Assessment of Product Maintainability for Two Space Domain Simulators

Tiago L. Alves
University of Minho, Portugal
Software Improvement Group, Netherlands
Email: t.alves@sig.eu

Abstract—The software life-cycle of applications supporting space missions follows a rigorous process in order to ensure the application compliance with all the specified requirements. Ensuring the correct behavior of the application is critical since an error can lead, ultimately, to the loss of a complete space mission. However, it is not only important to ensure the correct behavior of the application but also to achieve good product quality since the applications need to be maintained for several years. Then, the question arises, is a rigorous process enough to guarantee good product maintainability?

In this paper we assess the software product maintainability of two simulators used to support space missions. The assessment is done using both a standardized analysis, using the SIG quality model for maintainability, and a customized copyright license analysis. The assessment results revealed several quality problems leading to three lessons. First, rigorous process requirements by themselves do not ensure product quality. Second, quality models can be used not only to pinpoint code problems but also to reveal team issues. Finally, tailored analyses, complementing quality models, are necessary for in-depth investigation of quality.

I. INTRODUCTION

A space mission running a satellite is a long-term project that can take a decade to prepare and that may run for several decades. Simulators play an important role in the overall mission. Before the spacecraft launch, simulators are used to design, develop and validate many spacecraft components; validate communications and control infrastructure; train operations procedures. After the launch, they are used to diagnose problems or validate new conditions (e.g. hardware failure, changes in communication systems).

During such a long period of time, inevitably, glitches in both hardware and software will appear. To minimize the impact of these problems it was clear that standards were necessary to achieve very high quality [1]. The first standards were defined in 1977 [1] being currently under administration of the European Cooperation for Space Standardization\(^1\) (ECSS). The ECSS is represented by the European Space Agency (ESA), the Eurospace organization (representing European space industry), and several European national space agencies.

The ECSS standards are enforced by ESA and applicable to all projects developed for the space domain, covering project management, product assurance and space engineering. Two standards are specific for software. The space engineering software standard [2] defines the requirements for the software life-cycle process. The software product assurance standard [3] defines quality requirements for software development and maintenance activities.

In the ECSS space engineering standard [2] a rigorous software process is defined. This includes clear project phases (e.g. requirements), activities which can determine if the project is continued (e.g. reviews), and deliverables to be produced (e.g. documents). For example, in the requirements phase the Requirement Baseline document must be produced. Only after the review and approval of this document, done by all the involved parties in the System Requirements Review phase, the project is allowed to continue.

In the ECSS software product assurance standard [3], the use of a quality model is considered. However, the standard does not define or provide recommendations for any specific quality model. This omission in the standard explains why, typically, no quality model is used, or software suppliers are given the freedom to choose or propose a model. As consequence, the product quality of space software, in practice, relies only on the strict process standards.

Then, the question arises: is the current rigorous process enough to guarantee good product quality? To answer this question, we analyzed the software product maintainability of two simulators used in the space domain, developed with similar standards: EuroSim and SimSat. The EuroSim is a commercially available simulator system developed by a consortium of companies. The SimSat is a simulator owned by ESA and developed by external companies selected via a bidding process.

Both EuroSim and SimSat were analyzed using the Software Improvement Group (SIG) quality model for software product maintainability [4], based on the ISO/IEC 9126 standard for software product quality [5]. Additionally, for EuroSim we performed a custom analysis of the copyright licenses to check for possible software distribution restrictions.

From the results of the analysis three lessons were learned.

i) Rigorous process requirements do not assure good product maintainability, supported by the fact that both the EuroSim and SimSat ranked less than four starts in the SIG quality model.

ii) Quality models can reveal team problems, supported by the discovery that some of the code issues pinpointed by the quality model could be attributed to specific teams involved in the project.

\(^1\)http://www.ecss.nl/
iii) Tailored analysis are necessary for further investigation of product quality, supported by the discovery of code structure problems using copyright license analysis.

We conclude that having a quality model is a fundamental element to achieve good quality allowing to pinpoint potential problems and monitor quality degradation. Also, quality models should be complemented with tailored analysis in order to check for further potential problems.

