FAQAS MASS/DAMAt Toolsets

LuxSpace Assessment

FAQ-LXS-TN-0001

Fault-based, Automated Quality Assurance for Space Software (FAQAS) Project

Subcontract for the execution of the ESA Contract number 4000128969/19/NL/AS

Item Classification: CONFIDENTIAL

Version 1/- - date 01/10/2021

Document Meta Data

Title:	**FAQAS MASS/DAMAt Toolsets – LuxSpace Assessment**
Keywords:	FAQAS, MASS, DAMAt, Testing
Scope:	This document contains a description of the activities carried out by LuxSpace to use and evaluate the MASS ans DAMAt toolsets developed by SnT within the scope of the FAQAS project.
Document Number:	FAQ-LXS-TN-0001
Document Type:	Technical Note
Item Classification	CO
Document Requirements Definitions:	N/A
Contract Reference:	Subcontract for the execution of the ESA Contract number 4000128969/19/NL/AS
Document Status:	Issued
FileName:	FAQ-LXS-TN-0001_1 MASS_DAMAt_Toolsets_LXS_Assessment.docx
Issue/Revision Numbers:	1/-
Issue Date:	01/10/2021
BookCaptain:	LuxSpace/Antonio MARQUEZ

Signatures	
Author(s):	X [illegible] Antonio MARQUEZ Software Engineer Signed by: Antonio Marquez
Checked/Approved:	X [illegible] Emanuele GORINI Act. Product Assurance Manager Signed by: Emanuele Gorini
Authorised:	X [illegible] Emanuele GORINI Project Manager Signed by: Emanuele Gorini

Distribution List

Organisation	*Name*	*Electronic copy*	*Paper copy*
SnT/ University of Luxembourg	Fabrizio PASTORE <fabrizio.pastore@uni.lu>	1	-

Document Change Record

Issue	*Date*	*Page / paragraph affected*	*Change description*
1/-	01/10/2021	all	First Issue of the document

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

This Technical Note has been developed as a deliverable of the *Fault-based, Automated Quality Assurance for Space Software* (FAQAS) Project, in compliance with the contractual requirements (ref.[AD01]).

The purpose of this document is to assess, from the point of view of the final user, the MASS and DAMAt toolsets (ref. [RD01], [RD02], and [RD03]) developed by the *Interdisciplinary Centre for Security, Reliability and Trust* (SnT) of the University of Luxembourg.

For the evaluation of the MASS tool a test suite implemented in Google Test for a C++/Qt application has been used.

The DAMAt tool has been evaluated using the test suite for the E-SAIL On-board Satellite Control Software (OBSW) that was developed with the E-SAIL Software Validation Facility (SVF).

This document contains two main chapters for the assessment of the 2 tools.

Each one of these chapters provide information about:

- the use case used for the evaluation,
- the set-up of the environment,
- the execution of the tool,
- the evaluation of the results.

A final chapter summarizes the conclusion of the toolsets assessment

2 Documentation

2.1 Applicable Documents

AD01	UL-SnT Subcontract for the execution of ESA Contract No. 4000128969/19/NL/AS

2.2 Reference Documents

RD01	ITT-1-9873-ESA-FAQAS-D2 (Study of mutation testing applicability to space software)
RD02	ITT-1-9873-ESA-FAQAS-D4 (Validation of the toolset)
RD03	ITT-1-9873-ESA-FAQAS-SUM (Software User Manual)

3 Abbreviations

Abbreviation	**Definition**
FAQAS	Fault-based, Automatic Quality Assurance Assessment for Space Software
OBSW	On-board (Satellite Control) Software
SSE	Simulation Scenarios Editor
SVF	Software Validation Facility
SMP2	Simulation Model Portability
SUT	Software Under Test

4 MASS Toolset Evaluation

4.1 Brief Description of the toolset

The MASS toolset allows to evaluate a test suite using a code-driven approach. The toolset automatizes the process of generating and compiling a set of mutants from the SUT and executing and evaluating the test suite using each of these mutants.

4.2 Use Case Description: Simulation Scenarios Editor

The purpose of the Simulation Scenarios Editor (SSE) application is to allow the final user to generate files that can be used in an external simulator engine as input to establish the desired initial status of a simulation system.

The simulator attributes of the several simulation models, assembled into a system, may be initialized with values that the user can define/update by a graphical user interface (GUI).

This GUI is presented to the user as a set of fields.

Type, number, and description of these fields are fully customizable by the user through files with a specific format.

4.3 Software Required

To compile and run the software and the test suite, the following dependencies are required.

- Linux OS
- C++ compiler
- Qt5 libraries
- Boost libraries.
- Google Test libraries

4.4 Structure of the Project

The SSE application can be compiled and executed on a Linux OS distribution with Qt version 5 available. The project files are structured to provide all necessary items to deploy the given software and contains the following folders.

- **root**. Contains a *Makefile* to compile the SSE.
- **src**. Contains the code of the SSE.
- **examples.** Contains a default configuration to be loaded in the SSE.
- **styles.** Contains some styles to be applied to the SSE.
- **scripts.** Contains a default script to be executed in the SSE when starting a simulation in an external SMP2 engine.
- **test.** Contains the testsuite to be evaluated.

4.5 Test Suite

The test suite for the SSE software has been implemented by using the Google Test framework.

The *Makefile* supports some commands to execute the tests to check the functionalities of the application. There are two ways to execute these tests.

- **Automatic Tests**. All the tests will be executed without any interaction by the user.
    - The command to execute these tests is:

```
make test
```

- **Semi-interactive tests**. Some tests related to the GUI behavior requires interaction by the user.
    - The command to execute these tests is:

```
make test-interactive
```

    - The test scripts will display a message explaining the action to be performed by the user for several steps

For the MASS toolset evaluation purposes, only the Automatic Tests will be used.

4.6 Setup Environment

The configuration of the environment to use the MASS toolset with the SSE application has been performed inside a *Singularity* container provided by the University of Luxembourg. This container contains the MASS toolset pre-configured and all the required dependencies.

4.6.1 Installation of SSE in the Container

The provided Singularity does not contain the required dependencies to compile and execute the SSE application and the test suite.

It is necessary to install them before creating the MASS project.

