Requirements Document

Client:

RDW

Authors:

M. Bartelsman,
C. Dadashov-Khandan,
H. Monhemius,
J.Faber

Teaching Assistant:

Marten Struijk

v2.20210612

Introduction

Faultify is a mutation testing tool that will be used by the testers and developers of RDW's ICT division. Our task is to further expand and improve this application.

Faultify was initially developed by a different student team as a proof of concept, one team member of which kept working on the project afterwards in an open-source manner.

Since RDW plans to use Faultify as a core part of their daily testing suite, they wish to have Faultify work more efficiently and reliably, with more control and customization granted to the user.

Requirements

Overview

Critical requirements

RM01 Faultify must be able to use MSTest as a test host
RM02 Faultify must include debug symbols along with its reports
RM03 Faultify must allow the user fine-grained control over the settings

Important requirements

RS01 Faultify should produce user-friendly errors when something fails
RS02 Faultify should be able to use NUnit as a test host
RS03 Faultify should generate working copies of the test in the system's temp folder

Useful requirements

RC01 Faultify could be able to use XUnit as a test host
RC02 Faultify could produce reports in JSON format
RC03 Faultify could have more extensive unit tests
RC04 Faultify could have better documentation

Rejected requirements

RW01 Faultify will not be integratable with Azure DevOps Reports
RW02 Faultify will not be automatically integrated with Azure DevOps Pipelines
RW03 Faultify will not optimize the type and number of mutations performed
RW04 Faultify will not be further optimized
RW05 Faultify will not run on .NET Framework codebases
RW06 Faultify could display an accurate estimate for the testing duration

Non-functional requirements

RNF01 Faultify must not be substantially slower than other similar tools
RNF02 Faultify must produce reports that are clear and informative

Critical Requirements

RM01 Faultify must be able to use MSTest as a test host

RDW writes unit tests for their codebases using the MSTest testing framework. As Faultify is naturally going to be used on said codebases, it needs to be able to work with MSTest codebases.

RM02 Faultify must include debug symbols along with its reports

When a mutation is performed, the report should show information such as line numbers, test name, method names and class names. This would improve the legibility and usability of test reports for the user.

RM03 Faultify must give the user fine-grained control over the settings

Faultify must allow the user to have control over:

- Input project directory
- Test host framework
- Report output type
- Report output path
- Timeout limit
- Basic control over mutation detail
- Fine-grained control over mutations via a config file

Important Requirements

RS01 Faultify should produce user-friendly errors when something fails

At times, Faultify produces raw stack traces and debug messages when it fails. At other times, it fails silently or ignores exceptions. What the user sees in the console is often not user friendly or is hard to track down.
The logging and displaying of messages should be modified so that error messages are easy to understand and actionable. Debug messages should be written directly to log files that do not interrupt the user experience of the program.

RS02 Faultify should be able to use NUnit as a test host

NUnit is among the most popular testing frameworks for C#. It is also partially used by RDW. As such, it is important that Faultify supports NUnit as a testing host.

RS03 Faultify should generate working copies of the test in the system's temp folder

Currently, Faultify produces a number of working copies of the compiled test projects, upon which it performs the various mutations and tests. However, these copies are generated in the project's folder. Since these are temporary files used by the current instance of the program, it would be more appropriate to place them in the system's temporary directory.

Useful Requirements

RC01 Faultify could be able to use XUnit as a test host

XUnit is a widely used testing framework for the .NET environment, but it is not a requirement for RDW. As such, support for the XUnit testing framework will only be implemented if all other more critical features have been implemented themselves.

RC02 Faultify could produce reports in JSON format

JSON is a widely used format for data exchange. It is also used as the format for a number of testing dashboards such as the one provided by Stryker. As such, it would be useful if report data can be provided in JSON format.

RC03 Faultify could have more extensive unit tests

While Faultify has got good coverage already in terms of unit tests, some parts of faultify could be more extensively tested

RC04 Faultify could have better documentation

Given that Faultify will likely need to be updated in the future, it is useful for Faultify to have adequate documentation, to support and guide the future developers. To this extent, there will be Program Documentation, as well as documents outlining the architecture.

Rejected requirements

RW01 Faultify must be integratable with Azure DevOps Reports

RDW asked that Faultify was integratable to the Azure DevOps Reports system. However, Microsoft did not provide us access to a build system we could use to integrate and test Faultify with. As such, we were unable to realize this requirement.

RW02 Faultify will not be automatically integrated with Azure DevOps Pipelines

Automated integration into the Azure DevOps Pipeline system was deemed too costly in terms of development time for the utility it provides. Manually adding a task into the Pipeline system is a matter of either including a few statements into a configuration file or using a web dashboard to do the same via a GUI. It is an unnecessary endeavor to automate such a simple process.

RW03 Faultify will not optimize the type and number of mutations performed

Currently, Faultify performs so few mutations to the compiled code, that further reducing this amount of mutations would result in a substantial decrease of effectiveness. Instead, we have opted to give control over the performed mutations to the user (see here).

