Applicability of Mutation Testing Method for Flight Software: Fault-based, Automated Quality Assurance Assessment for Space Software (FAQAS)

ENRICO VIGANÒ, OSCAR CORNEJO, FABRIZIO PASTORE, University of Luxembourg, Luxembourg

CCS Concepts: • **Software and its engineering → Software verification and validation**.

Additional Key Words and Phrases: Mutation analysis, CPS, CPS Interoperability, Integration testing

1 INTRODUCTION

From spacecrafts to ground stations, software has a prominent role in space systems; for this reason, the success of space missions depends on the quality of the system hardware as much on the dependability of its software. Mission failures due to insufficient software sanity checks [3] are unfortunate examples, pointing to the necessity for systematic and predictable quality assurance procedures in space software.

Since one of the primary objectives of software testing is to identify the presence of software faults, an effective way to assess the quality of a test suite consists of artificially injecting faults in the software under test and verifying the extent to which the test suite can detect them. This approach is known as *mutation analysis* [2]. In mutation analysis, faults are automatically injected in the program through automated procedures referred to as mutation operators. Mutation operators enable the generation of faulty software versions that are referred to as *mutants*. Mutation analysis helps evaluate the effectiveness of a test suite, for a specific software system, based on its mutation score, which is the percentage of mutants leading to test failures. Also, mutation analysis enables *mutation testing*, which concerns the automated generation of test cases that discover mutants.

Despite its potential, mutation analysis is not widely adopted by industry. The main reasons include its limited scalability and the pertinence of the mutation score as an adequacy criterion [5]. Indeed, for a large software system, the number of generated mutants might prevent the execution of the test suite against all the mutated versions. Also, the generated mutants might be either semantically equivalent to the original software [4] or redundant with each other [6]. Equivalent and redundant mutants may bias the mutation score as an adequacy criterion. Finally, test generation approaches are preliminary and cannot be applied in industrial space context. For example, they can generate test inputs only for batch programs that can be compiled with the LLVM infrastructure [1].

The FAQAS activity addresses the problems above. It is a joint work between the SnT Centre of the University of Luxembourg[1], Gomspace Luxembourg[2] (GSL) and OHB Luxspace[3] (LXS). FAQAS led to the development of a toolset that addresses the challenges above. It includes four tools: *MASS* (Mutation Analysis for Space Software), *DAMAt* (DAta-driven Mutation Analysis with Tables), *SEMuS* (Symbolic Execution-based MUtant analysis for Space software), and *DAMTE* (DAta-driven Mutation TEsting).

Figure 1 provides an overview of the input and outputs of the FAQAS toolset. It relies on the idea of generating multiple modified versions of the software system under test (SUT), some are derived by modifying the implementation of the software (code-driven mutants) other by integrating a

