Software Testing Report

|  |  |  |
| --- | --- | --- |
| Appetito Giulio  Brinati Anastasia |  | Blekinge Institute of Technology  Karlskrona, Sweden |

Project

# Abstract

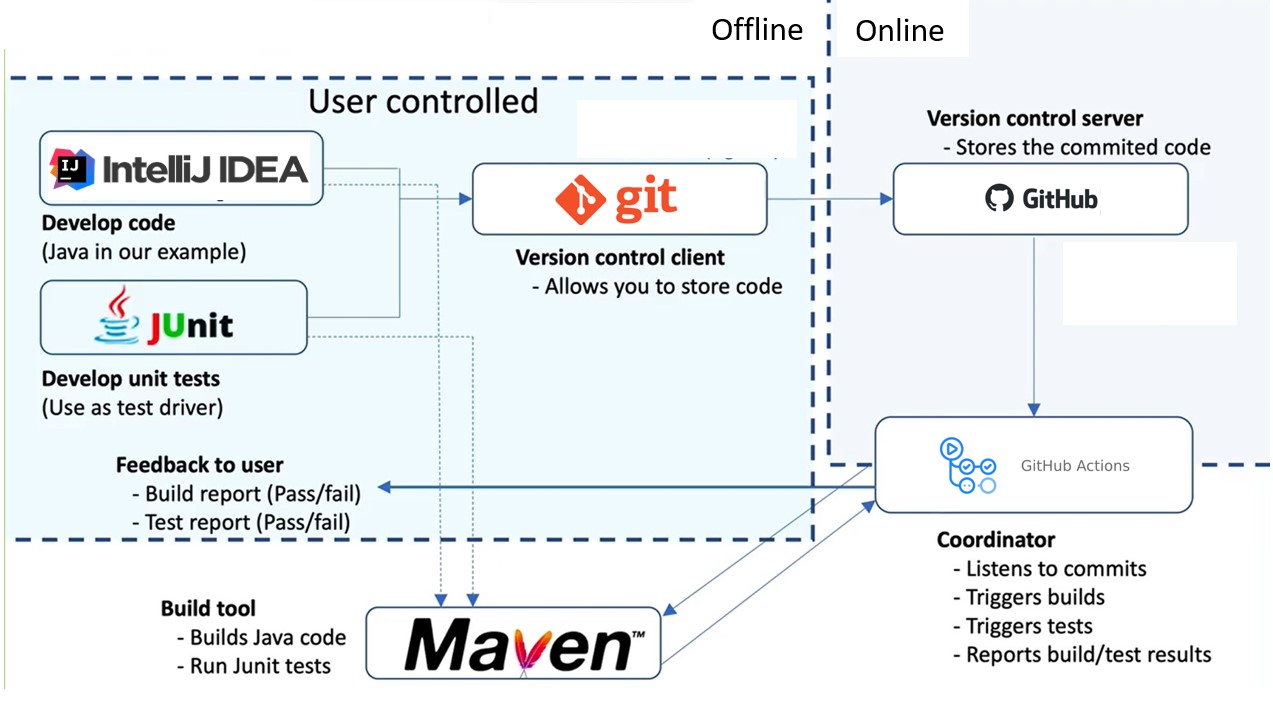
This report aims to present our group work for the final project for the course Software Testing. Starting with an overview of the structure of the system and the various adopted technicalities, we then continue with the reasonings and the challenges we faced during the development of the app, and lastly, we conclude with personal thoughts on the work.

# Introduction

As a group, we decided to re-adapt a project we worked on together for an assignment for a course in our bachelor’s. The system under test is a simple desktop GUI calculator developed using Java and JavaFX for the graphical interface, adopting MVC as an architectural pattern. The application consists of a graphic controller that administers the view (calculator. fxml), a main class to start the stage and panes, and then the Calculator.java, which is responsible for the business logic of the system.

# Technical choices of tools

We decided on IntelliJ IDEA as IDE, Junit to develop tests, Git as version control software, GitHub as a Version Control server, Maven as a build tool, and GitHub Actions as Coordinator. The choices for these tools were due mainly to our experience with using them, but also because in our opinion, the workflow comes out very smooth and it is very easy to just work alternating between an IDE and GitHub. JUnit was almost an obvious choice to develop tests since we developed our project in Java; moreover, we appreciated that IntelliJ IDEA has a lot of VCS commands, so we used the Git menu to Commit, Push and Update the project, allowing us to have a faster workflow directly from inside the IDE. Also, we decided to use GitHub Actions as the coordinator since it’s integrated into the GitHub environment, reducing the manual work to connect the Version Control Server to it.