This paper is structured as following. Section II provides a description of the two analyzed simulators. Section III introduces the quality model and the copyright license analysis used to evaluate product quality. Section IV presents the results of the quality assessment and describes the quality issues found and how they could be prevented. Section V summarizes the lessons learned and Section VI concludes the paper.

II. SYSTEMS UNDER ANALYSIS

In this section we provide a brief overview of the two simulators used in our analysis: EuroSim (European Real Time Operations Simulator) and SimSat (Simulation Infrastructure for the Modeling of Satellites).

A. EuroSim

EuroSim is a commercial simulator developed and owned by a consortium of companies including Dutch Space, NLR and TASK24.

The development of EuroSim started in 1997. It is mainly developed in C/C++, supporting interfaces for several programming languages (e.g. Ada, Fortran, Java and MATLAB). EuroSim supports hard real-time simulation with the possibility of hardware-in-the-loop and/or man-in-the-loop additions.

EuroSim is used to support the design, development and verification of critical systems. These include, for instance, the verification of spacecraft on-board software, communications systems and other on-board instruments. Additionally, outside the space domain, EuroSim was used for combat aircraft training purposes and to simulate autonomous underwater vehicles.

For the analysis EuroSim mk4.1.5 was used.

B. SimSat

SimSat is a simulator owned by ESA, developed and maintained by different external companies chosen via bidding process. In contrast to EuroSim, SimSat is not a commercial tool and is freely available to any member of the European Community.

The development of SimSat started in 1998 but its code has been rewritten several times. The analyzed version is based on the codebase developed in 2003. SimSat consists of two main modules: the simulation kernel, developed in C/C++; and the graphical user interface (GUI), developed in Java using Eclipse RCP [6]. Only soft real-time simulation is supported.

SimSat is used for spacecraft operation simulations. This involves the simulation of the spacecraft state and control communication (housekeeping telemetry and control). The on-board instruments (payload) are not simulated. The simulator is used to train the spacecraft operator team and validate operations software, such as the systems to control satellites, ground station antennas and diverse network equipment.

For the analysis SimSat 4.0.1 issue 2 was used.

EuroSim and SimSat have three commonalities. First, they are used to support the validation of space sub-systems. Second, according to companies involved in the development of EuroSim [7] and SimSat [8], they are both used for the simulation of (different) components of the European Global Navigation System (Galileo). Third, both EuroSim and SimSat were developed using strict equivalent software process standards, compatible with the ECSS standards.

III. SOFTWARE ANALYSES

Two types of analyses were done. One standardized analysis, applied to both EuroSim and SimSat, using the SIG quality model for software product maintainability. One custom analysis, applied only to EuroSim, for copyright license detection.

A. SIG Quality Model for Software Product Maintainability

The ISO/IEC 9126 standard for software product quality [5] defines a model to characterize software product quality according to 6 main characteristics: functionality, reliability, maintainability, usability, efficiency and portability. To assess maintainability, SIG developed a layered model using statically derived source code metrics [4]. An overview of the SIG quality model and its relation to the ISO/IEC 9126 standard is shown in Figure 1.

The SIG quality model has been used for software analysis [4], benchmarking [10] and certification [11], and is a core instrument in the SIG consultancy services. Also, Luijten et al. [9] found empirical evidence that systems with higher technical quality have higher issue solving efficiency.

The model is layered, allowing to drill down from the maintainability level, to sub-characteristic level (as defined in the ISO/IEC 9126: analyzability, changeability, stability and testability), to individual product properties. Quality is assessed using a five star ranking: five stars is used for very good quality, four stars for good, three stars for moderate, two stars for low and one star for very low. The star ranking is derived from source code measurements using thresholds calibrated using a large benchmark of software systems [11].

To assess maintainability, we used a simplified version of the quality model using the below described product properties. For each product property we provide a short description.

**Volume**: measures overall system size in staff-months estimated using the Programming Languages Table of Software Productivity Research LLC [12]). The smaller the system volume, the smaller the maintenance team required avoiding communication overhead of big teams.
**Volume**: measures the relative amount of code that has an exact copy (clone) somewhere else in the system. The smaller the duplication the easier to do bug fixes and testing, since functionality is specified in a single location.

