Tool-based Analysis of Space Domain Simulators

Tiago L. Alves[1,2]

[1] *University of Minho*
Departamento de Informática,
Campus de Gualtar
4710-057, Braga Portugal

[2] *Software Improvement Group*
Rembrandt Tower, 14th floor
Amstelplein 1
1096HA Amsterdam, The Netherlands

Email: t.alves@sig.eu

INTRODUCTION

Space domain simulators are fundamental pieces in a software project. As such, they are developed and maintained following strict software processes. However, is software process enough to guarantee good product quality?

In this paper we will report on the usage of the Software Analysis Toolkit (SAT), developed by the Software Improvement Group (SIG), to analyze product maintainability and support management decisions for two simulators used in the space domain: SimSat and EuroSim.

SimSat is a soft real-time simulator, owned by ESA and developed by different companies chosen via a bidding process. SimSat is used for the simulation of the spacecraft state and control communication (housekeeping telemetry and control). EuroSim is a hard real-time simulator, developed by a consortium of companies. EuroSim is used to support the design, development and verification of critical systems. Both simulators have been developed with equivalent software processes compatible with the ECSS standards.

The SAT is a tool that automatically analyzes software maintainability. Software maintainability is measured based on the ISO/IEC 9126 standard for software product quality. The SAT is used in the SIG consultancy services to support management decisions. SIG has experience analyzing software systems in the financial, insurance, government, logistical and IT domain.

The application of the SAT to both simulators identified several maintainability problems, indicating that a strict software process is not enough to guarantee good product quality. Additionally, these problems revealed lack of expertise of some of the team involved in the projects. This information can be used to support management decision, e.g., to provide training to the teams or not to use the teams.

This paper is structured as follows: the methodology for the analysis of product maintainability is presented; a description of the SIG Software Monitor tool is provided; the analysis of product maintainability of both SimSat and EuroSim is presented; the use of the SAT to support management decisions; finally, the paper is concluded.

The quality comparison present in this paper was previously published in [1].

METHODOLOGY

The SAT measures maintainability using quality model rating maintainability using a five stars scale: five stars are used for very good quality and one star for very low quality. The maintainability model is layered and can be decomposed into sub-characteristics, as defined in the ISO/IEC 9126 standard, and then into product properties defined by SIG [2]. The product properties are volume, duplication, unit complexity, unit size, unit interfacing, and testing. Figure 1 shows an overview of the quality model[1].

Volume: measures overall system size in staff-months (estimated using the Programming Languages Table of Software Productivity Research LLC). The smaller the system volume, the smaller the maintenance team required avoiding communication overhead of big teams.

[1] This figure was adapted from Luijten et al. [3] based on the original quality model introduced in Heitlager et al. [2].

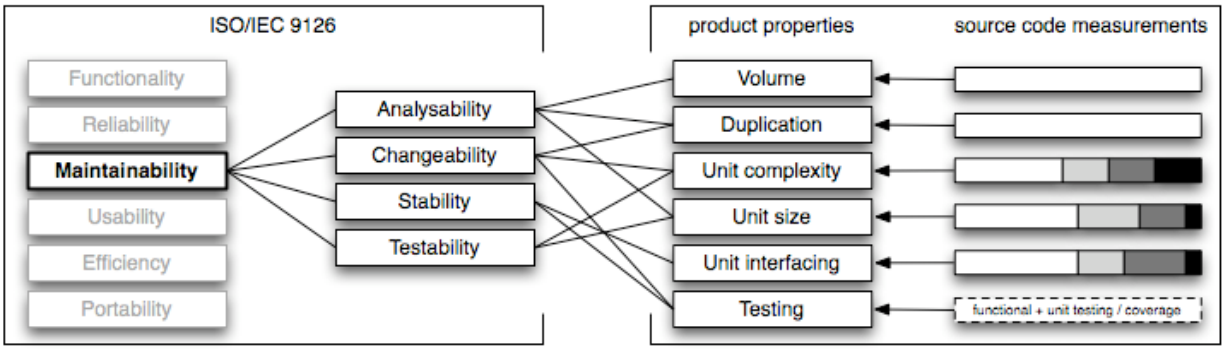

Figure 1: Quality model overview. On the left-hand side, the quality characteristics and the maintainability sub-characteristics of the ISO/IEC 9126 standard for software product quality are shown. On the right-hand side, the product properties defined by SIG and its relation with the maintainability sub-characteristics are shown.

