STAR 2025
Software Testing Project

March 3 - April 11, 2025

1 Goals of this project

In this project you show that you are able to:

- set up a test architecture. You select appropriate test automation tools, and implement an adapter for automated testing.
- design tests and models for checking the functionality of a system under test. You select of tests and test generation techniques.
- implement tests and models for checking the functionality of a software system using a Behaviour-Driven Development tool, a(nother) automated testing tool, and a model-based testing tool.
- compare the test results of the different testing techniques and tools that you used.

2 Expected project results

At the end of the project you deliver a software testing artefact and a report.

2.1 Software testing artefact

The software testing artefact is a zip consisting of the following:

1. All scripts/programs that you developed yourself for your testing environment, e.g. as adapter or stub.
2. All files containing your tests and models; in particular, for BDD, this includes both the BDD scenarios and the step definition files.
3. All test results, preferably as test reports of your tools. If the tool does not produce test report files, then include screen shots showing all results.
4. Any helper scripts that you used for collecting and organizing your test results.
5. A short video for each of your 3 testing tools, to show how the tool executes tests on the SUT.

2.2 Report

The report describes the following:

1. Your testing environment(s) with your adapter(s) and other used tools and libraries
2. Your developed tests
3. Your model(s)
4. Test results, and your conclusions based on these results
5. A comparison of your chosen testing tools and their test results
6. The contributions of each of the team members: who did what?
7. References to e.g links to web pages of tools, or papers.

Write your report such that it well-motivated, yet concise and to the point. Make sure that the text is well readable and consistent throughout. Put figures, diagrams, pictures, etc. on separate pages, directly after the page with text where they are *referenced*. Put references at the end. The report may contain at most 8 *text* pages. Hence, pages with figures, diagrams, pictures, and references do not count for the page limit. Exceeding the page limit for text pages may result in a reduced/insufficient grade.

3 Plagiarism

You are not allowed to reuse material from others without proper referencing. In particular:

- All report text needs to be written by yourself. If you write your text based on external sources, e.g. for describing a used tool, then you need to include a clear reference. A clear reference is e.g. a web page of a tool or a reference to a paper. Do not use or refer to text from generative AI tools such as ChatGPT.
- You may reuse code snippets from external sources, such as the Internet, provided that you give a clear reference (e.g. URL) in your report. You are not allowed to use code from another STAR team. Sources for code, without reliable reference, e.g. from generative AI tools such as CoPilot or ChatGPT, are not allowed.
- You are allowed (and encouraged) to use existing tools and libraries, provided that a clear reference is given in the report.

4 System Under Test

The System Under Test for this software testing project is Axini's NewsFeed application. The application can be run using the .jar file. The test interaction with the SUT is to be performed through a socket connection with the application. The README describes this communication, and in particular the format

of the messages. The *specification* of the application is included as well. This specification describes the expected behaviour of the SUT.

The NewsFeed application contains implementations of several manufacturers, which can be selected after starting the application. In particular, the implementation from manufacturer 'Axini' is expected to have no bugs w.r.t. the specification, while the one from 'Inixa' contains multiple bugs. You should test the Inixia implementation for this project, and report on your test results for this implementation. You should only use the Axini implementation to debug your test or model without being hampered by bugs from the system.

Note that the functionality testing to be performed for this project should target testing the conformance of the application w.r.t. the specification. The communication interface of the SUT is out of scope, e.g. you do not need to check responses to not well-formed messages. If you notice that the communication interface has a bug, then send an email to `p.vandenbos@utwente.nl` to report on this bug. Make sure you include a log of both the messages sent by your client (your testing tool), and the messages sent by the NewsFeed server.

5 Project organization

You work in a team of 4 or 5 students. You may just continue with the team of the risk assessment project, but it is possible to change team for this software testing project.

The deadline for submitting the deliverables as listed in section 2 is:

Friday April 11 2025 at 23:59

6 Project tasks

Below it is described what you should do, and which questions you should answer for completing the software testing project. Make sure you incorporate this in the deliverables for this project, i.e. your report and software artefact.

