1. Team Information

<Team D>

<Baiyu Huo, 40076004, boyu.huo.china@gmail.com>

<Hao Ma, 40057767, mh2923166@gmail.com>

<Runsen Tian, 40083990, Rinotrs@gmail.com>

<Liangzhao Lin, 40085480, 283477489@qq.com>

2. Selected Metrics and Correlation analysis

Metric 1&2: Statement coverage and branch coverage

Bench coverage is how many branches from each decision point is executed at least once thereby the more coverage percent it shows, the more opportunity to find the existing bug.

Statement coverage is how many statements are executed at least once during the test and thereby the more coverage percent it shows, the more opportunity to find the existing bug.

Metric 3: Mutation Score

The number of mutants depends on the definition of mutation operators and the syntax/structure of the software.

Score = (Killed Mutants / Total number of Mutants) * 100

Metric 4: McCabe metric (Cyclomatic complexity)

McCabe measures the number of linearly independent paths that can be executed through a piece of source code.

E = the number of edges in CFG

N = the number of nodes in CFG

P = the number of connected components in CFG

D = is the number of control predicate (or decision) statements

For a single method or function, P is equal to 1

Cyclomatic Complexity = E – N + 2P

Or Cyclomatic Complexity = D + 1

Metric 5: Maintainability index

Maintainability Index is a software metric which measures how maintainable (easy to support and change) the source code is. The maintainability index is calculated as a factored formula consisting of Lines of Code, Cyclomatic Complexity and Halstead volume. The calculation method as follows:

First, we need to measure the following metrics from the source code:

V = Halstead Volume

G = Cyclomatic Complexity

LOC = count of source Lines of Code (SLOC)

CM = percent of lines of Comment (optional)

From these measurements the MI can be calculated:

The original formula:

MI = 171 - 5.2 * ln(V) - 0.23 * (G) - 16.2 * ln (LOC)

The derivative used by SEI is calculated as follows:

MI = 171 - 5.2 * log2(V) - 0.23 * G - 16.2 * log2 (LOC) + 50 * sin (sqrt (2.4 * CM))

The derivative used by Microsoft Visual Studio (since v2008) is calculated as follows:

MI = MAX (0, (171 - 5.2 * ln (Halstead Volume) - 0.23 * (Cyclomatic Complexity) - 16.2 * ln (Lines of Code)) *100 / 171)

In all derivatives of the formula, the most major factor in MI is Lines of Code, which effectiveness has been subjected to debate.

Metric 6: Fix Backlog and Backlog Management Index

Introduction: A metric to manage the backlog of open, unresolved, problems is the backlog management index (BMI). As a ratio of the number of closed, or solved, problems to the number of problem arrivals during the month. If BMI is less than 100, then the backlog increased. With enough data points, the techniques of control charting can be used to calculate the backlog management capability of the maintenance process. More investigation and analysis should be triggered when the value of BMI exceeds the control limits. A BMI trend chart or control chart should be examined together with trend charts of defect arrivals, defects fixed (closed), and the number of problems in the backlog.

$$\text{BMI} = \frac{\text{Number of problems closed during the month}}{\text{Number of problem arrivals during the month}} \times 100\%$$

3. Correlation Analysis

① **Correlation between Metric 1 and Metric 2:** The rationale is that test case with higher statement cover mostly lead to higher.

② **Correlation between Metric 1&2 and Metric 3:** The principle is that test suites with higher coverage might show better test suite effectiveness.

③ **Correlation between Metric 1&2 and Metric 4:** The classes with higher complexity are less likely to have high coverage test suites.

④ **Correlation between Metric 1&2 and Metric 6:** There is an example that illustrates that the classes with low test coverage always contain more bugs.

⑤ **Correlation between Metric 5 and Metric 6:** According to understand the meaning of Maintainability index (MI) and Backlog Management Index (BMI). It can be found that if the value of MI is high while the value of BMI should be high. The high value of BMI means the bugs of the software are easier to handle, and it also means that the software is easier to be maintained. Thus, followed by high-valued MI.

4. Related Work

Metrics 1&2: It can be collected by using tools like JaCoCo, JCov and CodeCover, and most projects that we choose contains the developers' test case, so it should be easy to collect and analyze.

Metrics 3: The mutation score is defined as the percentage of killed mutants with the total number of mutants. It mainly tests the efficiency of test suit, so we need to collect the developers' test suit from projects.

Metrics 4: McCabe is based on control flow graph, so we need to collect data like SLOC, the number of control predicate from the projects that we chose. To collect that data, we can use tools such as MCCABE IQ.

Metrics 5: To calculate the maintainability index, we need to collect data from other metrics such as metrics 1&2&4, because while calculating this index, line of codes, Halstead Volume, Cyclomatic Complexity and percent of lines of Comment are needed.

Metrics 6: It can be collected monthly about the figures that how many issues are opening and the number of unsolved bugs and resolved bugs.

5. Selected Open-Source Systems

① **Apache Log4j 2**

Log4j is a large open source project (more than 100K SLOC) which offers many versions for evolution-related metrics, and has the issue tracking system for defect-related metrics (Metric 5), what's more, according to the project structure, it is a maven project and it has a formal development document, which makes it easier to be build up and using JaCoCo tool to test the coverage of existing developers' test case (Metric 1&2). Therefore, it is a good project that allowed us to better conduct our empirical study. We plan to use from 2.0 to 2.11.1, totally 21 version to analysis this project.