- **Qt 5 libraries**.
    - These libraries can be installed from the official repositories.

```
apt-get install qt5-default
apt-get install qtbase5-dev
```

- **Boost libraries**
    - These libraries can be installed from the official repository.

```
apt-get install libboost-all-dev
```

- **CMake**
  - This tool can be installed from the official repository.

```
apt-get install cmake
```

- **Google Test libraries.**
  - This library shall be downloaded and compiled from a remote repository.

```
wget https://github.com/google/googletest/archive/refs/tags/release-
1.11.0.tar.gz
tar -xf release-1.11.0.tar.gz
cd release-1.11.0.tar
mkdir build
cd build
cmake .. & make
```

- **SSE**
  - The SSE application shall be copied directly to the */home/sse* folder.
  - Include and binary files of Google Test shall be copied to */home/sse/lib/googletest*.
  - The application shall be compiled using the *make* command.

4.6.2 Initialization of the MASS Workspace

The MASS workspace for the SSE project will be stored in */opt/sse* folder. Before executing the MASS script to initialize the workspace, it is necessary to set the INSTALL_DIR environment variable to this folder.

```
export INSTALL_DIR=/opt/sse
cd /opt/srcirorfaqas/MASS/FAQAS-Setup
./install.sh
```

After executing these commands, the folder is created with all the necessary files.

4.6.3 Configuration of mass_conf.sh script

The */opt/sse/mass_conf.h* file contains a set of definitions of variables that will be used by the MASS toolset to evaluate the test suite.

The following entries has been modified to adapt the configuration to our application.

- SRCIROR folder.
  - `export SRCIROR=/opt/srcirorfaqas`

- Directory path where MASS files can be stored.
    - `export SRCIROR=/opt/srcirorfaqas`
- Building system.
    - `export BUILD_SYSTEM="Makefile"`
- Directory root path of the software under test.
    - `export PROJ=$HOME/sse`
- Directory src path of the SUT.
    - `export PROJ_SRC=$PROJ/src`
- Directory src path of the SUT.
    - `export PROJ_TST=$PROJ/test`
- Directory coverage path of the SUT.
    - `export PROJ_COV=$PROJ/build/tests/coverage`
- Directory path of the compiled binary.
    - `export PROJ_BUILD=$PROJ/build`
- Filename of the compiled file/library.
    - `export COMPILED=sse.o`
- Path to original Makefile.
    - `export ORIGINAL_MAKEFILE=$PROJ/Makefile`
- Compilation command of the SUT.
    - `export COMPILATION_CMD=(make)`
- Compilation command for TCE analysis.
    - `export TCE_COMPILE_CMD=(make)`
- Command to clean installation of the SUT.
    - `export CLEAN_CMD=(make clean)`
- Flag to execute MASS on a HPC.
    - `export HPC="false"`
- TCE flags to be tested.
    - `export FLAGS=("-O0")`
- Flag to execute MASS with a prioritized and reduced test suite.
    - `export PRIORITIZED="false"`
- Sampling technique.
    - `export SAMPLING="uniform"`
- Sampling rate.
    - `export RATE="1"`

4.6.4 Configuration of PrepareSUT.sh script

PrepareSUT.sh is the first script that will be executed in the evaluation process. This script shall be modified to set the commands to compile the application and execute the test suite.

- Compile SSE

```
cd $PROJ
make clean &&
bear make all &&
make clean &&
sed -i 's: src: /home/mlfs/sse/src:g' compile_commands.json &&
mv compile_commands.json $MUTANTS_DIR
eval "${COMPILATION_CMD[@]}"
```

- Prepare test scripts

```
make clean
make
make init-test-folders
```

- Execute test cases and store its execution times[1]

```
start=$(date +%s)
make test-interchangeparser
end=$(date +%s)
source $MASS/FAQAS-GenerateCodeCoverageMatrixes/FAQAS-
CollectCodeCoverage.sh test-interchangeparser "$(($end-$start))"

start=$(date +%s)
make test-xmlconfigurationparser
end=$(date +%s)
source $MASS/FAQAS-GenerateCodeCoverageMatrixes/FAQAS-
CollectCodeCoverage.sh test-xmlconfigurationparser "$(($end-$start))"


start=$(date +%s)
make test-formcontroller
end=$(date +%s)
source $MASS/FAQAS-GenerateCodeCoverageMatrixes/FAQAS-
CollectCodeCoverage.sh test-formcontroller "$(($end-$start))"


start=$(date +%s)
make test-qssemainwindow
end=$(date +%s)
source $MASS/FAQAS-GenerateCodeCoverageMatrixes/FAQAS-
CollectCodeCoverage.sh test-qssemainwindow "$(($end-$start))"
```

4.6.5 Configuration of mutation_additional_functions.sh script

The MASS toolset needs to know how to execute a test from the SSE test suite from its name and evaluate if the result returns *PASS* or *FAIL*.

For this purpose, the function *run_tst_case* has been implemented inside the */opt/sse/mutation_additional_functions.sh* file. This function receives the name of the test to be executed and returns **0** (if the test passes) or **1** (if the test fails).

```
run_tst_case() {

      # get test name passed as parameter
      tst_name=$1

      # run the test case and define if test passes or fails
      cd $PROJ
      if (make $tst_name); then
            return 0
```

[1] The execution times will be used to calculate the values of the timeout for each test.

```
        else
                return 1
        fi
}
```

4.6.6 Creation of script for compiler optimizations

It is necessary to create another *Makefile* (with filename *Makefile.template*) that will be used by the MASS toolset to compile the application using the optimization options specified in 4.6.3.

For the SSE application it is enough to copy the content of the original *Makefile* to the new one, removing all the debugging/coverage flags and adding a TCE placeholder to the compiler/linking commands.

4.7 Execution of the MASS tool

Once the environment is fully configured, it is possible to execute the scripts provided by the MASS toolset to evaluate the SSE test suite.

It is possible to execute all the scripts using the *Launcher.sh* script but instead they will be executed one by one to analyze the results of each of these executions.

4.7.1 PrepareSUT.sh

This script was previously modified in the 4.6.4section.

It creates the *COV_FILES* folder, where all the coverage files (*.gcda .gcno* files) for each test are stored.

It also creates the *COV_FILES/test_suite_order.txt* file that stores the timeouts for each test. These values are as follow.

