Test Suite Effectiveness: An Indicator for Open Source Software Quality

[1]Mamdouh Alenezi
College of Computer & Information Sciences
Prince Sultan University
Riyadh 11586, Saudi Arabia
malenezi@psu.edu.sa

[2]Mohammed Akour
Software Engineering Department
Yarmouk University
Irbid, Jordan
Mohammed.akour@yu.edu.jo

[3]Alaa Hussien
Computer Information System Department
Yarmouk University
Irbid, Jordan
alaa_84@ymail.com

[4]Mohammad Z. Al-Saad
Jordan University of Science and Technology
Irbid, Jordan
Mzalsaad15@cit.just.edu.jo

***Abstract*— Nowadays, open source software is playing a big role in several business contexts. Open source systems have moved from just educational projects to mainstream research area. Successful and active open source projects are numbering more than thousands that needs to be tested and their quality level needs to be determined. There are several ways to measure the quality of software systems. Mainly, measuring the ability of test cases to detect and find defects is used. This research focuses on finding a good technique to evaluate the effectiveness of test cases for finding defects in open source systems. Experiments are conducted on six OSS (open source software). The result shows how the effectiveness of test suite of could give an indication about studied systems quality.**

Keywords—Open Source Code;Test Coverage; Mutation Testing; Error Seeding; Software Quality

I. INTRODUCTION

Open source software systems have freely available source code that can be used by anyone. Some common eexamples of popular open source software are PureMVC [21], JoptSimple [2], IDOM [22] and Commons Codec [23]. One major factor that inhibits companies from adopting these open source systems is their quality level. The research described in this research paper focuses is to measure the quality of such systems and suggest an automated method to measure their quality.

An important of software development to assure the software quality and effectiveness is testing. In order to measure the effectiveness, the quality of test suites developed is measured. Good software testing have software tests than detect and find faults, if a test cases is unable to detect fault it will be hard to build a good software. [1] Defines two techniques to test the ability and effectiveness of test suites that are code coverage and fault injection. [9] and [10] used three techniques to measure the test case effectiveness, code coverage, error seeding, and mutation analysis. These three techniques are considered to be Fault Based Testing.

The main research objective is to determining which one of the aforementioned techniques is better than the others or combining some of them will yield a better result. Our research focuses on finding an empirical solution for finding the effectiveness of test suites and applying the mechanism on a selected open source system with their test cases, decide whether these systems are adequately tested or not, and generalize the results to open source systems. We will investigate and find the best mechanism for measuring the test case effectiveness as indicator of software quality. To do this, we will be using the available test cases that come with open source systems. The research will answer the question: Finding an effective technique to measure the quality and coverage of test cases? This research selects few open source system like PureMVC, Joptsimple, IDOM and Common Code as for testing their test cases effectiveness. To measure the effectiveness, we will use code coverage and mutation testing to test which is better for finding the effectiveness of test cases. The selected open source systems are based on different sizes (LOC) and different domains i.e. web, desktop, server, educational, business.

II. RELATED WORKS

David Schuler and Andreas Zeller [24] addressed the influence of the dynamic slice of covered statements on an oracle. In their experiments seven open-source projects show that checked coverage is a sure indicator for oracle quality and even more sensitive than mutation testing.

Mark Aberdour [25], described how fault detection by using black-box testing will not be done early and could reduced OSS quality. Moreover, he identified how testing

978-1-5090-4580-8/16/$31.00 ©2016 IEEE

effectiveness in finding errors will be improved by mixing three major types of testing in OSS projects (i.e., structured testing by the project team, regression test automation, and user testing).

Saifan et al [26], utilized meta-models for synchronizing test models with their corresponding structure model. When any change takes place in the component structure model of the system under test (reductive modification), component meta model will specify and transmit changes that should be taken to update the test model. To evaluate their proposed approach, they compare the effectiveness of reductive test suite in terms of the ability of finding errors with the re-test approach. The results show how test suite is reduced without negative impact on the effectiveness of the fault detection.

