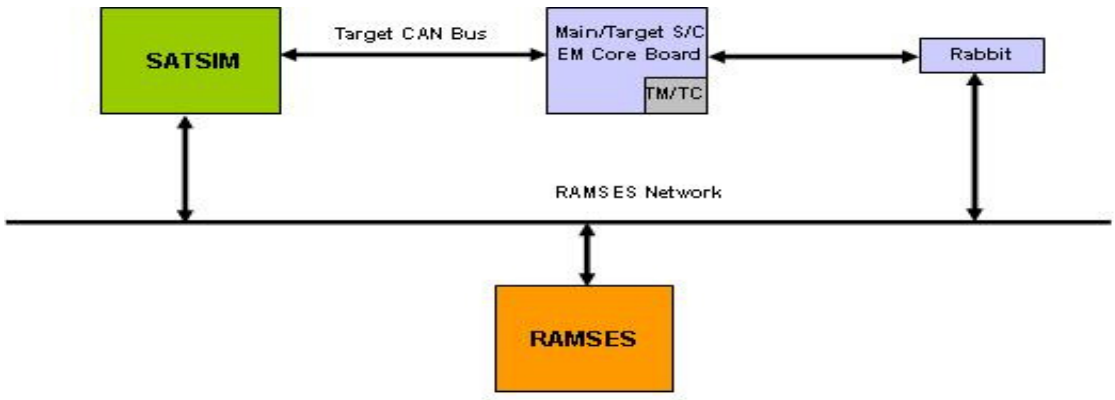

Figure 5. SATLAB configuration 1 [22]

- **PRISMA SATLAB Configuration 2**

Two core boards are connected. One configured as Main and the other as Target. They are connected via SATSIM, which simulates the ISL communication. Nominal/redundant settings are unspecified. This environment shall be used when verification of cooperation between Main and Target is included in the test case. It can also be used to obtain a better similarity with real world situation ain many other test cases, for example transmitting telemetry and telecommands over ISL instead of TM/TC unit on Target.

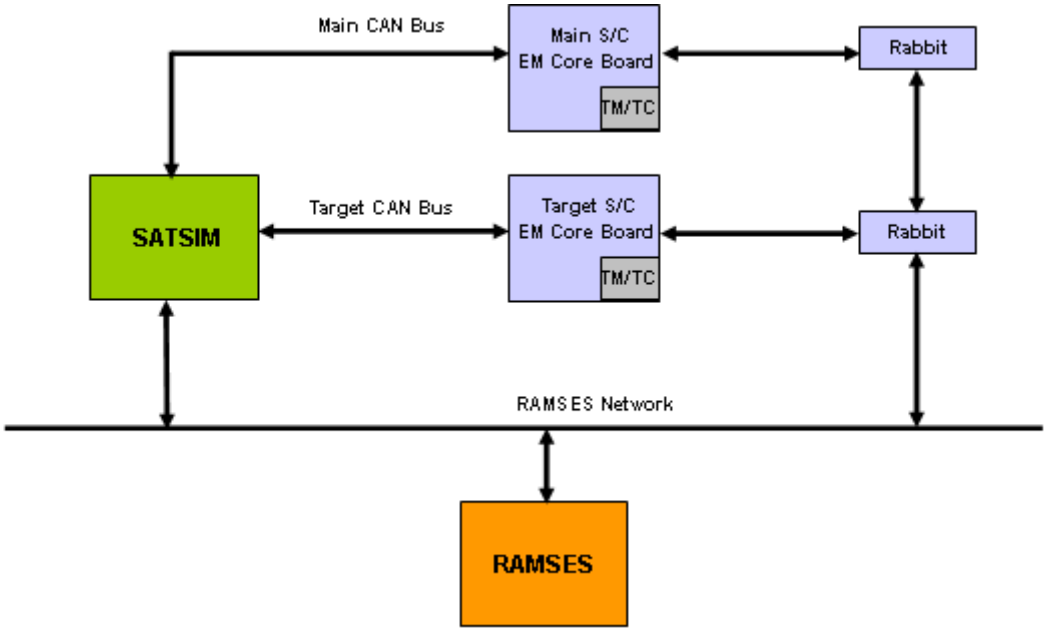

Figure 6. SATLAB configuration 2 [22]

- **PRISMA SATLAB Configuration 3**

Two core boards are connected. Both are alternated between Main and Target configuration. If one core board is used as nominal then the other one is used as redundant. They are connected via Spacewire link, which is equal to the satellite installation. The environment must be used for the best cases that verifies nominal/redundant core boards redundancy, for example when mass memory is located on redundant core board and OBSW is running on nominal.

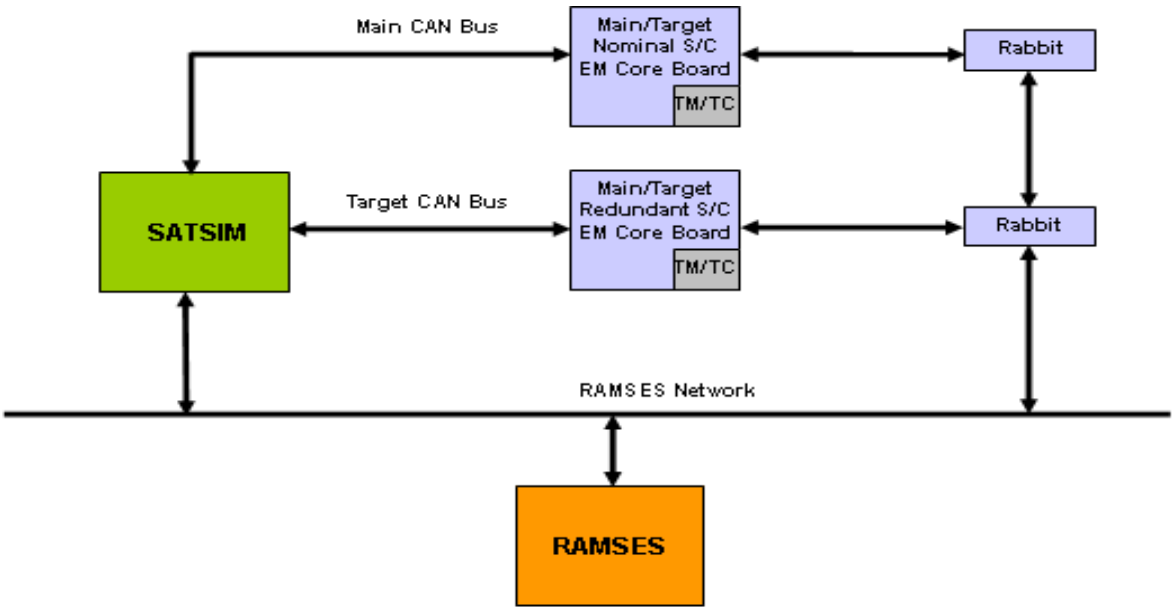

Figure 7. SATLAB configuration 3 [22]

- **SMART-OLEV SATSIM (PIL)**

SATSIM is SSC developed PIL and HIL system that has been extensively used in the PRISMA mission. SATSIM consists of:

- *Onboard Core Board* – This can be seen as the test object, running the OBSW
- *Environment Simulator* – A real-time simulator running models of the sensors, actuators and space environment
- *RAMSES* – Part of the electric ground support system that is developed by SSC. Contains a PLUTO interpreter that can be used for test automation

SATSIM will be used for early OBS verification and validation. In parallel the OSTF will be developed and SATSIM will in system phases act as a Independent Software Validation (ISV) facility.

- **SMART-OLEV OSTF (PIL)**

OSTF is a real-time test environment developed by Sener that contains the following parts:

- Onboard Core Board (Provided by SSC) – This can be seen as the test object
- Satellite Simulator (Made by SENER) – A real-time simulator running models of the sensors, actuators and space environment
- SCOS2000 – Part of the electric ground support system that shall be used for SMART-OLEV operations.
- Networks and Adapters – Used to connect different parts of the system and substitute the RF link between ground and spacecraft

- **SMART-OLEV OSTF HIL**

HIL incorporates several hardware units, sensors and Rendezvous and Docking Payload.

- **Tools**

CANalyzer can be used to monitor and generate CAN traffic. It runs on a standard PC and can be connected to any CAN bus via a CAN controller. *GRMON* is used to load, run and debug OBSW on core board from a PC, as an alternative to program software in EEPROM. A serial debug port on core board is used to connect the PC.

Summary of software testing performed in each increment is written to provide an overview of software status/quality. The document includes OBSW system and integration tests. Some examples of issues to be included:

- References to tools and documents where detailed test reports can be found
- Summary of each executed test case, including result and problem report references
- Summary of test cases that have not been executed, including a motivation for lack of execution
- Verification status of all applicable requirement
- Amount of time spent on verification and validation

This document is written once for each increment. Its purpose is to summarize test execution and result for one increment so that the reader obtains basic knowledge of software quality. Tests on previous versions within same increment might also be included since they unveil new problems and enhancements.

The document is called “Test Report” and stored in DOX. The same document is used and updated in each increment.

C.2.1.1.4 Regression Tests

The following test cases are executed for each new release/increment:

- Unit test cases for updated sub-products
- Resolved SPRs shall be verified
- Resolved NCR shall be verified
- All automated system and integration test cases.
- “Health checks” for all cores, e.g. Enter GNC Safe Mode

C.2.1.1.5 OBSW Unit Test Plan

OBSW has two main components BSW and ASW. The environment, approach and tools differ between these components.

- **Basic Software – BSW:**

Objective of BSW unit test plan is to the verification of BSW unit requirements. Two approaches are used for BSW unit testing:

- Manual ad hoc testing – most tests are done ad hoc and only test important enough to be regression test are formalized into test specifications
- Documented and automated regression tests – the goal is to automate a regression test suite by using the dedicated test task

For SMART-OLEV following additional test approaches are planned

- Board Support Package (BSP) tests – Tests are performed on flight-representative LEON hardware, to verify BSP unit requirements.
- BSW Application Layer test – Tests are performed using a stubbed BSP on a LEON instruction simulator or commercial LEON board, to verify BSW-AL unit requirements. All tests are automated and code coverage is measured.
- BSW Integration Testing – The BSP and BSP Application layer are tested together on flight-representative hardware to verify the SW/HW integration. Tests are automated when possible.

BSW also delivers some third party components that are not explicitly tested by OBSW team, for example the operating system RTEMS. Mercury Quality Center is used to specify test cases, document execution and found defects. All tests are performed in an environment similar to SATLAB. An extra task is added to the regular onboard software in compile time. The test task can receive commands and send results to/from a debug port on core board which in turn is connected to a PC which acts as test console.

- **Application Software – ASW**

Objective of BSW unit test plan is to the verification of ASW unit requirements. Three approaches are used for BSW unit testing:

- *Repeatable tests* – all tests must be repeatable, i.e. the sequential steps needed to rerun the test and to verify the result are documented or scripted. Whenever possible, tests are automated (both execution and verification).
- *Block box testing* – whenever possible, black box testing shall be done. That is, no modifications of the test object shall be necessary to perform the tests. A test harness shall be built that will feed the test object with stimuli and that will read outputs needed for verification.
- *Maximum execution time* – to ensure that a component can meet its time deadline and not consume too much of CPU resources, max-path tests in a LEON3 simulator are needed. For such tests, a test harness is built around the component that will execute maximum path taken by the component (i.e. the maximum execution time). The test harness is compiled and downloaded to a LEON3 instruction simulator and the resulting number of instruction cycles is returned.

Test cases and results are specified in a dedicated DOORS module and remaining defects shall be entered in Mercury Quality Center. Tests are primarily run in MATLAB/Simulink on a developer desktop. If this is not possible they are run in SATLAB environment. For SMART-OLEV Simulink Verification and Validation Toolbox is used for model coverage measurements while TSIM-LEON3 is used for testing of ASW hand-written code.

Testing of OBS components shall be done through a test harness that will provide the component with stimuli and read output.

The Stimuli are Telecommand packets, CAN data messages, Data Store inputs and additional BSW queue inputs (Spacewire buffers etc), if needed. The output can be Telemetry packets, CAN command messages, Data Store outputs and additional BSW queue output, if needed. The actual test procedure is executed through a MATLAB M-Script. This script prepares workspace variables needed by the test harness to feed the component with stimuli. When the simulation has finished, the script will read the generated output variables and automatically verify the data determine the test case is passed or failed. There is a master test script that calls all component test script.

C.2.1.1.6 OBSW System Test Plan

Objective for OBSW system is to verify correctness of the software developed by OBSW team. This can be by verification of all testable subsystem requirements and validation of the OBSW system via mission representatives, duration and high load (stress) scenarios.

OBSW system test activities are as under:

- Execution in target environment – tests shall be execution in an environment as similar to target as possible in respect in respect to processor board, EGSE and CAN bus data. For example EM core board and RAMSES system shall be used. The official SDB will also be used as much as possible.

- Black box testing – only black box tests shall be done. That is, no modifications or insight into the test object shall be necessary to perform the tests.

- Automated and manual execution – as many test cases as possible shall be both automatically and manually executable. There are always some test cases that are very rarely executed or for which the results cannot be clearly defined and should therefore not be automated. And some that requires a huge amount of events and verifications and cannot be manually executable. But in every where both alternatives are possible they shall be supported.

- Main and Target are tested separately – most functional OBS requirements are independent of main-target interactions and the two are therefore tested separately. Note though that some tests requires both satellites to be included which will then be done.

Mercury Quality Center is used to specify test cases, document execution and found defects. All tests will be performed in any of the SATLAB test environments specified. For implementation and execution of automated test cases the PLUTO interpreter included in RAMSES system is used.

Test case design may vary slightly depending on from perspective they are written, but all the test cases contains Procedure / sequence of events, Expected results / acceptance criteria, Short description of the purpose and preconditions for the test case, and Specification and motivation of whether the test case shall be atomized or not.

Test cases are specified upon the following perspectives:

- Based on Requirements – each test case aims to verify one, or couple of, level 2 requirement (s). Test procedures are defined in terms of what shall be done and observed.

- Based on Mission Representatives – test cases aim to verify that a flow event that may occur in real-world situation. Test case specifications are less detailed to allow for tester to execute it a way that he/she feels natural. Of course all scenarios that may occur cannot be tested in this way and those selected shall be motivated by importance or frequency of occurrence. A lot of test cases will be taken from System Functional and Performance Test (SFPT), which can be seen as a validation test for the software. Execution of these test cases is important since it will likely avoid delays in validation test campaign. Execution of the validation tests in system test environment is also standard procedure in software testing.

- Stress Tests – these test cases are similar to those based on mission representatives in that they try to catch a real-world scenario. On the other hand they do not have to be important or frequent; their only aim is to find circumstances when the software crashes or fails to meet its goal. Error cases that does not occur in nominal conditions, for example overload on CAN bus is also be included.

- Duration Tests – shall verify that the software functions remain constant overtime. A number of different scenarios are continued and verified over a long duration of time, typically a few days.

- Core Integration Tests – purpose of those tests are to verify that core ICD's are correctly implemented and that the specified input/output values are fetched from source (e.g. CAN bus, TM, other cores) correctly. This is a dynamic test that is carried out through by setting input/output and verification of telemetry/CAN bus.

C.2.1.1.7 Risks and Contingency approaches

Some contingency approaches are defined for OBSW system testing activities have been identified. Quality problems or delays in the RAMSES project. In particular the tool for execution of automated testes, Cheops, is very critical. A way to minimize impact of Cheops problems is to make all system test cases manually executable as well as automated. Quality problems or delays in SATLAB development. If SATLAB cannot be used when needed CANalyzer is an alternative way of generating CAN traffic. Tests of compliance with level 2 requirements in DOORS and ICD's shall be done and documented. System performance shall be measured and wide range of extensive real world scenarios shall have been executed in real-time. A small number of defects are accepted, if they are well investigated, not serve

and well documented. The deliverables of these testing activities are test case specifications, scripts for automated execution of test cases, defect reports for found software problems, execution reports for test case execution and a summary of software status. Logging of data during test executions is made on two buses via the following tools included in the test environment. Onboard CAN Traffic is logged via CANalyzer. RAMSES is used to log all traffic that is accessible in ground system, i.e. telemetry, telecommands, events and internal RAMSES data such as sync and acknowledged. Two applications are supplied by RAMSES for this purpose. Anubis logs all telemetry and Toth is used for other RAMSES traffic.