- *test-interchangeparser*, **93** seconds.
- *test-xmlconfigurationparser*, **108** seconds.
- *test-formcontroller*, **102** seconds.
- *test-qssemainwindow*, **153** seconds.

These timeouts are used when the MASS toolset executes the *run_tst_case* function for each mutation (if this function does not finish before the specific timeout, it will be killed and marked as FAIL).

4.7.2 GenerateMutants.sh

This script generates all the mutants to be used to evaluate the test suite by the MASS toolset. These mutants are stored in the *src-mutants* folder.

A total of **949** mutants are generated for this project.

4.7.3 CompileOptimizedMutants.sh

This script compiles each of one of the mutants generated in the previous step using the *Makefile.template* with the optimization options specified in the *mass_conf.sh* (only the *-O0* option was specified as seen in section 4.6.3).

It creates the *COMPILED* folder with a set of reports of the process for each of the used optimization flags.

From these reports we can find that **206** mutations fail when trying to compile, and **743** mutations are successfully compiled.

4.7.4 OptimizedPostProcessing.sh

This script analyses the results from the previous step in order to generate some additional reports in the *COMPILED* folder for the equivalent and redundant mutations.

A total of **12** equivalent and **56** redundant mutations were detected in this step.

4.7.5 GeneratePTS.sh

This script creates a list of reduced mutants to be used when creating a sample of mutations to be executed and tested. The list is stored in the *PRIORITIZED* folder.

The report for the SSE contains a total of **686** prioritized and reduced mutants.

4.7.6 ExecuteMutants.sh

This script extracts a sample from the prioritized tests and the sampling method specified in the *mass_conf.sh* and executes their associated tests for the compiled versions of the application for each of these mutations.

The values specified in the section 4.6.3 for the sampling method and rate are *uniform* and *1* respectively.

This is the longest step of all since it is necessary to compile and execute the associated tests for each of all the selected mutants.

The script creates a set of reports in the *MUTATION* folder with the results of the executions. The log and the coverage files generated for each execution are also stored in different folders inside this folder.

From these files, the following information is extracted.

- **460** mutants are killed by the test suite.
- **214** mutants are live (the test suite does not fail with these mutants).

4.7.7 IdentifyEquivalents.sh

This script analyzes the results from the executions performed in the previous step and generates a list in the *RESULTS* folder with a set of useful live mutations to be considered.

The list contains **34** live mutants. They are the cases that are analyzed in the section 4.8.

4.7.8 MutationScore.sh

This script generates the final report of the test suite evaluation.

These results are shown in the Table 1.

```
##### MASS Output #####
## Total mutants generated: 743
## Total mutants filtered by TCE: 68
## Sampling type: uniform
## Total mutants analyzed: 674
## Total killed mutants: 460
## Total live mutants: 214
## Total likely equivalent mutants: 0
## MASS mutation score (%): 68.24
## List A of useful undetected mutants:
/opt/sse/DETECTION/test_runs/useful_list_a.csv
## List B of useful undetected mutants:
/opt/sse/DETECTION/test_runs/useful_list_b.csv
## Number of statements covered: 692
## Statement coverage (%): 80.75
## Minimum lines covered per source file: 0
## Maximum lines covered per source file: 144
```

Table 1. Mutation score of the SSE test suite.

4.8 Evaluation of the Results

After executing all the steps of the toolset, it is possible to analyse the list of live mutations provided in the files *DETECTION/test_runs/useful_list_a.csv* and

DETECTION/test_runs/useful_list_b.csv to detect possible improvements to apply to the test suite. See Annex 1 attached to the PDF version of this document.

4.8.1 Class1.mut.47.1_1_2.SDL.~Class1

This mutation deletes the code of the destructor of the Class1 class. This class contains a list of controls, so the destructor is calling to the destructors of these controls.

The test could be improved to check if the memory is freed after calling the destructor, or if the Class1 destructor is calling to the destructor of all the controls.

4.8.2 Class2.mut.376.1_1_6.ROD.setClass6Value

This mutation removes the condition when trying to update a combobox with a non-existing value.

The test could be improved to check if the combobox is not updated when trying to set a value that are not in the list of the combobox.

4.8.3 Class3.mut.271.4_2_16.ROR.parseTab

This mutation changes the terminate condition of a loop (from < to <=), so an additional iteration will be executed. However, the subsequent conditions are false, so the change is not relevant for the result.

4.8.4 Class4.mut.444.1_1_2.SDL.confirmLoadSSEFile

This mutation deletes the return sentence of the *confirmLoadSSEFile* method. *confirmLoadSSEFile* is a method that requires interaction by the user.
The test provided to MASS are automatic, so in this case a mock/stub method is used Anyway the test suite has also a set of interactive tests that would detect these kinds of cases.

4.8.5 Class4.mut.117.1_1_2.SDL.createActions

This mutation deletes the call of a function that generates the View menu in the toolbar of the application. The View menu contains two entries related to functionalities that are not implemented, so the test suite does not check this menu.

The test could be improved to check if the View menu is generated.

4.8.6 Class2.mut.72.1_1_3.SDL.buildTabWidget

This mutation deletes a sentence that associates a button with a specific action. There is a testcase that check this action, but it was removed from the test suite before executing the MASS tool.
This is the only testcase that was removed. The MASS toolset detected this omission successfully.

4.8.7 Class5.mut.46.1_1_2.SDL.~Class5

This mutation deletes the code of the destructor of the Class5 class. This class contains a list of elements, so the destructor is calling to the destructors of these elements.

The test could be improved to check if the memory is freed after calling the destructor, or if the Class5 destructor is calling to the destructor of all the elements.

4.8.8 Class2.mut.295.1_1_2.SDL.buildTextcodeClass6

This mutation deletes the return sentence of the buildTextCodeClass6. The application will not build this kind of element. The test suite is overwriting this method in the mock of Class2.

The test could be improved removing the overwrite method in the mock of Class2 to use the real method.

4.8.9 Class3.mut.278.1_1_6.ROD.parseTab

This mutation set a condition always to false in the configuration parser. The consequence is that the tabs inside any other tab will be not created.

The test data could be improved adding a tab inside a tab in the configuration file. The test could be improved to check this tab.