[1]https://wwwen.uni.lu/snt
[2]https://gomspace.com/
[3]https://luxspace.lu/

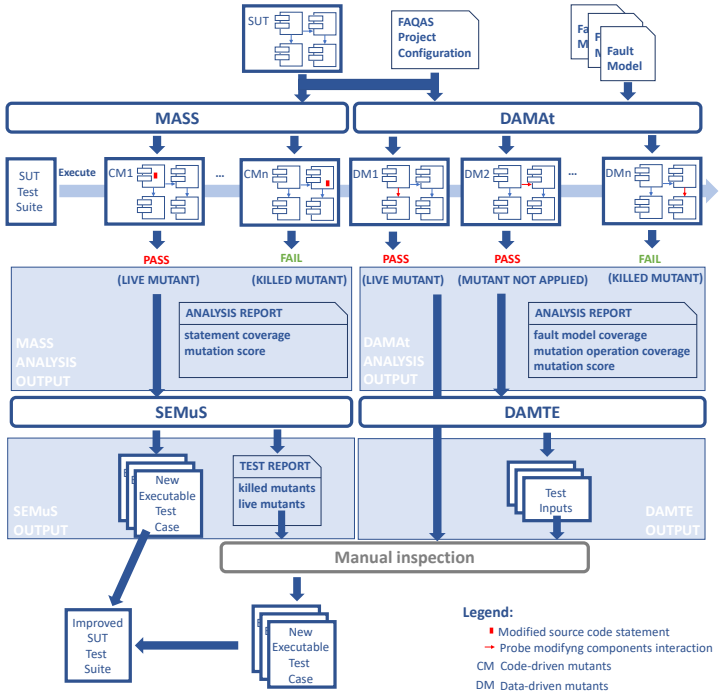

Fig. 1. Overview of the FAQAS toolset

mutation API that alters the messages exchanged by the software components of the SUT (data-driven mutants). The SUT test suite shall be executed with all the mutants, if it is effective then it shall fail with each of them. The mutants for which a failure is not observed are said to be *live* and indicate a pitfall in the test suite. All the FAQAS tools take as input the software under test (SUT), its test suite, and a set of configuration files.

MASS generates code-driven mutants. It integrates a pipeline of solutions that make mutation analysis feasible wit large SUT. The three main contributions of MASS are (1) the automated identification of trivially equivalent mutants using an ensemble of compiler optimization options, (2) the computation of the mutation score based on mutant sampling with fixed size confidence interval approach (FSCI), (3) the automated identification of equivalent mutants based on coverage. MASS reports the set of live mutants, the set of killed mutants (i.e., mutants that are discovered by the test suite), and information useful to draft a verification report, which includes the statement coverage of the SUT test suite and the mutation score (i.e., the percentage of mutants discovered by the test suite).

DAMAt generates mutants for data-driven mutation analysis. Data-driven mutation analysis is a research contribution of FAQAS. Instead of mutating the implementation of the SUT, it consists of altering the data exchanged by software components. DAMAt relies on fault models that specify how to mutate the data exchanged by software components through data-driven mutation operators. DAMAt can automatically alter data that is stored in data buffers (e.g., before serialization on the communication channel). DAMAt enables the simulation of faults that affect simulated components (e.g., sensors), which is not feasible with traditional, code-driven mutation analysis. DAMAt generates as output a set of killed mutants (i.e., mutants that, during testing, successfully

alter the data, and lead to test case failures), a set of live mutants (i.e., mutants that, during testing, successfully alter the data, but do not lead to test case failures), and a set of mutants not applied (i.e., mutants that, during testing, could not alter any data because the data they target is never exercised by the SUT); also, it provides information useful to draft a verification report, which includes the fault model coverage (i.e., percentage of fault models with at least one mutant applied), the mutation operation coverage (i.e., percentage of mutants applied), and the mutation score.

SEMuS automatically generates executable unit test cases based on code-driven mutation analysis results. The generated unit test cases detect mutants not detected by the original test suite. The generated test cases include test oracles that shall be manually validated by engineers, which enables detecting faults. The generated test cases can be integrated into regression test suites.

SEMuS takes as input the list of live mutants detected by MASS. It generates a set of additional test cases that can be integrated into the SUT test suite. Also, it reports the list of killed mutants and the list of mutants that remain live (i.e., for which SEMuS did not generate a test case that kill them). Live mutants shall be manually inspected by engineers to either determine if they are equivalent or to manually derive a test case capable of killing them.

DAMTE is a manual procedure supported by an automated symbolic execution toolset; it automatically identifies the test inputs that make software components exchange the data targeted by data-driven mutation operators. The derived test inputs can then be manually integrated into the SUT test suite.

The activity also included an extensive empirical evaluation demonstrating the feasibility, effectiveness, and scalability of the proposed toolsets in the space context, as described in the following sections.

2 EMPIRICAL EVALUATION

The FAQAS activity has ben evaluated through an extended empirical evaluation with six case study systems: ESAIL, *LIBGCSP*, *LIBParam*, *LIBUTIL*, MLFS, ASN1SCC.

2.1 MASS

Both GSL and LXS have manually inspected a subset of the live mutants identified by *MASS*. The inspection enabled industry partners to identify relevant shortcomings in their test suites: 57% of the live mutants are due to missing inputs, 23% of the live mutants were due to missing oracles. The few remaining live mutants had ben reported as either equivalent or not relevant (e.g., because concerning third party software). One fault was detected.