Duplication: measures the relative amount of code that has an exact copy (clone) somewhere else in the system. The smaller the duplication, the easier to fix bugs and do 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 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.

These product properties are automatically measured from the source code. For each product property a rating is derived by comparing measurements to thresholds, which have been calibrated with a large number of systems [4,5,6]. The calibration process ensures that five stars represents 5% of all systems (the highest quality), four, three and two stars represent each 30% of the systems, and one star represents the remaining 5% of the systems (the lowest quality).

SIG SOFTWARE ANALYSIS TOOLKIT

The SIG Software Analysis Toolkit (SAT) consists of two components: the Analysis tooling and the Software Monitor. The analysis tooling is responsible for analyzing the source code and storing the results in a database. The Software Monitor is a web application responsible for displaying the results in a user-friendly manner.

To use the Analysis tooling, the development of custom configurations is required. These configurations specify individual project needs, such as technologies to analyze (e.g. C/C++, Java, C#, XML), how to distinguish production and test code, and custom filters to ignore generated code, example code and third-party libraries.

The Software Monitor application displays the key indicators for software quality in a dashboard. Figure 2 shows a screenshot of the metrics table for the SimSat system. The key indicators are presented as charts or in metric tables. The Software Monitor allows drill-down from system-level measurements to the deepest level (e.g. method in Java). Duplicated parts of the code, clones, are shown individually reporting beginning and end lines for each clone. Finally, when multiple versions of the source code are available the Software Monitor also reports trends for all the metrics. The Software Monitor application is both available to the SIG consultants when analyzing software, as to the software owners.

SIMSAT AND EUROSIM QUALITY COMPARISON

The SAT maintainability analysis revealed an overall rating of three stars for EuroSim and two stars for SimSat. In-depth analysis of both systems revealed problems in most of the product properties. Figure 11 shows an overview of the product properties comparison between SimSat and EuroSim. Below, a more detailed analysis of each product property is presented.