RW04 Faultify will not be further optimized

Faultify's speed is defined by the tools it uses and the environments it runs on. Both of these aspects are beyond the scope of Faultify's development itself, therefore this task will not be carried out.

RW05 Faultify will not run on .NET Framework codebases

Faultify was not designed to operate on .NET Framework codebases, only .NET Core. Given that this has not been mentioned to be a priority, and that it would be an extremely expensive endeavor to implement, it will not be done.

RW06 Faultify could display an accurate estimate for the testing duration

Currently, Faultify displays a “worst-case scenario” time estimate before starting the mutation test runs. However, this estimate is inaccurate by orders of magnitude. Due to time constraints and low priority, this issue was not addressed and is instead logged as a known issue in the Architecture Document.

Non-Functional Requirements

RNF01 Faultify must not be substantially slower than other similar tools

Stryker, another mutation-testing tool for the .NET environment, was tested on Faultify's benchmark library and was able to test 3.7 mutations per second. Meanwhile, Faultify averages around 4 mutations a second.
Regardless of other changes, Faultify should remain in this same order of performance, if not better.

RNF02 Faultify must produce reports that are clear and informative

Currently, reports are either too primitive (PDFs show only summary statistics), or too complex (HTML shows each mutation along with a version of the original method and the mutated variant). Reports should be changed so that only information that is relevant to the user is displayed, and so that it is displayed in a simple way.

Traceability matrix

As stated in the Architecture Document:

> *Faultify's functionality has a complex nature that cannot be boiled down to a simple "Input -> Output" format that unit tests heavily rely on. For instance, to test Faultify's ability to run another codebase's tests, one must supply Faultify with a target test file and also use a lot of Faultify's packages in bulk without individually evaluating them.*
>
> *As a result, Faultify is impossible to fully test without creating bloated dummy set-up classes or using mocking frameworks. Both of these options were too resource and time expensive for the scope of the project. Therefore no new unit tests were written for our implemented requirements. Only the existing unit tests were updated and corrected, as they already provided good coverage of all code that is testable.*

For these reasons, we were unable to include a traceability matrix.

Meeting Log

A log of all meetings with the client and the decisions made there. Each requirement should have been discussed. This will help you with forgetful clients.

Meeting 0 - Week 07 (Introductory meeting)

- An overview of who RDW is and an introduction to our contacts in the organisation
    - René Tuinhout, Testing and Quality Department Manager.
    - Jasper van der Woude, Testing and Quality Manager, also the main supervisor of the project.
- An overview of the development environment that RDW uses. Specifically, they use the Microsoft Azure Stack.
- A broad introduction to the project, Faultify.
- It was decided that we would get in contact with the old team to get better guidance on how the project functions.
- We asked for the design documents for the project.
- Future meetings were arranged at 9 am on Thursdays

Meeting 1 - Week 08

- Initial primary requirements:
    - Improve speed of the library
    - Integrate with Azure DevOps
    - Improve mutations by using context cues and statistical analysis
    - Test on codebases focused on XML-based messaging
- Other potential improvements:
    - Integrate with web-reports
    - More control over configuration option
- Following a request from the team, Jasper plans to see if there is any software at RDW that would be usable as a testing target for Faultify.
- It was noted that RDW prioritizes the “functional correctness” of the application over “technical correctness”.

Meeting 2 - Week 09

- Each member shared their findings on their assigned research tasks
- For the Speed improvement research task, it was deduced that In-Memory testing fixes should be approached first
- The team decided to tackle Azure DevOps integration and In-Memory Testing fixes for the first sprint

Meeting 3 - Week 10

- Multiple proposals for mutation optimizations were listed off (noted in sprint 3 document)
- **From this point forward, sprints have been extended to 2 weeks due to the size of the issues the team must resolve. The client approved this change.**
- The client requests for MSTest-based testing to be implemented.
- A solution for XUnit and NUnit testing has been found and partially implemented, but needs more time to be completed.

Meeting 4 - Week 12

- The codebase has been deemed too low quality to comfortably implement the needed features.
- The team and the client both have decided that the next sprint should be dedicated to refactoring. The first week will be dedicated to investigating usefulness and feasibility, while the second week will either carry out the refactor or switch to implementing different features (e.g. MSTest and Azure DevOps)

Meeting 5 - Week 14

- The refactor was deemed useful and work was dedicated to it in the second half of the sprint
- The team considers the refactor to have been a worthwhile and successful

endeavour
- A small regression bug was run into with the Dotnet testing framework after the refactor, it will be worked on and corrected during the following sprint.
- The team now wishes to continue working on the high priority features, namely MSTest implementation and speed optimizations.
- The client states that they would like the application to have more detailed information on the mutations, e.g. which lines of code were mutated and how.
- The requested feature will be looked into during the sprint as well.

