TESTING CRITICAL SOFTWARE: A CASE STUDY FOR AN AEROSPACE APPLICATION

Denis S. Loubach, João C. S. Nobre, Adilson M. da Cunha, Luiz A. V. Dias, Marcos R. Nascimento, Brazilian Aeronautical Institute of Technology (ITA)

Walter A. Dos Santos, National Institute for Space Research (INPE), São José dos Campos, São Paulo, Brazil

Abstract

This paper reports an academic experience at the Brazilian Aeronautical Institute of Technology using automated software testing tools applied to a critical real time embedded systems. The work uses a Problem-Based Learning - PBL teaching methodology, a Rational Unified Process - RUP tailoring, and the IBM-Rational Quality Architect RealTime and the IBM-Rational Test RealTime tools. This academic experience has been originated on the needs and specifications of the Brazilian Aeronautics and Space Institute - IAE and the Brazilian National Institute for Space Research – INPE. The Case Study is based upon three ongoing realistic Software Projects, all of them using real time embedded software respectively named Unmanned Aerial Vehicles - UAV, Student Satellite - SSAT, and Ground Control Station - GCS. This experience has involved 15 senior computer engineering undergrads and 18 graduate students to develop, test, verify, and validate the system, in just 17 academic weeks, and four aggregation levels: Computer Software Units - CSU, Computer Software Components - CSC, Computer Software Configuration Items - CSCI in just one Computer Software System - CSS demanding software engineering state-of-the-art processes, and real time embedded software respectively named Unmanned Aerial Vehicles – UAV, Student Satellites - SSAT, and their Ground Control Stations – GCS.

The Problem-Based Learning Methodology – PBL on this Case Study was applied using Real Time Embedded Software, and Software Quality, Reliability and Safety disciplines [1, 2], resulting into a more dynamic teaching.

From students’ point of view, the PBL [3] has provided the development of group workability, and the enhancement of their individual study stimulus in the academic environment within a realistic framework, following individual students’ rhythm, concerning reaching courses’ deadlines. In this way, the instructor or professor was able to get used to each student.

On the other hand, the professor had to change his habits, adding new roles and/or replacing old ones. He no longer was the only disseminator of new knowledge. The students researching over the Internet, books, consultancy with other professors, and even visiting factories and enterprises were able to add important new theoretical and practical knowledge to the project.

The paper starts by top-down describing a Computer Software System – CSS, and then introduces the tools used to model and implement a software prototype. After that, performed tests shown together with software quality metrics are discussed. It ends up presenting some practical results and key conclusions.

Introduction

This article describes an Academic Case Study based upon real needs of the Aeronautics and Space Institute - IAE and the National Institute for Space Research – INPE, involving real time embedded software development, to be utilized in Unmanned Aerial Vehicles – UAV, Student Satellites - SSAT, and their Ground Control Stations – GCS.

What is testing?

Testing is a disciplined process that consists of checking that applications (including components) behavior, performance, and robustness match expected criteria. Although usually implicit, one of
the main accepted standards is that applications should be as defect-free as possible. Therefore, expected behavior, performance, and robustness should be formally described and measurable. In order to meet this demanding, debugging is necessary as it literally means removing defects or "bugs" and is considered only part of the testing process [4].

**Computer Software System - CSS**

This section presents the main scenario and the software prototype mission.

**Main Scenario**

The CSS prototype is called VANT-EC-SAT, which in Portuguese is an acronym for Unmanned Aerial Vehicle - UAV, Ground Control Station - GCS, and Student Satellite - SSAT. This Case Study is a realistic project aiming the implementation of a CSS [5] prototype. The CSS has been developed preserving quality, reliability and safety, during 17 weeks on the second academic semester of 2005 at the Brazilian Aeronautical Institute of Technology - ITA.

A group of 15 senior computer engineering and 18 graduate, masters and doctorate students were involved to build up the software prototype and perform tests.