4.8.10 Class3.mut.239.1_1_2.SDL.parseTextlineDatatype

This mutation deletes the return sentence of the *parseTextlineDatatype*. The tests are not checking the returned element because is it not necessary.

The test could be improved to check the returned element of the function.

4.8.11 Class4.mut.119.1_1_2.SDL.createActions

This mutation deletes the call of a function that generates the Help menu in the toolbar of the application. The View menu contains an entry to show information about the application.

The test could be improved to check if the Help menu is generated.

4.8.12 Class4.mut.454.1_1_2.SDL.showFieldsNotMappedWarningMessage

This mutation deletes a sentence that shows a warning message to the user. This warning message appears in the interactive tests, but the tests are not checking that this message is displayed.

The test could be improved to check that the method to show the warning message is called with the correct message.

4.8.13 Class6.mut.48.1_1_2.SDL.~Class6

This mutation deletes the code of the destructor of the Class6 class. This class contains a datatype, so the method is calling to its destructor.

The test could be improved to check if the memory is freed after calling the destructor, or if the Class1 destructor is calling to the destructor of the datatype.

4.8.14 Class3.mut.167.1_1_2.SDL.parseNumericDatatype

Same case as explained in Class3.mut.239.1_1_2.SDL.parseTextlineDatatype.

4.8.15 Class4.mut.80.7_1_5.ROR.loadClass5

This mutation negates a condition that checks if there is a style provided from command line to be applied to the application.

The test could be improved to check that a style provided by the user from command line is applied.

4.8.16 Class4.mut.281.7_1_6.ROR.loadScenarioFile

This mutation negates a condition that checks if there are no mapped attributes to show a warning message.
It is similar to Class4.mut.454.1_1_2.SDL.showFieldsNotMappedWarningMessage.

The test could be improved to check that the method to show the warning message is called with the correct message.

4.8.17 Class4.mut.281.1_1_3.SDL.loadScenarioFile

This mutation changes the terminate condition of a loop (from < to <=), so an additional iteration will be executed. However, the subsequent conditions are false, so the change is not relevant for the result.

4.8.18 Class4.mut.279.7_1_5.ROR.loadScenarioFile

This mutation changes a condition that checks if there is any error when loading a Scenario File to show an error message.

The test could be improved to check that the method to show the error message is called with the correct message.

4.8.19 Class4.mut.279.3_1_5.UOI.loadScenarioFile

Same case as explained in Class4.mut.279.7_1_5.ROR.loadScenarioFile.

4.8.20 Class3.mut.85.4_2_16.ROR.parseDatatypes

Same case as explained in Class3.mut.271.4_2_16.ROR.parseTab.

4.8.21 Class3.mut.294.4_2_16.ROR.parseSection

Same case as explained in Class3.mut.271.4_2_16.ROR.parseTab.

4.8.22 Class4.mut.205.1_1_2.SDL.createViewActions

This mutation deletes a sentence that associate a GUI control with an action. This action is not implemented (just show a warning message), so it is not necessary to check it.

4.8.23 Class2.mut.72.3_2_16.ROR.buildTabWidget

This mutation changes a condition to ignore the root elements that are not containers.

The test data could be improved adding a not container element in the root. The test could be improved to check that this element is ignored.

4.8.24 Class4.mut.451.1_1_2.SDL.showFieldsNotMappedWarningMessage

This mutation changes a warning message to be displayed.
Same case as explained in Class4.mut.454.1_1_2.SDL.showFieldsNotMappedWarningMessage.

4.8.25 Class3.mut.215.1_1_2.SDL.parseTextfileDatatype

Same case as explained in Class3.mut.239.1_1_2.SDL.parseTextlineDatatype.

4.8.26 Class3.mut.192.1_1_2.SDL.parseFileDatatype

Same case as explained in Class3.mut.239.1_1_2.SDL.parseTextlineDatatype.

4.8.27 Class3.mut.203.1_1_2.SDL.parseTextboxDatatype

Same case as explained in Class3.mut.239.1_1_2.SDL.parseTextlineDatatype.

4.8.28 Class3.mut.227.1_1_2.SDL.parseTextcodeDatatype

Same case as explained in Class3.mut.239.1_1_2.SDL.parseTextlineDatatype.

4.8.29 Class3.mut.180.1_1_2.SDL.parseDatetimeDatatype

Same case as explained in Class3.mut.239.1_1_2.SDL.parseTextlineDatatype.

4.8.30 Class3.mut.153.1_1_2.SDL.parseEnumeratedDatatype

Same case as explained in Class3.mut.239.1_1_2.SDL.parseTextlineDatatype.

4.8.31 Class4.mut.204.1_1_2.SDL.createViewActions

This mutation deletes the call of a function that generates an item inside the View Menu related to a not implemented functionality.

4.8.32 Class4.mut.189.1_1_2.SDL.createEditActions

This mutation deletes a sentence that delete a instance of a class. It is not necessary to check it.

4.8.33 Class4.mut.199.1_1_2.SDL.createViewActions

Same case as explained in *Class4.mut.204.1_1_2.SDL.createViewActions*.

4.8.34 Class4.mut.251.1_1_2.SDL.createHelpActions

This mutation deletes the call of a function that generates an item inside the Help Menu related to the About functionality.

The test could be improved to check that the About functionality display information about the application.

5 DAMAt Toolset Evaluation

5.1 Brief Description of the Toolset

DAMAT is a toolset that generates an API to generate mutations of a set of messages with a specified format. This toolset receives as input a CSV file with the format of the messages to be mutated and the properties of the desired mutations. The toolset allows to execute and evaluate a test suite with these mutations.

The toolset has been successfully used to analyse the ESAIL SVF test suite. Some possible improvements to the test suite have been detected by the toolset.

5.2 Use Case Description: ESAIL SVF

ESAIL is a satellite that was launched in Kourou (French Guiana) on September 3, 2020. During the development of the OBSW of the satellite, a SVF that simulates the components of the satellite was implemented to execute a set of tests without need for hardware.

This SVF was implemented in C ++ following the SMP2 standard and using a development framework and a SMP2 simulator engine (Rufos).

The target of the test suite is the evaluation of the ESAIL OBSW. The OBSW binary is loaded in a simulated computer implemented inside of the SVF to execute all the required tests in the simulation environment.

