1. Introduction: Enhancing Mutation Testing with the PiTest Plugin for IntelliJ IDE

In the ever-evolving landscape of software development, the pursuit of higher quality code is paramount. Testing methodologies play a pivotal role in achieving this goal, with Mutation Testing standing out as a powerful approach to assess the resilience of test suites. In this context, our project emerges with a mission and vision focused on advancing Mutation Testing capabilities within the popular IntelliJ IDE through a dedicated plugin that seamlessly integrates with PiTest.

2. Project Mission

Our mission is to enhance software mutation testing within the IntelliJ IDE by implementing a specifically designed plugin that integrates with PiTest. The approach involves several key steps:

Integration Development: We will develop a plugin that integrates with IntelliJ IDE, ensuring that PiTest's functionalities are easily accessible within the developer's primary workspace.

Dynamic Test Configuration: A core feature of our plugin will be to enable dynamic configuration of test scopes. This will allow developers to selectively fine-tune their testing efforts, focusing on specific classes or modules.

Result Visualization: The plugin will provide visualizations of Mutation Testing results. This will make it more comfortable for developers to interpret PiTest outputs.

User-Centric Design: The interface and functionality of the plugin will be designed with a strong focus on user experience, ensuring that it is both powerful and easy to use.

By following these steps, we aim to not only enhance PiTest’s functionality within IntelliJ IDE but also empower developers with more efficient, precise, and user-friendly software testing tools, ultimately leading to higher quality software development.

3. Project Vision

Software quality hinges on robust testing practices. While code coverage remains a prevalent metric, evaluating the true effectiveness of tests in ensuring expected behaviour often gets overlooked. This

is where Mutation Testing steps in—a method that generates code variations to evaluate the ability of tests to detect changes.

PiTest, a leading tool in Mutation Testing, falls short due to its limited integration capabilities. It lacks the functionality to display test run results and configure test scope dynamically, creating a gap in assessing test effectiveness within the environment best known to the developer.

Our product vision is to introduce an IntelliJ IDE plugin that not only presents PiTest results but also empowers users to seamlessly fine-tune test scopes, even down to specific classes. By integrating these features, we aim to bridge the existing gap, providing enhanced visibility and control within the familiar IntelliJ environment, thereby ensuring higher-quality test outcomes.

4. Software Architecture

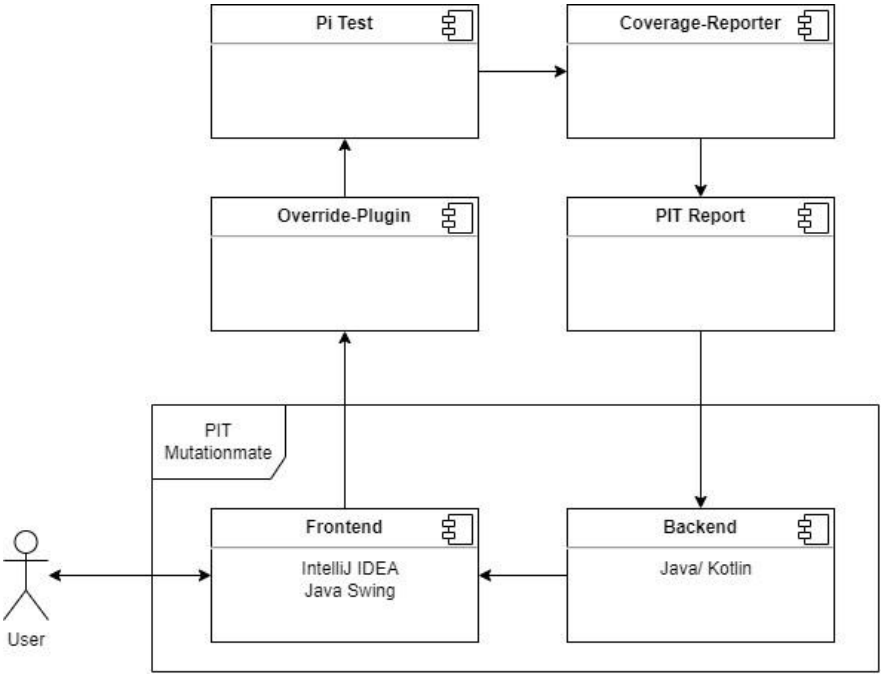
4.1 Definitions, Acronyms and Abbreviations

PiTest: PIT mutation testing

IDE: Intellij IDEA

4.2 Architecture Overview

4.2.1 Runtime Components



After installing our plugin, users will be able to configure mutation tests from within the IDE. This can be achieved through a separate window where users can specify values for verbosity, mutators, target tests, and more. Additionally, buttons will be provided within the code editor for initiating, re-running, and stopping the execution of PITest runs.

