## 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 an 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 behavior 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.

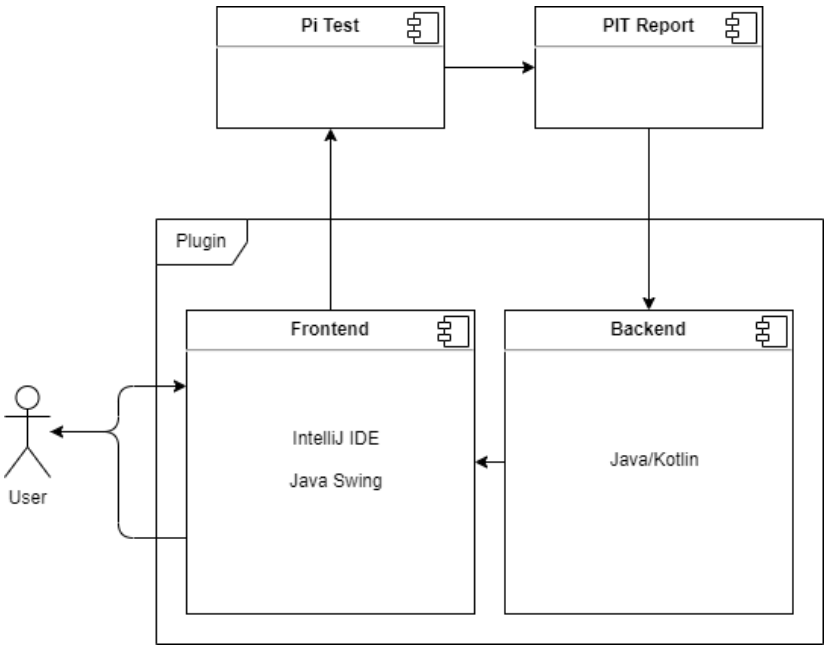
# 4. Software Architecture

## 4.1 Definitions, Acronyms and Abbreviations

PITest: PIT mutation testing

## 4.2 Architecture Overview

### 4.2.1 Runtime Components

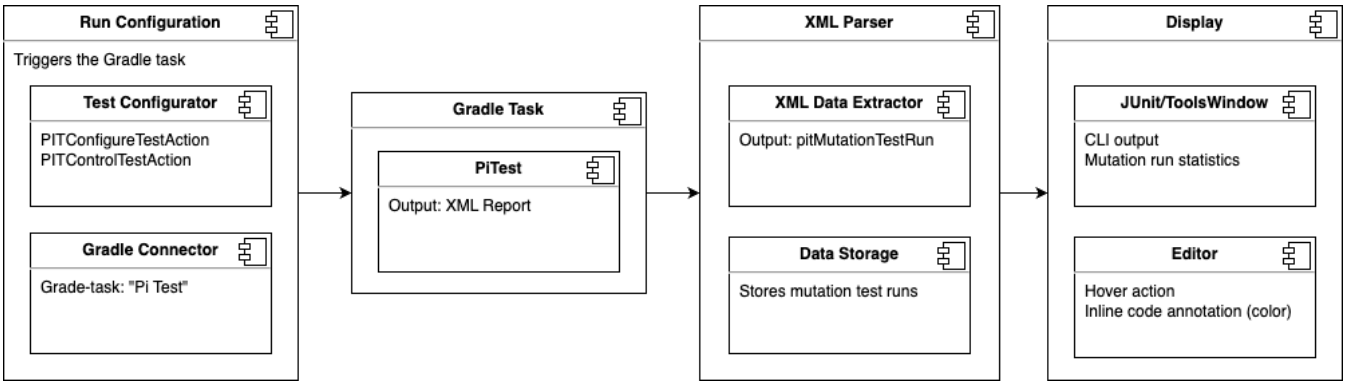


After installing our plugin, users gain the capability to configure mutation tests. This can be achieved by specifying values for verbosity, mutators, targeting tests, and more in a separate window. Additionally, within the code editor, buttons are provided for initiating, rerunning, and stopping the execution of the Pitest runs.

For the plugin implementation, we've opted the IntelliJ Platform Plugin SDK, which offers all the infrastructure that IDEs need to provide rich language tooling support. The SDK utilizes Java Swing for frontend development. The frontend components establish communication with the backend, which is implemented in Java/Kotlin, through Gradle. The choice of Java/Kotlin aligns with the directory structure that is generated by the IntelliJ IDE Plugin generator.

Upon clicking the "start" button, the Gradle-Task "pitest" is executed in the background. Our plugin parses the generated report to present Line Coverage, Mutation Coverage, and Test Strength. Passed tests are depicted in green, while failed tests are highlighted in red. Furthermore, mutation test results for each code line are presented as inline annotations.

### 4.2.2 Code Components




The TestConfigurator provides users in the editor with the ability to configure and manage mutation test runs. Our plugin will configure the test runs by editing the corresponding Gradle build file. The GradleConnector class enables us to establish a connection with Gradle, allowing us to execute the Gradle-Task "pitest". This execution will generate an XML report containing the results of the mutation tests. Our XMLParser will extract the necessary information and store it for further processing.

The results will be presented in the ToolsWindow as red and green bars. Additionally, we will print the results on the command line. When a user hovers over a specific code line in the editor, the results for that line will be displayed.

## 5. Technology Stack

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

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: GradleConnector: Allows the plugin to run a gradle task, which in turn will execute the pitest mutation testing. Alternatively Maven Invoker API if maven is used as a build system.

## 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