Meeting 6 - Week 16

- Faultify is not able to speed up much more, as its mutation strategy is already rather minimal and it has a very similar per-mutation speed as other popular mutation tools such as Stryker.
- The team has decided to primarily focus on tailoring Faultify to RDW's needs.
- The client states that they would like Faultify to be configurable, i.e. the user can specify which mutations to run and which ones to skip.
- Faultify has issues running on large codebases, so the error must be fixed.
- The team will focus on Azure Devops Integration

Meeting 7 - Week 18

- During the last sprint, the bug with Faultify not working with large codebases turned out to be an even bigger issue once all the other branches were merged, as another branch caused a regression where Faultify could no longer use dotnet tests at all. Debugging attempts were made during the entire sprint by Chingiz, but they failed. Miguel will take over the task for the next sprint.
- Faultify now has more detailed HTML reports with detailed line numbers and code location data. The next step is to integrate them into Azure DevOps.
- Faultify now has configurable mutations that can either be specified in the command line, or inserted as a JSON.

- PDF reports lack the detail that HTML reports have and they should be adjusted.
- Now that the reports are at the level desired by the client, they can be integrated into Azure DevOps
- **From this point forward, sprints have once again been reduced to one week, as the team now has a bulk of small features to implement, and only 3 weeks of development time left.**
- The next sprint meeting will only be an e-mail, as the client is on holiday for the week.

Meeting 8 - Week 19

During this week's meeting, the client was on holiday, thus we had a meeting among ourselves and informed the client of what was decided during the meeting.

- Azure DevOps integration might not be completed during the project time due to Microsoft's failure to provide pipeline access. Fortunately, Faultify should already be ready for integration with Azure, with the use of this extension to view the reports. It is merely a matter of having a pipeline to test and use to get this matter resolved.
- Faultify needs to be unit tested more extensively.
- Faultify should work in the system's temp folder instead of tampering with local build files.
- For the next sprint, we will be working on the unit tests and changing the files in which faultify works

Meeting 9 - Week 20

- We will be working on finding bugs that are currently in the system, and either fix or report them, to make sure we know of any bug that could not be fixed.
- For future maintenance to the codebase, we will make sure to properly document in any place that is currently lacking. Additionally, we will make an architecture document that gives an overview of Faultify and it's inner workings

- Finalize all the documents and reports that need to be made for the project

Meeting 10 - Week 21

- The team is working on resolving a bug where Faultify fails on one of RDW's proprietary codebases. The problem could not be resolved during the past sprint as it is difficult to reproduce the bug without being able to see the codebase due to company policy.
- The client is satisfied with the feature set of Faultify, therefore the team will entirely focus on polishing and bug-fixing for the rest of the project's development time.

Meeting 11 - Week 22

- The previous bug was caused by an environment issue, once a different environment was used the bug was no longer present. Instead, a different issue cropped up where Faultify fails to show the correct code for the mutated source. The team had been working on that and will continue working on it in the coming final sprint.
- A new issue was identified on the codebase where Faultify fails to resolve any tests under certain conditions, the team will examine that issue to see what can be done to address it.

Meeting 12 - Week 23 (Final Meeting)

- The team has done as much work as possible on fixing/identifying bugs. It will focus on finalizing the documentation for the last few days of the project.

Change Log

Who	**When**	**What**
M. Bartelsman, C. Dadashov-Khandan, H. Monhemius, J.Faber	Feb. 25, 2021	Creating the document and adding the first draft of the requirements
M. Bartelsman	March 10, 2021	Updated requirements to match with the current state of the project.
C. Dadashov-Khandan	March 10, 2021	Updated Meeting log to include meeting 2. updated critical requirements for clarity.
C. Dadashov-Khandan	March 16, 2021	Updated meeting log to include meeting 3.
C. Dadashov-Khandan	March 25, 2021	Updated meeting log to include meeting 4.
C.Dadashov-Khandan	April 16, 2021	Updated meeting log to include meeting 5.
M. Bartelsman, C. Dadashov-Khandan, H. Monhemius, J.Faber	April 21, 2021	Update the requirements document with readjusted milestones
M. Bartelsman	April 23, 2021	Expanded on various incomplete requirements
C.Dadashov-Khandan	May 8, 2021	Update Meeting log to include meetings 6 and 7
J. Faber	May 24, 2021	Update the Meeting log and requirements document to include meetings 8 and 9

C. Dadashov-Khandan, J Faber	May 30, 2021	Update the Meeting log to include meeting 10. Went over the requirements to rephrase where needed.
C. Dadashov-Khandan, J. Faber, M. Bartelsman Mejia	May 31, 2021	Adjust requirement priorities and polish the document.
M. Bartelsman Mejia	June 03, 2021	Update the Meeting log to include meeting 11.
Chingiz Dadashov-Khandan, Joran Faber, Miguel Bartelsman, Hessel Monhemius	June 12, 2021	Finalize Requirements document