② **Apache Cassandra**

Cassandra is a large project (more than 100K SLOC) which is used by many companies. And it builds up by using Ant. It has 8 versions (from 1.0 to 3.11) which we can use for evolution-related metrics and a perfect issue tracking system for calculating the number of post-release defects for Fix Backlog and Backlog Management Index (metrics 5). In addition, it has JaCoCo plugin to calculate the coverage of statement testing and branch testing (metrics 1&2).

③ **Apache Struts**

Struts is an Apache project which has more than 100K SLOC, bug tracking system for *Fix Backlog and Backlog Management Index* (Metrics 5) and many versions for us to analyze the version rev. We plan to use 4 version (from 2.3 – 2.5x) for the analysis of this project. It’s built up by Ant, and it is easier to get the branch coverage and statement coverage (metrics 1&2) by tools. And it contains the developers’ test suit.

④ **Apache bookkeeper**

Bookkeeper is a large project (more than 100k SLOC) offers a reliable replicated log service. We plan to use from 4.0 to 4.9, 10 versions for evolution-related metrics. It is a maven project and contains JaCoCo plugin which will be easier for us to test the coverage (metrics 1&2) of the existing developer’s test suit. What’s more, there is a bug tracking system on the website of this project, so it will be used for metrics 5. Above all, it is a good project for us to study and practice analysis a open source object and get an in-depth study in software measurement.

⑤ **Apache commons email**

Apache commons email is a project that offers the API for sending emails. It has many versions (from 1.0 to 1.5) for evolution-related metrics. In addition, there is a formal issue tracking system on this project website, so we can use it for defect-related metrics (metrics 5). It is built by maven and there are so many documents for us to tracking the problem and help us to solve the problem during the analysis work. It contains the developers’ test suit, which will be used for metrics 1&2&4. Therefore, It is a good project to conduct our empirical study.

6. Resource Planning

Name	Tasks
Baiyu Huo	Build up the source code; Coding the report script; Write mutant for metrics 3; Collect data for metrics 3 and metrics 4; Study statistics and exploring statistical tools, documentation.
Runsen Tian	Studying statistics and exploring statistical tools; Collecting data from issue tracking system for metrics 6; Documentation; Manage the resources of the whole team; Statistical data analysis.
Hao Ma	Studying about coverage testing tools; Collecting data and report by using Jacoco for matrics 1&2; Combine the result of metrics 1&2&4 to do the metrics 5; Documentation;
Liangzhao Lin	Studying about coverage testing tools; analysis the version resolution data; collect data for metrics 3; Documentation;

7. References

[1] M. D. Weiser, J. D. Gannon and P. R. McMullin, "COMPARISON OF STRUCTURAL TEST COVERAGE METRICS," IEEE Software, vol. 2, (2), pp. 80-85, 1985. Available: http://dx.doi.org/10.1109/MS.1985.230356.
DOI:10.1109/MS.1985.230356.

[2] Laura Inozemtseva and Reid Holmes. 2014. Coverage is not strongly correlated with test suite effectiveness. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014).ACM, New York, NY, USA, 435-445. DOI: https://doi.org/10.1145/2568225.2568271

[3] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2014. The impact of code review coverage and code review participation on software quality: a case study of the qt, VTK, and ITK projects. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). ACM, New York, NY, USA, 192-201. DOI: http://dx.doi.org/10.1145/2597073.2597076

[4] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014). ACM, New York, NY, USA, 654-665. DOI: https://doi.org/10.1145/2635868.2635929

[5] Luca Cardelli, Serge Abiteboul.2015. Software Quality Management, p75-77. https://www.tutorialspoint.com/software_quality_management/index.htm

[6] M. H. Moghadam and S. M. Babamir, "Mutation score evaluation in terms of object-oriented metrics," in 4th International Conference on Computer and Knowledge Engineering, ICCKE 2014, October 29, 2014 - October 30, 2014, Available: http://dx.doi.org/10.1109/ICCKE.2014.6993419. DOI: 10.1109/ICCKE.2014.6993419.

[7] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," IEEE Trans. Software Eng., vol. 17, (12), pp. 1284-1288, 1991. Available: http://dx.doi.org/10.1109/32.106988.DOI:10.1109/32.106988.

[8] H. Sun, "Knowledge for software quality control and measurement," in Proceedings of the 2011 International Conference on Business Computing and Global Informatization, 2011, Available: http://dx.doi.org/10.1109/BCGIn.2011.123. DOI: 10.1109/BCGIn.2011.123.

[9] M. M. Suleman Sarwar, S. Shahzad and I. Ahmad, "Cyclomatic complexity: The nesting problem," in 8th International Conference on Digital Information Management, ICDIM 2013, September 10, 2013 - September 12, 2013,
Available: http://dx.doi.org/10.1109/ICDIM.2013.6693981.
DOI: 10.1109/ICDIM.2013.6693981.

[10] Y. H. Yang, "Software quality management and ISO 9000 implementation," Industrial Management + Data Systems, vol. 101,
(7), pp. 329-38, 2001. Available: http://dx.doi.org/10.1108/EUM0000000005821. DOI: 10.1108/EUM0000000005821.