Pavneet et al. [27] studied two open source systems to measure the relationship of code coverage and its effectiveness in killing real bugs from the software systems. Randoop is used as a random test generation tool to generate test suites with varying levels of coverage and run them to analyze if the test suites can kill each of the real bugs or not. The result of the experiment shows that there is a tangible statistically significant correlation between code coverage and bug kill effectiveness, which have an indeed relationship with the software quality.

III. BACKGROUND

A. Code coverage Technique

Code coverage test is a systematic method that can help with monitoring the quality of testing, based on control flow of programs. Code coverage is considered as a white box testing technique. It's used for the assurances of the test case quality [1]. For code coverage there are multiple techniques like branch coverage, function or statement coverage and condition coverage can be used. Test coverage shows a percentage of how thoroughly a program is exercised by a given test suite. Code coverage or coverage is ability of test suits to cover aspects of code and its functions, Maximum coverage results in good software's.

Code coverage satisfaction doesn't necessarily mean that faults can be detected effectively. Actually, test cases that cover 100% code coverage may not find any fault. Code coverage only evaluates the test data generation aspect of the code while it doesn't cover faults for those fault-based techniques should be applied to find the faults. Fault-based techniques are usually applied with the aim of increasing the confidence that the developed code do not have a certain type of defects. Fault-based testing is an adequacy criterion that measures the ability of test cases in detecting injected faults.

B. Error seeding Technique

Error seeding is standard fault-based technique in which experienced developers inject faults in the source code. Error seeding is a manual process where defects i.e. "errors" are inserted by experienced developers into the source code to find the effectiveness of developed. Test cases by detecting the inserted defects. However, these injected faults are usually not reproducible. Furthermore, these injected faults highly dependent on the developer opinion and not generalizable for large systems. Error seeding is a laborious and time consuming activity and hence cannot be applied for large projects.

C. Mutation Testing Technique

Mutation testing is also referred by some as white box testing is an analysis technique to test the error seeding method in a systematic way to evaluate the quality of test suite. Mutation refers to creating of small modification of program and creating a mutant of original source code. Idea is to identify the mutant and kill it. Thus, detection of fault, when are introduced to assess the quality of test case [3]. For this the code that is mutated and one that is original are compared, if the results are similar the mutant remains alive and if detected the mutant is killed. Effectiveness of this technique is measured by amount of mutant that can killed and use case is said to be effective if kill maximum possible mutants. [5]. There are three types of mutation, first value mutation; it involves modifying the value of constants or variables. Second is statement mutation where a certain line is deleted or the orders of lines in the code are being changed. The third is decision mutation the conditional statements are modified [7].

IV. RESEARCH METHODOLOGY

A. Research Questions

As described the research focuses on evaluation of techniques for measuring the effectiveness of test cases suites. The research will focus on two things line coverage and mutation coverage. This research aims to answer the following research questions for large Java programs

1. *How can we measure the effectiveness of test cases of open source software systems?*
2. *What is the acceptable quality level of open source software systems?*

B. Research Workflow

Goal of this research is to evaluate the two techniques mutation and line coverage testing and finding an effective technique to measure the test suites quality. Following steps have been adapted to answer the research question following four main steps were adapted.

- Mechanism collection, the best mechanism to measure the effectiveness of test cases on finding and detecting defects in the source code will be collected. This part will be achieved by exploring several literature reviews, related works that have

been published in a credible conference or journal on how to measure the effectiveness of test cases. Then applying the recommended techniques to chosen open sources systems' test cases.

- Open source code collection, here we will select several systems with their available test cases and apply the best measure that we found in the first part into the selected systems.
- Analysis technique collection, here the qualitative data analysis techniques will be used to analyze the collected data that are extracted from a literature reviews and papers.
- Result collection, by comparing and contrasting the findings of the output of the tests we will be doing to open sources systems test cases.

