An Evaluation of Automated Test Coverage Tools Project Proposal

CS 230 Software Engineering

Project url: https://github.com/PARanOiA1120/TestCoverage

Yuanqi Li 504759910
Haotian Xu 504778094
Hou Liu 904759994
Shun Cao 404705197
Yue Xin 204775695
Zhengliang Wu 204134580

Motivation

Test coverage is a crucial aspect of software maintenance in evaluating the effectiveness of the testing suite, and is also one of the important indicators of software quality. It is basically a measurement of how thoroughly the software is tested by telling which portion of the code has been executed by the testing suite. Test-coverage based tools are widely used in practice, but there are too many of them available in the market, and is thus difficult for developers or program managers to pick the tool that can best fit their requirements and testing needs. This project aims to examine several popular test coverage tools in the market from different aspects, compare and conclude the advantages and limitations for each.

Related Work

Test Coverage Definition. Test coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs.

Test Coverage Process. The test coverage analysis is generally divided into three tasks: code instrumentation, data gathering and code coverage analysis. Code instrumentation consists of inserting some additional code to measure coverage results. Data gathering means to store coverage data collected during test runtime. The code coverage analysis step is to analyze the collected results and provide test strategy recommendations in order to reduce, feed or modify the relevant test suite. [3]

Methods for Code Instrumentation. Instrumentation can be done at the source level in a separate pre-processing phase or at runtime by instrumenting bytecode [3]. *Source code instrumentation* adds instructions to the source code before compilation, and these instructions are used to trace which portion of the code have been executed. *Offline bytecode instrumentation* adds those same instructions, but after the compilation, directly into the bytecode. *On-the-fly bytecode instrumentation* adds instructions in the bytecode at runtime, when the bytecode is loaded by JVM.

Metrics of Test Coverage. *Line/ Statement/ Method/ Class/ Block coverage* measures the fraction of the total number of lines of code/ statements/ method/ classes/ blocks that have been executed by the test data. *Branch coverage* measures if each branch (true or false) of control structure has been executed by the test suite. *Path coverage* checks if every possible route through a given part of a code has been executed.

Test Coverage Tools.
EMMA is an open-source toolkit for measuring and reporting Java code coverage. It can instrument classes for coverage either offline or on-the-fly (using an instrumenting application classloader). It can support several test metrics for coverage: class, method, line and basic block. To give users a more visualized report, its output format include plain text, HTML, XML[9] .

JaCoCo is code coverage tool evolving and improving continuously. JaCoCo creates instrumented versions of the original class definitions. The instrumentation process happens on-the-fly during class loading using so called Java agents.[11] JaCoCo is integrated with eclipse and intellij well. It measures the coverage of classes, methods and lines as metrics of code coverage. It can output HTML coverage report for tested codes.

Clover is a code coverage tool developed by Atlassian, and it has recently become an open source project. Clover uses source code instrumentation for gathering its test coverage metrics. Clover provides three types of test coverage metrics: statement coverage, branch coverage and method coverage. There are multiple ways in which Clover can be adopted by developers. It can be used as either an Eclipse or Intellij plugin, or as an integration with a either Maven or Ant build tool. The test coverage results can be shown in the IDE, or as a generated report in chosen file formats including PDF and HTML.

JMockit [10] is an open source software with many APIs includes mocking, faking, integration testing and code coverage. In our study, we will focus on its function of code coverage. Its code coverage tool provides line coverage metrics and path coverage metrics which are calculated and shown at the source file, package, and global level, and displays them in the HTML report. There are two main differentiating factors for JMockit. One is that it generates the coverage report as easily as possible and without depending on special conditions or requiring specific plugins for Java IDEs or build tools. The other is that it provides newer and more sophisticated coverage metrics, such as path coverage and true line coverage.

Cobertura is an an open source code coverage tool. It is based on a older testing tool called Jcoverage. Cobertura does code coverage analysis by instrumenting the bytecode of original java classes. Then run test cases on those instrumented classes and generate reports of code coverage analysis in XML and HTML format. The code coverage analysis report contains line coverage and branch coverage data of each classes in the project. The report could also identify which line (and branch) has been covered and which line (and branch) has not by test cases.

Approach Description

We plan to evaluate five most popular Java test coverage tools and do the evaluation in two phases by using most starred open-source Java repository on Github and our customized project, accordingly. The five tools that we picked to evaluate are *Clover[6], JaCoCo[7], Cobertura[8], EMMA[9], JMockit[10],* and we have done proof-of-concept for each of them to make sure we can set up and run each tool properly.

Phase I:
We plan to start by comparing features supported by each tool, and then pick one or two (the number might change as we approach) open-source Java repositories with well-designed testing suite from Github. The next step is to run each repository with all five tools, and compare their results. The results produced by the five tools might be and should be different, so then we move forward to look into the mechanisms they used to measure code coverage. We might be able to explain some of the differences and conclude some advantages and limitations of each tool after this step, but those might be just our own assumptions. We are going to verify those assumptions in Phase II.

Phase II:
It might be hard to catch the exact differences of each tool by purely looking at the testing report returned by each tool in Phase I, so in this phase, we plan to create our own Java program along with its test cases to see if the five test-coverage tools can handle some special test cases, for example, parameterized tests.

Evaluation Plan

Test coverage is normally measured in terms of blocks, branches, lines, computation-uses, predicate-uses, and etc. that are covered by the testing suite. Not all tools provide measures in all of these metrics, so we are going to evaluate each tool based on the evaluation metrics they provided, which is, probably a subset of the following list:

1. line/ statement/ method/ class/ block/ branch/ path coverage percentage
2. tool running/analysis time
3. tool features/capabilities

Research questions for evaluation:

1. What mechanism does this tool use to measure test coverage?
2. What metric does this tool use to represent test-coverage?
3. How reliable is this tool in measuring the test coverage?
4. Are there any special test cases that the tool is failed to analyze?
5. Does the measured test coverage percentage correctly represent the thoroughness of testing?
6. How long does this tool take to run the analysis?

References

[1] Zhu, H., Hall, P.A.V. and May, J.H.R. (1997) Software Unit Test Coverage and Adequacy. ACM Comput. Surv., 29, 366–427

[2] Y.K. Malaiya, M.N. Li, J.M. Bieman and R. Karcich. Software Reliability Growth With Test Coverage. IEEE. Volume: 51, Issue: 4, Dec 2002. http://ieeexplore.ieee.org/abstract/document/1044339/

[3] Muhammad S. and Suhaimi I. (2011) An Evaluation of Test Coverage Tools in Software Testing. http://library.ndsu.edu/tools/dspace/load/?file=/repository/bitstream/handle/10365/23045/Alemerien_Evaluation%20of%20Software%20Testing%20Coverage%20Tools.pdf?sequence=1

[4] "Comparison Of Code Coverage Tools - Atlassian Documentation". N.p., 2017. Web. 1 May 2017. https://confluence.atlassian.com/clover/comparison-of-code-coverage-tools-681706101.html.

[5] Khalid Ali Alemerien, “Evaluation of Software Testing Coverage Tools: An Empirical Study”. https://pdfs.semanticscholar.org/fd48/e9a1decfb12fba13f9760538d74b6baa23fb.pdf

[6] JCover, http://www.mmsindia.com/JCover.html

[7] JaCoCo, http://www.jacoco.org/

[8] Cobertura, https://github.com/cobertura/cobertura

[9] Emma, http://emma.sourceforge.net/

[10] JMockit, http://jmockit.org/about.html

[11] Implementation of JaCoCo, http://www.eclemma.org/jacoco/trunk/doc/implementation.html