An Independent Software Verification and Validation Process for Space Applications

Ana Maria Ambrosio¹, Fatima Mattiello-Francisco² and Eliane Martins³

National Institute for Space Research - INPE, São José dos Campos, Brazil, 12227-010

This paper presents an Independent Software Verification and Validation process that applies reviews for verification and a systematic testing methodology to guide validation. This process was applied to a pilot project named Quality Software Embedded in Space Missions (QSEE) at INPE and pointed very good results. The main feature of the process is that it uses a particular testing methodology named CoFI and an automatic test cases generation tool based in state-models. These features allowed systematizing validation activities which were carried on by a team not involved with the software development. The main activities of the process, the results in terms of the errors found not only through the reviews but also through the tests are presented. Lessons learned including drawbacks and benefits are discussed as well.

I. Introduction

The space agencies has being outsourcing their products in order to fulfill the planning of spacecraft building program. The spacecraft hardware already has standards and testing guidelines for qualification and acceptance purposes¹. Currently, there are not similar standards for software. Each agency defines its own acceptance rules and adopts different techniques in order to assure the software has achieved an adequate quality level for flight. Different techniques for assuring software quality does exist but what should be the most effective and efficient are yet an open question.

The most frequent techniques being applied to improve and assure quality in space software are reviews and tests. Reviews along the software development assure good quality of documents and are a good opportunity to customers to check, gradually, the comprehension of the suppliers about the software details. Tests are used to demonstrate that the implementation meets the requirements.

For critical embedded software, safety and criticality analysis are recommended. Some examples of techniques usually applied are: Software Failure Mode and Effect Analysis (SFMEA), Software Failure Mode Effect and Criticality Analysis (SFMECA), Hardware-Software Interaction Analysis (HSIA), Software Common Cause Failure Analysis (SCCFA). These techniques help the verification of robustness of a critical software design. However, reviews and analysis techniques are not enough to support validation especially for software acceptance; a good set of tests is invaluable. Using the Voas² analogy that dirty water may flow by clean pipes, meaning that a software product that not conforms to the corresponding specification may be produced even though a good development process is adopted to produce it. Similarly, we believe that only performing traditional reviews are not enough to a client accept a software product and that to execute tests are invaluable before accepting software.

We propose an Independent Software Verification and Validation process that includes reviews and test activities for test design, test execution and test result analysis, which improve confidence in validation and acceptance of space applications software. The reviews are tailored from those required in ECSS-E-40³ and ECSS-Q-80B⁴ standards.

The main contribution of this ISVV process is the establishment of a systematic method to design test cases as part of the validation. The tests are to be executed against the software product delivered to the client. Although it is impossible to prove correctness by testing all possible states of a medium to large software system within a reasonable amount of time, the tests are the only way to check the final executable version of the software⁵.

¹ Technologist, DSS, Av. Dos Astronautas, 1758 São Jose dos Campos -12227-010 Brazil, ana@dss.inpe.br
² Technologist, ETE, Av. Dos Astronautas, 1758 São Jose dos Campos -12227-010 Brazil, fatima@dss.inpe.br
³ Researcher, UNICAMP, Campinas - Brazil, eliane@ic.inpe.br

1 American Institute of Aeronautics and Astronautics

Copyright © 2008 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.
According to the IBM report\(^6\), almost 30% of the testing task can be the writing of test cases so; to guide the test design via a well-established methodology will reduce the ISVV effort and improve quality in the final product. The testing methodology adopted in the ISVV process is named CoFI\(^7\). This methodology includes:

(i) to use tools to automatically generate test cases based on state-based models;
(ii) to guide the creation of test cases taking into account abnormal inputs and
(iii) to apply fault-injection concepts for creating hardware fault based test cases in order to evaluate the fault tolerance mechanisms.

The objective of adopting this test methodology as part of the ISVV process is to help detecting errors that had continued to exist in spite of the reviews and the tests applied by the supplier during the development phase.

The ISVV process was applied in the project named Quality Software Embedded in Space Missions (QSEE)\(^8\) developed at INPE. In this project the supplier was a Brazilian software company certified with Capability Maturity Model Integration (CMMI) level-3 process. The ISVV team was composed by people from the Computing Institute of State University (UNICAMP) and people from INPE not involved with client responsibilities.

This paper is organized as follows: Section 2 presents the ISVV process and its activities; Section 3 discusses the results obtained during the verification activities; Section 4 presents the validation activities focusing on the testing methodology and discusses the results as well. Section 5 addresses the lessons learned, and Section 6 concludes the paper.