The VANT-EC-SAT CSS was divided into three Computer Software Configuration Items – CSCI, presented on Figure 1. These Items represents the UAV, the GCS and the SSAT, respectively. Each CSCI was formed from its CSC, which in turn generated their respective CSU.

![Figure 1. Scenario in Packages Diagram](image)

Each CSU was developed following a quality oriented method, including quality standards, procedures, principles of good engineering management, and best practices of the Software Engineering [6], to assure the software quality as product, process, and service.

In order to keep acceptable reliability level, the Failure Modes and Effect Analysis – FMEA [7] metrics was used to:

- Evaluate additional needs;
- Estimate project defects;
- Define, a priori, safe configurations regarding the occurrence of failures in critical components or supplies; and
- Determine and set procedures priorities for contingency operations, when system enters on risk situations due to high probability of errors or unstructured actions.

The software safety is implemented based upon the HAZard and OPerability – HAZOP analysis technique, together with black and white boxes techniques allowing to find out which CSC has contributed for preventing each hazard [8]. This is achieved by means of formal specifications from the Safety Requirements Specification Analysis, following the Rational Unified Process – RUP [9]. In addition, automated tests are also performed using Integrated - Computer Aided Software Engineering - Environment – I-CASE-E tools, during the entire development process, assuring construction compliance with specifications and requirements.

At the end of the software prototype development, it was generated 77 KLOCs of C++. For each modeled Use Case it was also build at least one Use Case Test.

**Mission of the Software Prototype**

In order to provide a VANT-EC-SAT CSS prototype assessment, as far as quality, reliability, and safety goes, a fictitious search and rescue mission was built as a test scenario.

A scenario fragment is shown in Figure 2. The search and rescue mission starts with the GCS asking for the SSAT to image a certain area, at a known latitude and longitude. The SSAT acquires the low resolution image of the selected area and sends it to the GCS. After appropriate analysis of...
the received image, the GCS assigns a first UAV to perform an aerial reconnaissance mission. The UAV receives the GCS assignment and flies right to the latitude and longitude assigned point, at a pre-determined altitude, as its first mission. Then the UAV acquires the assigned point image and transmits it to the GCS. The GCS operators analyze the image, assigning then a second UAV mission, this time for a rescue mission. After that, a request is sent to SSAT to image again the target area. Finally, a third UAV is assigned to confirm successful target rescue and mission accomplishment.

Figure 2. Use Cases Diagram of a Scenario Fragment

Tools

During the software development it was used an I-CASE-E [10], and a Model Driven Architecture – MDA [11], allowing software construction supported by formal methods embedded on tools, providing substantial improvements on the quality of the VANT-EC-SAT prototype.

Most of the used tools, shown on Table1, come from the IBM-Rational Suite, like the Rational Test RealTime – RT-RT [12] and the Rational Quality Architect RealTime– RQA-RT [13, 14]. These tools allow the software testing, verification and validation.

Table 1. Tools

<table>
<thead>
<tr>
<th>Software</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Rational Rose RealTime, 2003.06.00</td>
<td>RealTime System Development based upon the Unified Modeling Language allows source code generation and runs the model.</td>
</tr>
<tr>
<td>IBM Rational RequisitePro, 2003.06.15</td>
<td>Allows performing software requirements management.</td>
</tr>
<tr>
<td>IBM Rational ClearCase, 2003</td>
<td>Allows version control management.</td>
</tr>
<tr>
<td>IBM Rational Test RealTime, 2003.06.00</td>
<td>Cross-platform solution for component testing and runtime analysis.</td>
</tr>
</tbody>
</table>

Modeling and Implementation

The development process of the VANT-EC-SAT CSS prototype was based upon the IBM-Rational Suite and RUP, tailored to attend the specific needs of small projects. This prototype has used 12 RUP artifact documents.

Each RUP phase had at least 2 iterations, allowing the project incremental elaboration. These figures were previously established so that students could synchronize with their academic schedules.