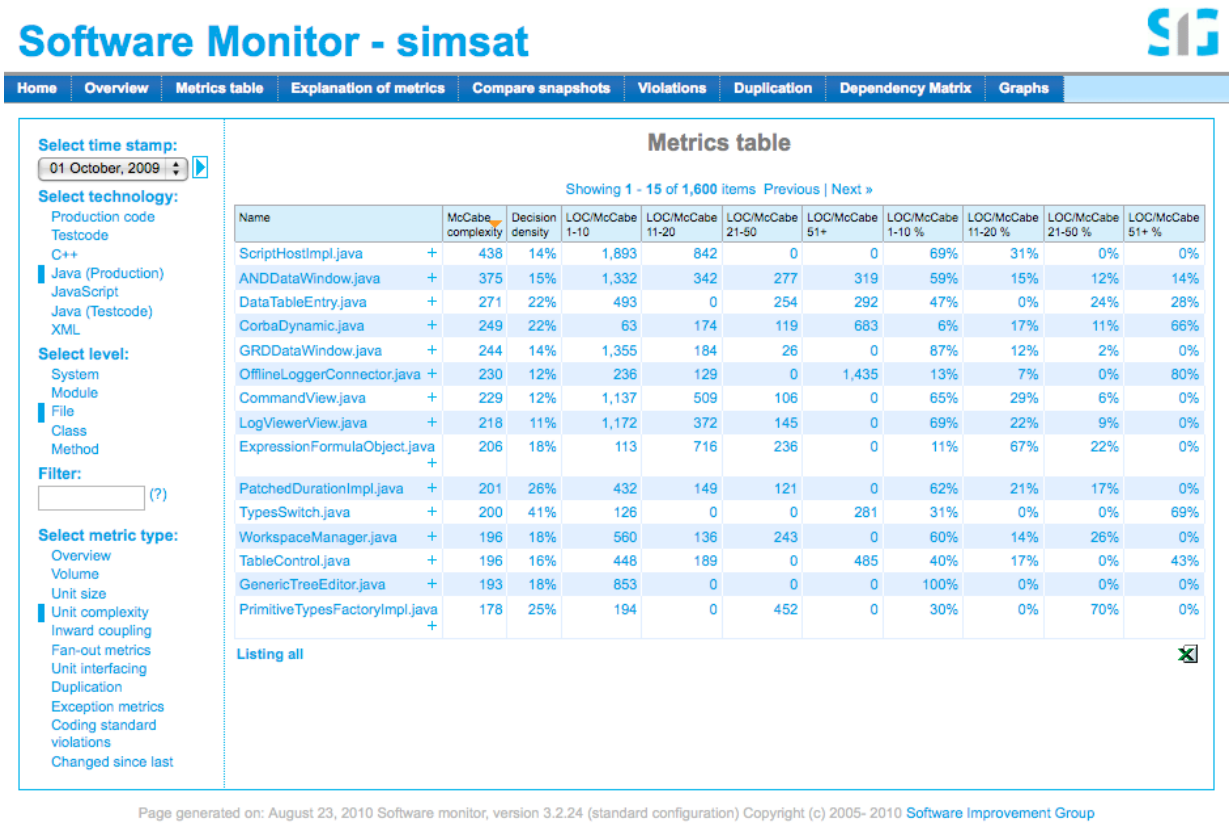

Figure 2: Software Monitor screenshot for the SimSat system showing most complex files.

Volume

Figure 3 compares the volume of SimSat and EuroSim. Each section of the scale corresponds to a rating, from five stars (left-most section), to one star (right-most section). SimSat ranks 3 stars, representing an estimated rebuild value of 32 staff/year (calculated from 122K LOC of C/C++ and 189K LOC of Java). EuroSim ranks 4 stars, representing an estimated rebuild value of 24 staff/year (calculated from 275K LOC of C/C++ and 4K LOC of Java).

Since both simulators rank three stars or higher for volume, this indicates that maintenance is possible with a small team (two or three people), though the maintenance effort would be higher for SimSat than for EuroSim. For SimSat, it is advisable to monitor the system growth and, if continuous growth is observed, divide the system into different parts in order to reduce maintenance effort (e.g. separate the SimSat Kernel from the MMI, treating them as two separate projects). For EuroSim no changes are recommended.

Duplication

Figure 4 compares the duplication of SimSat and EuroSim. Each section of the scale corresponds to a rating, from five to on star (from left to right). SimSat ranks two stars, containing 10.4% of overall duplication. EuroSim ranks three stars, containing 7.1% of duplication.

In both SimSat and EuroSim duplication problems were found in several modules, showing that it is not a localized problem. Also, for both systems, files that were duplicated in their entirely were found. For SimSat, surprisingly, we found a large number of duplicates for the newly-developed Java part, indicating lack of reuse and abstraction. 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 different.

As final observations, although SimSat has a more recent codebase than EuroSim, and was hence less exposed to code erosion, the overall duplication is higher in SimSat than in EuroSim. 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 and its growth prevented (specially for SimSat, since part of it was recently built).

Unit size

Figure 5 and Figure 6 show, side-by-side, the unit size distribution using risk profiles of SimSat and EuroSim, respectively. Both SimSat and EuroSim 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 SimSat and EuroSim, we can see that they have similar amounts of code in the (very) high risk categories, indicating the presence of very-large (over 100 LOC) methods and functions.

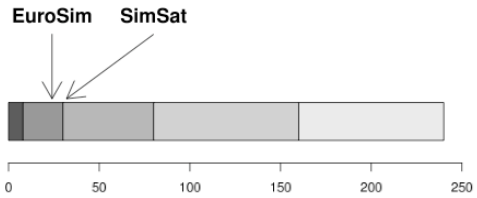

Figure 3: Volume comparison for SimSat and EuroSim.