C.2.1.1.8 OBSW Integration Test Plan

As different parts of the software are developed by different teams some test campaigns dedicated to integration of the software is needed. Such test campaigns are described in this section. Primary objectives are to verify that those parts of that the software developed by OBSW team integrates correctly with its surrounding, e.g. ground segment and external units. Secondary objectives are to verify that those parts of the onboard software that are developed by other teams can be executed without crashes or other deterioration of the software environment (this is done since OBSW team is ultimately responsible for the onboard software). And to verify the Main-Target ISL link, which cannot be fully tested in system test since it tests the software separately.

There is no need for a dedicated RAMSES test campaign since it is used through system tests. Mercury Quality Center is used to specify test cases, document execution and found defects. SATLAB is used for as a testing environment. Since some tests include hardware that is not accessible in SATLAB it will probably be necessary to use AIV environment. Test object is a formal build of the complete onboard software for both Main and Target. GNC/ACS cores and DLR's GIF module are included build, but verification of them is excluded. Some important integration test campaigns are as under:

- Embedded Software Conformance Test – Objectives of this test activity are to verify that all embedded software modules that are developed by other parties are "Well behaved". Examples of things that are verified is that they don't use too much CPU time or generate software reboots. Test is done for GNC (Main), ORB (Main), DLR GIF (Main), and ACS (Target).

- Ground Segment Software Integration Test – These tests shall verify compatibility with specific ground segments tools that are not part of the standard ground segment equipment, i.e. RAMSES. The reason why there is no RAMSES integration is that such compatibility is tested in all other test campaigns. This test campaign is done for patching tool testing.

- Core Consistency Check – Purpose of this check is to verify that the inputs/outputs of course are consistent with respect to data types, number of elements in vectors etc. This is a static test and is carried out through inspection of ICD's.

- Payload Hardware Integration Test – These tests intend to verify data flow to and from different onboard instruments. That data flows are compatible with ICDs for the instruments are verified in different system tests, but this one focuses on compatibility with actual hardware instrument and specific ground tools, when applicable. This test campaign is used for Star Tracker, Visual Based Sensor, GPS and FFRF.

C.2.1.2 SMALL-GEO AOCS Verification and Validation Plan

In order to meet SGEO AOCS project needs, improve the efficiency of software integration and provide the early evidence that SGEO AOCS Software will be acceptable; the incremental and iterative software development process based on well-proven ECSS and Scrum standards has been chosen.

C.2.1.2.1 OBSW Test Process, Methods and Roles

- **SGEO AOCS SW Product Owner**

The SGEO AOCS SW Product Owner has the following responsibilities:

- Represent all stakeholders' and present their requirements on verification and validation
- Define verification and validation objectives and strategies
- Decide test completeness criteria for Sprints' deliverables
- Define Sprints' goals (inclusive testing goals and test coverage)
- Accept or reject the Team's verification and validation results (Sprint deliveries)
- Support the Team and Scrum Master in the day-to-day testing activities
- Control status of NCRs/ RFDs/ RFWs/ RIDs/ SPRs/ SCRs/Actions

- **SGEO AOCS SW Scrum Master**

The SGEO AOCS SW Scrum Master (SM) has the following responsibilities:

- Employ cross-functional team, including verification and validation experience and competence
- Inform and train project members in software verification and validation matters.
- See to that the Team understands and works towards verification and validation objectives
- Support the Team in the day to day testing work
- Describe terms, standards, procedures for verification and validation
- Implement measure and report software testing coverage (metrics) to SGEO AOCS PO
- Suggest and perform actions to effectuate verification and validation activities
- Ensure that the test documentation contains the appropriate level of information for maintenance activities
- Justify the use of software testing tools

- Keep track and report to Product Owner status of NCRs, RFDs, RFWs, SPRs, SCRs, RIDs, Actions
- Produce status accounting reports for Software Problem Reports/Software Change Requests (SPR/SCR)

- **SGEO AOCS SW Team**

The SGEO AOCS SW Team has the following responsibilities:

- Define methods, rules, templates and tools for software verification and validation.
- Design and develop the test suits (test scenarios, procedures, scripts, harness). To enhance quality, unit tests will in general be performed independently of the development.
- Perform verification and validation in accordance with a strategy for each testing level (unit, integration, validation, and acceptance)
- Participate to Daily Scrum (Stand-Up meetings) and display verification and validation progress and status on daily basis
- Demonstrate testing results at Sprint demo
- Report impediments, NCRs, RFDs, RFWs, SPRs, SCRs, RIDs
- Apply defined standards and procedures in all verification and validation stages. Not reducing products' internal quality and creating technical debt.
- Produce test documentation that contains the appropriate level of information for regression testing and maintenance activities.
- Suggest and perform actions to effectuate testing work

- **Test Approach**

Three types of test are performed:

- SGEO AOC Core Software Static Checks – SGEO AOC Core design guidelines are implemented in the Simulink Verification and Validation modeling standards checker, which can find unwanted model properties, such as incorrect or deprecated blocks and block parameters, incorrect fonts, and misplaced objects. Analysis reports can be generated from the tool.
- SGEO AOC Core Software Unit Test –Unit test has as its only goal to verify unit requirements and to obtain test coverage of the model in a Simulink environment.

- SGEO AOC Core Software Integration Test –Integration test aims to guarantee that the software that is excluded in system test still can be integrated correctly into SGEO OBSW without CPU overload or crashes.

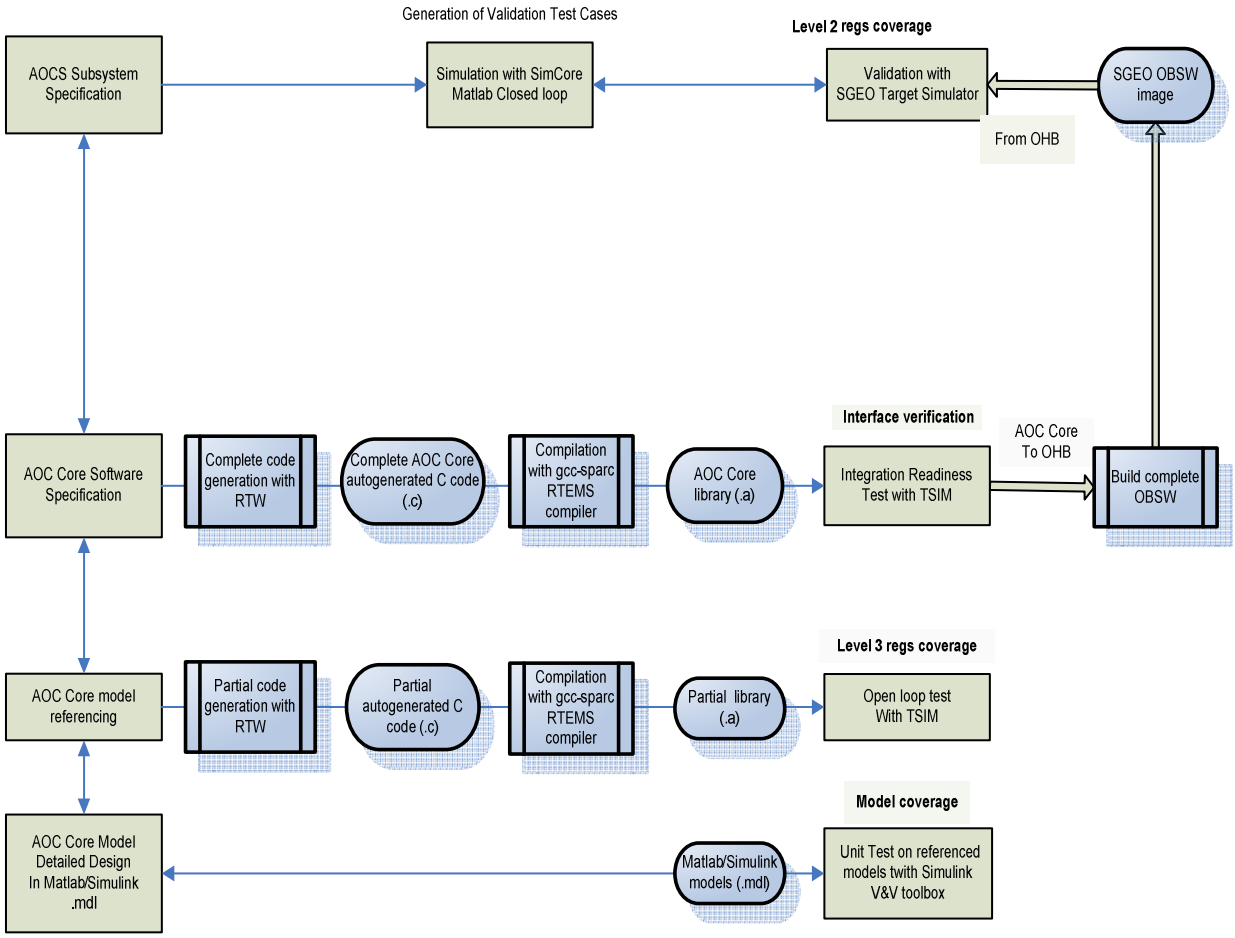

Figure 8: SGEO AOC Core Development and Test logic [22]

C.2.1.2.2 Test Management Tools

- **Telelogic DOORS**

  DOORS is used for requirement like OBSW.

- **Simulink Verification and Validation Toolbox**

The model based design paradigm encourages the developers to closely link requirements to models. The Simulink V&V toolbox provides a handy requirements management interface (RMI) that allows designer to associate requirements with Simulink models, subsystems and blocks, as well as with State flow charts, states, transitions, boxes and functions.

Requirements stored in Telelogic DOORS can be linked to Simulink and State flow models using this RMI.

- **Hansoft project manager**

Hansoft project manager QA part is planned to be used for managing defects. The bug workflow in Hansoft will be customized to fulfill project needs.

C.2.1.2.3 Test Environments

- **MATLAB / Simulink**

AOCS Core software is developed in Simulink. Simulink is a tool for modeling, simulating and analyzing multi-domain dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can both drive MATLAB or be scripted from it. Coupled with Real-Time Workshop, another product from The MathWorks. Simulink can automatically generate C code for real-time implementation of systems or digital controllers.

Simulink Validation and Verification toolbox as well as internally developed test tools are used to run open-loop tests, i.e. input-output vectors tests in Matlab/Simulink. These tests can be executed both on unit level and Core level. Simulink Validation and Verification toolbox will also be used to provide model coverage tests.

- **Target Simulator (TSIM)**

GAPSLER Research's TSIM is an instruction-level simulator capable of emulating ERC32- and LEON-based computer systems. The tests designed and executed in the Matlab/Simulink environment is rerun on TSIM and the outputs from either test can be compared. TSIM can be connected to a GNU gdb debugger.

C.2.1.2.4 Regression Tests

The following test cases are executed for each new release/increment:

- Unit test cases for updated models.
- All automated integration test cases.

C.2.1.2.5 SOFTWARE UNIT TEST PLAN

Unit test has as its only goal to verify unit requirements and to obtain test coverage of the model in a target emulator environment, i.e. TSIM. The units are implemented as model reference blocks in Simulink, thereby ensuring that the code generated from a unit will be the same as the code generated later for the entire flight code.

The unit tests are used to verify the 3rd level requirements applicable to the AOCS software.

C.2.1.2.6 Test Objects

Test items are the code generated from AOC Core sub models, i.e. functional units implemented as referenced models. Examples of such functional units are mode handler; GYRO-, RWA-, SADM-, SPS-, ST-signal conditioning functions; safe pointing control functions; celestial attitude and angular momentum estimation functions; momentum management functions; guidance functions etc.

C.2.1.2.7 Test Objectives

- Verification of all testable AOC Core requirements (3rd level requirements applicable to the AOC Core software) for each referenced model.
- 100% model coverage of referenced model, deviations from this goal are to be motivated

C.2.1.2.8 Test Approach

The tests are constructed as a Simulink "test harness" model first that may take source data from a file as input. The test is then run on the TSIM target emulator. It shall be possible to execute the tests automatically. Each test shall consist of an expected result output sequence to which the test result shall be autonomously compared for a pass/fail check of the result. The test shall be "model free" in the sense that the test shall not require any closed loop simulation of on-board functions in combination with simulation models.

All the unit tests are executed as an autonomous "batch" job before any release of the AOCS software.

- **For every Sprint**
  - Create test cases in Simulink to verify all testable requirements implemented in this Sprint.
  - Run test cases on unit model.
  - Compile test report.
  - Review design within the team to ensure that the Coding Style Guidelines is followed.
- **In the eve of major AOCS software delivery:**
  - Create test cases in Simulink to ensure maximum module coverage,
  - Run test cases using Simulink Verification & Validation toolbox
  - Justify/explain deviations from 100% model coverage
- **In the eve of major AOCS software delivery**
  - Generate code from the Simulink unit test script for TSIM.
  - Generate TSIM test harness.
  - Re-run unit test on TSIM.
  - Evaluate results against the results in the Simulink test cases.
  - Compile test report.

- **As input to SGEO S/W analysis**
  - Generate max path test cases from for TSIM.
  - Generate TSIM test harness.
  - Run unit test on TSIM.
  - Compile test report.

C.2.1.2.9 Applicable Requirements

The following requirements are verified by AOC Core unit test:

- AOC Core software requirements
- AOC Core internal ICDs

C.2.1.2.10 Test Environment

MATLAB/Simulink on Intel Win-NT systems is used for AOC Core model testing. The Simulink environment consists of Matlab, Simulink, State flow, RTW, State flow Coder and Simulink Verification &Validation toolbox. Unit tests will be re-run on TSIM.

C.2.1.2.11 Test Case Design

Testing of SGEO AOC Core components are done through a test harness that will provide the component with stimuli and read outputs.

Following figure C-5 explains AOCS Unit test design process

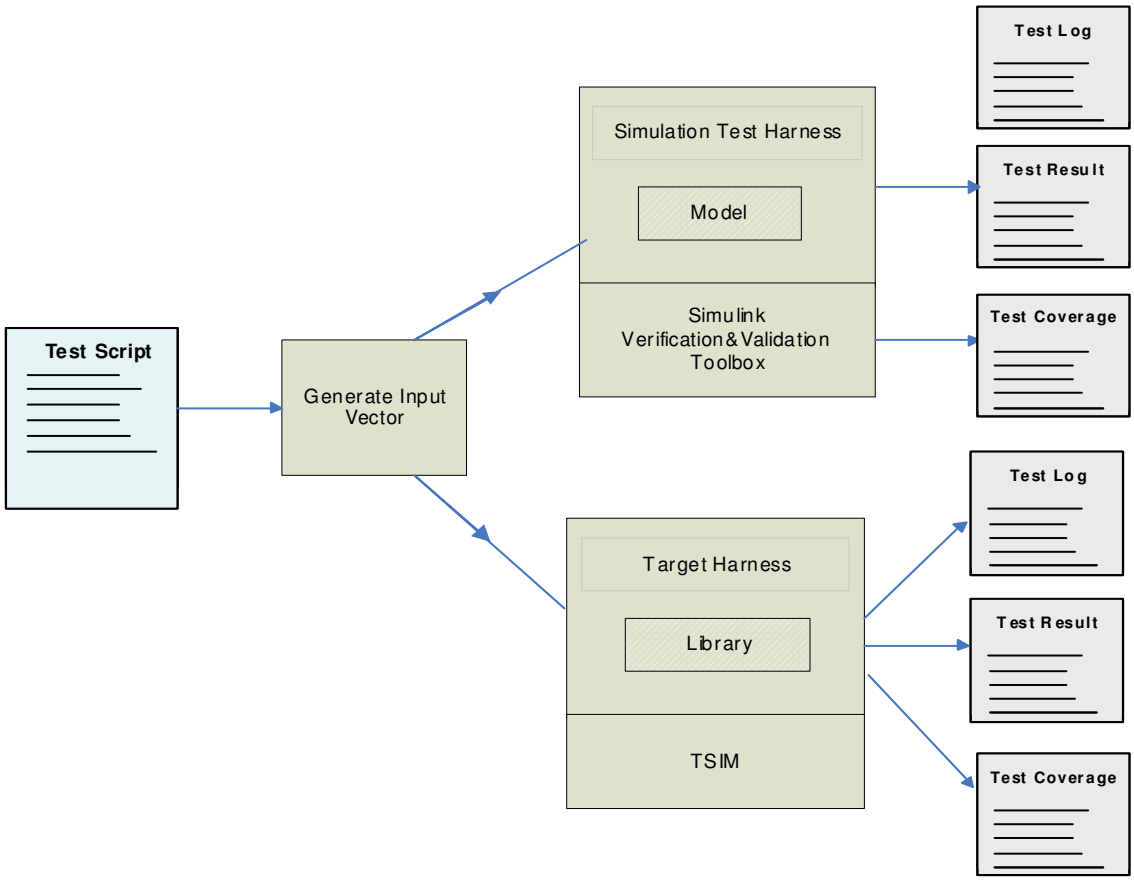

Figure 9: SGEO AOC Core Unit Test [22]

C.2.1.2.12 Software integration test plan

Objectives for AOC Core Integration Test are:

- Guarantee consistent implementation of AOC Core ICD
- To verify AOC Core functionality in target run-time environment.
- To verify AOC Core performance in target run-time environment.

