

Project Software Reengineering

Robbe Claessens
robbe.claessens@student.uantwerpen.be

Tim Leys
tim.leys@student.uantwerpen.be

April 12, 2019

1 Introduction

2 Tool Choice

2.1 Tools for Duplicate Code Analysis

For the duplicate code analysis, the tools that were reviewed for this project are IntelliJ, iClones, DuDe, Clone Doctor and PMD CPD.

The IntelliJ integrated tool for duplicate code analysis analyzed code syntactically, but enables you to choose whether to anonymize local variable, fields, and method names. Which allows us to detect duplicates of type 3: structurally identical clones with gaps. The tool is also integrated in the IDE that was mainly used for this project. It allows for fast navigation through the source code, with text highlighting.

iClones is a research tool by the university of Bremen. In contrast to IntelliJ, the tool allows for more detailed configuration of finding near miss clones. In IntelliJ, could choose whether variable names or method invocations are anonymized, in iClones, we can determine the minimum of identical tokens that can be merged as near miss clones. The creators of iClones also provide a tool to visualize the code clones in a clear and understandable way.

Instead of looking at token similarity, like iClones, DuDe uses line similarity to discover clones. Because of this, the tool is language independent, but is also able to determine near-miss clones. DuDe features it's own GUI that clearly represents each of the clones found and where they are located in the source code.

	EMMA	JaCoCo	IJ IDEA	Clover
Line Coverage	71.2%	71%	71%	57.2% (statement)
Branch Coverage	46.7%	46%	/	48.7%
Method Coverage	72.1%	71%	71%	62.8%

Table 1: Test coverage according to EMMA

While iClones is a token based matcher, Clone Doctor inspects the abstract syntax tree of the analyzed code. The extra information we have by inspecting the abstract syntax tree might result in fewer false-positives, however, the results will be very similar to the token-based tools.

PMD offers a wide collection of static code analysis tools. The CPD or Copy Paste Detector can be used to discover code duplication. This tool is, just like iClones, a token based detector with similar workflow.

For the project, we will use the IntelliJ integrated tool for detecting the clones throughout the whole source code. Code clones that are an exact copy are the most important, since they are most vulnerable to the problems of duplicate code. For the classes that we determine to be feature hotspots (and thus are expected to change a lot), will be analyzed with a token based tool, to find more complex code clones. For this, we will use the iClones tool, because of our experience with the tool and the good visualization options.

2.2 Tools for Test Coverage

To select the tool for the test coverage, we selected four of the most widely used coverage tools for Java and compared their results. The tools that were reviewed for this project were: EMMA, JaCoCo, IntelliJ IDEA built in analyzer, and Clover. The results of each tool are presented in Table 1.

As seen in Table 1, most tools produce similar results, with the exception of Clover. Clover also does not feature line coverage, but features statement coverage. However, though we would expect similar results to the line coverage, it is much less.

The built-in tool of IntelliJ does not feature any way of showing the branch coverage and produced a different output in a previous run.

This left us with two remaining tools, EMMA and JaCoCo. Both tools are able to compute the line coverage, the branch coverage, and the method coverage and produce similar results. Because the JaCoCo tool is integrated in the IDE that was used in this project and is compatible with the maven site plugin, this was the best choice for the project.

	Coverage
Line	71.2%
Branch	46.7%
Method	72.1%

Table 2: Test coverage according to EMMA

	Coverage
Line	58.3%
Class	82.8%
Method	63.3%

Table 3: Test coverage according to IntelliJ IDEA

2.3 Tools for Mutation Tests

As for mutation tests, the available tools were LittleDarwin, PITest and Major Mutation Framework.

The Major Mutation Framework only works file per file or in combination with the Apache Ant tool. Since the current project is managed by Maven and other tools were available that worked out of the box, we chose not to look into this tool.

PITest seemed very promising, it is a mutation testing tool that is available as plugin for IntelliJ that promises to be much faster and much more efficient than competing tools. However, we were not able to make it work with the JFreeChart project.

Though Little Darwin is a research tool, it was the only tool that was able to work out of the box. It also provides a good and precise report of the coverage of each class in each package. Therefore, it is our preferred tool for mutation testing.

3 Test Coverage

3.1 Initial Test Coverage

	Coverage
Line	71%
Branch	46%
Method	71%
Class	89%

Table 4: Test coverage according to JaCoCo