On the Inception phase, functional and non-functional requirements were surveyed and documented in artifacts, allowing the initial Use Cases elaboration. On the second iteration of this phase the CSU artifacts belonging to a CSC were integrated, increasing the development complexity.

On the Elaboration phase, the Rational Rose RealTime – RRRT was used, and each CSU was modeled and complemented, in order to allow CSC constructions. Then, from Structure and State Diagrams, structure and behavior implementations were generated.
On the Construction phase different CSCs were integrated into CSCIs. Their Structure and State Diagrams were aggregated in order to facilitate components’ understanding and visualizations. At this point, classes, protocols, and associated diagrams were organized into packages, as shown in Figure 3.

![Figure 3. Model Organized in Packages](image)

Nevertheless, this is not always possible, considering the needs of using test activities to minimize or extinguish persistent errors.

Tests were performed in four levels:

- White box CSU testing;
- Black box CSUs testing;
- Black box CSCs testing; and
- Black box CSCIs testing.

**Testing strategy**

From the 12 RUP artifacts, only the Test Plan and the Use Cases Test are concerned with the test discipline. The elaboration of these artifacts started on the first Inception phase iteration, considering the prototype quality.

On the Inception phase, the first requirements were identified together with a software testing strategy in order to assure functionality mapping onto one or more use cases to meet median and long range software quality, reliability, and safety.

Among other information included on test artifacts, four requirements were fulfilled to cover test criteria: 1) test type objectives; 2) techniques; 3) exit criteria; and 4) special considerations.

Following the RUP early testing philosophy, it was possible to identify soon in the development, which points would need more attention.

The following tests were applied to all RUP phases:

- Inception phase – Unit testing on isolated CSUs to identify possible defects on the model. Drivers and stubs were created for testing capsules. These tests were based in collaboration and sequence diagrams applied to the RQA-RT tool;

- Elaboration phase – Tests on the CSCs were performed. For this phase, it was not necessary to create drivers and stubs, because existing diagrams were used. On this phase, collaboration and sequence diagrams were also created to simulate message exchanging among CSCs. Only RQA-RT functionalities were used;
• Construction phase – Tests on the CSCIs, the second integration level have grouped similar CSCs into three CSCIs: UAV, GCS, and SSAT. After this point, the RQA-RT and RT-RT functionalities were used. Tests were performed for code and model coverage, and also static metrics. Tests were based upon Collaboration and Sequence Diagrams, and also on the C++ source code semi-automatically generated by the RRRT; and

• Transition phase – Tests of the CSS, the third integration level, were performed to assure the absence of its defects, considering the integration of three CSCIs.

The quality verification of both model and generated source code were performed on the RT-RT and RQA-RT tools.

Testing Techniques

The RT-RT had used the object oriented test technique to generate the automated tests applied to the VANT-EC-SAT. Black Box Method, Functional Test, with Equivalence Partitioning Class and Boundary Value Analysis were applied. White Box Method, Structural Test, Coverage Tests based upon Requirements, and Coverage Tests based upon source code were also applied. The last one was considered important for safety-critical systems [15].

On the Inception phase, the White Box Method was used to test CSUs to analyze states activation, and to validate the generated code from the model. On the next three phases, after some iterations and integrations, the Black Box Method was used.

Running Tests

Model and code coverage were also performed using the RT-RT, during the CSS development. These tests together allowed code identification pieces (functions and statements blocks), and model pieces (states, transitions and statement blocks) started during the software runtime, as shown in Figure 4.

![Collaboration Diagram creation, to interconnect capsules involved in the test, and also by a](image)

Figure 4. Tests execution scheme for Model and Code Coverage

The execution of Test Harness for each CSU, supported by the RQA-RT, was preceded by a
Sequence Diagram creation, providing message exchanging throughout the lifeline of each capsule, in order to assure compliance with specified requirements, as illustrated on Figure 5.

Figure 5. Sequence Diagram for testing a capsule that monitors the UAV data bus

For each integration level, a new Test Harness was built to verify intermediate incrementally generated software prototypes without losing identity and functionality from previous phases.

