Project Proposal

Taishan Wang
txw171430

Mengru Wang
mxw171430

Tianrou Chang
txc172430

ABSTRACT

In software development, test automation is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes to predicted outcomes. Test Coverage is an important indicator of software quality and an essential part of software maintenance. It helps in evaluating the effectiveness of testing by providing data on different coverage items. Automated testing tools can be used to enhance the maintainability, testability and stability of the software. This project aims to provide an automated test coverage collection tools in software testing.

KEYWORDS

Automation test, ASM, Java agent, Junit Listener, Maven

1. INTRODUCTION

Software testing is considered now as an essential activity in software maintenance life cycle. It is a practice often used to determine and improve software quality. Where development is more systematic, organizations seek measures of testing completeness and goodness to establish test completion criteria.

Code coverage is one such measure. “Coverage is the extent that a structure has been exercised as a percentage of the items being covered. If coverage is not 100%, then more tests may be designed to test those items that were missed and therefore, increase coverage”. Test coverage can help in monitoring the quality of testing, and assist in directing the test generators to create tests that cover areas that have not been tested before.

Now we plan to build an automated coverage collection tool using ASM byte-code manipulation framework which can capture the statement coverage for the program under test.

The implementation details of automated coverage collection tool :

1.1 Phase 1:

(1) Use Java Agent to perform on-the-fly code instrumentation. Java Agent can be directly combined with ASM to do bytecode instrumentation on-the-fly. ASM defines several interfaces, such as ClassVisitor, FieldVisitor, MethodVisitor and AnnotationVisitor. The Core package of ASM follows the "Visitor" design pattern to walk through complex bytecode structures. These interfaces interact with each other to implement bytecode transformations and capture information from the bytecode.

(2) Store the coverage for each test method in the file system. In order to record the coverage information for each JUnit test method, we should use JUnit RunListener to capture the start and end events for each JUnit test method, means that it starts recording when the test starts and stops recording when the test ends.

(3) Integrated with the Maven build system so that the tool can be triggered by simply typing “mvn test” after changing the pom.xml file of the project under test.

(4) Output a file named “stmt-cov.txt” under the project under test while “mvn test” is firing.
The file should include statement coverage information for each test method.

Specification:
Each line in the file represents a test method or a covered statement, while the test method line will have the [TEST] prefix. Each test method should be followed by the set of statements covered by it (with no duplication). Each test method is represented by the full name of the test class +”:”+the test method name, while each statement is represented by the full name of the class +”:”+the line number.

(5) Generate a statics report for the time consuming from different aspects.
eg: time consuming by pre project, time consuming by per line of code.

1.2 Phase 2:

(6) Augment the tool to trace more information about program internal states: branch coverage, c-use and p-use coverage.

2. TECHNIQUES DETAILS

2.1 Java Agent

The code instrumentation way that we learnt in the class has to modify the bytecode file on the file system. There are several limitations for that. First, the time can be wasted when reading and writing back the bytecode files from/to file systems. Second, the bytecode file on the file systems may be changed into a mess. Therefore, Java Agent was proposed to change the bytecode file only in the memory during the JVM load time. Whenever, a Java bytecode file is loaded into JVM (for execution), Java Agent will transform that file and add instrumentations in the memory, leaving the actual bytecode file on file systems unchanged. There is a much cleaner and faster way to do code instrumentation. Java Agent can be directly combined with ASM to do bytecode instrumentation on-the-fly .

2.2 JUnit Listener

In order to record the coverage information for each JUnit test method, you need to capture
the start and end events for each JUnit test method, so that you can start recording when the
test starts and stop recording when the test ends. JUnit RunListener provides a direct way to do that. There are plenty of example online.

2.3 Integration with Maven

Maven is one of the most widely used build system for Java. It enables the users to build
Java projects easily without worrying about the 3rd-party libraries, since the Maven build system will automatically download the 3rd-party libraries based on the Maven configuration file (pom.xml). The “mvn test” command is used to execute all the JUnit tests for a Maven project. When firing the “mvn test” command, the Maven surefire plugin is used to find and execute all the JUnit tests. You can easily configure your Java Agent and JUnit listener for the surefire plugin in the Maven pom.xml file.

3. PLAN

phase 1:
Prerequisite Knowledge
Feb.14 - Feb.17
Official Document Understanding
Feb.18 - Feb.21
Create own automated coverage collection tool
Feb.22 - Mar.12
Conclusion and experiment analyse
Mar.12 - Mar.14

phase 2:
Improve tool in phase 1 to trace program variable values
Mar.15 - Apr.14
Test and improve tool accuracy and efficiency
Apr.14 - Apr.30
Conclusion and experiment analyse
May.1 - May.6

4 EVALUATION

The project will be evaluated on 10 real-world Java projects form Github. Each project has more than 1000 lines of code and is tested under more 50 JUnit tests to collect the statement coverage information. The two metrics of evaluation are the efficiency and the effectiveness.
The efficiency is how much time your tool consumes in collecting coverage for a project. We will generate a statics report for the time consuming from different aspects eg. pre project, per line of code.
The effectiveness is correctness of the coverage collected by your tool and how much information you traced.The common coverage items used include; Statement, Basic Block, Branch, Path, Conditions, Loop, Condition/ Decision, Modified, Condition/Decision (MCDC), Method or Function, Class, Package, Design and Requirement. Some other coverage types are Data Flow and Control Flow coverage, c-use and p-use coverage. In the phase 1, we are mainly focused on statement coverage. In the phase 2, we will add more coverage items including branch coverage, c-use and p-use coverage.

5 REFERENCES

[1] http://asm.ow2.org/
[2] https://github.com/
[3]https://docs.oracle.com/javase/7/docs/api/java/lang/instrument/package-summary.html
[4]https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
[5]http://www.tomsquest.com/blog/2014/01/intro-java-agent-and-bytecode-manipulation/
[6]http://memorynotfound.com/add-junit-listener-example/
[7]https://dzone.com/articles/junit-listener
[8]Shahid, Dr Muhammad & Ibrahim, Suhaimi. (2011). An Evaluation of Test Coverage Tools in Software Testing.