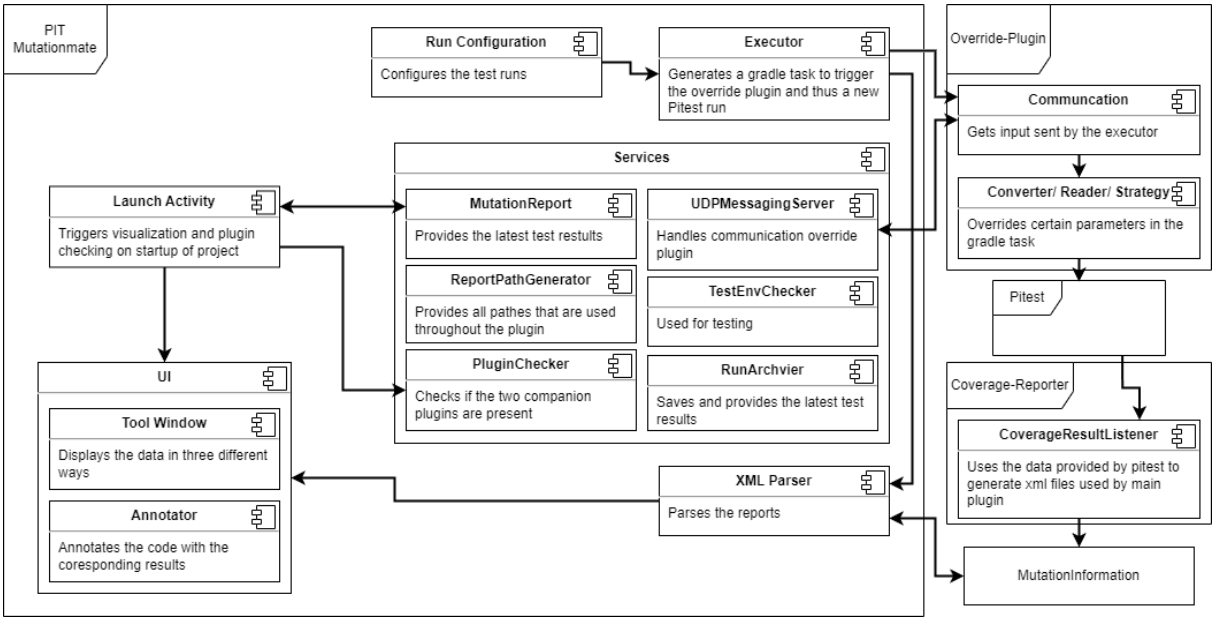
For the plugin implementation, we've chosen the IntelliJ Platform Plugin SDK, which provides all the infrastructure IDEs need to support rich language tools. The SDK uses Java Swing for front-end development. The front-end components communicate with the back-end, which is implemented in

Java/Kotlin, using Gradle. The choice of Java/Kotlin is based on the directory structure generated by the IntelliJ IDE plugin generator.

The user can start a PIT run either by clicking on the "Run" button in the IDE or by right-clicking on an appropriate class/package in their project folder and selecting "Run PIT MutationMate". This action will send either the previously set run configuration or a default configuration to the override plugin, which will then start a PITest run in the background. This PITest run then triggers the third plugin, the coverage reporter, which generates two XML reports containing all the information gathered by PITest.

Our plugin then parses the generated report to show line coverage, mutation coverage, and test strength. Passed tests are highlighted in green, while failed tests are highlighted in red. In addition, mutation test results for each line of code are presented as inline annotations.

4.2.2 Code Components



The TestConfigurator empowers users to configure and manage mutation test runs directly within the editor.

Our plugin orchestrates test runs by executing a Gradle task through the executor. This task triggers the Override plugin, which modifies specific aspects of the task and then initiates a new PITest run.

Upon completion of the PITest run, the coverage reporter plugin is activated, generating XML files essential for subsequent processing within our plugin.

The executor actively monitors the execution of the previously triggered task. Upon detection, it proceeds to initiate the XML parser.

The XML parser extracts crucial information from the generated files, preserving it for further analysis, and then activates the visualization component.

The results are seamlessly presented in the tool window for easy reference and analysis.

Additionally, upon opening a project in the IDE, the LaunchActivity automatically triggers the visualization of the latest PITest run results. Furthermore, it initiates the plugin checker to verify the presence of companion plugins.

5. Technology Stack

Programming Language: Kotlin. Java can be used if strictly necessary for certain tasks and no satisfying kotlin solution is available.

- Intellij IDE: The user will interface with the plugin through the Intellij IDEA IDE or Android Studio.
- Intellij Plugin SDK: main framework for development of the actual plugin
- PiTest: The targeted tool is the PIT Mutation Testing system.
- Gradle: as the build system
- PiTest-Gradle/Maven-Plugin: Setup by the user to integrate running their mutation tests into the build environment.
- Git/Github: for source code management & collaboration
- JUnit: as a test framework
- Frontend: The UI for the plugin will be developed using Swing or JavaFX
- Plugin dependencies: The developed plugin will have the following dependencies:
- Pitmutationmate-Override-Plugin: Allows the plugin to run a gradle task, which in turn will execute the PiTest mutation testing.

6. Team Members

Erben Emanuel

Nützel Felix

Heimbs Lennart

Böhm Luca

Malliaros Nikolaos

Herzig Tim Niklas

Fogarty Liam

Oberson Brianne

Dargel Olivia

6.1 Roles and Responsibilities

Product owners: Erben Emanuel, Nützel Felix

Software Developers: Heimbs Lennart, Böhm Luca, Malliaros Nikolaos, Herzig Tim Niklas, Fogarty Liam, Oberson Brianne

Scrum Master: Dargel Olivia