**Unit size**: measures the size of units (methods or functions) in source lines of code (LOC). The smaller the units the lower the complexity and the easier it is to understand and reuse.

**Unit complexity**: measures the McCabe cyclomatic complexity \([13]\) of units (methods or functions). The lower the complexity the easier to understand, test and modify.

**Unit interfacing**: measures the number of arguments of units (methods or functions). The smaller the unit interface the better encapsulation and therefore the smaller the impact of changes.

**Testing**: provides an indication of how testing is done taking into account the presence of unit and integration testing, usage of a test framework and the amount of test cases. The better the test quality the better the quality of the code.

As we can observe, the ratings for product properties are derived in different ways. **Volume** and **Duplication** are calculated at system level. **Unit size**, **Unit complexity** and **Unit interfacing** are metrics calculated at method or function level, and aggregated to system level using quality profiles. A quality profile characterizes a metric through the percentages of overall lines of code that fall into four categories: low risk, moderate risk, high risk and very-high risk. The methods are categorized in these categories using metric thresholds. The ratings are calculated using (a different set of) thresholds to ensure that five stars represents 5\% of the (best) systems, four, three and two stars represent each 30\% of the systems, and that one star represents the 5\% of the (worse) systems. Finally, the rating for **Test quality** is derived using the following criteria: five stars for unit and functional testing with high coverage; four stars for unit or function testing with high coverage; three stars for unit or functional testing with good or fair coverage; two stars for function or unit testing with poor coverage; one star for no tests.

For a more detailed explanation of the quality model, we defer the readers to [4], [14] and [15].

### B. Copyright License Detection

A customized analysis was developed to find and extract copyright licenses used in EuroSim. This was done to investigate if any of the used licenses poses legal restrictions in the distribution of EuroSim.

The copyright license analysis was executed in two steps. First, by implementing and running an automatic script to detect copyright and license statements. Second, by manually checking each license type using Table I.

The developed script was implemented by defining regular expressions in \texttt{grep}, to match keywords such as \texttt{license}, \texttt{copyright}, and \texttt{author}. Although the approach is simple, it allows a powerful and generic way of detecting copyright statements. This is necessary since there is not a standardized way to specify copyright information (this is mostly available as free form text in code comments).

This list with the copyright statements found was then manually processed to detect false positives (keywords recognized that do not reference actual licenses or authorship...
information). After validation, false positives were removed from the list. Table I was then used to help checking if a license found poses any risk to the project. For instance, finding an open-source copyleft license, such as the GNU GPL license, would not only mandate to distribute library changes (if any) but also mandate EuroSim to be available under the GNU GPL license. As consequence, this would legally allow the free or commercial distribution of EuroSim by third-parties. The use of open-source copyright licenses, or licenses from consortium companies does no pose any restriction in software distribution. However, if an external license is found then, it should be investigated what the license conditions are.

IV. ANALYSES RESULTS

In this section we describe the results of the application of the SIG quality model for software product maintainability to both EuroSim and SimSat, and from the copyright license detection analysis done for EuroSim.

A. Software Product Maintainability

Using the SIG quality model, described in Section III-A, EuroSim ranks three stars while SimSat ranks two stars. In the following sections, we will provide a more detailed analysis of the 6 product properties measured by the quality model: Volume, Duplication, Unit complexity, Unit size, Unit interfacing and Testing. Volume and Duplication are measured at system-level and are presented in a scale, from five stars to one star, read from left to right. All the other metrics, except testing, are measured at unit level and are presented in a pie-chart, in which very-high risk is depicted in black, high risk in gray, moderate risk in light gray, and low risk in white.

1) Volume: Figure 2 compares the volume of EuroSim and SimSat. EuroSim contains 275K LOC of C/C++ and 4K LOC of Java, representing an estimated rebuild value of 24 staff/year, ranking four stars. SimSat contains 122K LOC of C/C++ and 189K LOC of Java, representing an estimated rebuild value of 32 staff/year, ranking three stars.