5.3 Software Required

To compile and run the software and the test suite, the following dependencies are required.

- CentOS 7.
- C++ compiler.
- Rufos.
- ESAIL OBSW binary.
- OHB Simulation Platform.
- SMP2 model of GR712RC computer.
- Python 2.7.

5.4 Structure of the Project

The SVF can be compiled on a CentOS distribution with the OHB Simulation Platform that contains all the libraries needed to develop SMP2 models.

The output of the compilation is a dynamic library that can be loaded by Rufos (a SMP2 engine).

The code of the models of the SVF are separated in different folders related to one of each module of the ESAIL satellite to be simulated.

5.5 Test Suite

The test suite has been implemented in Python by using the OHB Simulation Platform. This platform uses the Python unittest framework as base.

5.5.1 Tests implemented

With the aim of abstracting the implementation of the tests, a framework has been implemented that acts as layer to handle the ESAIL SVF.

The implemented tests have been grouped in different folders according to the module being tested.

- ADCS: **10** tests.
- CAN: **18** tests.
- EPS: **10** tests.
- FDIR: **31** tests.
- OPSE: **8** tests.
- Services: **35** tests.
- TCS: **3** tests.
- TMCT: **8** tests.

5.5.2 esail.sh

A tool has been implemented in bash (*esail.sh*) to facilitate the execution of the test suite. The available commands for this tool are listed below.

- **help** (h): Display help.
- **jobs** (j): Maximum number of jobs (tests) to run in parallel. Defaults to 1.
- **svf** (v) <*version*>: SVF version to use. Can be use the path to a directory containing the SVF library, a path to a directory containing several SVF directories, form which the one corresponding to the largest version will be obtained, or a version X, which will load /opt/Sim-E-SAIL/share/SvfX.
- **source** (s) <*dir*>: Path to the Source directory. Needed to update Python enumerations that are automatically generated from header files. If not provided, it may be inferred from the path of the OBSW binary file, if specified.
- **obsw (**o) <*bin*>: Path to the OBSW binary file. Can also be specified through the environment variable OBSW_BIN. If not defined, it will be taken from _binaries/OBSW.exe inside the source directory, if provided.
- **version** <*version*>: Version of the OBSW as major.minor.revision.
- **persistent** (p): Make ASW MRAM region non-volatile. Equivalent to providing both --save-mram and --restore-mram options.

- **mram-file** *<path>*: Path to the file where MRAM shall be saved to / restored from. By default, *Test/System/common/asw_mram.bin*.
- **save-mram**: If provided, contents of ASW MRAM region will be saved at the end of the test to a binary file.
- **restore-mram:** If provided, contents of ASW MRAM region will be restored at the beginning of the test from a binary file, if it exists.
- **continue** (c): Continue automatically after loading the OBSW.
- **test** (t) *<script>* Test script(s) to run. Either:
    - Absolute or relative path to python script.
    - Integer n -> *testcases/*/E-SAIL-ASW-TST-n.py*
    - Directory d -> *testcases/d/E-SAIL-ASW-TST-*.py*
    - 'all' -> *testcases/*/E-SAIL-ASW-TST-*.py*
- **unit-test** (u): Whether the test to be run is a unit test. If not provided, the Python script is assumed to contain a system test.
- **nogui** (n): Do not launch the GUI.
- **redundant-obc**: Use the redundant OBC at boot-up.
- **redundant-can-bus**: Use the redundant CAN BUS at boot-up.
- **edac**: Enable error detection and correction in the FLASH model.
- **authentication** (a) Enable TC authentication after boot up.
- **realtime**: Configure the GPS models to report the time advancing with the current UTC.
- **faster** (f): Run faster than real time.
- **log** (l): Display Rufos log in terminal (true by default if nogui option specified, otherwise false).
- **display-prints** (d): Enable the log property of the GR712RC APBUART1 line to redirect print messages from the OBSW to the Rufos log.
- **define** (D) *<var>*: Define a Python variable as True before the test is run. Can be provided multiple times to set several variables. When at least one variable is defined and multiple tests are about to be run, the tests not containing "# filter: var" for any of the defined variables will be skipped.
- **gdb** (g): Enable debugging from an external GDB client (e.g. eclipse). When this option is enabled and the OBC FPGA watchdog is not kicked for more than 20 seconds, instead of an OBC reset the simulation is paused to enable further debugging. This option is not compatible with the task-trace option.
- **break** (b): Set a breakpoint at rtemsInit.c:Init.

- **gdb-script** *<arg>* : Path to a GDB script to be run after loading the OBSW.
- **port** *<arg>*: Base port (if not defined, chosen automatically).
- **which-port** *<user@module>*: Get the port that will be used for a specific user and module.The module parameter can take the next values:
    - gdb: GDB port
    - tmtc: TM/TC port
- **which-svf**: Get the path to the directory from which the SVF library will be loaded.
- **update-resources**: Forces some resources in Python files, which by default are automatically regenerated from resources in the Source directory when these change, to be regenerated.
- **coverage**: Force printing of coverage results at the end of the test. An instrumented OBSW binary file must be provided.
- **import** *<dir>*: Automatically import the coverage results into a VectorCAST project located at the directory <dir>.
- **results** (r): Runs the test(s) without GUI and prints only a summary of the test results. The GDB-related options are ignored. When more than one test is to be run, this is activated by default.
- **final**: Only show the final results when the tests finish instead of displaying the progress as they run.
- **silent**: Whether no output should be provided in terminal when running tests in batch.
- **retry** <n>: Number of times to retry to run a test if it fails. By default, 0. Only applicable if the option --results is provided or when running tests in batch without a GUI.
- **testpath** *<arg>*: Absolute path to the Test directory inside the repository. By default, the parent directory of the directory containing the esail.sh script that is being run. Can be set in order to run tests from working copies different than the esail.sh's working copy.
- **history-depth** *<n>*: Number of test results from previous runs to keep in the history folder. Defaults to 5. If set to 0, only the results of the current run are kept.
- **timeout** *<t>*: Number of seconds without kicking the PCDU FPGA watchdog after which the simulation will be aborted. Defaults to 0 (timeout disabled) or to 60 s if the option --results is provided.
- **task-trace:** Enable automatic generation of TaskTrace.bin. If the OBC FPGA watchdog is not kicked and this option is not provided, power to the OBC will be cut immediately. When this option is enabled, power will be cut 5 seconds later to allow sufficient time for dumping the TaskTrace. The OBSW binary must be compiled with task_trace=true for this to work. This option is not compatible with the gdb option.