II. The ISVV Process

According to ECSS-E-40B standard\(^3\), software verification aims to confirm that adequate specifications and inputs exist for any activity, and that the outputs of the activities are correct and consistent with the specifications and inputs, whereas validation is to confirm that the requirements baseline functions and performances are correctly and completely implemented in the final product. The ISVV process comprises verification, validation and management activities. These activities are organized as follows: one for covering management, six for covering verification and three for validation. The activities and their corresponding techniques are shown in Table 1.

<table>
<thead>
<tr>
<th>Activity type</th>
<th>Activity</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>System Requirement Review (SRR)</td>
<td>review</td>
</tr>
<tr>
<td>Verification</td>
<td>Preliminary Design Review (PDR)</td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td>Critical Design Review (CDR)</td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td>Detailed Design Review (DDR)</td>
<td></td>
</tr>
<tr>
<td>Verification</td>
<td>Qualification Review (QR)</td>
<td></td>
</tr>
<tr>
<td>Validation</td>
<td>Acceptance Review (AR)</td>
<td></td>
</tr>
<tr>
<td>Validation</td>
<td>Test Design Activity (TDA)</td>
<td>testing</td>
</tr>
<tr>
<td>Validation</td>
<td>Test Execution Activity (TEA)</td>
<td></td>
</tr>
<tr>
<td>Validation</td>
<td>Test Result Analysis (TRA)</td>
<td></td>
</tr>
</tbody>
</table>

The V&V Management (VVM) activity consists of the establishment of personnel, costs, schedule, methods to be applied, the exact inputs and outputs to be required in each activity, the tasks and subtasks, among others.

The verification activities comprise basically reviews. In reviews, an independent team examines the delivered documents. During a review meeting, an overview of the documents is presented and the reviewers point out the founded discrepancies. After an agreement, reviews item discrepancies (RIDs) are registered. Providences must be taken in order to correct the discrepancies.

The validation activities are related with the following test tasks: (i) design of test cases associated to the Test Design Activity (TDA), (ii) execution of the test cases associated to the Test Execution Activity (TEA) and (iii) analysis of the test results associated to the Test Result Analysis (TRA) activity.

Table 2 lists the input and output associated to each process’s activity. These activities were defined to comply with the requirements of the QSEE project. Column two shows the document name and indicates the team responsible to deliver it. Three independent teams are referred: client, supplier and isvv.
<table>
<thead>
<tr>
<th>Activity</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>VVM</td>
<td>Baseline Requirements - client</td>
<td>Independent Sw V&amp;V Plan – isvv &amp; client</td>
</tr>
<tr>
<td>SSR</td>
<td>Baseline Requirements - client</td>
<td>SRR Report - client</td>
</tr>
<tr>
<td></td>
<td>Independent Software V&amp;V Plan - client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software Development Plan - supplier</td>
<td></td>
</tr>
<tr>
<td>PDR</td>
<td>Baseline Requirements - client</td>
<td>PDR Report - client</td>
</tr>
<tr>
<td></td>
<td>PDC-OBDH Communication Protocol - client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PDC-EPPs Communication Protocol - client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software Development Plan - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software Technical Specification - supplier</td>
<td></td>
</tr>
<tr>
<td>DDR</td>
<td>Software Technical Specification - supplier</td>
<td>DDR Report - client</td>
</tr>
<tr>
<td></td>
<td>Software Design Document - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software Test Plan - supplier</td>
<td></td>
</tr>
<tr>
<td>CDR</td>
<td>Independent Software V&amp;V Plan - client</td>
<td>CDR Report - client</td>
</tr>
<tr>
<td></td>
<td>Test Specification for Instrument Level - isvv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Specification for Subsystem Integration Level - isvv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software Design Document - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software Test Plan - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software Test Report - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software User Manual - supplier</td>
<td></td>
</tr>
<tr>
<td>TDA</td>
<td>Baseline Requirements - client</td>
<td>Test Specification for Instrument Level - isvv</td>
</tr>
<tr>
<td></td>
<td>PDC-OBDH Communication Protocol - client</td>
<td>Test Specification for Subsystem Integration Level - isvv</td>
</tr>
<tr>
<td></td>
<td>PDC-EPPs Communication Protocol - client</td>
<td>Test Specification for System Level - isvv</td>
</tr>
<tr>
<td></td>
<td>Software Technical Specification - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software Design Document - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software User Manual - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OBDH Simulator Manual – supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EPPs Simulator Manual – supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Simulator and testing tool Manual – client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EGSE Description - client</td>
<td></td>
</tr>
<tr>
<td>TEA</td>
<td>OBDH Simulator Manual – supplier</td>
<td>Instrument Level Test Report - isvv</td>
</tr>
<tr>
<td></td>
<td>EPPs Simulator Manual – supplier</td>
<td>Subsystem Integration Level Test Report - isvv</td>
</tr>
<tr>
<td></td>
<td>Simulator and testing tool Manual – client</td>
<td>System Level Test Report - isvv</td>
</tr>
<tr>
<td></td>
<td>EGSE Description -client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Specification for Instrument Level - isvv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Specification for Subsystem Integration Level - isvv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Specification for System Level - isvv</td>
<td></td>
</tr>
<tr>
<td>TRA</td>
<td>All input used in TDA</td>
<td>Report of non-conformances – isvv &amp; client</td>
</tr>
<tr>
<td></td>
<td>All reports produced in TEA</td>
<td></td>
</tr>
<tr>
<td>QR</td>
<td>Instrument Level Test Report - isvv</td>
<td>QR report</td>
</tr>
<tr>
<td></td>
<td>Subsystem Integration Level Test Report - isvv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software User Manual - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Product of Software - supplier</td>
<td></td>
</tr>
<tr>
<td>AR</td>
<td>System Level Test Report - isvv</td>
<td>AR report</td>
</tr>
<tr>
<td></td>
<td>Software User Manual - supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Product of Software - supplier</td>
<td></td>
</tr>
</tbody>
</table>
III. Verification