For EuroSim, the Java part is responsible only for exposing API access to Java programs. Both GUI and core of the application are developed in C/C++.

For SimSat, the Java part is responsible for the GUI, while the C/C++ is responsible for the core of the application. It is interesting to observe that the SimSat GUI is larger than the core of the application.

Since both simulators rank three stars or higher for volume, this indicates that the maintenance effort is possible with a small team composed by a couple of elements, albeit the maintenance effort is smaller for EuroSim than for SimSat.

While for EuroSim no recommendations are necessary, for SimSat it is advisable to monitor the system growth and, if continuous growth is observed, to divide the system into two parts in order to reduce maintenance effort.

2) Duplication: Figure 3 compares the duplication of EuroSim and SimSat. EuroSim contains 7.1% of duplication, ranking three stars, while SimSat contains 10.4% of duplication, ranking two stars.

In both EuroSim and SimSat, duplication problems were found in several modules, showing that it is not a localized problem. Also, for both systems, duplicated files were found. For EuroSim, we uncovered several clones in the library code supporting three different operating systems. This fact surprised the EuroSim maintainers as they expected the code...
to be completely independent.

For SimSat, surprisingly, we found a large number of duplicates for the Java part which, in newly-developed code, indicates lack of reuse and abstraction.

As final observations, although EuroSim is a much older system than SimSat, and hence more exposed to code erosion, the overall duplication in EuroSim is smaller than SimSat. Also, for both systems, several clones found were due to the (different) implementations of the ESA Simulation Model Portability library (SMP).

As recommendation for both systems, code duplication should be monitored (specially for SimSat, since part of it was recently built).

3) Unit size: Figure 4 compares the unit size of EuroSim and SimSat using quality profiles. Both EuroSim and SimSat contain a large percentage of code in the high-risk category, 17% of the overall code, leading to a ranking of two stars.

Looking at the distribution of the risk categories for both EuroSim and SimSat, we can see that they have similar amounts of code in (very) high risk categories, indicating the presence of very-large (over 100 LOC) methods and functions.

In EuroSim, we were surprised to find a method with over 600 LOC, and a few over 300 LOC, most of them regarding the implementation of device drivers. On results validation, this was explained due to the manual optimization of code.

In SimSat, the biggest C/C++ function contains over 6000 LOC, however inspection of the function revealed that it is an initialization function, having only a single argument and a single logic condition. Several methods over 300 LOC were also found, however most of them are in the Java part.

As recommendation, it is important to monitor if the large methods are subject of frequent maintenance and, if so, refactor them. In case of SimSat, the large methods in the recently built Java part, indicate, again, lack of encapsulation and reuse.

4) Unit complexity: Figure 5 compares the unit complexity of EuroSim and SimSat using quality profiles. Both EuroSim and SimSat contain a similar (large) percentage of code in the high-risk category, 4% and 3%, respectively, which lead to a ranking of two stars.

Considering the percentage of code of the three highest risk categories (very-high, high and moderate risk), EuroSim contains 30% of unit complexity while SimSat contains 20%. For both systems, the highest McCabe value found is around 170 decisions in a single C/C++ function (for EuroSim) and Java method (for SimSat).

In EuroSim, methods with very-high complexity were spread in the system. Interestingly, from our measurements at module-level, the EuroSim maintainers observed that particular consortium members were responsible for modules with the worse complexity.

In SimSat, it is worth to note that, the majority of the methods with very-high complexity were localized in just a dozen of files.

As recommendation, it is important to monitor and check for high complex units, since they will not only require more maintenance effort, but also make the testing process harder.

5) Unit interfacing: Figure 6 compares the Unit interfacing of EuroSim and SimSat using quality profiles. EuroSim ranks two stars while SimSat ranks three stars.

While EuroSim contains 42% of the overall code in the moderate and (very) high risk categories, SimSat contains 13%. For both systems, the highest number of parameters is around 15. Also, for both systems, methods considered as very-high risk are found spread over the system.