The following picture shows the constellation of tools we used for the development. 

# Discussion

Intending to follow the Test Driven Development approach, we had to make some changes to the project: first of all, we started by creating a new JavaFX project in Intellij IDEA with a Maven archetype; then, we shared our project on GitHub using the Intellij IDEA feature ‘*Share Project on GitHub*’, to create a new repository on GitHub that we both could work on as collaborators. Once we had a common workspace, the following step was creating a new directory in the repository called ‘.github/workflows’, where we added the workflow file necessary for GitHub Actions to build and test the system. To run the build remotely we created the workflow ‘maven.yml’ that builds an Ubuntu environment with JDK 17 and executes the build till Maven’s “package” phase. The pom.xml contains all the dependencies and plugins necessary to run the application, like the maven-surefire-plugin for XXX and the JavaFX-maven-plugin.

To begin with, we created the ‘LogicTestSuite’ to run together all the tests used to verify the behavior of the SUT, meaning the four operations that should be available within the calculator. We wrote four test cases, one for each operation (addition, subtraction, multiplication, and division), that took into consideration simple cases, not considering borderline values, and from them, we originated the four relative stubs in the Calculator.java class. At this point, we committed the tests and waited for the results of the remote build, which of course failed because the tests failed, having the code had not been developed yet. Secondly, we went back to Calculator.java and implemented the four operations. After this commit, the result of the workflow was a successful build, with all the tests going green. Next, we went back to the tests and implemented a new one for the division operation, to take into account the division by zero. With this commit the remote build failed, presenting 4 successful tests and 1 test failure. We went back to the logic and added to the division method the ‘DivisionByZeroException’ that sends the message ‘Math Error’, and then again back to the test case to change the assert to check that the exception is thrown. Another commit and another build later we have a successful run.

After developing these unit tests, we developed integration tests to combine and test the different operations as a group. We added three test cases for each operation in its respective test class, with each one of the three testing that a certain operation worked properly when combined with another operation. As expected, after this commit we obtained a successful build.

Lastly, we tried to implement some GUI test cases using EyeAutomate in Java, but unfortunately we had a lot of problems to make it work, and we will talk about this in the challenges. The test case we tried to implement involved the operation ‘9999999999999999999999\*9999999999999999999999’, so that the calculator would display ‘infinity’ as the result. The idea was to firstly implement this test case, expect a failing build-run, and then add the View of the application using our old code. In the end we just added on the View without being able to implement any tests on it.

# Challenges and solutions

The first challenge we had to face was non-software related because Anastasia’s computer accidentally broke two days after starting the project, so unfortunately, she only managed to do some initial commits with her actual name on it; but, luckily at the moment we share our accommodation and so we were able to cooperate 100% on the project, even if using only one computer.

Another challenge we had to face was dealing with the Maven Surefire plugin, necessary to find and execute the JUnit tests in the project because the latest version of the plugin wasn’t able to find the required test suite. We solved this problem by modifying the plugin version inside the *pom.xml* and selecting an older one that managed to find the test suite.

To conclude, we had several problems with EyeAutomate, but this was also due to the fact that we worked on this project while on a trip organized by the European Student Network for Erasmus students, with only one computer, reducing ourselves to the last day to try and make the GUI testing work. Of course this was a problem that came out of our bad organization system and bad time scheduling skills.

# Personal experiences

Here we present our personal experiences with setting up and using the environment.

# Giulio’s experience

Setting up the environment was challenging in some parts since we had to make some research to solve issues that came up during the process. However, this allowed me to understand more deeply how the various components of the CI environment worked, how they cooperated, and why each one of them was fundamental to make the system run properly. After the setup, using the environment was quite easy, also thanks to the choice of tools. I think that this will be useful in the future, since in a future job it may not be necessary to set up a CI environment from scratch, but understanding how it works, how it’s composed, and why it’s important will help me in having more awareness while working on a project.

# Anastasia’s experience

In my opinion, I find that the most interesting part of the assignment was using the Test Driven Development, which I never used before because it helped me realize how much the features implemented depend on ‘the way they should be working in the end’ and not only how they are described in the requirements. The environment setup was something not completely new to me, but I learned a lot more about GitHub Actions, and I hope I will be using it again.

# Collaboration

[…]

# Conclusion

[…]

Link to the GitHub repository: <https://github.com/AnastasiaBrinati/SoftwareTesting_FinalProject.git>