The SSR, PDR, DDR and CDR reviews are named development reviews as they are performed during the software development lifecycle. These reviews were tailored from ECSS\(^4\) to the QSEE project\(^6\). The review team was composed by: ISVV team, supplier, client and external people. As shown in Table 2, during each review, documents provided by different teams were analyzed. In order to show the results of the reviews applied in QSEE project, Table 3 summarizes the number of RIDs and indicates what team should answer the RIDs. It is important to notice that not only the supplier was under development reviews, instead all team that produced a document were.

| TABLE 3. Number of RIDs produced during the software development reviews. |
|-----------------------------|----------------|----------------|----------------|----------------|
|                             | SRR | PDR | DDR | CDR | Total |
| client                      | 6   | 16  | -   | -   | 22    |
| supplier                    | 26  | 40  | 37  | 45  | 148   |
| isvv                        | 4   | -   | -   | 21  | 25    |
| total                       | 36  | 56  | 37  | 66  | 195   |

In all the reviews 195 RIDs were formally produced. Although many of the RIDs indicated minor problems, some critical problems were detected through this verification technique. Table 4 exemplifies some of the most significant problems raised in each review and indicates the team responsible to solve them.

| TABLE 4. Example of problems detected during the software development reviews. |
|-----------------------------|----------------|----------------|----------------|----------------|
|                             | SRR | PDR | DDR | CDR |
| Protocols specification and a description of the operation modes were not provided by the client. | Delivers and deadlines were not specified by the supplier. Technical specification did not include interruptions for data acquisition and faults treatment (supplier). | Software design did not include behavior of how to deal with commands sequencing. Message Sequencing was misunderstood (supplier). | Test specifications did not included performance testing and a testing coverage analysis (isvv). |

The two last reviews QR and AR are performed after the integration of the equipment embedding the software into the satellite. These reviews did not occurred yet in the QSEE project.

IV. Validation

The Validation of the ISVV process was conducted as systematically as possible. The validation included the acceptance aspect of the system comprising embedded software. The design of effective sets of test cases was hard and expensive but it was compensatory in this project. The testing activities were conducted in order to reduce costs as more repeatable they were. The testing activities comprised specifying the test cases, adjusting the test cases to the test execution mean, executing the test, analyzing the results and producing reports about the test execution. The validation process was based on previous experience in defining the CoFI process\(^7\). The testing tools adopted in this process were based on those proposed in the research project named ATIFS\(^11\). The testing execution tools, used by the isvv team, were those produced by the development team.

The activities TDA, TEA and TRA were repeated for the 3 levels of integration defined in QSEE project: instrument (simulators are used to provide all external interfaces), subsystem (real instruments operate in parallel and simulators are used to provide the external interfaces) and system (real equipments provide the external interfaces).