- **Interface with customer**

A common simple tool (e.g. XML-based) will be agreed between SSC (supplier) and OHB (onboard software prime) to enable coherent and simplified exchange of test cases specification.

- **Test Environment**

TSIM shall be used for integration readiness tests at SSC.

- **Test Object**

Test object is the target AOC Core software.

C.2.1.3 PRISMA GNC Test Plan

Testing of the GNC Software consists of four different campaigns:

- Unit Tests
- Integration Tests
- Software System Tests
- All-Soft Testing

The Unit Tests verify the 3rd level requirements applicable to each Core in a Simulink environment. The Integration Tests consist of a "max-path" "max-load" test for each Core as well as of an interface test for integration in the all-over on-board S/W. The Software System Tests are scenario based system level tests executed on flight representative computer in real-time. The All-Soft tests consist of faster than real-time tests of all flight representative procedures.

C.2.1.3.1 Unit Testing

The unit tests are used to verify the 3rd level requirements applicable to the GNC S/W cores.

The tests are constructed as a Simulink "test harness" model what may take source data from a file as input. The test shall be possible to execute automatically and for each test there shall be an expected result output sequence to which the test result shall be autonomously compared for a pass/fail check of the result. As much as possible, the test shall be "model free" in the sense that the test shall not require any closed loop simulation of on-board functions in combination with simulation models.

All the unit tests will be executed as an autonomous "batch" job before any release of the GNC Cores. The tests are executed in the MATLAB/Simulink environment only. Similar types of unit tests are implemented for the on-board S/W cores as for the simulator modules.

C.2.1.3.2 Integration Testing

Integration testing is performed on core level as well as on software system level. On core level, much functionality is included in Simulink that guarantees the correct integration of the modules contained in the core. For this reason, the core level integration testing is limited to a "max-path" input/output sequence test that exercises as much as possible of the core's functionality simultaneously.

In addition to the core level tests, a dedicated integration test is performed on software system level. This test is an interface test that takes "dummy" cores containing the correct signal I/F and selectable signal sources. These dummy cores are then included in the on-board S/W and executed in the SATLAB environment. The tests systematically verify the complete signal chains from Simulator to CAN to TM and from TC to CAN to Actuator.

C.2.1.3.3 Software System Testing

Software system testing takes place in the real-time SATLAB Environment. One such simulation environment is dedicated to GNC tests.

The GNC Software System Tests are executed in the framework of test scenarios that correspond to the fundamental mission phases as well as to basic S/C functionality. Testing focuses on 2nd level GNC requirements and all-over performance. The tests are script-based and shall be executed w/o any manual interaction. In this way, the tests are fully repeatable allowing for automated results analysis. To the highest extent as possible pass/fail criteria shall be checked in the test scripts.

C.2.1.3.4 SATLAB Environment

The SATLAB Environment is a real-time test environment for the verification of the PRISMA on-board S/W. The environment consists of a real-time simulator (SatSim) running models of the MAIN and TARGET S/C, their sensors and actuators and the space environment. The simulator is connected via CAN busses to one EM Core Board running the MAIN S/W and one EM Core Board running the TARGET S/W. The simulator is also connected to RAMSES for its command I/F. RAMSES is also connected to the TM/TC I/F of the MAIN Core Board. The SATLAB environment also has the possibility to include an EM ST/VBS in the loop.

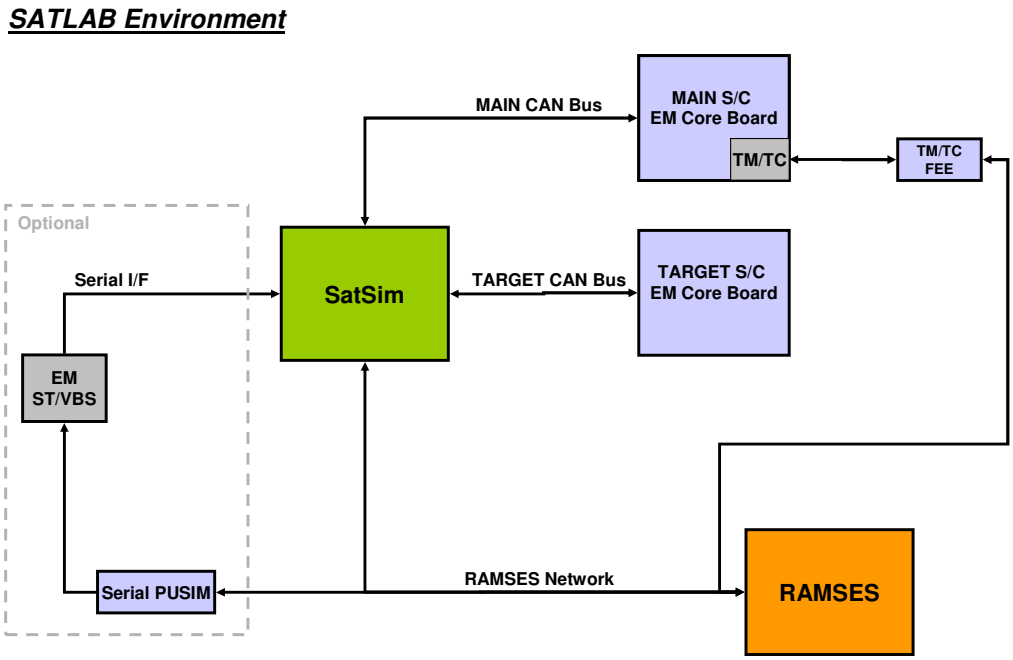

Figure 10: SATLAB Environment [23]

C.2.1.3.5 Test Execution

The Software System Tests are executed within the framework of a set of Test Scenarios. These scenarios represent fundamental attitude functionality of the MAIN and TARGET spacecraft, initial acquisition of the COMBINE S/C, separation of TARGET from MAIN and specific functionality of the AFF, ARV, and PROX/FARM Modes. In addition, tests are executed by CNES and DLR in their separate modes.

A number of Test Cases are executed within each of these scenarios. The test cases verify basic functionality of different functions of the GNC software, handling of failures in different sensors and actuators and the performance of a set of representative maneuvers in

the AFF, ARV, and PROX/FARM modes. Several of the test cases are executed in more than one of the test scenarios.

In addition to the Software System Testing, an all-soft simulator is implemented. This simulator consists of a compiled version of a MATLAB/Simulink model containing the simulator Simulink models as well as the Simulink models used for on-board S/W generation. This model uses a command interface similar to the RAMSES command scripts. The all-soft model executes as fast ass possible, preferably faster than real-time. The model is used to verify all the GNC related mission phases and cases.

The all-soft tests verify in particular the GNC subsystem performance and the sequence order of the different mission phases and execution cases.

C.2.1.3.6 All-Soft Testing

In addition to the Software System Testing, an all-soft simulator is implemented. This simulator consists of a compiled version of a MATLAB/Simulink model containing the simulator Simulink models as well as the Simulink models used for on-board S/W generation. This model uses a command interface similar to the RAMSES command scripts. The all-soft model executes as fast ass possible, preferably faster than real-time. The model is used to verify all the GNC related mission phases and cases.

The all-soft tests verify in particular the GNC subsystem performance and the sequence order of the different mission phases and execution cases.

The GNC Subsystem is verified with a series of H/W in-the-loop tests. Both Open Loop Tests and Closed Loop Tests are executed. The tests are executed in two different environments i.e. BTM and AIT.

C.2.1.3.7 Open Loop Tests

The purpose of the open loop tests is to verify the correct response in the complete sensor-software-actuator chain. the purpose is to also verify the correct magnitude and characteristics of the response

C.2.1.3.8 Closed Loop Tests

The purpose of the closed loop tests is to verify the system behavior in a closed loop setting running the on-board S/W involving as much H/W as possible in the loop. The principle behind the AIT/BTM closed loop testing is to include as much as possible of the on-board H/W.

The closed loop test setup has the following limitations:

- It is not truly closed loop since the S/C does not move as a result of its actuators. Rather, the test is a simulation involving H/W in the loop.
- It does not use the real sensors since sensors are in general not equipped with stimulation ports. Exceptions are the ST which can be directly stimulated and possibly the rate sensors.

On the other hand, the test setup includes as much hardware in the loop as is possible with reasonable effort.

C.2.2 Interview Analysis Results

C.2.2.1 Theme 1: Verification Activities in practice

Everything is mostly testing. The whole team is involved in it. Requirements and test cases are written at the same time. Test cases are functional oriented and driven by mission definition. They had a consultant who worked on ESA projects previously and brought the idea of V&V activities and then they further developed their ideas based upon it. Unit testing is performed by the developer in the same development environment. If the units are too large then they need some independent resource, it also depends upon the requirements from the customer. Defects are not logged at unit testing.

System requirements or equipment requirements comes from the client which are then broken down by subsystem managers in to sub-system (level 2) requirements which are then further broken down by the developers into unit or implementation requirements (level 3). Unit testing is based upon level 3 requirements. Assurance of the completeness of testing process is done by 3 things:

- Use of SIMULINK, for automated generation of code which is better than hand written code and minimizes syntax errors
- Unit tests are based on level 3 requirements which are written by the developer himself and he keeps in mind to write testable requirements
- Architecture is made to be testable

The developer is responsible to deliver the whole working unit. Each unit has input and output interface. The scripts are written for input data and the output is compared with the requirements. Input can be hardcoded are well. Regression testing is done in case of any change in implementation due to change in requirements are made.

Requirements are stored in DOORS, and test cases and test procedures are defined in mercury. One requirement can have multiple test cases and one test case can be linked to multiple defects. NCR is generated for each defect which is linked with SPR database. They use model driven approach and perform model coverage but not code coverage. Automated code is generated from Matlab and testing is done for models and not on automated generated code.

They define the test cases but run them automatically. The trick is to find the initial condition and the success criteria. System testing is independent and is carried out at integrated environment. System testing is focused on single functionality at a time. Python script is used to check the output. Unit tests are run automatically at the GNC level and in the end a report is generated if they pass or not. This also checks if any module is missing. Same tests are run on TSIM and SIMULINK and then the results are compared. Stress testing and duration testing are done after system testing by using telecommands and telemetry on

isolated cores. After subsystem testing clean run process is done, if defects are found at this stage, then changes are made by doing some scenarios and after that the complete hardware and software is tested on satellite.

In a project called ODEN, SSC was responsible for independent validation and it was carried out after the development by a separate company. They had to deal with many issues like quality discrepancies, misunderstanding and communication and the project was not completed on time.

Mercury Quality Center, SATLAB, MATLAB/SIMULINK, TSIM by GAISLER, DOORS, Hansoft, In house simulator by AOCS, ESA coverage Tool Box are tools which are used in the V&V process. OHB will provide testing environment for SMALL GEO, which will not be validated by SSC. They have informal reviews on requirements, test readiness and preliminary design without checklists. Having V&V experts early in the process will improve the quality of requirements.

They reuse the same testing environment, but in Small Geo OHB (prime contractor) will provide them with it. They don't start plans from scratch, if they have used models then they also use the code and V&V plans for it as well, but they are behind in reusability. The idea is to have portable models, codes and test cases and not ad hoc solutions. Things are implementation specific but some can be reused. They use ideas of Smart1 with improvements; however the whole test plan can't be reused because of different mission specific variables. Some mission variables differ at higher level, which are more general at the end. Software becomes very costly in space projects especially if they include man hours for development and V&V and they want to improve themselves in reusability.

The projects at SSC are critical because they can't change anything except the software. Once it is in operation they can't take any feedback on it, so they have more emphasis on testing and validation and mission critical things. They have longer projects and large number of subsystems are working together. It is difficult to get interaction between all of them as huge amount of data flows between them and most of the things are newly built. So it takes a longer time to test the whole system. The approach at SSC is very rigorous as they have single failure tolerant systems. Unit testing is very extensive at SSC. In space craft sometimes they have to run specific duration tests for a certain period of time to check if it behaves accordingly.

C.2.2.2 Theme 2: efficiency of Verification Activities

They found defects which are not local at that stage because their testing environment is not fully representative of the target environment. A common mistake they do is they are too much in a hurry to show progress in the beginning that can have problems in the design which can be costly to remove in the later stages. In design review they find defects relating to conflicting requirements and requirements discrepancies. At the highest level many people are involved and they don't have the idea about the constraints on requirements. At unit level sometimes they implement things which are not linked to the top level due to the conflicts in requirements, sometimes they implement things based upon their own knowledge. In some cases while working with unit testing they find problems in the requirements because

everyone doesn't have the full picture. At unit testing they find logical errors, boundary errors and clashes between requirements. Change in requirements can also have clashes between requirements.

They don't have formal specification of what each module should do. There are few things that are interpreted by the designer himself this is also one of the reasons for faults to slip through.

All unit tests must pass and each time an error is found they fix it, without logging anything in the database. They log defects at the sub system level and later stages but have different reporting systems and map them manually. The defects are reported as software problem reports (SPRs), software change requests (SCRs) and non conformance reports (NCRs) and they use different reporting systems for NCRs and SPRs, SCRs. It is not evident if something is logged as a defect or a change. Defects are connected to test cases in mercury and test cases are connected to requirements in DOORS. They document test data, time to resolve, criticality, files, baseline software version and also test set up for a particular defect in mercury and then it is mapped to the specific sub project and assigned to the developer to fix it. Defects are mainly classified according to priority and severity and sub systems, which is a vague classification. Test cases are functionally oriented and they correspond to the parts of the mission. Mercury is a very efficient tool for keeping track of passed and failed test cases and their coverage.

They measure the total development time and that includes the fixing of bugs as well. This is done at a very high level, and they use these estimates in other projects. They are using scrum and in each sprint set goals and measure and later check whether they succeeded in it or not. Each sprint has other activities as well. V&V activities should be separated from other activities and they should use matrices to measure the efficiencies of them as well. They calculate the time spent on fixing SPRs. They have matrices for the number of verified requirements, model coverage and test cases.

They are beginning to use test driven development in their projects. The main idea is to let the test cases be the requirements and instead of writing detailed requirements (level 3) and then write code and then do unit testing, they want to skip the detailed requirements. Main purpose is to have more focus on testing in the beginning and not push it to the end and to have more focus on testing than implementation. They will develop test cases and system test cases from the beginning which will reduce the number of defects. They will try to measure the progress of a project in terms of test cases that are covered. It will force them to think ahead and identify much earlier if they need certain testing environments. It can also give them an estimate about the efforts they need and the time it would take. The development cycles would go much faster and will enable them to focus more on important things. But no workshops or training sessions has been held for test driven development.