Finally, our results show that MASS helps addressing scalability problems to a significant extent by reducing mutation analysis time by more than 70% across subjects. In practice, for large software systems like *ESAIL*, such reduction can make mutation analysis practically feasible; indeed, with 100 HPC nodes available for computation, *MASS* can perform the mutation analysis of *ESAIL-CSW* in half a day.

2.2 SEMuS

The empirical evaluation demonstrated the scalability of *SEMuS* for the case study subjects in which it can be successfully applied (i.e., ASN1CC, MLFS, and *LIBUTIL*). Our results also demonstrated the usefulness of *SEMuS*. Indeed, *SEMuS* enabled the identification of two faults in our case studies. Also, the generated test cases concerned inputs that are relevant (according to specifications) but not tested by the test suites.

2.3 DAMAt

The empirical evaluation of DAMAt has demonstrated the effectiveness of the approach. Indeed, LXS has indicated that 57% out of the overall amount of 102 test suite problems detected by DAMAt were spotting major limitations of the test suite. Also, GSL has confirmed that the approach enabled the detection of relevant test suite shortcomings. One possible limitation of the approach is that it may introduce slow-downs that lead to non-deterministic failures when the test suite exercises brief interaction scenarios in which most of the operations performed concern the encapsulation of data into the network.

2.4 DAMTE

DAMTE aims to address a task (i.e., test generation at system and integration level) that is particularly difficult to address with state-of-the-art technology (e.g., test generation toolsets based on symbolic execution). For this reason, FAQAS only concerned the evaluation of the feasibility of DAMTE by applying it to the *LIBParam* client API functions. Overall, we conclude that the DAMTE approach may be feasible; however, it requires some manual effort for the configuration and execution of test cases which may limit its usefulness.

3 CONCLUSION

The FAQAS activity had been motivated by the need for high-quality software in space systems; indeed, the success of space missions depends on the quality of the system hardware as much on the dependability of its software. Before FAQAS there was no work on identifying and assessing feasible and effective mutation analysis and testing approaches for space software.

The main output of the FAQAS activity had been a toolset that implements three main features: code-driven mutation analysis (MASS), data-driven mutation analysis (DAMAt), code-driven mutation testing (SEMuS).

The empirical evaluation conducted with the aid of industrial case study providers have highlighted the practical usefulness of the FAQAS toolset, which led to the identification of relevant test suite shortcomings (test input partitions not exercised and missing test oracles) and bugs in the case study subjects. Since all the case study subjects considered in our empirical evaluation are space software systems that either undergo an extensive testing procedure and are deployed on orbit, the identification of such shortcomings and bugs indicate that the current procedures in place are not sufficient to guarantee software quality. Our results thus highlight the necessity for the adoption of an automated quality assessment toolset into development practices for space software — the FAQAS toolset has demonstrated to be an effective solution.

REFERENCES

[1] Thierry Titcheu Chekam, Mike Papadakis, Maxime Cordy, and Yves Le Traon. 2021. Killing stubborn mutants with symbolic execution. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 2 (2021), 1–23.

[2] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer* 11, 4 (April 1978), 34–41. https://doi.org/10.1109/C-M.1978.218136

[3] European Space Agency. 2017. ExoMars 2016 - Schiaparelli Anomaly Inquiry. *DG-I/2017/546/TTN* (2017). http://exploration.esa.int/mars/59176-exomars-2016-schiaparelli-anomaly-inquiry/

[4] Lech Madeyski, Wojciech Orzeszyna, Richard Torkar, and Mariusz Jozala. 2013. Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. *IEEE Transactions on Software Engineering* 40, 1 (2013), 23–42.

[5] Mike Papadakis, Christopher Henard, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Threats to the validity of mutation-based test assessment. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 354–365.

[6] D. Shin, S. Yoo, and D. Bae. 2018. A Theoretical and Empirical Study of Diversity-Aware Mutation Adequacy Criterion. *IEEE Transactions on Software Engineering* 44, 10 (Oct 2018), 914–931. https://doi.org/10.1109/TSE.2017.2732347