In SimSat, surprisingly, no very-high risk methods were found in the Java code, only in the C/C++ code. This was surprising since for all other product properties we observed an abnormal quantity of problems in the Java code.

6) Testing: Both EuroSim and SimSat have similar test practices, both ranking two stars for testing due to the existence of only functional tests. Most of the testing is done manually, testing teams follow scenarios to check if the functionality is correctly implemented.

Automatic functional/component tests are available for both systems. However, none of the systems revealed unit tests, i.e. tests to check the behavior of specific functions or methods.

Test frameworks were only found for the SimSat system, but restricted to the Java code only, and without test coverage measuring. For the C/C++ code, for both systems, no test framework was used in the development of the tests.

For both systems test coverage information was not available. However, comparing the ratio between test code and production code, for both systems, we observed roughly 1 line of test code per 10 lines of production code, which typically indicates low test coverage.

Regarding to EuroSim, we observed that for different parts of the system, slightly different naming conventions were used, indicating that tests were developed in a non-structured way.

In summary, the testing practices could be improved for both systems. As recommendation, a clear separation between production and test code should be made and testing frameworks with coverage support should be adopted.

B. Copyright License Detection

Our analysis of the copyright licenses identified a total of 25 different licenses: 2 LGPL licenses, 7 licenses from consortium members, 11 licenses from hardware vendors and libraries, and 5 licenses copyrighting software to individuals.

None of the copyright licenses found poses restrictions on software distribution. However, the copyright license analysis revealed open-source library code mixed with code developed by the EuroSim consortium. While for some external libraries specific folders were used, this practice was not consistent, indicating code structure problems. This issue is particularly important when updating libraries, requiring to keep track of the locations of these libraries (or to manually determine them), requiring extra maintenance effort.

Additionally, for software developed by a consortium of companies, it was expected that the code would be under a unique license defined by the consortium. Instead, we verified that each consortium company used its own license. While this
is not a problem, extra effort is required in case any of the licenses needs to be updated.

Finally, it was surprising to find copyright belonging to individual developers in some files. Since this happened only for device drivers extensions code this also poses no serious problem.

The analysis, despite using a very simple technique, provided valuable information to the EuroSim team. Not only the team was not aware of the number and diversity of licenses in the source code, as they were surprised to discover copyright statements to individuals.

In summary, although our analysis did not reveal copyright license issues, it helped us to uncover code structure issues.

As recommendation, code from external libraries should be clearly separated from production code. Also, the use of an unique license for all the consortium should be considered.

V. LESSONS LEARNED
A. STRICT PROCESS REQUIREMENTS DO NOT ASSURE PRODUCT QUALITY

Both EuroSim and SimSat were developed using similar process requirements. Although the development is done following a rigorous process, the product quality analyses revealed moderate maintainability for EuroSim (three stars) and low maintainability for SimSat (two stars). For both systems, in-depth analysis revealed duplication, size and complexity problems causing several product properties to rank two stars.

To assure good software product quality it is necessary to define clear criteria to assess quality, establish quality targets and provide means to check if the targets are being met (preferably using tool-based analyses). This can be achieved using a tool-supported quality model.

The quality model defines criteria, i.e., defines the metrics/analyses that are going to be applied to the software, and check if the results are within boundaries. When using the SIG quality model, an example of a quality target is that the overall maintainability should rank a minimum of four stars. Finally, to validate that the quality target is being met it is necessary to continuously apply the quality model to monitor the evolution of the system quality.

The continuous application of a quality model during the software life-cycle offers many advantages: it allows, at any moment, to check if the quality targets are being met; it can provide early-warning when a potential quality problem is introduced - pinpointing the problem; and it allows the monitoring of code quality degradation.

B. QUALITY MODELS CAN REVEAL TEAM PROBLEMS

When assessing software product maintainability using a quality model, potential team problems can be revealed by the observation of rapid software degradation or by the unusual lack of quality in newly-built software. Team problems were revealed for both EuroSim and SimSat.

EuroSim was developed by several teams (from the consortium of companies) which, through time, contributed with different parts of the system. When analyzing software quality we observed quality differences that could be attributed to specific teams, i.e., some teams delivered code with worse quality than others. Homogeneous code quality between the consortium members can be achieved by using a shared quality model and establishment of common quality targets.