It is a limitation that they don't have any internal verification board or team which overlooks their activities. Everybody is involved in their own projects. At some stage they ought to have an independent validation but in having completely separate team doing all the validation can result in the loss of information. Developers can interchange code and review each other's work as well.

C.2.2.3 Theme 3: ECSS standards

In Smart1 they used PSS05 which were more strict and prescriptive than ECSS. SSC had less formal testing procedures and they had clashes with ESA. ESA couldn't understand the way SSC develop things and SSC thought that ESA only wants useless documentation from them that requires a lot of work. ECSS standards are high level, general and abstract, they don't follow it formally but the general flow is the same. Sometimes the people are following it but they don't know about it. It prescribes to produce certain documents which drive the project. In Smart-Olev, the Dutch prime contractor took ECSS standards very seriously and asked all the sub contractors to use them as it is. This joint venture didn't work because the sub contractors came back with their cost estimates. That is why tailoring of ECSS standards is very important. ESA has many technical experts and they can gather people from academia as well to comment upon what should be done and what should they prove and it is very difficult to completely comply with all the requirements. The commercial customers don't have that insight. SSC has allocated a person who makes strategies to make their processes in compliance with ECSS standards and keep track of its requirements and documents to produce.

ECSS is a standard that ESA wants all sub contractors to use, to have common documentation and language. It improves communication between primary and sub contractors. It also give knowledge about which activities to follow for non ESA projects as well like for instance in PRISMA the way they worked was based on ESA standards but they tailored it.

At the same time they also feel that good quality doesn't mean if they have read ECSS standards or not, its like an ISO 9000 standard. It is very comprehensive and general and has a lot of text. It is not applicable for small projects due to high cost associated with it. It is abstract and people can misunderstand them and refrain from using things which are very beneficial, only to make sure that they stay in compliance with ECSS standards. It can also be misused as a tool by the customers who may not have the idea about the cost of certain activities like formal proof checking and tree analysis. It is hard to map things from ECSS to projects and investment varies from project to project. People become reactive due to these standards and only produce the list of things which are asked to such an extent that it hampers innovation. There are no implied processes mentioned in the ECSS standards and producing same set of documents for every release is very costly.

C.2.3 Web-based Questionnaire Analysis Results

C.2.3.1 Introduction

An online survey has been conducted at SSC to get a certain level of understanding about the V&V activities which they perform. The total respondents were 18, having varying level of expertise in terms of experience, age, areas of software development and testing and projects in which they worked at SSC.

C.2.3.2 Areas of software development

The survey results show that 83% of respondents are involved in software requirements engineering but only 39% are involved in system requirements engineering. 67% are involved in architectural design and validation/system testing where as 61% are involved in detailed design, implementation, integration testing, requirements review, tool development and unit testing. 55% are involved in design review and project management. 44% are involved in acceptance testing and code review. 28% in quality assurance, 22% in development support, 16% in management and only 5% in mission analysis and coordination between projects.

C.2.3.3 Experience in software development

33% of the respondents had an experience in software development between 5 to 10 years, 22% between 10 to 15 years, 16% between 15 to 20 years, 5% had greater than 20 years, 16% between 3-5 years and 5% had less than 1 year.

C.2.3.4 Experience in V&V

23.5% of the respondents had 5 to 10 years of experience in v&v, 17.6 % had between 1 to 2 years, 10 to 15 and less than 1 year, separately, 11.7% had 3 to 5 years and 5.8% had between 15 to 20 years and greater than 20 years, separately.

C.2.3.5 Knowledge about ECSS standards

22.2% of the respondents knew the contents and how it affects their software development activities where as 33.3% had never heard about it. 11.1% knew in detail what it prescribes and its consequences for their software development and 33.3% knew them roughly.

C.2.3.6 Degree to which ECSS affect their software development activities

16 % of the respondents were of the opinion that the degree to which ECSS effect their software development activities is high, 44.4% said its low where as 38.8 % said its vey low

C.2.3.7 Effect of ECSS on the quality of software

33% didn't answer about the effects of ECSS on the quality of software. However, amongst the remaining respondents 58.3% said somewhat positive, 25% said its mostly positive and 16.7% said its mostly negative

C.2.3.8 Effect of ECSS on the efficiency of software development

33% of the respondents didn't answer about the effect of ECSS on the efficiency of software. However, amongst the remaining respondents 41.6% said its somewhat positive, 33.3% said its somewhat negative, 16.6% said its mostly negative and only 8.3% said its mostly positive.

C.2.3.9 Effectiveness of verification activities

- Requirements review - 27.8% are not involved in it, 16.7 said it is very effective, 38.9% said effective and 16.7 % said ineffective
- Design review - 27.8% are not involved in it, 16.7% said it is very effective, 38.9% said effective and 16.7% said ineffective
- Code review - 44.4% are not involved in it, 38.9% said it is effective and 16.7 said ineffective
- Unit Testing - 38.9:% are not involved in it, 44.4% said it is very effective, 5.6% said effective and 11.1 said ineffective
- Integration Testing - 44.4% are not involved in it, 38.9% said it is very effective, 11.1% said effective and 5.6% said very ineffective
- System Testing - 22.2% are not involved in it, 50% said it is very effective, 16.7 said effective 5.6% said ineffective and the same percentage said very ineffective
- Validation Testing - 38.9% are not involved in it, 22.2% said it is very effective, the other 22.2% said effective and 11.1% said ineffective and 5.6% said very ineffective
- Acceptance Testing - 50% are not involved in it, 16.7% said it is very effective, the other 16.7 said effective, 11.1% said ineffective and 5.6% said very ineffective

C.2.3.10 Effort which the verification activities require

- Requirements review - 22.2% are not involved in it, 11.1% said that it requires very high effort, 27.8 said high effort and 38.9% said low effort
- Design review - 27.8% are not involved in it, 5.6% said that it requires very high, 33.3 said high effort and 33.3 said low effort
- Code review - 38.9% are not involved in it, 16.7% said that it requires very high, other 16.7% said high effort, other 16.7 said low effort and 11.1% said very low effort
- Unit Testing - 33.3% are not involved in it, 22.2% said that it requires very high effort, 27.8 said high effort and 16.7% said low effort
- Integration Testing - 38.9% are not involved in it, 50% said it requires high effort and 11.1% said low effort
- System Testing - 16.7% are not involved in it, 33.3% said that it requires very high effort, 44.4 % said high effort and 5.6% said low effort

- Validation Testing - 33.3% are not involved in it, 11.1% said that it requires very high effort, 44.4 said high effort and 11.1% said low effort
- Acceptance Testing - 44.4% are not involved in it, 11.1% said that it requires very high effort, 22.2 said high effort and the other 22.2% said low effort

C.2.3.11 Change in VA, if ECSS is not relevant

- Requirements review - 16.7% are not involved in it, 38.9% won't change the effort on it where as 38.9% said they would like to put more effort on it where as only 5.6% said they would put less effort on it
- Design review - 22.2% are not involved in it, 55.6% won't change the effort on it, 16.7% would put more effort on it and 5.6% would put less effort on it.
- Code review - 38.9% are not involved in it, 27.8% won't change the effort on it, 16.7% would put less effort on it and the other 16.7 would more effort on it.
- Unit Testing - 33.3% are not involved in it, 38.9% won't change the effort on it and 27.8% would put more effort
- Integration Testing - 33.3% are not involved in it, 50% won't change the effort and 16.7% would put more effort
- System Testing - 22.2% are not involved in it, 66.7% won't change the effort and 11.1% would put more effort
- Validation Testing - 33.3% are not involved in it and 66.7% won't change the effort
- Acceptance Testing - 38.9% are not involved in it, and 55.6% won't change the effort and 5.6% would put more effort

C.2.3.12 Percentage of V&V activities of the total development costs

29.4% said the cost should be 60%, 23.5% said the cost should be 30%, 17.6% said it should be 50%,the other 17.6 % said that it should be 70% and 11.7% said it should be 80%

C.2.3.13 Weighted-average calculation

Knowledge about ECSS Standards

I have never heard of it - 1
I know roughly what it is about – 2
I know its contents and how it affects the SWD activities - 3
I know it detail what it prescribes and its consequences for SWD – 4
I am an ECSS expert and have worked actively in implementing and adopting it – 5

18 people answered

(6*1) + (6*2) + (4*3) + (2*4) + (0*5)/18 = 2.1

Degree to which ECSS affect the SWD

Very high – 4
High – 3
Low – 2
Very low – 1

18 people answered

(7*1) + (8*2) + (3*3) + (0*4)/18 = 1.8

Degree to which ECSS affect the quality of SW

Mostly positive – 4
Somewhat Positive – 3
Somewhat Negative – 2
Mostly negative - 1

12 people answered

(3*4) + (7*3) + (0*2) + (2*1) /12 = 2.9

Effect of ECSS on the efficiency of software development

Mostly positive – 4
Somewhat Positive – 3
Somewhat Negative – 2
Mostly negative - 1

12 people answered

(1*4) + (5*3) + (4*2) + (2*1) /12 =2.4

Effectiveness of verification activities
Very effective- 4
Effective-3
Ineffective-2
Very Ineffective-1

Requirements Review
5 people are not involved in requirements review
(3*4) + (7*3) + (3*2) + (0*1) /13=3

Design review
5 people are not involved in design review
(3*4) + (7*3) + (3*2) + (0*1) /13=3

Code review
8 people are not involved in code review
(0*4) + (7*3) + (3*2) + (0*1)/10=2.7

Unit testing
7 people are not involved in unit testing
(8*4) + (1*3) + (2*2) + (0*1)/11=3.5

Integration testing
8 people are not involved in integration testing
(7*4) + (2*3) + (0*2) + (1*1)/10=3.5

System testing
4 people are not involved in system testing
(9*4) + (3*3) + (1*2) + (1*1)/14=3.4

Validation testing
7 people are not involved in validation testing
(4*4) + (4*3) + (2*2) + (1*1)/11=3

Acceptance testing
9 people are not involved in acceptance testing
(3*4) + (3*3) + (2*2) + (1*1)/9=2.9

Effort which the verification activities require
Very high effort- 4
High effort -3
Low effort-2
Very low effort-1

Requirements Review
4 people are not involved in requirements review
(2*4) + (5*3) + (7*2) + (0*1)/14=2.6

Design review
5 people are not involved in design review
(1*4) + (6*3) + (6*2) + (0*1) /13=2.6

Code review
7 people are not involved in code review
(3*4) + (3*3) + (3*2) + (2*1)/11=2.6

Unit testing
6 people are not involved in unit testing
(4*4) + (5*3) + (3*2) + (0*1)/12=3.1

Integration testing
7 people are not involved in integration testing
(0*4) + (9*3) + (2*2) + (0*1)/11=2.8

System testing
3 people are not involved in system testing
(6*4) + (8*3) + (1*2) + (0*1)/15=3.3

Validation testing
6 people are not involved in validation testing
(2*4) + (8*3) + (2*2) + (0*1)/12=3

Acceptance testing

8 people are not involved in acceptance testing
(2*4) + (4*3) + (4*2) + (0*1)/10=2.8

Change in VA, if ECSS is not relevant

Would put more effort on - 4
No change - 3
Would put less effort on - 2
Would not do activity at all - 1

Requirements review
3 people are not involved in requirements review
(7*4) + (7*3) + (1*2) + (0*1)/15=3.4

Design review
4 people are not involved in design review
(3*4) + (10*3) + (1*2) + (0*1) /14=3.1

Code review
7 people are not involved in code review
(3*4) + (5*3) + (3*2) + (0*1)/11=3

Unit testing
6 people are not involved in unit testing
(5*4) + (7*3) + (0*2) + (0*1)/12=3.4

Integration testing
6 people are not involved in integration testing
(3*4) + (9*3) + (0*2) + (0*1)/12=3.25

System testing
4 people are not involved in system testing
(2*4) + (12*3) + (0*2) + (0*1)/14=3.14

Validation testing
6 people are not involved in validation testing
(0*4) + (12*3) + (0*2) + (0*1)/12=3

Acceptance testing
7 people are not involved in acceptance testing
(1*4) + (10*3) + (0*2) + (0*1)/11=3.1

Appendix D

This appendix provides information about RUAG Aerospace Sweden AB. It provides the summaries of interview, survey and document analysis conducted for data collection at RUAG Aerospace.

D.1 Introduction

RUAG Aerospace Sweden AB (RUAG) was formerly known as SAAB Space AB but was recently acquired by RUAG Aerospace, and thus changed their name. RUAG has a very long and vast experience concerning design, development and delivery of both hardware and software for computer and data handling related products in space programs. The main product areas are Data management systems, Fault-tolerant computers and processor products, Payload control computers, Data processing and Small mass memories. The software developed by RUAG for these computers is in the range from small boot software to full application software, but the main focus is on hardware-near, embedded, real-time software. The software development process used is based on the ECSS standards, mixed with an integration driven development approach.

D.2 Data Collection/Study Results

D.2.1 Document Analysis Results

RUAG develops a Software Development Plan (SDP) at the start of each project which addresses customers and different team like development, validation and maintenance. This document defines the overall planning for the development of the software and covers all the activities concerning development, verification, validation and maintenance of the software during whole software life cycle. SDP is a main document and defines what activities will be performed. It does not explain how these activities will be performed. The details of those activities are discussed in relevant plans like Software Configuration Management Plan (SCMP), Software Verification Plan (SVerP), Software Validation Plan (SvalP), Software Product Assurance Plan (SPAP) etc. RUAG has defined three organizational levels for software development:

- Software level – The level where are the activities defined in SDP are performed.
- Product/System/Project level – The level where overall planning and management of the project is discussed. It is the higher level than software level and is referred as next-higher level in SDP. It defines Requirement Baseline (RB) for the software and performs RAMS activities.
- Customer level – This level corresponds to all external customers.

RUAG has a well defined software object organization setup with predefined roles and responsibilities. Object Manager (OM), SW Verification Responsible (SVeR), SW Specification Responsible (SR), SW Design Responsible (DR), SW Design Engineer (DE), SW Validation Engineering (VR) and SW Validation Responsible (VR) are the major roles in the software object organization. Manager of software department is responsible for selecting team for the project and for conducting a just-in-Time introduction to each software life cycle phase for those who need it. RUAG has classified software into three main categories:

- Boot Software – Known as critical software. It should be fully verified (code inspected, unit tested and integration tested) and validation tested. HSIA will be performed.

- Monitor Software – It is used for test purposes and is only verified through code inspection. Validation testing is performed as well.

- AT95 – it is not critical software but is fully verified through code inspection, unit tested and integration tested. Validation testing is performed at the end.

SDP has a section which explains the activities necessary to be performed at each development stage (Planning, Specification, Production and Validation) for the above mentioned software categories. It also lists down the software which will be reused for that particular project. Details on the reuse are further explained in Software Reuse File (SRF). Software deliveries are planned and described briefly in SDP.

D.2.1.1 Software Development life cycle

Software development life cycle provides a reference framework for software phases and activities. It specifies the relationship between project phases, transition criteria, feedback mechanism, milestones, baselines, reviews and deliverables.

Integration-driven Development (IDD) approach is used along with waterfall reviews for software development. IDD is introduced to develop, test and integrate software increments concurrently. Software development life cycle at RUAG has five phases: Software planning phase, Software specification phase, Software production phase, Software RB-validation phase and Software maintenance phase. Figure D-1 further explains software development life cycle along with IDD and ECSS reviews. In this figure SRR: System Requirement Review, SW-PDR: Software Preliminary Design Review, SW-CDR: Software Critical Design Review, SW-QR: Software Qualification Review.