- **plot** <*name*>: Generate plots. Can be provided multiple times to enable several plots. Currently supported names are:
  - cpu_usage: a plot with the CPU usage of the top tasks over time (requires option task-trace).
- **procedure**: Generate a procedure report at the end of the test, containing a summary of the steps, checks and asserts, and their results.
- **junit**: Generate a JUnit report to be used by Jenkins.
- **clean**: Whether the entire testresults directory should be removed before execution of test(s).
- **incremental:** Whether only the tests that have changed (or whose dependencies have changed) since last run, amongst those specified to be run, should be run. If not specified, all indicated tests are run regardless of whether no changes occurred.
- **req:** Generate a requirement text file including all the requirements intended to be covered by the test.
- **req-coverage** <*reqid*>: Display the test scripts that cover a given requirement.
- **tsc**: Allow external connections from TSC to receive telecommands in CCS IF.
- **monitor** <*ids*>: Enable monitoring of trigger sources by ID or name. Can be provided up to two monitor two trigger sources simultaneously.
- **debug** <*opts*>: Enable debugging options for testing platform (SVF, Python library, scripts). Can be provided multiple times to set several debug options. Currently supported options are:
  - obt
- **kill:** Kill processes currently running the same test.
- **config** <*file*>: Path to a file from which command-line arguments can be loaded. The file can contain multiple lines. Lines starting with # will be ignored. Command-line arguments specified in terminal will override those in the config file. By default, a file called esail.config in the current working directory will be used if it exists.
- **flash**: Load an SVF library containing the FLASH model in the nominal OBC (*deprecated*).
- **ignore-config-file**: Ignores esail.config file in the current directory.
- **ignore-script-config**: Ignore config arguments provided in the test script.
- **skip-header**: Do not print the header line in top of the test results summary.
- **dry-run:** Do not run the tests, only print results if results file already exists.
- **print-paths:** Do not run the tests, instead only print the paths of the test scripts that would be run if this option was not provided.

- **attempts** <*n*>: Number of attempts before the current run. Purely informative.

5.6 Set Up Environment

A virtual machine with CentOS 7 as operating system has been prepared with all the dependencies for the execution of the test suite.

DAMAt toolset has been provided by the University of Luxembourg as a set of files, scripts and fault models to be used as inputs. These files can be copied in any folder of the virtual machine.

5.7 Execution of the DAMAt Tool

The DAMAt workflow is composed by several steps, which are explained in the next sections.

5.7.1 Preparation of Fault Models

The main input of the DAMAt toolset is the definition of the mutations to be applied in the messages. This definition shall be in a CSV file with the following columns.

- **FaultModel**. The model that sends/receives the messages to be mutated.
- **DataItem**. The item of the message to be mutated.
- **Span**. The start (in bits) of the item to be mutated.
- **Type**. Type of the data item (BIN, HEX or DOUBLE).
- **FaultClass**. The operator of the mutation to be applied.
  - BF: Bit flip.
  - VAT: Value above threshold.
  - VBT: Value below threshold.
  - VOR: Value out of range.
  - INV: Invalid Numeric Value.
  - IV: Illegal Value.
  - ASA: Anomalous Signal Amplitude.
  - SS: Signal Shift.
  - HV: Hold Value.
  - FVAT: Fix value above threshold.
  - FVBT: Fix value below threshold.
  - FVOR: Fix value out of range.
- **Min**. Parameter to be applied by VOR, BF, INV and FVOR mutation operators.
- **Max.** Parameter to be applied by VOR, BF, INV and FVOR mutation operators.
- **Threshold**. Parameter to be applied by VAT, VBT, ASA, FVAT and FVBT mutation operators.
- **Delta**. Parameter to be applied by VAT, VBT, VOR, ASA, SS, FVAT and FVBT mutation operators.
- **State**. Parameter to be applied by BF mutation operator.

- **Value**. Parameter to be applied by BF, IV, ASA and HV mutation operators.

A set of CSV files has been provided by the University of Luxembourg for the definitions of the mutations of the ADCS IF, PDHU and GPS messages.

5.7.2 DAMAt_probe_generation.sh

This script generates a mutation API with the functions that modify the messages according to the inputs prepared in the previous step.

After executing the script, three files are generated.

- *FAQAS_dataDrivenMutator.h*: C++ header containing the mutation API.
- *FAQAS_mutants_table.csv*: CSV file containing the mutation operators applied.
- *function_calls.out*: Helper document with the template of the functions to be applied to the target classes.

```
To mutate insert probes using the following templates:

void mutate_FM_MagnetorquerSetPWMRSP(std::vector<unsigned char> *v);
void mutate_FM_SSTP(std::vector<unsigned char> *v);
void mutate_FM_SunSensorTM(std::vector<unsigned char> *v);
void   mutate_FM_MagnetorquerSetPWMRSPFailure(std::vector<unsigned   char>
*v);
void mutate_FM_SunSensorTMFailure(std::vector<unsigned char> *v);
void mutate_FM_SpaceCraftHK(std::vector<unsigned char> *v);
void mutate_FM_IfStatus(std::vector<unsigned char> *v);
void mutate_FM_SSTPFailure(std::vector<unsigned char> *v);
void mutate_FM_GYTMFailure(std::vector<unsigned char> *v);
void mutate_FM_IfHK(std::vector<unsigned char> *v);
void mutate_FM_FaultModel(std::vector<unsigned char> *v);
void mutate_FM_GYTM(std::vector<unsigned char> *v);
```

Table 2. Content of the function_calls.out file after executing the DAMAt_probe_generation.sh with the ADCS fault model.

5.7.3 Instrument Code with Mutation API

The *FAQAS_dataDrivenMutator.h* can be copied to the same folder of the class to be mutated. This class is modified by calling to the specific method of the API for each message to mutate.

The *Makefile* is also modified to instrument this code.

5.7.4 DAMAt_mutants_launcher.sh

This script performs the following actions.

