GomSpace A/S
Langagervej 6
9220 Aalborg E

EVALUATION OF MASS TOOLSET

GomSpace

Revision: 1.0
Date: 2021-09-15

GomSpace A/S
Langagervej 6
9220 Aalborg E

Document Title:	Evaluation of MASS toolset		
Revision number:	1.0	**Date:**	2021-09-15

Release Table:

Action	Name	Function	Date
Prepared / Owned by:	Adrian Michalik	Software Engineer	2021-09-05
Verified / Reviewed by:	Alastair Isaacs	Mission Lead	2021-09-15

Document Change Log

Revision	Date	Name	Description
0.1	2021-09-05	ADMI	DRAFT for internal release
1.0	2021-09-15	ALIS	FINAL for release

TABLE OF CONTENTS

1 Introduction

1.1 Purpose

This document reports the evaluation activities performed by GomSpace Luxembourg on the toolset implementing code-driven mutation analysis (MASS) delivered by the Interdisciplinary Centre for Security, Reliability and Trust (SNT) at the University of Luxembourg. This toolset represents a portion of the overall FAQAS framework, namely code-driven mutation analysis.

2 Documents

2.1 Reference Documents

Reference	Document Title	Issue	Date
RD-01	D2: Study of mutation testing applicability to space software	Issue 1 Rev 1	2020-06-24
RD-02	https://github.com/libcsp/libcsp/blob/master/README.rst	N/A	2021-03-09
RD-03	https://github.com/libcsp/libcsp	N/A	N/A
RD-04	FAQAS Framework Software User Manual	Issue 1 Rev 1	2021-03-26

3 Overview

3.1 General

Mutation analysis, or mutation testing, is a process that aims to evaluate the quality of a suite of test cases by introducing small syntactical changes (“mutations”) into the code under test. The quality of the test suite may be evaluated by the proportion of mutations that are detected or missed by the test suite. In an ideal situation the test suite can be expected to catch 100% of the mutations, however various real-world factors mean the actual proportion caught is likely to be lower.

To perform the mutation testing, a large number of copies of the code under test are generated, each with a distinct mutation. These copies are known as “mutants”. Mutants which result in a failed test are said to be “killed” by the test suite. Ideally a test suite would kill all the mutants, however it is likely that some mutants will survive. This ratio of survived vs killed mutants provides a metric for evaluating test suite quality.

Several challenges present on a closer inspection. Some mutants may result in “equivalent” code, that is, the same defect generated in two ways. Other mutants may alter the code but not in a way that produces defective code. These factors can affect the metric in ways that reduce its ability to measure the test suite quality.

The aim of the FAQAS framework is to implement a system for mutation analysis and for automatic generation of test cases based on the results of the analysis. In full, this framework includes:

- A toolset implementing code-driven mutation analysis
- A toolset implementing data-driven mutation analysis
- A toolset implementing code-driven test generation
- A toolset implementing data-driven test generation

The MASS (Mutation Analysis for Space Software) toolset implements the first of these four items; that is, implementation of code-driven mutation analysis. The toolset includes:

I. Trivial compiler optimizations, to remove equivalent and redundant mutants after their generation

II. Mutation sampling, to reduce the number of mutants to execute

III. Code coverage information regarding the original software, to prioritize and select test cases

IV. Code coverage information regarding the mutated files, to further detect equivalent and redundant mutants

For further details on the mutation analysis approach used by the MASS toolset, see RD01.

3.2 Background

GomSpace is a manufacturer and operator of small satellites known as cubesats. As well as hardware, GomSpace also produces software for commanding and operating satellites. This software is based around the CubeSat Space Protocol (CSP), a protocol stack written in C and used for communication and commanding of the satellite. Further information about CSP can be found in RD02.

The software libraries developed by GomSpace for use onboard satellites are tested using an automated test approach on ground. The test suites involved are prepared by software developers and aim to ensure a high quality of code before deployment and placement in the space environment. GomSpace does not currently employ mutation testing to evaluate the quality of these test suites.

CubeSats are built, integrated, and tested on a shorter timescale than is normal in more traditional satellite manufacturing. This means that the time available for testing and quality control is limited, and test cases must therefore be as complete as possible while executing in a short period. To this end, GomSpace applies a process of continuous integration and deployment (CI/CD) that emphasizes short development and test cycles. To be successful, the MASS toolset should be usable within this CI/CD framework, and capable of rapidly (i.e., on timescales as short as daily) evaluating changing code and test suites.

3.3 Software Libraries

The libcsp library was used to evaluate the performance of the MASS toolset. libcsp is an opensource implementation of the Cubesat Space Protocol (CSP). CSP is designed to ease communication between distributed embedded systems in smaller networks, such as Cubesats. The design follows the TCP/IP model and includes a transport protocol, a routing protocol and several MAC-layer interfaces. The core of libcsp includes a router, a connection-oriented socket API and message/connection pools.

The libcsp library is open source and available at RD03.

3.4 Objectives of Evaluation

The evaluation performed by GomSpace had the following objectives:

- To evaluate the effectiveness, scalability, efficiency, and applicability of FAQAS to space software
- To compare the effectiveness with state-of-the-art approaches in terms of the capability of spotting deficiencies in test suites
- To evaluate the effort required to put it in practice, its potential to uncover errors and the verification and validation activities to which it can be applied

3.4.1 Effectiveness

Effectiveness can be defined as the degree to which something – in this case the MASS toolset – is successful in producing a desired result. The desired result in this case is an evaluation of the quality of the test suite and highlighting of the mutants that survive the test suite; and to do so in a reasonable time.