RUAG is working with ESA, so they have to follow certain software standards like ECSS and Galileo. These standards mostly recommend waterfall approach with predefined reviews, for software development. Since the space projects are very critical and waterfall in nature, so before starting the incremental IDD approach they have to ensure some pre-requisites:

- The RB must have been specified at a comprehensive level along with non-functional requirements (Performance and design constraints)

- SDP, SCMP, SPAP, SUTP, SITP, SVerP and SValP must be developed. A top-level architectural software design must exist

- Tools and development environment must be set up

ESA allows using IDD but at the same time it demands for all the reviews for each increment. These reviews are very costly and sometimes are impossible to be fully performed at some particular stages. For example it is not possible to perform full PDR using IDD because they don't have complete details. IDD uses four internal reviews for each increment.

D.2.1.2 Requirements Review (T0):

Requirements Review is performed before the start of each integration step to ensure:

- The requirements in RB for this integration step are reviewed and approved
- The requirements in TS for this integration step are reviewed and approved
- Architectural design for this integration step is reviewed and approved

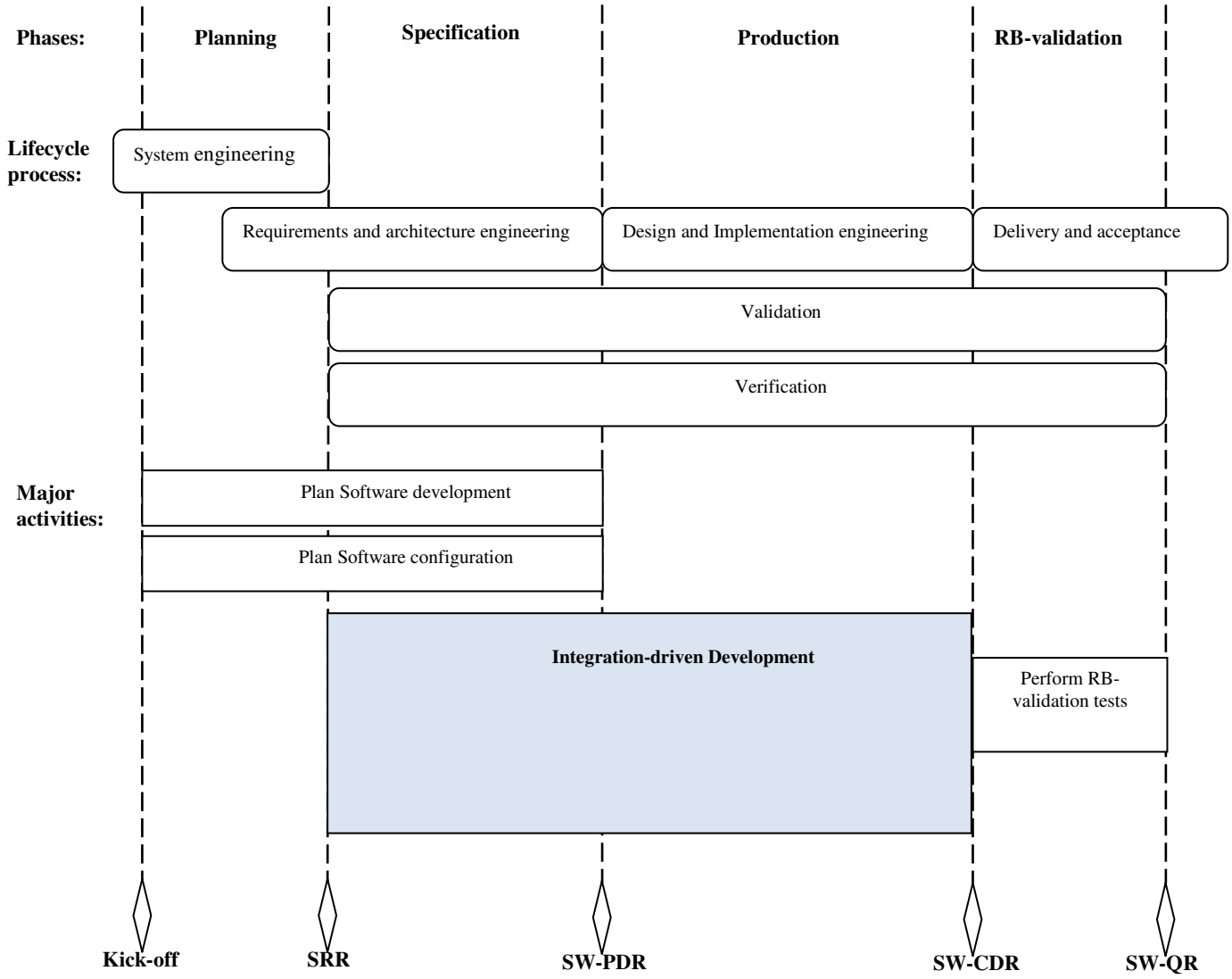

Figure 11: Software development life cycle [24]

D.2.1.3 Integration Readiness Review (T1):

This review is performed after the development and verification of the unit. It is done to ensure:

- The detailed design for the integration step is described
- Source code is implemented and successfully implemented and code inspected

- Source code is successfully unit tested

- Validation test cases are identified, specified and implemented

D.2.1.4 Integration Closure Review (T2):

This review is performed after the integration has been performed to ensure:

- All validation tests/analysis related with the integration step have been executed successfully.

D.2.1.5 Test Readiness Review (TRR):

TRR is performed before the validation test campaign of the whole software starts to ensure:

- All the integration steps successfully performed

- Software documentations, source code and test facilities are under configuration control

- No open SPRs or NCRs exist

A review success criterion has been defined for each review. A review is considered successful if the review objectives are met and all RIDs and review related actions are closed. A review can end up as successful, successful with rework to done and not successful. Figure D-2 further explains this idea:

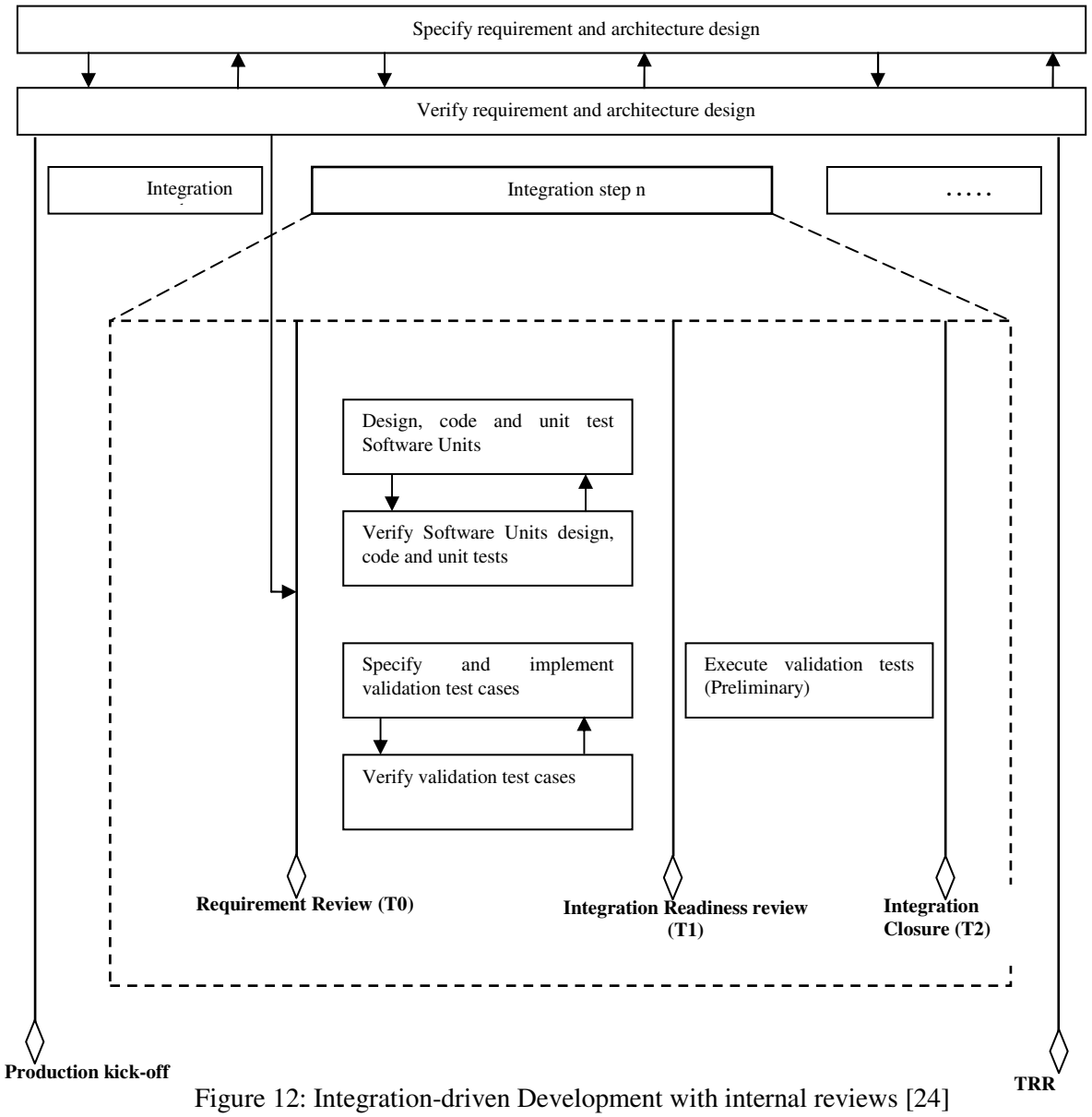

Figure 12: Integration-driven Development with internal reviews [24]

D.2.1.6 Software Verification and Validation Plan (SVVP)

RUAG develops a comprehensive verification and validation plan for each project. This document defines overall planning of the verification and of the software and covers tasks and decisions related to verification and validation through the whole software development life cycle. It mainly addresses customers, verification and validation team and maintenance team. Role and responsibilities along with schedule for software verification and validation are explained in SDP.

The administrative procedure of software verification is well defined by explaining anomaly reporting, software verification reporting, task iteration policy, deviation policy, control procedures, and standards. Software design team is responsible for major verification activities (Requirements verification, Design verification, Code verification, Module testing, Module test verification, Integration testing) and software validation team is responsible for verification of the validation test cases and analysis specification explained in validation test specification against Technical Specification (TS) and Requirements Baseline (RB).

Most of the software modules are written in C language and module tested in UNIX environment on ERC32 (TSim) and Cantata framework. Some parts are written in Assembler and are tested in a special environment called SDTE.

D.2.1.6.1 Verification of software requirements

Software requirements are complied in Technical Specification (TS) which are derived from Requirements Baseline (RB). The requirements (TS) are verified against the next higher level (RB). The requirements are verified with respect to compatibility with RB, Accuracy and consistency, Feasibility, Validation of software, Clarity, Absence of unnecessary constraints and conformance to standards.

Software requirements are verified by means of reviews of the TS. These reviews are conducted both internally and externally with customers as well. A comprehensive checklist for internal reviews along with prototype is used to check the consistency and feasibility of the review.

D.2.1.6.2 Verification of Software architecture design

Software architectural design is verified with respect to compatibility with software requirements, consistency, shared resources, and compatibility with target computer, verifiability, clarity and conformance to standards. Software architecture design is verified by the means of reviews of the software design, timing and sizing budget, module integration testing.

D.2.1.6.3 Verification of Software design and code

Software design and code is verified with respect to compatibility with software requirements, compliance with module specifications, compliance with software architecture, verifiability, clarity, and conformance to standards. Software code is verified by means of code inspection and module testing. Code inspection is performed using a checklist. Verification of the design is performed by means of integration testing.

D.2.1.6.4 Verification of traceability

RB requirements are traced to the TS requirements which are in turn traced to the architectural design classes. At module level, a matrix is used to trace the module to the corresponding module test cases.

D.2.1.6.5 Validation of built software product

Validation of the software is validated with respect to compliance to functional and performance requirement to assure that the software product reliably behaves in accordance with TS and requirements RB. Software product is also verified with respect to miscellaneous software requirements to assure the confidence on the reliability, availability etc.

RUAG uses a re-verification approach instead of regression approach to show that a base line software that has been changed still meets functional and performance requirements. This approach at integration testing verifies and validates correct implementation of software problem reports/non conformance reports/change requests (SPRs/NCRs/CRs), any change in functionally or testing environment, and components calling or called by the changed module. Re-verification is carried out in same original verification environment. Same set of tools, techniques and methods are used.

D.2.1.6.6 Software validation:

Software validation team is responsible for validation of Technical Specifications (TS-validation), Requirements Validation (RB-validation) and related non-regression testing. Non-regression testing is performed to ensure the correctness of the software when it has changed after validation.

Role and responsibilities for software validation are specified in SDP. Validation is either performed by testing or analysis depending on the nature of the requirement. Validation by analysis is performed when testing is not feasible. Software Development and Testing Environment (SDTE) described in SDP along with the scheduled activities is used for validation. Validation tasks are mainly performed in the software integration and TS-validation phase and software RB-validation phase.

D.2.1.6.7 Software testing standards

RUAG performs testing at three stages: module testing, integration testing and validation testing. Test coverage during whole test campaign is defined by testing/analyses to verify all the TS/RB requirement, 100% statement and decision coverage at source code level and all types of test cases applicable to module, integration or validation testing must be taken into account.

D.2.1.6.8 Module testing:

Module testing aims to verify the behavior of the module. Module test cases normally include stress testing, robustness testing, tests for data, logic coverage tests. Data test includes equivalence portioning and boundary value analysis. Different approaches are used for boot software and non-critical or test software which is explained in related plans.

D.2.1.6.9 Integration testing

Integration testing aims to verify the dynamic interaction of the modules to ensure that they are working correctly together. The testing is performed on the hardware, each couple

of “calling-called” software component is activated and functions of each software are activated at-least once.

D.2.1.6.10 Validation testing

Validation testing is performed to ensure that the software meets functional and performance requirements and is designed to find out the discrepancies between software and its requirements. The strategy for the validation testing is as follows:

- Testing is done to find the faults
- Testing is planned for zero fault tolerant
- No assumptions should be made about implementation of the code
- If testing is not possible then the requirement will followed by analysis
- Validation is done independently
- Test cases are written for valid and expected, valid unexpected conditions
- Both normal and abnormal behavior should be taken into account.

The validation of the boot software is separated from the validation of the service or test software as it requires different testing environment. Functional, interface, fault injection and robustness testing is taken into account at this stage.

D.2.2 Interview Analysis Results

D.2.2.1 Theme 1: Verification Activities in practice

At RUAG, they have rigorous VAs, as the software systems for space industry are very critical. The main activities are code inspection, module testing and qualification testing which is now called SW validation testing and which is also the term used in ECSS as well. They also do integration testing which is not the main activity. They try to re use module tests or validation tests as basis for integration tests to keep the effort down but they don’t skip this process completely but try to minimize the work effort in between. As the requirements are only in the form of text so they review them against check lists, this is the same with design, as it is in UML which can’t be executed like code.

Documents such as requirements specification, detailed design and all sort of produced documents are reviewed and reviewers are set up depending upon the kind of documents. The verification performed at earliest stages of the project is reviewing the requirements or documents and then they break the equipment requirements into module requirements which are the system level requirements at different parts of the unit. Then reviewing the module requirements against the equipment requirements, focusing on equipment requirements which cannot be fulfilled by one module and check for their consistency.

Then the information is passed onto software development team (software object). They interpret requirements from the module requirements document also called the software system specification and the requirements baseline. They start interpreting system level requirements to preliminary design and define their requirements specification which are reviewed and verified by the software system manager. He also reviews the architecture design and preliminary design.

The next step is writing validation plan describing what they are going to do, the kind of test techniques they will use and how the test equipment should behave by specifying and allocating each test case for every requirement. Different test cases draw up the outline what the tests will do. The next step is implementing everything and reviewing test code to make sure they are correct with respect to the test specification and if required make any final configuration to the test equipment. If they can't test everything they do the manual analysis and document them in the report.