- Generates all the mutants according to the code instrumented with the mutation API.
- Compile all the generated mutants.
- Execute the test suite with all the generated mutants.

This step consumes around 1000 hours just for the ADCS fault model cases since it is necessary to execute all the test for each of the defined mutations.

The script generates several files in the *RESULTS* folder.

- *mutation_sum_up.csv*: CSV file with the main metrics of the mutations executed.
- *final_mutants_table.csv*: CSV file with an overview of all the generated mutants.
- *mutation_score_by_*.csv*: CSV files with the mutation score for all the generated mutants categorized in several ways.

5.8 Evaluation of the Results

After executing all the steps using the mutations for ADCS, PDHU and GPS modules, it is possible to analyse the list of live mutations that can be found in the generated CSV files to detect possible improvements to apply to the test suite. See also Annex 2, 3 and 4 attached to the PDF version of this document.

5.8.1 ADCS Module

5.8.1.1 IfStatus Model

5.8.1.1.1 OBC communication error

These values are not verified in the SVF. One possible way to verify these values would be by reading the log generated by the ADCS IF unit (**adcsif.log**). In this file we can find all the commands/responses sent/received by the unit.
Another way would be by checking the OBDI entries associated with these values or by requesting the related housekeeping groups.

5.8.1.1.2 Unit communication error

Same as case explained in 5.8.1.1.1.

5.8.1.1.3 Unit in error

Same as case explained in 5.8.1.1.1.

5.8.1.1.4 Gyroscope enable

Same as case explained in 5.8.1.1.1.

5.8.1.1.5 Reaction Wheel enable

Same as case explained in 5.8.1.1.1.

5.8.1.1.6 3 axis Magnetorquer enable

Same as case explained in 5.8.1.1.1.

5.8.1.1.7 Sun Sensor board ADC enable

Same as case explained in 5.8.1.1.1.

5.8.1.2 IfHK

5.8.1.2.1 VCCb

Same as case explained in 5.8.1.1.1.

5.8.1.2.2 VBUS

Same as case explained in 5.8.1.1.1.

5.8.1.2.3 VCC_SW1

Same as case explained in 5.8.1.1.1.

5.8.1.2.4 T_PCB_TEMP2

Same as case explained in 5.8.1.1.1.

5.8.1.2.5 T_PCB_TEMP3a

Same as case explained in 5.8.1.1.1.

5.8.1.2.6 T_PCB_TEMP3b

Same as case explained in 5.8.1.1.1.

5.8.1.2.7 T_PCB_TEMP4

Same as case explained in 5.8.1.1.1.

5.8.1.3 GYTM

5.8.1.3.1 Unit Identifier

Same as case explained in 5.8.1.1.1.

5.8.1.4 GYTM Failure

5.8.1.4.1 Error Type

In test 3077 an error is simulated in Gyroscope by modifying some attributes of the model (*forcedResponse*, *forcedResponseCmdId*, *forcedResponseSubcmdId*, *forcedResponseErrorCode*), since it is not possible to provoke it from the SVF.

It would be necessary to perform the mutation in this part of the code (line 107 file *AdcsIf.cpp*).

5.8.1.5 Sun Sensor

5.8.1.5.1 Photodiode Q1 ADC(2..6)

It is possible to configure the simulator to put non-nominal scenarios for these values.

But there are no tests that check these values. The test 6275 illuminates a set of photodiodes to force the Karman filter divergence.

The solution would be read the answer from ADCS IF (using log file as explained in row 1) and compare them with current values stored in the SVF.

5.8.1.5.2 Photodiode Q2 ADC(2..6)

Same as case explained in 5.8.1.5.15.8.1.1.1.

5.8.1.5.3 Photodiode Q3 ADC(2..6)

Same as case explained in 5.8.1.5.15.8.1.1.1.

5.8.1.5.4 Photodiode Q4 ADC(2..6)

Same as case explained in 5.8.1.5.15.8.1.1.1.

5.8.1.6 Sun Sensor TM Failure

5.8.1.6.1 Error Type

There are no tests to check different error codes returned by ADCS IF. The solution would be force the desired error and check that the ADCS IF build the answer properly.

5.8.1.7 SSTP

5.8.1.7.1 Temperature reading from ADC #(3..7)

Same as case explained in 5.8.1.1.1.

5.8.1.8 SSTP Failure

5.8.1.8.1 Error Type

Same as case explained in 5.8.1.1.1.

5.8.1.9 SpaceCraftHK

5.8.1.9.1 TMTC_SW1

Same as case explained in 5.8.1.1.1.

5.8.1.9.2 TMTC_SW2

Same as case explained in 5.8.1.1.1.

5.8.1.9.3 SC_TEMP5

Same as case explained in 5.8.1.1.1.

5.8.1.9.4 SC_TEMP6

Same as case explained in 5.8.1.1.1.

5.8.1.10 MagnetorquerSetPWMRSP

5.8.1.10.1 Unit identifier Magnetometer

Same as case explained in 5.8.1.1.1.

5.8.1.10.2 Magnetorquer nX Current - On

Same as case explained in 5.8.1.1.1.

5.8.1.10.3 Magnetorquer nX Current - Off

Same as case explained in 5.8.1.1.1.

5.8.1.10.4 Magnetorquer pX Current - On

Same as case explained in 5.8.1.1.1.

5.8.1.10.5 Magnetorquer pX Current – Off

Same as case explained in 5.8.1.1.1.

5.8.1.10.6 Magnetorquer nY Current - On

Same as case explained in 5.8.1.1.1.

5.8.1.10.7 Magnetorquer nY Current – Off

Same as case explained in 5.8.1.1.1.

5.8.1.10.8 Magnetorquer pY Current - On

Same as case explained in 5.8.1.1.1.

5.8.1.10.9 Magnetorquer pY Current – Off

Same as case explained in 5.8.1.1.1.

5.8.1.10.10 Magnetorquer nZ Current - On

Same as case explained in 5.8.1.1.1.

5.8.1.10.11 Magnetorquer nZ Current – Off

Same as case explained in 5.8.1.1.1.

5.8.1.10.12 Magnetorquer pZ Current - On

Same as case explained in 5.8.1.1.1.