A. Test Design Activity (TDA)

The Test Design Activity adopted the CoFI testing methodology\(^7\) that consists of a systematic way to create test cases for space software. The CoFI comprises steps to identify a set of services; to each service to create different models represented in finite state machines (state diagrams). These models represent the behavior of the system under test (SUT) under the following kind of inputs arriving: normal, specified exceptions, corrects but in wrong moments and inputs caused by hardware faults. The software behavior is represented in small models taking into account decomposition in terms of: (i) the services provided by software and (ii) the types of behavior under classes

4

American Institute of Aeronautics and Astronautics
of inputs. Figure 1 shows schematic graphic where the horizontal lines indicate the types of behavior (Fault Tolerance, Sneak Path, Specified Exception and Normal) and the vertical the services. Each element, represented by a box is the set of models. Observe that, more than one model may be created to represent a type of behavior of a given service, as illustrated by the dots into the boxes.

![Diagram of system behavior broken into small models.](image)

**Figure 1.** System behavior broken into small models.

1. **Services identification**

   The software under test in the QSEE project is a software embedded into the Payload Data Handling Computer that will be part of a scientific X-ray instrument onboard of a scientific satellite under development at INPE. The eleven services identified in this software for test purposes were: initialization, scientific data, housekeeping data, test data, diagnostic data, memory dump, changing operation mode, load&execute program, OBDH syntax, EPP syntax and special commands.

2. **Hardware fault, exceptions and Models**

   In QSEE project context, the following hardware faults were taken into account: communication, memory and processor faults.

   The types of behavior of the system under test were referred as: Comm (fault tolerance under communication faults), M&Pro (fault tolerance under memory and processor faults), SExc (specified exceptions), Spat (sneak path) and Norm (normal behavior), as shown in columns of Table 5. The number of models created by service x type of behavior is presented in Table 5.

**TABLE 5. Number of state-based models created to QSEE considering the services in isolation.**

<table>
<thead>
<tr>
<th>Services</th>
<th>Behavior</th>
<th>Norm</th>
<th>SExc</th>
<th>Spat</th>
<th>Comm</th>
<th>M&amp;Pro</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Scientific data</td>
<td></td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Housekeeping</td>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Test data</td>
<td></td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Diagnostics</td>
<td></td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>Memory dump</td>
<td></td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>Change operation mode</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Load &amp; execute program</td>
<td></td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>OBDH msg syntax</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>EPPs msg syntax</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Special commands</td>
<td></td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td><strong>System load</strong></td>
<td></td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td><strong>Service combination</strong></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>24</td>
<td>24</td>
<td>22</td>
<td>13</td>
<td>14</td>
<td>99</td>
</tr>
</tbody>
</table>

5

American Institute of Aeronautics and Astronautics
One may observe that 2 models were created to represent the normal behavior of the Initialization service. This means that, in QSEE project, the initialization service is equally possible using one or two processors. So the behavior to the system under test when only one processor is available is represented in one Norm model and when two processors are available, this behavior is represented by a different model; but both are considered as normal behavior for the operational point of view.

Two types of models were specially created to complete the tests activities for validation purpose: System load and Service combination, as illustrated in the last lines of Table 5. Besides the models for services in isolation, the models, for system load and combination of service, were created to: (i) cover repetition of commands in order to cause memory overflow, and overload in command buffers, and (ii) to combine all services to be executed in long sequences.

In total, the software behavior was represented in 99 different models, designed by the isvv team. The models reflected the information obtained from the documents provided by the client and supplier teams.

3. Models and Test cases

The test cases were derived automatically from each model. Each model were submitted to the Condado tool. The set of test cases created in this activity are named abstract test suite as some information must yet be provided before execution. In total 771 test cases were derived from the models of services in isolation more the 3 modes for System load. Table 6 shows the number of test cases created by service.

The test cases obtained from the Service combination models detected the same errors founded before (services in isolation) so these test cases were used for regression test.

B. Test Execution Activity (TEA)

In the Test Execution Activity, the abstract test suite generated in TDA is converted into a parameterized executable test suite, and the isvv team is ready to the starting of the testing campaigns against the software product delivered by the supplier. The test cases, comprising the parameterized executable test suite, have two main parts: inputs, outputs. The inputs are the events produced to excite the software under test whereas the outputs are the expected answer produced. All the output observed during the execution of each test case is logged not only for result analysis but also for future reference. The observable output is associated to each test case and compared with the expected output. According to the result of the comparison a verdict is given as follows: passed, passed with restriction, failed, or inconclusive.

C. Test Result Analysis (TRA)

The Test Result Analysis activity consists of an analysis meeting; during which client and isvv teams analyze the test cases results. Only the test case with failed and passed with restriction verdicts are analyzed. As a result of such analysis, non-conformance reports (NCRs) are created whenever it is mandatory a correction in the software product or in documents provide by the supplier.