SimSat was developed and maintained by external companies selected via a bidding process. In the last iteration of the product, a new technology, Eclipse RCP [6], was used to develop the GUI. When analyzing software quality for the GUI we observed unusual duplication, size and complexity for the newly developed part, responsible for the system low maintainability rating. Our hypothesis, was that these quality problems were attributed to the low expertise of the contracted company with this new technology, which was confirmed by ESA. A shared quality model between ESA and the external company responsible for software development would allow to reveal the before mentioned quality issues during development. Also, by including in the outsourcing contract the quality model to be used and the quality targets to be met, would create legal obligations for the external company to deliver high quality software.

In summary, the adoption of a quality model and establishing common quality targets allows to create independence between code quality and the development team. This is particular important in environments where team composition change over time.

C. TAILORED ANALYSES ARE NECESSARY FOR FURTHER INVESTIGATION OF PRODUCT QUALITY

Quality models evaluate a fixed set of the software characteristics hence only revealing a limited set of potential problems. If under the suspicion of a particular problem, quality models should be complemented with tailored analyses to check for the existence of that problem. Additionally, even if evidence of that particular problem is not found, the analyses can reveal other problems.

For EuroSim, the customized analysis to check for copyright licenses revealed the existence of open-source code mixed with production code without clear separation, providing evidence of problems in code structure. Some suspicion of this problem already existed, when we observed, during test quality analysis, that test code was not available from folders named in a consistent way.

This is an example of a problem that can not be detected using a quality model. Quality models are important, because they can automatically produce an overview of the system quality and provide basis for quality comparison between systems. However, as we learned from this example, it is important to complement the quality assessment of a quality model with tailored analysis and expert opinion.

VI. CONCLUSION

This paper presented a quality assessment of two simulators used in the space domain: EuroSim and SimSat. To analyze both systems the SIG quality model for software product maintainability based on ISO/IEC 9126 standard for software product quality was used. Although both systems followed
similar strict standards, quality problems were found in both systems. Not only do both systems rank less than four stars, also problems in duplication, size and complexity were found.

From the analyses of both the EuroSim and SimSat three lessons were learned.

i) Rigorous process requirements do not assure product quality: To achieve good code quality it is necessary to measure it directly. Hence, in addition to rigorous process requirements it is essential to use a quality model to ensure product quality.

ii) Quality models can reveal team problems: The use of quality models is not restricted to rank quality or identify code issues. They can also be used as a tool to gain knowledge about the software project, helping in the diagnosis of overall problems in development and maintenance activities.

iii) Tailored analyses are necessary for further investigation of quality: While in general quality models provide a good overview of the system quality they are not suitable for pinpointing all quality issues. In addition to quality model, tool-based tailored analyses can help unraveling system knowledge helping to diagnose further problems.

The analyses reported in this paper provide evidence that, although a quality model does not reveal all problems, it can be an important instrument to manage software quality and to steer the software development and maintenance activities.

CHALLENGES

Having a quality model is an important step to achieve better software quality, however there are many challenges to be met, both by industry and academia.

To industry, the challenge is the adoption of already existent quality models and make them part of the software development/maintenance process. Also, we challenge industries in helping research by providing data to build software quality benchmarks in order to allow more general understanding of software quality.

To academia, the challenge is to propose even better quality models. New analyses and metrics are required to further reveal issues in software. Catalogs of analysis and possible diagnoses are needed to guide industries on how to solve problems. Finally, cost-models for maintenance activities, to estimate the cost of fixing software issues, would be valuable.

ACKNOWLEDGMENTS

Thanks to Joost Visser and other colleagues of the SIG, to Coen Claassens, Leon Bremer and José Fernandez-Alcon of Dutch Space, and to Mauro Pecchioli, James Eggleston and Eduardo Gomez of the European Space Agency (ESA) for providing comments to earlier versions of this work. The author is supported by the Fundação para a Ciência e a Tecnologia, grant SFRH/BD/30215/2006.

REFERENCES