Validation is performed against the requirements baseline. They have issues at different levels of requirements. In one of the projects (NSG) they have acceptance testing which is one level above validation. At the lower levels of verification they have strict conditions for fulfilling tests by having 100 % statement coverage and 100% coverage of requirements baseline. They have basic foundation about the things they should test and for each module test they state which units are tested and which aspects and things are covered. External interfaces are more thoroughly tested because internal parts are also covered by reviews, especially if they provide an API they focus more on external interfaces.

They cover the code by unit testing and check if they have gone through the whole module. When it comes to requirements, architecture design, test specification, coverage of tests against requirements they do manual review of documents and when it comes to code they do manual review against requirements, the design document, coding standards, hardware-software interfaces, control documents.

They use requirements database DOORS to specify the requirements and test cases and it provides traceability between them. It also gives them an idea about the kind of test equipment they will need in the project. They try to state them as such that they are verified using black box testing. If they have non functional requirement in their testing plan then they verify them, but they don't have all the required figures related to them. Mostly, non-functional requirements are verified by inspections. They also use special performance tests to stress all the channels at the same time and get CPU load very high and apply test campaigns with different scenarios.

They don't have any requirements on how to do inspections or reviews on the code or requirements. The requirement they have is that code shall be verified and that it shall be verified against some rules. Their inspection and reviews are not completely formal and not completely informal either. They have well established check lists and clear scope of what they should inspect in documents. The manual inspection is complemented by Splint and Flexelint-tool, which cover certain aspects of code and static analysis of code metrics.

They inspect 100% code but they always have an issue on how much is covered from the coding standards. Review is dependent upon the person doing. It is better if the person doing it have the knowledge about the coding standards and the way they implement things. But the problem is the check list which they use, although they have an intention to reduce it but it has been expanded which is difficult to manage.

The module testing is either performed by the developer himself or by the separate developers, depending upon the requirements from the customer. Inspection is performed independently but typically by the person who is involved in the development of software. For each change in the module they do re inspection. In some projects they also carried out inspection of module tests and validation tests depending upon the requirements of project.

Unit testing is performed by the help of a tool called Cantata. It has many scripts which configures everything and makes an automated report about them and if any part is missed some additional test cases are made. In unit test level, they focus on functionality and try to get complete coverage. After checking the main functionality they test the borders by

boundary value and parameters. There are pros and cons of having independent unit testing. There could be a loss of information while carrying it out independently, on the other hand, the developer doing the unit testing may miss some of his defects. Mostly, the developer also do unit testing as well but they have independent inspection of the code to avoid the cons.

At unit level testing they don't have any specific design for the tests. They use T95 framework for test design and write test sequences in C++ but those are more or less sequential activities and checks to perform during testing. GSWS required them to have inspection of the module tests.

They are not good at doing integration testing. They don't have formal integration testing rather they try to cover integration testing at the validation level where they test software and hardware together. But some form of integration of software module testing is performed by the software development team which is not formal. The validation department requires that the software must be fully module tested and code reviewed by the software department before sending it to them.

They mainly produce hardware platforms and along with it they deliver software. The size of the software varies, it can be small from just a boot up application to a full software application. It differs from project to project but is always related to hardware. Previously all software activities in flight software development were performed in the software development section. Then they had two validation teams one for hardware validation and the other for software validation and often they find the same problems with same test but they discover them separately. Since the hardware team need software to test their hardware and vice versa. They are making validation testing by testing the software and hardware together more and more.

Previously, they ran the same test cases for software and hardware, they are now trying to work together in one test specification and tank them together so they don't have to implement the same tests twice. In the customer requirements the hardware and software requirements are not separated so they see them as a function when they test them. The validation department is taking more and more responsibility to save time because previously the same type of tests were performed by both departments, separately. In the latest space craft management unit projects, they had low level hardware drivers and there was a tight connection between the drivers and the hardware therefore they think it is better to conduct the testing on drivers together with hardware.

They have templates for documents which are traced to ECSS so if they use that template without making any changes, they would be compliant to ECCS-E-40. If they make any changes they have a trace to verify and evaluate if the changes made them non complaint to it. These templates are used in different projects and improvements are made in them continuously. They built similar kind of systems, so they try to reuse the system requirements and technical specifications. Requirement Baseline is provided by the customer so they try to match them with system requirements and technical specification by making small changes. Sometimes they have to rewrite the technical specifications.

After the Ariane5 accident, in which they reused some part of the code from Ariane4, ESA and other companies became very skeptical about reusing code. Although, it is not forbidden but is not encouraged either. ECSS requires too much work to reuse things. It requires them to do some extra tasks which can become more expensive and at times they end up doing more work to make analysis and justify it. They have to justify why they are reusing it and explain how it works.

Often the software they are testing is not the final application and is usually more or less a driver for hardware so they needed to add test application and guess how the customer will

use it. They have started to use more simulations but normally they have in-house prototypes of breadboard for hardware testing. In most cases their validation testing is the acceptance testing. The validation team always validates a complete software. Sometimes they get part of it to check the functionality but the official testing is always done for complete software. For calibration, they make sure they execute the software on real hardware which has gone through a level of level of verification. They also have traceability from requirements to design, to test cases and to customer requirements to ensure that they have full coverage.

They use a number of tools like Splint for static analysis, FlexeLint for checking coding standards and Cantata for unit testing and checking structural coverage. They have structural coverage on unit testing level but not at the validation level. The equipment also depends upon projects and it varies. They have parallel development of testing environment and actual software. Testing of the software tools and testing equipments are less formal than the validation of the actual software. For test equipment they don't trace the requirement because it is an internal project nor do they have test scripts. They perform inspection of the software used by the instruments but they don't have any module test or review test. COTS are also not formally tested.

The validation team has very less involvement in the requirements phase.

D.2.2.2 Theme 2: efficiency of Verification Activities

They find defects which could have been found earlier. Sometimes at validation level they find defects which should have been found at the unit level. It also happens the other way around that at code inspection and unit testing they find defects which could have been detected by validation tests but weren't in their test specification. Code inspection is a manual task and is dependent upon the engineers and usually they have different engineers doing code inspection and unit testing. It is very tiring and if they do it over and over then they start to miss things. They had problems with unit testing because they focused highly on the coverage figures. The problem is that people are looking at what code does instead of looking at what the code should do and test that. Other reasons for faults slip through are the tight schedules in the project. In such circumstances they move away from optimal ways of doing things and rush through them. They have had problems with the in house tools and sometimes it requires too much effort to improve such tools and methodologies which are project driven. They use the same environments and tools for other projects as well but always with certain changes so the main components are the same but there are always differences and those differences take a lot of time.

They are lacking in metrics which are easy to use and follow up and they think it is hard to get them. Another problem is they don't have statistics to use in metrics and they are not good at following up afterwards to see what method was good or bad. They haven't measured the efficiency of different VA's and they don't even measure the number of defects found at different levels. They are not good at measuring the results of what they are doing why they are doing it. They don't have a formal list which says these categories of faults should be detected at this stage and so on. One opinion is that code inspection is more efficient that module testing but they don't have exact scientific figures to back that up.

Code inspection is one of the most valuable things at RUAG, but the downside of code inspection is the dependence upon the person doing it. It is different when an experienced employee is doing it or when a consultant is doing it. During code inspection they find a lot of issues like maintenance issues, bad choices of algorithms, implementation inefficiencies, these are the kind of issues which they don't find in module testing or validation testing where they check the input and output result. They think that they are very good at code inspection and they even get better in it. They have well established check lists which keep

on improving in every project. They keep on looking for automated tools to help them in the manual inspection.

Reviews of requirements and design are cheap as it only takes few days and a number of engineers. People have less interest in module testing compared to writing code. They don't use any tool for logging defects during inspection or module testing, they use sheets to record them and make a note where the defect was found and remove them. But at validation level or after the delivery of the first version to the customer they generate proper SPRs and NCRs and use database tool, because it is carried out by another team who need to communicate with the design team.

They classify problems on the basis of criticality which is defined in their development plan. They also classify them according to artifacts in which they were found like requirements, coding and design. They don't have the support for dividing code into criticality levels.

D.2.2.3 Theme 3: ECSS Standards

ESA reviews their documents to ensure they are following ECSS standards. For each major review in the lifecycle, they review documents like requirements, design, interfaces, etc. After reviewing them they send Reviews Items Discrepancy (RIDs) based upon them. RIDs are discussed in their meetings, arranged at RUAG to check whether a particular RID need to be changed or it is not clear.

D.2.2.3.1 Positive effects of ECSS standards:

The major positive effect is the standardization itself. It is required in business area like space, to improve their way of working and to have cooperation with different countries as well. If they don't have the standards, this can create lot of problems in the cooperation. It helps in understanding each others way of working.

ECSS is a European wide standard and all their customers are aware of it. When they make proposals on new projects, it is often the same standard every time so it makes easier for them to reuse their own internal standards and processes since they know the requirements from customers about the development process. They don't have to write their development plans from the scratch and don't have to spend a lot of time explaining their internal processes.

Everyone has to follow the same design. It gives the idea what has to be in the documents, which data to provide and which task has to be performed at each stage. ECSS keep all the stakeholders on their toes. When they follow the standards and the templates in the standard then it is easier to show the customers that they have included all the necessary information and everyone knows a place where to look for that information. Even though they may not agree with few things in the standards but it is better to follow it.

D.2.2.3.2 Negative effects of ECSS standards:

Although ECSS creates common understanding among different stakeholders but requirements of it can be interpreted differently by different people especially when the people have different cultural background like Italian, Swedes, French, and Germans etc. Since the standards are considered as consensus among different stakeholders and they must leave room for interpretation because if the standard would decide everything then it would be too rigid for innovation. It is very person dependant and everyone can interpret it

differently so they have issues about how these standards should be interpreted while writing development plans and tailoring those standards. This opens up a number of new discussions with the customers.

ECSS does not provide enough support for iterative development. They are more or less forced to follow waterfall model with costly reviews and lot of overheads. It is very difficult to work iteratively and in small integration steps because ECSS focuses on the external review of customer and if they divide their software project into 5 integration steps then they are forced to have 5 or 6 reviews for each increment. It is an overhead in each project and is not feasible for each and every increment. Incremental development is not impossible with ECSS but it is not supported by ECSS.

They think that agile approaches are quite hard to be introduced with ECSS, because of the reviews required by ESA for each increment. ECSS standard doesn't support model based software engineering. The C version of the standards, according to ESA doesn't forbid model based software engineering anymore. It doesn't forbid the use of model based software engineering but it doesn't promote it either.

ECSS standards ask for detailed documents and then they have to prove it as well e.g. in case of unit testing they are very specific how they should do it. They require RUAG to have a plans, reports, test specifications, test execution in a certain way and then also prove it that all is done according to standards. Testers, project manager and quality assurance manager have to agree that they are only doing a certain activity because they are forced to do this by ECSS so they may not achieve the desired results from that activity and by using some other activity they can have better results.

Next version of the ECSS standard, version C, is on public review now it seems that it is taking a wrong turn because it requires more detailed documentation in a specific way. There are even more requirements on how to do things in a certain way. They are forced to do things in specific ways and prove every step they take. In this way, ECSS hampers innovation and good ways of doing things.

ECSS is sometimes a bit fussy and unclear to be interpreted properly. It requires doing a lot of things which are not clear. In case of code inspection there are only 2 or 3 requirements about them which don't help much. They need to be broken down for clarity. People writing the standards should think about what they need to standardize and why this is? They are aware that software starts by writing requirements and then a design is made that will fulfill these requirements. If we look at the standards, it is basically a design and ECSS standard is a solution to something. They haven't thought of the requirements first.

At RUAG, they are trying to follow ECSS standards in every new document and try to follow it exactly to avoid the amount of questions by customer. Each document explains how they will work and has special tables telling where to find the information or deviation from the standards. They will now try to adopt the new C standard system because customers will use that in future so they sent some comments to ESA about this but have not received any response from them. But they are looking for more effective way of being complaint with ECSS so that they could have a good argumentation for that according to standard to show that they fulfill the standard even they work in their own way. They try to limit them to few details and send in their statements of compliance in the proposal.

The validation department needs to learn more about ECSS standards and see how it can affect them for upcoming flight software projects. After the Galileo project they have realized that they can increase quality by following the standards. They need to educate themselves more about ECSS standards. They can also realize what customer requirements effectively by using the standards along with customer requirements.

It would be good if everyone knows at least the basics of the standards and then object leaders who are responsible for the overall activities and create most of the documents can fully aware of the standards. If there is a manager who has explained everything to developers and testers and shielded them from standards then they may not need to have full knowledge about them. Some basic knowledge is enough. For large projects the object leaders might not have enough time for all the hardware and software standards so they might need one person in software who should know about what is expected in the documents by ECSS standards.

Tailoring can be very helpful for solving problems. RUAG is at the lowest level of the customer-supplier chain [17] and often have 3 or 4 levels of customers above them. Mostly they can't really tailor standards because they get tailored standards from their customers. The customers usually add few tweaks to ECSS and they can't do any tailoring from that. If they were at the top level prime it would have been possible for them to tailor ECSS but in most of their projects they have ESA on the top which have their own opinions about tailoring.

Sometimes they are allowed to do things which are not in the ECSS and that can be in the form of tailoring of any particular standard. But it is very much up to the customer. Especially if the prime is confident in supplier's work and has worked with them before and if they have a good relationship.

ECSS promote reusability but they made it almost impossible. There are certain requirements concerning reusability about considering reusing already their own existing software or commercial-of-the-shelf. It is not enough that the reused software has been fully verified and validated in other projects. They have to make sure that it is fully verified it in the new project as well. They have to show how they are going to remedy this, fix this and how would they do the re-verification.

Every new mission has its own metric limits which are imposed by the standards. They have to show that the development standard of the reused software is compatible with the current development standard. There is a lot of work in reusing software and it is easy to say that we are not reusing software.

D.2.2.3.3 Galileo Software Standards (GSWS)

GSWS started as a tailoring of ECSS, it has many similarities of ECSS. In a way it is easier to follow because it is clearer. It has taken out some of the fussiness but it is strict as it has added number of details on how to do things or what to do. It requires more but is easier to follow. New ECSS standard will be more like the Galileo software standards. They have taken requirements from Galileo and put them in the new version of ECSS. They are more precise in what it requires. Following GSWS made them more compliant to ECSS.

It is strict on some source code metrics but better in clarity. Some of the requirements are tough and not worthwhile and it is not open to alternative approaches. Some of the requirements don't improve the quality as it requires independent module testing.

GSWS has tailored different irrelevant levels that ECSS didn't have and has imposed the tailorization. At RUAG, they would like to use ECSS standards with some interpretations of GSWS. They have used interpretation of GSWS to interpret some requirements of ECSS to make it easier to understand and implement.

.

D.2.2.4 Theme 4: Software Development Methodology

They have hard toll gates, like requirements review, architecture design review, detail design review (DDR) by their customers. It means that they are not allowed to do any activity which follows that review prior to it. Since they have strict toll gates like DDR all the design should be finished regardless if they are implementing them incrementally.

They are required to use waterfall method, so they start by writing the requirements and conduct reviews along with customers on the requirements. Once the requirements are finalized, the basic architecture of the software is designed after which they are allowed to start coding. They have early reviews like Software Requirement Review (SRR), Preliminary Design Review (PDR) where requirements are reviewed and they make sure if the requirements are ready.

They wanted to start coding before all the requirements are finished so they have introduced iterative aspect in waterfall development and named it as Integration-driven Development (IDD). The motivation is to built software faster, so they take one part of the software and more of less complete it and start testing it. IDD is introduced to have more interaction between different teams. Before IDD the time span between requirements specification, design, implementation and testing was very large and there was loss of information because of that.

D.2.3 Web-based Questionnaire Analysis Results

D.2.3.1 Introduction

An online survey has been conducted at RUAG to get a certain level of understanding about the V&V activities which they perform. The total respondents were 19, having varying level of expertise in terms of experience, age, areas of software development and testing and projects in which they worked at RUAG.