5.8.1.10.13 Magnetorquer pZ Current – Off

Same as case explained in 5.8.1.1.1.

5.8.1.11 MagnetorquerSetPWMRSPFailure

5.8.1.11.1 Error Type

Same as case explained in 5.8.1.1.1.

5.8.2 PDHU Module

5.8.2.1 STO_GET_HEAD

5.8.2.1.1 Command Status

There are no testcases that check the different error codes.

Same as case explained in 5.8.1.1.1.

5.8.2.2 STO_GET_TAIL

5.8.2.2.1 Command Status

There are no testcases that check the different error codes.

Same as case explained in 5.8.1.1.1.

5.8.2.3 CRC_COMPUTE

5.8.2.3.1 Command Status

There are no testcases that check the different error codes.

Same as case explained in 5.8.1.1.1.

5.8.3 GPS Module

5.8.3.1 F40

5.8.3.1.1 GPS seconds of week [s] (of navigation solution)

The testcase could compare the current date with the F40 answer.

In the test performed in the real satellite, this current date had to be introduced by user.

6 Conclusions

The analysis of the results of the toolsets shows a clear effectiveness in detecting flaws of the testsuites in different environments: different programming languages, different testing frameworks.

For the MASS tool, the percentage of detected problems that can be classified as major is the 38.24% out of the overall amount of 34 testsuite problems detected.

For the DAMAt tool, the percentage of detected problems that can be classified as major is the 57% out of the overall amount of 102 testsuite problems detected.

It is interesting to note the following in terms of potential costs saving.

Assumptions:

- Cost of correction of a software bug:
  - K at Software Validation Level,
  - 10K and System Integration level,
  - 100K at Final Satellite Qualification level,
  - 10000K at Mission Time.
- *P%* of the above mentioned major testsuite problems that causes a software problem going undetected at software validation level and is instead detected at System Integration (A%) or Final Satellite Qualification (B%), or Mission Time (C%)
- The cost to:
  - analyse the tools results,
  - implement the corrective actions for the testsuite major problems,
  - implement the corrective actions for the percentage of software problems newly detected.

  can be estimated as E1 + NxE2 + *P%*xNxK.

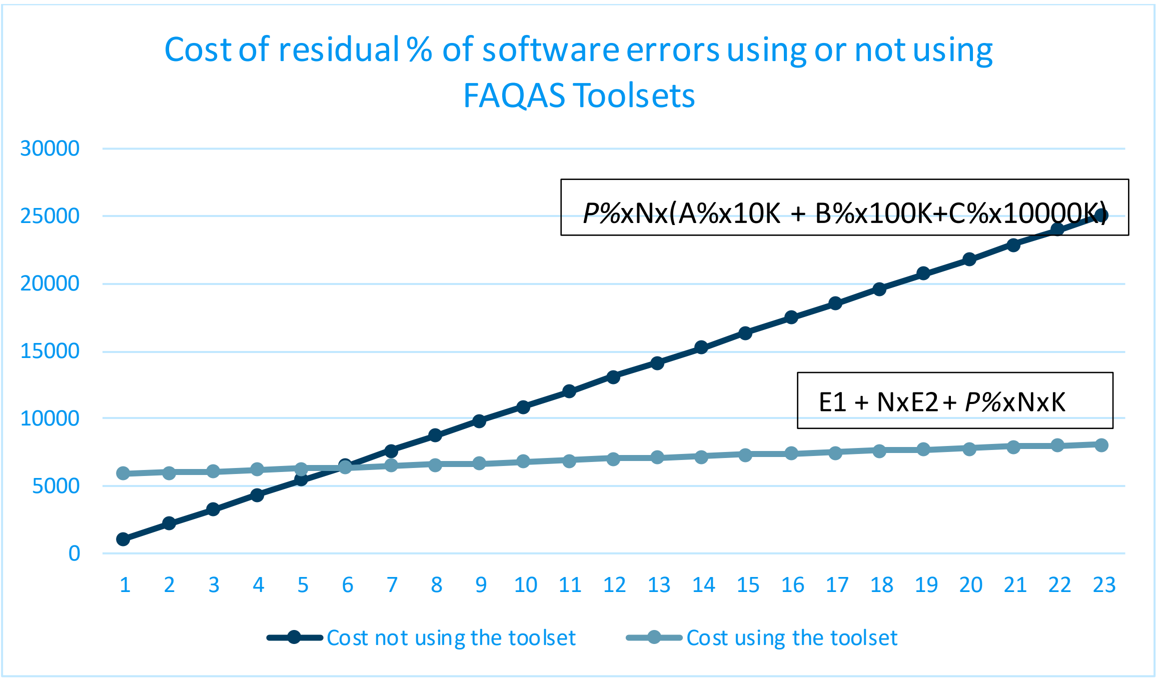
  Being:

  - E1: cost for analysing the toolset results
  - N the absolute number of major problems of the testsuite detected by the toolset.
  - E2: cost for implementing corrective actions to fix one major problem, identified by the toolset in the testsuite
- Convenience in terms of costs is when:
  - *P%*xNx(A%x10K + B%x100K+C%x10000K) > E1 + NxE2 + *P%*xNxK

Considering the following values for the identified parameters:

N	100
A	0.99
B	0.009999
C	0.000001
E1	800
E2	50
K	100

E.g.
the following chart shows that using the FAQAS toolset is cost effective when more than the 6% of the detected major problems of the testsuite could have determined a software error would have gone undetected.



Beyond the obvious observation that the value of the parameters, taken here into consideration, are not empirically validated, and that different values of them may lead to different results, an interesting aspect of the inequation is that it shows analytically where the toolset can be improved to increase its cost effectiveness:

$$P\%xNx(A\%x10K + B\%x100K+C\%x10000K) > E1 + NxE2 + P\%xNxK$$

Cost effectiveness increases when the value of the parametersE1 and E2 decrease.

The decreasing of the first parameter E1 depends on the capability of the toolset to decrease the time of analysis of the results (e.g. by providing results along with accurate root cause analysis considerations).

The decreasing of the paramenter E2 is instead clearly linked to the capability of the toolset to generate automatically additional/updated test cases that may "patch" the current failing testsuite.

Both these fields of research are considered from LuxSpace worth to be explored in a possible extension of the FAQAS project.

End of Document