**Tool analysis**

Paper title: PIT: a practical mutation testing tool for Java

DOI: 10.1145/2931037.2948707

Team members: Buiga Andreea-Diana, Ceontea Vlad-Călin, Chirea Liviu

**Installation**

This tool requires Maven or Gradle. We used Maven. The installation consists of adding it as a dependency in the Maven configuration file. It requires a plugin to work with each testing framework. We used JUnit 5.

**Tool Usage**

Faults (or mutations) are automatically seeded into your code, then your tests are run. If your tests fail then the mutation is killed, if your tests pass then the mutation lived. The quality of your tests can be gauged from the percentage of mutations killed.  
An analysis is then done on the efficiency with which the test cases cover the lines of code, and whether or not they were able to find different types of mutations.  
These mutations are performed in a smart way, looking to bring to the surface partially tested code that has full coverage but would be easily broken by small changes to the code.  
The most effective way to use mutation testing is to run it frequently against only the code that has been changed.  
Once it has been integrated into the build file, it can be run locally by developers, or automatically against pull requests.  
A user has the ability to specify which classes to mutate and which test cases to run. The reports produced are in an easy to read format combining line coverage and mutation coverage information.

**Tool Findings**

- It allows us to build confidence in the codebase and test coverage

- Helps find untested code easily

- Helps find redundant code that, when mutated, still implements the functionality in the same way

- Offers an interesting alternative when it comes to testing that can improve the way we understand the code