D.2.3.2 Areas of software development

The questionnaire results show that the percentage of respondents involved in code review, requirements review and unit testing are 58%, 53% and 53% respectively.47% of the respondents have been involved in architectural design, validation testing and design review. 42% in implementation, tool development and software requirements engineering where as 37% in detail design and integration testing. 26% has been involved in management where as 16% in acceptance testing, project management, quality assurance and system requirements engineering. Only 10% and 5% have been involved in development support and proposals respectively.

The above data shows that most of the people have been involved in the static verification activities.

D.2.3.3 SWD methodology

31% of respondents said they use integration driven development where as only 5% said its waterfall and the same percentage said its test driven development. 58% of the respondents didn't have clear idea about this, either they didn't understand the question or they didn't know about it.

D.2.3.4 Knowledge about ECSS standards

37% of the respondents knew the contents and how it affects their software development activities where as 21% knew in detail what it prescribes and its consequences for their software development. 31% knew it roughly. 5% were an expert who actively worked and implemented it whereas the other same percentage has never heard about it.

D.2.3.5 Degree to which ECSS affect their software development activities

42 % of the respondents were of the opinion that, the degree to which ECSS effect their software development activities is high, 31% said its very high where as only 21 % and 5% said its low and very low, respectively.

D.2.3.6 Effect of ECSS on the quality of software

21% didn't answer about the effects of ECSS on the quality of software. However, amongst the remaining respondents 47% said its mostly positive and the same percentage said its somewhat positive. Only 6% were of the opinion that its somewhat negative.

D.2.3.7 Effect of ECSS on the efficiency of software development

15% of the respondents didn't answer about the effect of ECSS on the efficiency of software development but 87% of the remaining respondents said that it is somewhat negative. 6% said that it is somewhat positive and the same percentage said that it is negative.

D.2.3.8 Effectiveness of verification activities

- Requirements review - 21% are not involved in it, the remaining 79% opined that it is effective
- Design review - 32% are not involved in it, 47% says its effective and only 21% thinks its ineffective
- Code review - 21% are not involved in it, 73% thinks its effective and only 6% thinks its ineffective
- Unit Testing - 21% are not involved in it, 68% thinks its effective and only 11% thinks its ineffective
- Integration Testing - 47% are not involved in it, 32% thinks its effective and 21% thinks its ineffective
- System Testing - 47% are not involved in it, the same percentage thinks its effective and only 6% thinks its ineffective
- Validation Testing - 21% are not involved in it, 68% thinks its effective and only 11% thinks its ineffective
- Acceptance Testing - 53% are not involved in it, 10% thinks its effective and 37% thinks its ineffective

D.2.3.9 Effort which the verification activities require

- Requirements review - 21% are not involved in it, 42% said that it requires high effort where as 37% said that it requires low effort
- Design review - 32% are not involved in it, 31% said that it requires high effort and 37% said that it requires low effort
- Code review - 21% are not involved in it, 58% said that it requires high effort and only 21% said that it requires low effort
- Unit Testing - 21% are not involved in it, 68% thinks that it requires high effort and only 11% it requires low effort
- Integration Testing - 47% are not involved in it, 37% thinks that it requires high effort and 16% thinks requires low effort
- System Testing - 47% are not involved in it, the same percentage thinks it requires high effort and only 6% thinks requires low effort
- Validation Testing - 21% are not involved in it, 68% thinks it requires high effort and only 11% thinks it requires low effort
- Acceptance Testing - 58% are not involved in it, 26% thinks it requires high effort and 16% thinks it requires low effort

D.2.3.10 Change in VA, if ECSS is not relevant

- Requirements review - 21% are not involved in it, 42% won't change the effort on it where as 32% said they would like to put more effort on it where as only 5% said they would put less effort on it
- Design review - 31% are not involved in it, 37% won't change the effort on it and , 16% would put more effort on it and the same percentage would put less effort on it.
- Code review - 21% are not involved in it, 53% won't change the effort on it and only 26% would put less effort on it.
- Unit Testing - 21% are not involved in it, 42% would put less effort on it, 31% won't change the effort on it and 6% would put more effort
- Integration Testing - 36% are not involved in it, 42% would put more effort and 16% won't change the effort and 6% would put less effort
- System Testing - 47% are not involved in it, 26% won't change the effort and 21% would put less effort and 6% would put more effort
- Validation Testing - 21% are not involved in it, 37% won't change the effort, 32% would put less effort and 10% would put more effort

- Acceptance Testing - 63% are not involved in it, and 16% won't change the effort, 10.5% would put less effort and the same percentage would not do this at all

D.2.3.11 Percentage of V&V activities of the total development costs

10% of the respondents didn't answer this, amongst the remaining people 41% said the cost should be 70%, 29% said the cost should be 60%, 12% said it should be 40%, 5.8 % said that it should be 30%, the same percentage said it should be 100% and 80%

D.2.3.12 Knowledge about ECSS standards

I have never heard of it - 1
I know roughly what it is about – 2
I know its contents and how it affects the SWD activities - 3
I know it detail what it prescribes and its consequences for SWD – 4
I am an ECSS expert and have worked actively in implementing and adopting it – 5

19 people answered

(1*1) + (6*2) + (7*3) + (4*4) + (1*5)/ 19= 2.9

Degree to which ECSS affect the SWD

Very high – 4
High – 3
Low – 2
Very low – 1

19 people answered
(1*1) + (3*2) + (8*3) + (6*4)/19 = 2.9

Degree to which ECSS affect the quality of SW

Mostly positive – 4
Somewhat Positive – 3
Somewhat Negative – 2
Mostly negative - 1

15 people answered
(7*4) + (7*3) + (1*2) + (0*1) /15 =3.4

D.2.3.13 Effect of ECSS on the efficiency of software development

Mostly positive – 4
Somewhat Positive – 3
Somewhat Negative – 2
Mostly negative - 1

16 people answered

(0*4) + (1*3) + (14*2) + (1*1) /16 = 2

Effectiveness of verification activities
Very effective- 4
Effective-3
Ineffective-2
Very Ineffective-1

Requirements Review
4 people are not involved in requirements review
(1*4) + (14*3) + (0*2) + (0*1) /15=3.1

Design review
6 people are not involved in design review
(1*4) + (8*3) + (4*2) + (0*1) /13=2.8

Code review
4 people are not involved in code review
(7*4) + (7*3) + (1*2) + (0*1)/15=3.4

Unit testing
4 people are not involved in unit testing
(3*4) + (10*3) + (2*2) + (0*1)/15=3.1

Integration testing
9 people are not involved in integration testing
(2*4) + (4*3) + (3*2) + (1*1)/10=2.7

System testing
9 people are not involved in system testing
(2*4) + (7*3) + (1*2) + (0*1)/10=3.1

Validation testing
4 people are not involved in validation testing
(7*4) + (6*3) + (2*2) + (0*1)/15=3.3

Acceptance testing
10 people are not involved in acceptance testing
(0*4) + (2*3) + (7*2) + (0*1)/9=2.2

Effort which the verification activities require
Very high effort- 4
High effort -3
Low effort-2
Very low effort-1

Requirements Review
4 people are not involved in requirements review
(3*4) + (5*3) + (6*2) + (1*1)/15=2.6

Design review
6 people are not involved in design review
(1*4) + (5*3) + (6*2) + (1*1) /13=2.4

Code review

4 people are not involved in code review
(1*4) + (10*3) + (4*2) + (0*1)/15=2.8

Unit testing
4 people are not involved in unit testing
(5*4) + (8*3) + (2*2) + (0*1)/15=3.2

Integration testing
9 people are not involved in integration testing
(2*4) + (5*3) + (3*2) + (0*1)/10=2.9

System testing
9 people are not involved in system testing
(7*4) + (2*3) + (1*2) + (0*1)/10=3.6

Validation testing
4 people are not involved in validation testing
(13*4) + (2*3) + (0*2) + (0*1)/15=3.8

Acceptance testing
11 people are not involved in acceptance testing
(1*4) + (4*3) + (3*2) + (0*1)/8=2.7

D.2.3.14 Change in VA, if ECSS is not relevant

Would put more effort on - 4
No change - 3
Would put less effort on - 2
Would not do activity at all - 1

Requirements review
4people are not involved in requirements review
(6*4) + (8*3) + (1*2) + (0*1)/15=3.3

Design review
6 people are not involved in design review
(3*4) + (7*3) + (3*2) + (0*1) /13=3

Code review
4 people are not involved in code review
(0*4) + (10*3) + (5*2) + (0*1)/15=2.6

Unit testing
4people are not involved in unit testing
(1*4) + (6*3) + (8*2) + (0*1)/15=2.5

Integration testing
7people are not involved in integration testing
(8*4) + (3*3) + (1*2) + (0*1)/12=3.6

System testing
9 people are not involved in system testing
(1*4) + (5*3) + (4*2) + (0*1)/10=2.7

Validation testing
4 people are not involved in validation testing
(2*4) + (7*3) + (6*2) + (0*1)/15=2.7

Acceptance testing
12 people are not involved in acceptance testing
(0*4) + (3*3) + (2*2) + (2*1)/7=2.1

Appendix E

This appendix contains discussion about the key findings, challenges and motivation for the solutions

Discussion

In this study, we identified challenges faced by two space companies during their V&V process. A triangulated research process has been used to for the validation of the results.

The results collected from web-based questionnaire are compared to understand differences between the work processes and to identify the factors causing those differences. The questionnaire was divided into four themes to cover different aspects of the study. Theme 1 is related to ECSS standards. Questions were asked about knowledge, effects of ECSS on software development, effects of ECSS on software Quality and effects of ECSS on the Efficiency of software development. Following figure 13 presents the comparison between RUAG and SSC over Theme 1, with important factors on X-axis and weighted average on Y-axis. See Appendix C and D, for further information about weighted average.

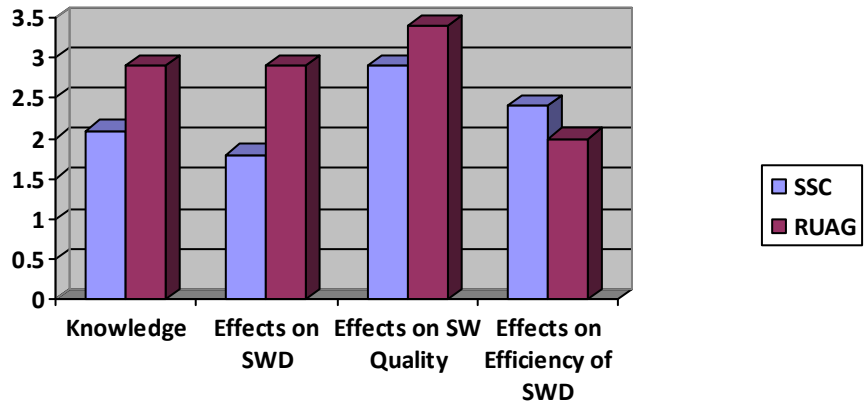

Figure 13: Comparison between SSC and RUAG over ECSS Standards

The above graph shows that ECSS knowledge distribution within the organization can affect the results of other three factors. Both SSC and RUAG are of the opinion that ECSS standards affect their software development process positively and it improves the quality of the software as well. But it was interesting to find out that both the case companies are of the opinion that ECSS standard has a negative effect on the efficiency of their software development. The Challenge-cause analysis shows that this negative effect is because of difference in interpretation of ECSS and documenting requirements for compliance proofs. ECSS standards are general in nature and even within ESA they are interpreted differently among different reviewers. Tailoring of ECSS standards can be a solution to cope with the challenge of documenting proofs for compliance but sometimes it is very hard for the organization to perform appropriate tailoring due to different reasons. For example in the case of SSC, due to inappropriate knowledge distribution about ECSS, they are unclear how to tailor ECSS standards. RUAG on the other hand is at a lower level of recursive customer-supplier relationship [17] hierarchy and get tailored requirements about ECSS from their customers. As results, they have very little chance to tailor ECSS according to their needs.

Theme 2 of the questionnaire was related to the effectiveness VAs. This was to judge the effectiveness of different VAs on this scale Very Ineffective –1, Ineffective – 2, Effective – 3 and Very effective – 4. Requirement Review, Design Review, Code Review, Unit Testing, Integration Testing, System Testing, Validation Testing and Acceptance Testing were judged on the above mentioned scale by both the case companies. Following figure 14 presents the comparison between RUAG and SSC over Theme 2, with VAs on X-axis and weighted average on Y-axis. See Appendix C and D, for further information about weighted average.

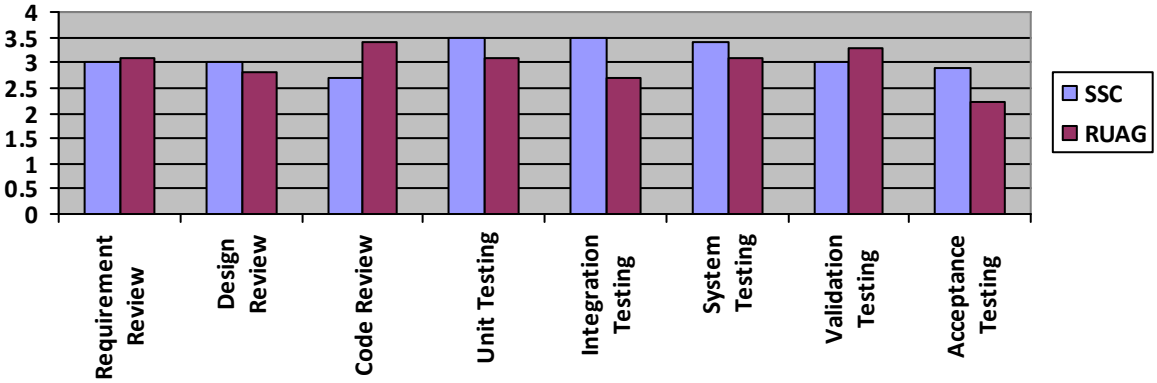

Figure 14 Comparison b/w SSC and RUAG over effectiveness of VAs

For RUAG the most effective VA's are 'code review' and 'validation testing' whereas for SSC 'unit testing' and 'integration testing', are more effective than other VA's. The least effective according to RUAG is acceptance testing whereas for SSC it is code review.

Theme 3 of the questionnaire was related to the effort required for VAs. This was to determine the effort required for different VAs on the scale Very low effort –1, Low effort – 2, High effort – 3 and Very high effort – 4. Requirement Review, Design Review, Code Review, Unit Testing, Integration Testing, System Testing, Validation Testing and Acceptance Testing were judged on the above mentioned scale by both the case companies. Following figure 15 presents the comparison between RUAG and SSC over Theme 3, with VAs on X-axis and weighted average on Y-axis. See Appendix C and D, for further information about weighted average.

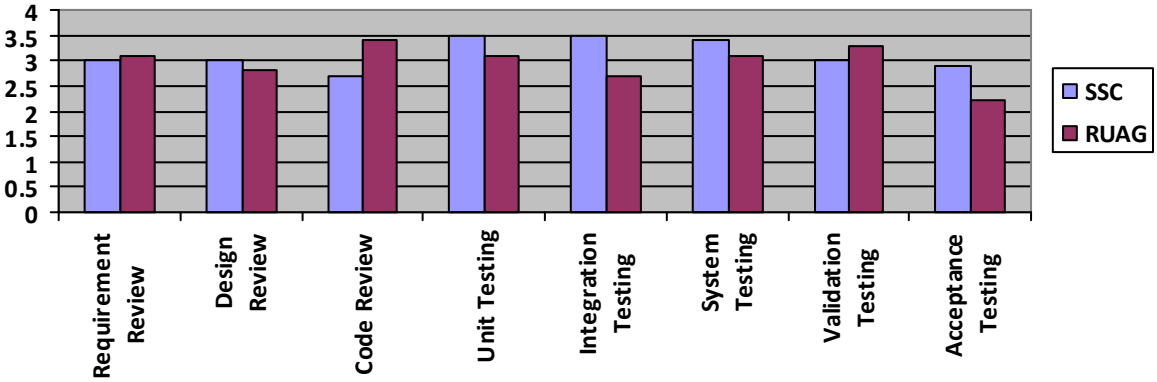

Figure 15 Comparison b/w SSC and RUAG over effort required for VAs

For RUAG 'validation testing' and 'system testing' requires more effort compared to other VA's whereas for SSC it is system testing. Both the companies think that 'requirements review' and 'design review' requires less effort compared to other activities.