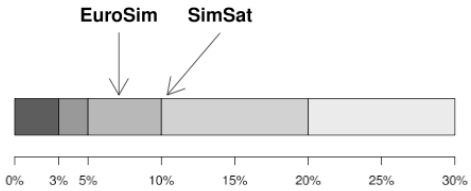

Figure 4: Duplication comparison for SimSat and EuroSim.

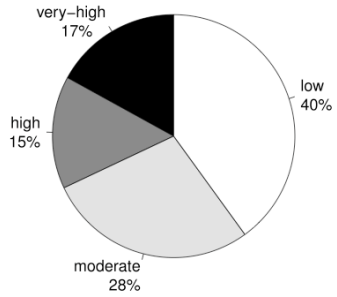

Figure 5: Unit size distribution for SimSat.

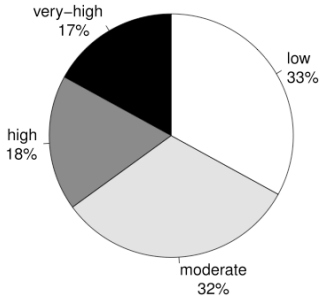

Figure 6: Unit size distribution for EuroSim.

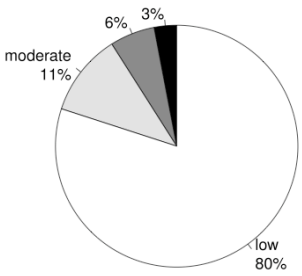

Figure 7: Unit complexity distribution for SimSat.

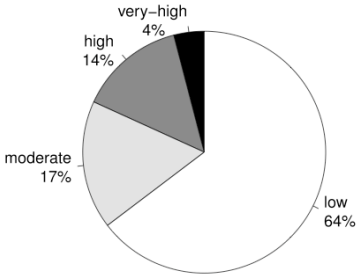

Figure 8: Unit complexity distribution for EuroSim.

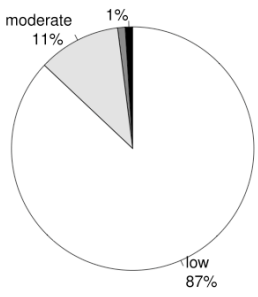

Figure 9: Unit interfacing distribution for SimSat.

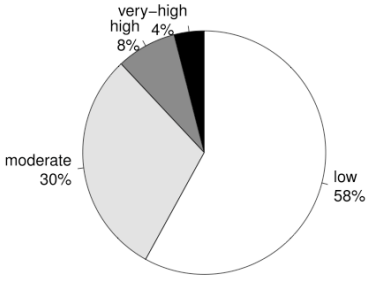

Figure 10: Unit interfacing distribution for EuroSim.

Figure 11: Product property comparison between SimSat and EuroSim. The bar charts are used to depict a scale from five stars (left) to one star (right). The pie charts depict risk profiles. A risk profile is a technique, developed by SIG, to separate measurements into four risk categories: very-high risk (depicted in black), high risk (depicted in dark gray), moderate risk (depicted in light gray) and low risk (depicted in white). The risk areas are defined using calibrated thresholds per product property.

In SimSat, the biggest C/C++ function contains over 6,000 LOC, however inspection of the function revealed that it is an initialization function, having only a single argument and a single logic condition. 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. When validating this finding with EuroSim maintainers, this was explained due to the manual optimization of code. 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.

Unit complexity

Figure 7 and Figure 8 show, side-to-side, the unit complexity distribution using risk profiles of SimSat and EuroSim, respectively. Both SimSat and EuroSim contain a similar (large) percentage of code in the very-high risk category, 3% and 4%, 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), SimSat contains 20% of unit complexity, while EuroSim contains 30%. For both systems, the highest McCabe value found is around 170 decisions in a single Java method (for SimSat) and for a single C/C++ function (for EuroSim).

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. In EuroSim, on the other hand, functions with very-high complexity were spread out throughout 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.