Table 6 shows the number of test cases that detected errors and the corresponding number of NCRs.

<table>
<thead>
<tr>
<th>Services</th>
<th># test cases</th>
<th># founded errors</th>
<th># NCRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>46</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Scientific data</td>
<td>98</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>Housekeeping</td>
<td>109</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test data</td>
<td>123</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Diagnostics</td>
<td>69</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>Memory dump</td>
<td>119</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Change operation mode</td>
<td>42</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Load &amp; execute program</td>
<td>29</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>OBDH msg syntax</td>
<td>64</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>EPPs msg syntax</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Special commands</td>
<td>52</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>System load</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>771</strong></td>
<td><strong>104</strong></td>
<td><strong>36</strong></td>
</tr>
</tbody>
</table>

| TABLE 6. Test cases that detected errors and indicated NCRs. |
The significant reduction in number of detected errors compared to the number of RNCs, particularly in services Scientific data and Diagnostic Data, is due to that the automation causes repeated test cases. Then, generally, a single problem is detected by several test cases.

Another important observation is that, the number of founded errors and NCRs are related to both code and textual documents. Since the models represent the software behavior according to the information obtained from the textual documents (such as Protocol Specification, Software Design, and Manuals), all non-conformances between code and document were computed as a detected error, resulting approximately 75% code errors and 25% document non-conformance errors.

V. Lessons learned

System load and combination of services are important types of tests to complete the validation activities. As they were not previously considered in COFI methodology, we carried out some experiments in QSEE project:

(i) in one experiment we used an existing set of test cases derived from a service, in normal behavior, and included manually repetitions of commands;
(ii) in another, we combined only normal behavior models of several different services;
(iii) a third experiment, we combined in a single model all the models that detected errors.

From the two first experiments, no error was founded. With the third experiment, i.e., when generating test from a model that combined several other models, we found that such a combined model detected the same set of errors with a significantly reduced number of test cases. So it was taken as reference to be used in Regression tests.

Taking into account that the SUT were developed under rigorously software engineering standards of quality by the company and by the reviews, the results with validation shown that it was very effective to apply the CoFI testing methodology as a systematic method to generate the test cases in TDA activity.

The test methodology served as a guideline to focus the tester’s attention to the faults and exceptions that occur during the software’s operation, leading to the design of situations the developers had not thought of.

The effort to create many models is compensated in different point of views. First, the number of faults detected, not only in the code, but also in documents surprised us as the documents had already been submitted to about four reviews. Second, we could realize that the models may be reused for similar software, mainly because they represent services not the implementation solutions. Considering that in space systems the set of services is very similar from one mission to another, many of the models may be reused. The organization of the test cases and the precision on how the tests are interpreted via models are superior when compared to tests created without any methodology.

VI. Conclusion

The Independent Software Verification and Validation (ISVV) Process presented in this article was applied to a pilot project named Quality Software Embedded in Space Missions (QSEE) carried on at INPE in cooperation with a national software company and with the State University of Campinas, a Brazilian university. The main objective of the QSEE project was not only to transfer technology to national software industry but also to establish a process to accept space software at INPE. In this process we introduced the use a testing methodology named CoFI.

One of the goals of applying a testing methodology as part of an ISVV process was to reduce cost with the systematization of the test design activity. The application of CoFI in a real space project was a great challenge and proved to be feasible and very effective. The set of test cases generated according to CoFI methodology detected many different errors in real space embedded software. The errors were found not only in code but also in the documents, so, it improved reliability in acceptance of all kind of the deliverables: documents and code.

The good results with this process have encouraged us to continue improving it and motivated other project managers to adopt the ISVV process in future satellite planed in the Brazilian space missions Program. It represented a step towards achieving remaining challenges on space software validation.

Acknowledgments

The authors acknowledge the financial support from Financiadora de Estudos e Projetos (FINEP) to the QSEE research project and all those involved.

References

2 Voas, J.M. and Graw, G. M. V.; “Software Fault Injection – Inoculating programs against errors”.

American Institute of Aeronautics and Astronautics


7Ambrosio, A M., Martins, E.; Santiago, V; Mattiello-Francisco, MF; N.L.Vijaykumar, S.V. de Carvalho. A methodology for designing fault injection experiments as an addition to communication systems conformance testing Proceedings of the First Workshop on Dependable Software - Tools and Methods in the IEEE Conference on Dependable System and Network, 28 June – 1 July 2005, Yokohama, Japan.