Theme 4 of the questionnaire was related to the Change in VAs if ECSS is not relevant. It identified the change in different VAs on the scale; would not do activity at all –1, Would put less effort on –2, No Change –3 and Would put more effort on – 4. Requirement Review, Design Review, Code Review, Unit Testing, Integration Testing, System Testing, Validation Testing and Acceptance Testing were judged on the above mentioned scale by both the case companies. Following figure 16

presents the comparison between RUAG and SSC over Theme 4, with VAs on X-axis and weighted average on Y-axis. See Appendix C and D, for further information about weighted average.

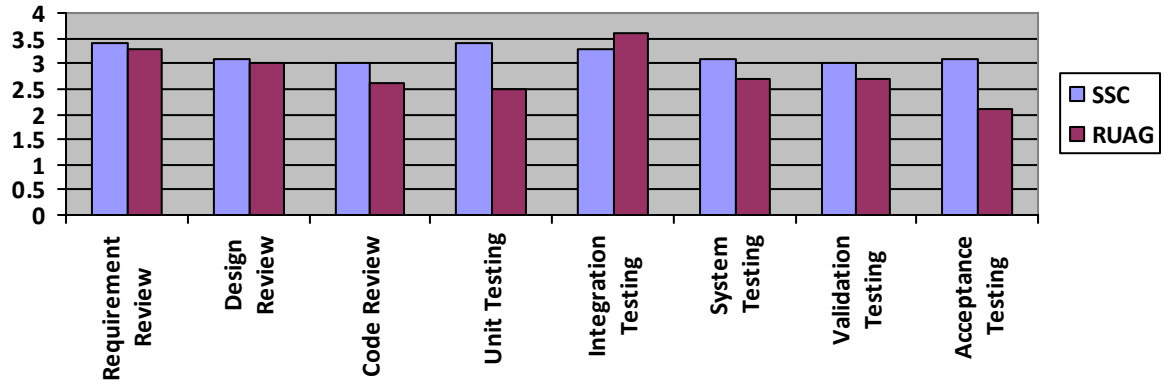

Figure 16 Comparison b/w SSC and RUAG over change in VAs

RUAG would like to put effort on 'integration testing' and less effort on 'acceptance testing' whereas SSC would like to put more effort on 'unit testing' and 'requirements review' and less effort on 'integration testing'.

By the help of challenge-cause analysis we concluded that both the companies are facing problems due to three main causes; ECSS Standards, Faults-slip-through among different stages and Inappropriate selection of cost-effective VAs. Challenges regarding ECSS and the recommendation solutions are presented above in the 'Paper' section of this thesis. Current Reality Tree (CRT) helped us to identify the main causes of the challenges and from the CRTs it is clear the both companies are facing problems due to inappropriate selection of VAs at different stages of software development life cycle. For example, at SSC, the developer himself performs unit testing of his code which can cause the fault slippage to the next stage. On the other hand at RUAG, code inspection is performed by an independent person at unit level to ensure full structural coverage, however they are not focusing on integration testing properly. Both the companies are facing problems in identifying the appropriate VAs at different stages. So there is a need for a strategy to select appropriate VAs.

Both companies wants to have simple measurements so they may evaluate the efficiency of their VAs, the improvement potential they can have and a method by which they can have a combination of VAs to ensure that defects are covered. Lars-Ola et al. [15] proposed a method at Ericsson for faults-slip-through measurements which has three steps, in the first step a strategy is developed about what should be tested at which phase. This will have a direct mapping about what types of faults a certain phase should cover. In the second step, the average cost of finding defects in various stages is determined; this can be obtained through the reporting system or by expert judgments. In the third step, an improvement potential is determined by calculating the difference between the costs of finding defects at the stage they were found to the cost of finding defects from the stage where they slipped through. The approach described the definitions and instructions about how to apply and follow up on the measurements. But the pre requisite for applying this method is a strategy and classification of defects and measurements about the cost of finding defects at various stages.

Software architectures based on the concept of *design diversity* are widely used in industry. VAs differ in efficacy of finding defects so their different combination can lead to an optimal verification process. Littlewood et al used this idea to convince practitioners for using combination of different approaches for fault detection as the diversity of defect finding approaches results in high quality software [1]. Koster further enriched this idea by combining it with Modern Portfolio Theory [3]. He investigated three factors which are important for the interplay between different techniques i.e. effectiveness level, resource allocation and variability of effectiveness. Experiments conducted by Myers, Selby and Murray et al. proved that combination of different VAs such as structural testing, functional testing and code review is much more effective than using them alone [4, 5,6]. Freimut et al introduced a customizable defect classification strategy based on domain specific knowledge of developers and software engineering knowledge of measurement experts [7]. The strategy is

developed by keeping in view the requirements of embedded systems only. It provides a mechanism for classifying the defects but no guidelines are given for selecting particular VA or combination of VAs for this classification. In [8], Nakamura et al explained an approach for identifying and classifying domain specific defects by emphasizing on inspection and change history. However, the prerequisites such as availability of source code, availability of analysts and availability of verifiers can make this approach non practical for small and medium sized companies due to the cost associated with it. Wagner summarized the state-of-the-art for defining domain specific defect classification approaches and discussed the factors affecting the industry usage of these approaches [8]. According to him, domain specific artefacts, defect classifications, defect classification di9mensions and their connection to quality models must be defined in advance, in order to achieve efficient VAs.

There are some strategies focusing on the selection of combination of VAs. Baret el al. [25] used the idea of mapping matrix for optimizing test process. The matrix is filled with the VAs and defects types in rows are columns, respectively. If any VA has the ability to detect a specific defect type, then the cell representing that VA in row and defect type in column is marked with "X". Wagner's model of quality economics presents a cost versus benefit analysis using more detailed metrics and equations [9, 26]. This model requires lot of initial data and cannot be a candidate strategy for the selection of cost-effective VAs, for the case companies. Murnane et al [27], presented a method for the selection of test cases by tailoring Black box testing. The limitation of this method is that it only checks one aspect of V&V process. Combination framework by Bradbury et al [28] focuses on mutation for defects and automated VAs and does not provide any guideline for the selection of VAs. The strategy presented by Strooper et al [10] for the selection of cost-effective VAs is a candidate solution for this challenge. It aims at selecting and evaluating different combinations of VAs by focusing on maximizing completeness and minimizing effort thus reducing cost and enhancing the efficiency, in four steps. Systematic way of applying empirical information makes this strategy a competitive approach for our study.

Strategy presented by Strooper et al [10] requires a calculation of improvement potential at the end of each iteration. FST [15] has been used effectively in telecom industry for the calculation of improvement potential for different stages of verification and validation. FST can be used effectively at the start of each iteration to calculate improvement potential for that stage. After applying the selected combination VAs identified in the step 2 of Strooper et al strategy, improvement potential is again calculated. Both improvement potentials are compared to validate whether the selected combination worked as expected or not. Following figure 17 further explains this idea. The stop means that particular combination for which ip-2 has been calculated is effective and can be used at this particular stage. As the Strooper et al strategy is iterative in nature so same process can be repeated for the next iteration until all the combinations have been applied or a tradeoff has been suggested by the management.

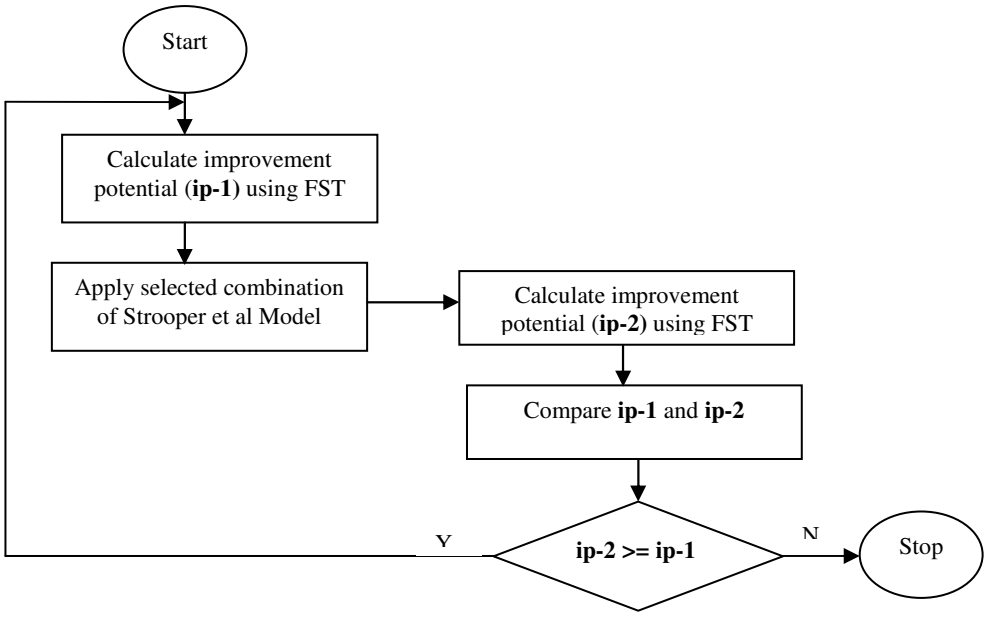

Figure 17: Strategy for using FST and Strooper et al Model

Appendix F

This appendix sets direction to the future work, identified during the study

Future Work

Based on this study we have identified several potential ideas to evaluate in reaching the goal for having cost efficient software projects in the space industry.

There are three different activities which are followed to get in compliance with the standards like ECSS. They are differentiated as the activities which:

- add to the actual quality of the software (eliminates existing or avoids the introduction of future defects),
- add to the confidence in the software,
- does neither of the above and only imposes an administrative cost to the process itself

The idea is to increase the use of VAs for the first type, adapt activities of the second type to the level of confidence of the customers, and reduce the costs for the third type to minimum. Our intention is to evaluate and refine this 'Cost of Compliance' model into a framework that can be used for V&V optimization. Also we could combine this model with a selection framework for V&V activities as the one presented in [10].

References:

1. B. Littlewood, P. Popov, L. Strigini, N. Shryane. (2000) "Modelling the effects of combining diverse software fault removal techniques". *IEEE Transactions on Software Engineering*, vol. 26(12). pp. 1157-1167. IEEE Computer Society, Washington, DC.

2. Jones. M, Mazza. C, Mortensen. K, Scheffer. A, (1997) "1977 – 1997: Twenty Years of Software Engineering Standardization in ESA", Bulletin Nr.90, ESA Publication Division, Noordwijk, the Netherlands.

3. K. Koster, "Using Portfolio Theory for Better and More Consistent Quality", in *Proceedings of The International Symposium on Software Testing and Analysis (London, United Kingdom, 9 – 12 July 2007),* ISSTA'07.

4. J. Myers. "A controlled experiment in program, testing and code walkthroughs inspections". *Communications of the ACM*, vol. 21/9, September 1978, ACM, New York, NY, USA.

5. R. W. Selby. "Combining software testing strategies: An empirical evaluation", in *Proceedings of the Workshop on Software Testing (Banff, Canada, July 1986)*, IEEE Computer Society, Washington, DC. USA.

6. M. Wood, M. Roper, A. Brooks, J. Miller. "Comparing and combining software defect detection techniques: a replicated empirical study", in *Proceedings of 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering (Zurich, Switzerland 22-25 September 1997),* ESEC '97/FSE-5, ACM New York, NY, USA, Springer-Verlag, New York, USA.

7. B. Freimut, D. Christian, M. Ketterer. "An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management", in *Proceedings of 11th IEEE International Software Metrics Symposium(Como, Italy, 19-22 September 2005)*, METRICS'05, IEEE Computer Society, Washington, DC. USA.

8. T. Nakamura, L. Hochstein, V. R. Basili. "Identifying domain-specific defect classes using inspections and change history", *in Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering (Rio de Janeiro, Brazil, 21-22 September 2006),* ISESE '06, ACM New York, NY, USA.

9. S. Wagner. "Defect Classification and Defect Types Revisited", in *Proceedings of International Workshop on Defects in Large Software Systems (Washington, USA, July 2008),* DEFECTS'08, ACM New York, NY, USA.

10. Margaret A. Wojcicki, Paul Strooper, "An Iterative Empirical Strategy for the Systematic Selection of a Combination of Verification and Validation Technologies", in *Proceedings of 29th International Conference on Software Engineering (Minneapols, MN, USA, 20 - 26 May 2007)*, ICSE 2007, IEEE Computer Society, Washington, DC. USA.

11. Creswell, J. (2002) Research Design: Qualitative, Quantitative, and Mixed Methods Approaches (Second Edition), Sage Publications, London, UK.

12. Seaman. C. (1999) "Qualitative Methods in Empirical Studies of Software Engineering", *IEEE transactions on Software Engineering*, vol 25(4), pp. 557-572. IEEE Computer Society, Washington, DC. USA.

13. Swedish Space Corporation, http://www.ssc.se, (last visited: November 17, 2009).

14. European Corporation for Space Standardization, http://www.ecss.nl, (last visited: November 17, 2009).

15. Damm. L. (2006) "Faults-Slip-Through - A Concept for Measuring the Efficiency of the Test Process", *Journal of Software Process: Improvement and Practice*, vol 11(1), pp. 47-59. Wiley InterScience.

16. Brat. G, Denney. E, Giannakopoulou. D, Frank. J, Jonsson. A, "Verification of Autonomous Systems for Space Applications", in *Proceedings of IEEE Aerospace Conference (NASA Ames Res. Centre, Moffett Field,CA, USA, 04 - 11 March 2007)*, IEEE Computer Society, Washington, DC. USA.

17. ECSS Std ECSS-E-40 Part 1B, (2003) "Software product assurance", ESA-ESTEC, Requirements & Standards Division, Noordwijk, the Netherlands.

18. Asquier. J, Battrick. B, (2005) "The Newsletter of the European Cooperation for Space Standardization No.8", ESA Publication Division, Noordwijk, the Netherlands.

19. ECSS Std ECSS-Q-80B, (2003) "Software - Part 1: Principles and requirements", ESA-ESTEC, Requirements & Standards Division, Noordwijk, the Netherlands

20. Ponz. D, Spada. M, "Three Years of ECSS Software Standards, An Appraisal and Outlook", in discussion of monthly Seminar OPS-G(Darmstadt, Germany, 2o January 2006), European Space Operations Centre (ESOC), Darmstadt, Germany.

21. Balestra. L, "European Cooperation for Space Standardization (ECSS)", *in Proceedings of Trilateral Safety & Mission Assurance Conference (Noordwijk, the Netherlands, 14 – 16 April 2008)* TRISMAC 2008.

22. Software Development Plan (SDP) SMALL GEO, Space Division at Swedish Space Corporation, Stockholm, Sweden

23. Software Verification and Validation Plan SMALL GEO, Space Division at Swedish Space Corporation, Stockholm, Sweden

24. Software Development Plan (SDP) , RUAG Aerospace AB, Goteborg, Sweden

25. N. Barret, S. Martin, and C. Dislis, "Test Process Optimization: Closing the Gap in the Defect Spectrum," *In Proceedings of the International Test Conference, 1999*, pp. 124-129.
26. S. Wagner, "Software Quality Economics for Combining Defect-Detection Techniques", *In Proceedings of the Net. Object Days 2005 Workshop on Software Quality (WOSQ'06)*, 2006, pp. 69-74.
27. T. Murnane, K. Reed, and R. Hall, "Tailoring of Black-Box Testing Methods," *In Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06), 2006,* pp. 292-299.
28. J. S. Bradbury, J. R. Cordy, and J. Dingel, "An Empirical Framework for Comparing Effectiveness of Testing and Property-Based Formal Analysis," *In Proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, 2005*, pp. 2-5