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

Unit interfacing

Figure 9 and Figure 10 show, side-by-side, the unit interfacing distribution using risk profiles of SimSat and EuroSim. SimSat ranks three stars since it contains 13% of code in the moderate and the (very) high risk categories. EuroSim, ranks two stars since 42% of the overall code is in the moderate and the (very) high risk categories. For both systems, the highest number of parameters in a function is around 15.

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. For EuroSim, no issues worth to note were found.

For this product property no recommendations are necessary. However, refactoring the code in the very-high risk category would improve the overall quality.

Test quality

Both SimSat and EuroSim 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, in which 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 contained unit tests, i.e. tests to check the behavior of specific functions or methods.

The use of test frameworks was 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.

A heuristic to determine test coverage is to compare the ratio between test code and production code. Smaller ratios, typically, mean low test coverage. For both systems, we observed roughly 1 line of test code per 10 lines of production code, which is a very low ratio, consequently, indicating low test coverage.

As recommendation, the introduction of a test framework with coverage support should be adopted. Also, the gradual introduction of unit testing for the newly developed code or maintained code is recommended.

SAT TO SUPPORT MANAGEMENT DECISIONS

The SAT can be an important instrument to support management decisions. In SimSat, the unusual duplication, size and complexity found for recently developed code, helped to uncover that the team involved with that part of the project lack familiarization with the technology used. In EuroSim, the complexity analysis helped to uncover that specific teams were responsible for the modules with lower quality. In both cases, a decision could be made not to use these teams, or to provide the teams with specific training for producing higher quality code.

In the two examples above, the analysis of the system was done after the systems were developed. Hence these findings can only help future management decisions. However, the application of SAT from the beginning of the process could help the early detection of code problems. Such problems, like team lack of expertise, could be handled in a much earlier stage avoiding later extra costs.

CONCLUSION

The SAT provides a pragmatic and practically feasible approach to measure product quality, which can be used to support management decisions. The quality model implemented in SAT uses a small set of key metrics whose measurements are aggregated in a five-star ranking. By design, the metrics are operational, understandable and generic. The star rating enables comparison and easy communication between all the stakeholders of a project. The analyses of EuroSim and SimSat revealed that although both systems were developed with rigorous software processes, implementing strict standards, quality problems were found. Hence, it is important to have mechanisms to identify and quantify quality problems during both development and maintenance activities. These problems, if identified early in the process can be used to back up management decisions.

Acknowledgements

The author is supported by the *Fundação para a Ciência e a Tecnologia*, grant SFRH/BD/30215/2006.

REFERENCES

[1] T. Alves, "Assessment of Product Maintainability for Two Space Domain Simulators", Proc. of the 26th IEEE International Conference on Software Maintenance (ICSM 2010), to appear, IEEE Computer Society Press, 2010.

[2] I. Heitlager, T. Kuipers, and J. Visser, "A Practical Model for Measuring Maintainability", Proc. of the 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007), pages 30-39, IEEE Computer Society Press, 2007.

[3] B. Luijten, J. Visser, "Faster Defect Resolution with Higher Technical Quality of Software", Proc. of the 4th International Workshop on Software Quality and Maintainability (SQM 2010), IEEE Computer Society Press, 2010.

[4] T. Alves, C. Ypma, J. Visser, "Deriving Metric Thresholds from Benchmark Data", Proc. of the 26th IEEE International Conference on Software Maintenance (ICSM 2010), to appear, IEEE Computer Society, 2010.

[5] J. P. Correia, J. Visser, "Benchmarking Technical Quality of Software Products", Proc. of the 15th Working Conference on Reverse Engineering (WCRE 2008), pages 297-300, IEEE Computer Society Press, 2008.

[6] J. P. Correia, J. Visser, "Certification of Technical Quality of Software", Proc. of the 2nd IEEE International Workshop on Foundations and Techniques for Open Source Software Certification (OpenCert 2008), stellite event of the 4th International Conference on Open Source Systems (OSS 2008), Research Report 398, pages 35-51, United Nations University – International Institute for Software Technology (UNU-IIST), 2008.