Management and Control Tests

IBM-Rational RequisitePro and IBM-Rational ClearCase were used to manage requirements and control artifacts versions.

For automated tests the RT-RT and RQA-RT tools were used. For the RT-RT it was necessary to manually create a new project for version control each time new tests were created, while for the RQA-RT this kind of control was automatically performed by the tool using timestamps.

Quality and Software Metrics Analysis

For the static metric applications over the source code (".h" and ".cpp") generated by the RRRT, it was used RT-RT tool.

Assessment Metrics were divided into four groups: Halstead Complexity Metrics; Lines & Comments; Statements & Levels; and the Cyclomatic Index – \( V(G) \), as follows.

Halstead complexity measurement was developed to measure program modules complexity directly from source code, with emphasis on computational complexity. This measurement was developed based upon the late Maurice Halstead as a means of determining quantitative measures of complexity directly from operators and operands in the module.

The use of Halstead metrics has provided various modules complexity indicators allowing evaluating the testing time of any C/C++ source code. Nevertheless, they only make sense at the source file level and they vary with parameters and formulas, shown on Figures 6 and 7 [16].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_1 )</td>
<td>Number of distinct operators</td>
<td></td>
</tr>
<tr>
<td>( n_2 )</td>
<td>Number of distinct operands</td>
<td></td>
</tr>
<tr>
<td>( N_1 )</td>
<td>Number of operators instances</td>
<td></td>
</tr>
<tr>
<td>( N_2 )</td>
<td>Number of operands instances</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Halstead Metric Parameters

<table>
<thead>
<tr>
<th>Metric</th>
<th>Meaning</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>Vocabulary</td>
<td>( n_1 + n_2 )</td>
</tr>
<tr>
<td>( N )</td>
<td>Size</td>
<td>( N_1 + N_2 )</td>
</tr>
<tr>
<td>( V )</td>
<td>Volume</td>
<td>( N^* \log_2 n )</td>
</tr>
<tr>
<td>( D )</td>
<td>Difficulty</td>
<td>( n_1/2 * N_1/n_2 )</td>
</tr>
<tr>
<td>( E )</td>
<td>Effort</td>
<td>( V * D )</td>
</tr>
<tr>
<td>( B )</td>
<td>Errors</td>
<td>( V / 3000 )</td>
</tr>
<tr>
<td>( T )</td>
<td>Testing time</td>
<td>( E / k )</td>
</tr>
</tbody>
</table>

Figure 7. Halstead Metric Formulas

The use of Lines & Comments Metric was related to the File Level Metrics, as follows:

- Comment-only lines (the number of comment lines that do not contain any source code);
- Comments (the total number of comment lines);
- Empty lines (the number of lines with no content);
• Source-only lines (the number of lines of code that do not contain any comments);
• Source and Comments Lines (the number of lines containing both source code and comments);
• Lines (the number of lines in the source file); and
• Comment rate (percentage of comment lines against the total number of lines).

File, Class or Package, and Root Level Metrics were related with the Statement & Levels, according to the following label meanings:

• Maximum of Nested Level and Sum of Nested Level (shows the highest nesting level reached in the software component);
• Maximum statements (the maximum number of statements); and
• Total statements (total number of statement in child nodes).

The V(G), Cyclomatic Number or Index represents the complexity measure of a function which is correlated with difficulty in testing. It was introduced by Thomas McCabe in 1976 with the purpose to measure the number of linearly-independent paths through a program module. This measure provides a single ordinal number that can be compared to the complexity of other programs. Cyclomatic complexity is often referred to simply as program complexity, or as McCabe’s complexity. It is also used together with other software metrics. As one of the most widely-accepted software metrics, it is intended to be independent of language and language format [17].

Cyclomatic complexity is based entirely on the structure of a software control flow graph. Control flow graphs describe the logic structure of software modules. A module corresponds to a single function or subroutine in typical languages, it has a single entry and an exit point, and is able to be used as a design component via a call/return mechanism [18].