6.1 Testing environment

1. Perform some manual tests to get familiar with the SUT (e.g. using `telnet`)

2. Search for a Behaviour-Driven Development tool (in a language of your choice). Make sure that the BDD tool support scenarios in the Given-When-Then style. Shortly describe the BDD tool, and also shortly motivate your choice.

3. Search for a tool or library that facilitates automatic execution of test cases written as programs/script (in a language of your choice). Shortly describe the automated testing tools or libraries that you chose, and also shortly motivate your choice.

4. Write an adapter for the SUT that can send and receive its messages. In particular, use a tool or library for dealing with JSON objects. Preferably your adapter works with both your BDD tool and automated testing tool.

Describe the tools or libraries you use for implementing your adapter, and shortly motivate your choices.

5. If you use any stubs or drivers, or other additional tooling, then also describe these, and shortly motivate your choices.
6. Provide a diagram in which you position the SUT, its interfaces, your adapter, and any, stubs, drivers, or other test tools.
7. Describe the architecture of your testing environment, referring to its diagram.
8. Describe any assumptions that you make. For example: assumptions on behaviour of the test environment, or assumptions on the execution of the SUT itself.

6.2 Behaviour-Driven Development

1. Design at least 10 Behaviour-Driven Development scenarios using your selected Behaviour-Driven Development tool.
2. Implement the corresponding tests as supported by your chosen BDD tool, e.g. by implementing the step definitions.
3. If you discover now that you need to adapt your testing environment, then make sure you update what you wrote for subsection 6.1.
4. Describe what you test with your Behaviour-Driven Development scenarios
5. Shortly motivate why you chose these for applying BDD
6. Describe any relevant details w.r.t. the scenarios' test implementation.
7. Describe the test results: which tests did pass or fail?
8. Draw some conclusions from the test results, i.e. describe any bugs that you found, or comment on the tests' successful execution. How confident are you about the correctness of your SUT for the tested functionalities?

6.3 Automated tests

1. Design and implement at least 10 automated tests, using your selected test automation tool. Make sure you test different aspects than were tested by the BDD scenarios.
2. If you discover now that you need to adapt your testing environment, then make sure you update what you wrote for subsection 6.1.
3. Describe your automated tests. What functionality do they cover? And how are they different from the BDD tests?
4. Describe the test results: which tests did pass or fail?
5. Draw some conclusions from the test results, i.e. describe any bugs that you found, or comment on the tests' successful execution. How confident are you about the correctness of your SUT for the tested functionalities?

6.4 Model-based testing

1. Implement an adapter for AMP. Try to reuse parts from the testing environment you already developed for BDD and automated testing.
2. Revisit your description of the testing environment of subsection 6.1. Write new descriptions for all parts that differ with respect to BDD and automated testing.
3. Identify the functionalities that you can model in AMP.
4. Write a small model. Then extend your model step by step, while executing tests with AMP after every extension.
5. Explain your final model (structure, components, actions, conditions, etc.). It may be helpful to include screen shots of parts of the graphical or textual representation to support your explanation.
6. Execute several test runs with AMP. You may also experiment with the configurations of AMP to obtain different test runs.
7. Describe the test results: which tests did pass or fail?
8. Also report on the achieved coverage as reported by AMP.
9. Draw some conclusions from the test results, i.e. describe any bugs that you found, or comment on the tests' successful execution. How confident are you about the correctness of your SUT for the tested functionalities?

6.5 Compare tools and test results

Evaluate and compare the testing tools and their test results. Consider (at least) the following aspects in your comparison:

1. Detected bugs by the tests
2. Confidence in the quality of the SUT based on test results
3. Test selection and generation methods enabled by the test tool
4. Ease of use of test tool
5. Effort needed to:
   (a) embed test tool in test environment
   (b) design and implement tests or model
   (c) to execute tests and obtain test results