3.4.2 Scalability

Scalability is the ability of a system to adapt to increased demands in term of performance, maintenance, and availability. In the case of MASS, the goal is to achieve easy horizontal scaling to perform mutation testing on many software components. Horizontal scaling means adding more instances of software instead of increasing computing power.

3.4.3 Efficiency

The efficiency of test software can be measured as number of test cases executed divided by unit of time (for example number of executed test cases per minute). The goal is to have a quick feedback loop that allows the introduction of mutation testing as part of CI/CD pipelines. In this way a company could monitor quality of a test suite after every code or test case change.

3.4.4 Applicability

The applicability of software refers to how useful it is. The goal is to assess if MASS is a tool needed in the space industry to improve software reliability. As part of this assessment, GomSpace's libcsp library will be placed under such mutation testing.

The need for reliable software and high-quality test suites is especially acute when working with the space environment. Objects in space cannot be directly accessed for repair or modification, and as a result a missed defect, whether in hardware or software, can be fatal to a mission. Traditionally, satellites have undergone a large degree of testing before launch to reduce as much as possible the risk of a defect. However, as timelines are often shorter, cubesats and the newspace industry require a more rapid approach to testing and quality. There is thus a need for a large degree of automation in testing and more rapid iterations of software and testing, more akin to the software practices seen in other parts of the technology industry. The challenge for tools like MASS is to enable this automation and rapid development without compromising the reliability of the software.

4 Evaluation

4.1 Ease of Use

4.1.1 Documentation

Before starting the MASS toolset, the project must be configured. The steps for this configuration are provided in the Software User Manual (SUM) (see RD04).

The SUM contains all necessary information in a well written style. This includes an explanation of the library structure and the purpose of contained files, a description of all configuration variables within those files and instructions for running the toolset. Each important file is provided with its own subsection, allowing the end-user to get a clear understanding of the framework configuration.

The evaluation did identify that a large number of abbreviations and acronyms are used within the SUM, some of them without explanation in the document. An expansion of the list of abbreviations and acronyms is recommended to improve the overall user experience.

4.1.2 Configuration

The MASS toolset offers many configuration options to the end-user. All these configuration options and parameters are well described in the SUM. However, the large degree of possibility is challenging for a new user to learn and can be overwhelming for first use. GomSpace recommends providing an interactive script with default values to ease this process.

Such a script should contain default values that allow users to follow a “standard” approach. This enables them to test their software quickly and provides a guide to configuring MASS properly for each particular use case.

4.2 Effectiveness

MASS was used to evaluate the test suite used with libcsp. The toolset executed successfully and produced a report with the number of created, killed and surviving mutants. Analysis of the surviving mutants shows the toolset does identify valid (potential) defects that the test suites currently miss. This implies the presence of missing test cases, often in areas that can be considered as challenging edge cases that are difficult for a developer or dedicated software tester to anticipate. In this the MASS tool is effective at identifying potential defects that are unanticipated by our current test suites. It is also worth mentioning that closer examination of the code inspired by this approach did seem to reveal actual defects in the software that were previously unknown.

4.3 Scalability

MASS can be containerized. The FAQAS team used Singularity as a container system, however it is also possible to create a Docker image that allows running MASS mutation tests in a docker container. Configuration files can be stored together with source code and mounted as volumes. In principle this makes it trivial to spawn a new instance (horizontal scaling).

4.4 Efficiency

Currently it takes a few hours to perform mutation testing against libcsp from scratch on a regular PC (eight steps – prepare SUT, generate mutants, compile optimized mutants, compile optimized mutants post-processing, generate prioritised and reduced test suites, execute mutants, identify equivalent mutants based on code coverage, compute mutation score).

Keeping in mind how many operations are performed during such testing this is still a very good result – however it is too long to run such tests after any change to a codebase which would be necessary for integration with a CI/CD pipeline. This, at least without efficiency improvements, means this analysis would be reserved for more infrequent milestones (for example at the end of a SCRUM sprint, before a project review, after developing new tests etc.).

4.5 Applicability

The evaluation showed that the MASS tool would be considered as a useful addition to GomSpace's testing processes.

After checking the output report, GomSpace realised that the test suite of libcsp does not address certain side effects of functions. This already demonstrates the ability of the toolset to identify improvements that would raise the quality of existing test suites. Overall, however, the number of killed mutants was encouragingly high, which provides GomSpace with a degree of confidence in the libcsp test suite.

Applying this approach to other libraries maintained by GomSpace could allow managers to direct efforts towards improving test suites identified as lower quality (i.e., those with more surviving mutants) or to set certain gateway thresholds (i.e., a certain proportion of mutants must be caught before a test suite is considered high enough quality to proceed). This would lead to an overall improvement in both test suite and code quality.

5 Concluding Remarks

The evaluation shows that the MASS toolset fulfils the expected role. It can also, according to the evaluation results, help to build more reliable software and be incorporated as a part of standard testing approaches at GomSpace.

The evaluation also concludes that:

- The setup process is generally well documented and practical to implement
- The user documentation should include a more complete list of abbreviations and acronyms
- The initial (first-time) configuration process should be simplified to reduce the steep learning curve
- The toolset takes too long to complete to form a part of the regular CI/CD pipeline, however it could be run less regularly to evaluate quality of test suites, and indirectly, code.
- The approach and toolset are useful enough to provide a meaningful benefit to the current GomSpace test and quality process