Their parameters are:

• Maximum V(G) (the highest encountered cyclomatic number);
• Mean V(G) (the average cyclomatic number);
• Standard deviation from V(G) (deviation from the average V(G)); and
• Sum of V(G) (total V(G) for the scope).

Table 2 shows the parameter value and its meaning in terms of risk.

<table>
<thead>
<tr>
<th>V(G)</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>easy program, low risk</td>
</tr>
<tr>
<td>11-20</td>
<td>complex program, tolerable risk</td>
</tr>
<tr>
<td>21-50</td>
<td>complex program, high risk</td>
</tr>
<tr>
<td>&gt;50</td>
<td>impossible to test, extremely high risk</td>
</tr>
</tbody>
</table>

For a practical case, one of the tests performed, was the source code analysis of a file named “VBAD_MI.cpp.” It is a CSU belonging to the CSCI UAV. This file implemented the data bus interruption monitoring. Figure 8 shows static metric application results on this file, given by the RT-RT tool.

According to Table 2, the referred file could be ranked as a complex program with high risk as using the tool, its Maximum V(G) was rated 21.
Reliability and Safety

Software Reliability is concerned with keeping the quality along with a period of time, which means that the software should not fail during the specified period of time. A fault means that the software will not meet a functional requirement.

The FMEA metric was applied to detect the existence of a high level of Severity and Probability of Failure, which could generate the need for changes and corrections within the project.

Software safety involves studies to assure that the software will execute its functions without unacceptable risks for the safety of the whole computer software system. It concentrates in the identification and evaluation of potential causalities that could exert a negative impact on the software.

Following the early testing philosophy, the quality was assured as a process and product, which means that tests were applied in all phases of the development.

To effectively test the functionalities of the CSS, it was designed a fictitious scenario based on a Search and Rescue Mission, which originated a simulated assignment for the system, as previously described on the “Software Prototype Mission” section. Accomplishment of this assignment has assured the quality of the system.

Conclusions

IBM-Rational I-CASE-E Suite together with RUP methodology has allowed the development of a real time embedded software prototype, with quality, reliability and safety assurance.

Additionally, if proper technologies comprising of standards, methods, techniques, and tools are correctly used, such guarantees are provided. In this work, software and methodology were applied, using automated test tools provided by the IBM-Rational Suite.

In special, the adoption of the RT-RT and RQA-RT test tools during all phases of the CSS construction has reduced developer’s effort, rework, and time, contributing for higher quality, reliability and safety. This increases the development team efficiency.

The main results can be verified on the software prototype developed by the 33 students, in 17 academic weeks, ending up with a medium-size software complexity, which has generated, tested, verified and validated 77 KLOCs.

For each modeled Use Case, at least one Use Case Test was generated with an average Test Harness of 8 KLOC each.

In spite of the particular focus on an aerospace Case Study, these test processes can be applied for a wide variety of software projects involving a large spectrum of other correlated domains.

Acknowledgements

The authors of this paper would like to thank the following institutions for their support to this work: the Brazilian Department of Defense (Ministério da Defesa - MD); the General-Command of Airspace Technology (Comando-Geral de Tecnologia Aeroespacial – CTA); the Aeronautics and Space Institute (Instituto de Aeronáutica e Espaço – IAE); the Aeronautical Institute of Technology (Instituto Tecnológico de Aeronáutica – ITA); the Brazilian National Council for Research and Development (Conselho Nacional de Pesquisa e Desenvolvimento – CNPq); the Casimiro Montenegro Filho Foundation (Fundação Casimiro Montenegro Filho – FCMF); the Brazilian Department of Science and Technology (Ministério da Ciência e Tecnologia – MCT); the Brazilian Airspace Agency (Agência Espacial Brasileira – AEB); and the Brazilian National Institute for Space Research (Instituto Nacional de Pesquisas Espaciais – INPE).

References


25th Digital Avionics Systems Conference
October 15, 2006

6C6-9