Then, we will try to generalize the results to open source systems since we will apply the best techniques to open source systems testing. The results will tell us if these systems are adequately tested or not.

C. *Studied System*

For evaluation purposes, we collect six, different related domains, free open source applications that are implemented using Java programming language. Table 1 list the selected applications for research along with its no of test cases that made with each open source project and number of classes that each open source project have.

PureMVC is a well-known concept of building the applications as model, view and controller; it has 25 classes and test case suite of 26. Cinema is an open source application to handle the movie tickets and scheduling of movies, it is 6 no of classes with 212 test case suite. ApaCLi is an open source application to pass programs with command line options, it has 18 classes and 121 test cases. JOpt Simple is a way to pass command line options like javac to programs. It has 545 test cases for 62 classes. Common Codec contains different Base 64 and Hexadecimal encoders and decoders. It has 501 test cases suite for just 22 classes. Idom2 is an open source project for xml parsing for different platforms and it is based on libxml2, msxmmls to provide various interfaces and implementations. It has 28 classes and has a test case suite of 519.

Table 1: A Summary of Selected Open source Projects in java along with number of classes and test cases provided

No	Project Name	Number of Classes	NO. Test Case
1	ApaCLI	18	121
2	Cinema	6	212
3	PureMVC	25	26
4	Joptsimple	62	545
5	IDOM2	22	501
6	commons-codec-1.10	28	519

V. RESULT AND DISCUSSION

The following table 2 shows that mutation coverage of each selected open source project. The mutation coverage of each project varies from 0.01to 77% that means ranging from poor mutants to effective mutant.

Table 2: Showing Mutation coverage of selected project

No	Project Name	Mutation Coverage
1	ApaCLI	401/520 - 77%
2	Cinema	170/391 - 43%
3	PureMVC	81/133 - 61%
4	joptsimple	78/7821 - 0.01%
5	IDOM2	383/ 1913 - 0.2%
6	commons-codec-1.10	53/1488 -0.4

The details of mutation coverage can be seen in figure 1

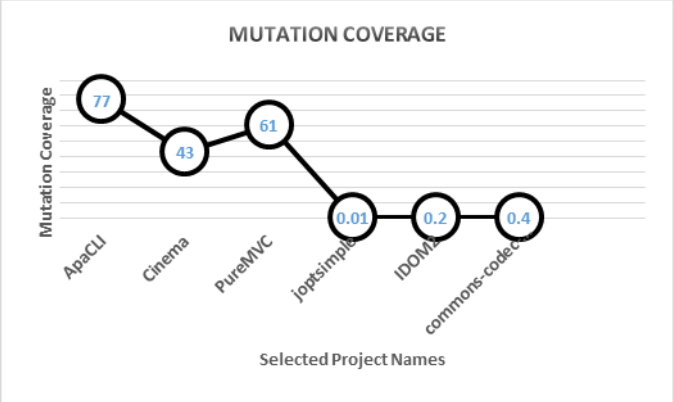

Figure 1: Showing the mutation coverage for each project

The calculation for above figure is total number of lines of code with the mutant coverage in actual. Mutant introduced were simple like logical or conditional. APICli has the maximum coverage of 77% mutation coverage that means 401 out of 520 variants were detected while PureMVC has 61% mutation coverage that were 81 out of 133 variants. Minimum is IDOM2, common codec and least is joptsimple that is able to detect only 78 variations out of 7821 variants.

The Table 3 shows the Line Coverage aspect of the selected open source projects. It shows that test cases of ApaCli are well designed while for IDOM2 are very poorly designed. In Figure 2 the Line coverage is shown for each project according to percentage that each line covers.

Table 3: Showing Line coverage of selected project

No	Project Name	Line Coverage
1	ApaCLI	770/880 - 88%
2	Cinema	410/532 - 77%
3	PureMVC	240/329 - 73%
4	joptsimple	182/1233 - 0.148%
5	IDOM2	193/2692 - 0.07%
6	commons-codec-1.10	84/1805 0.5

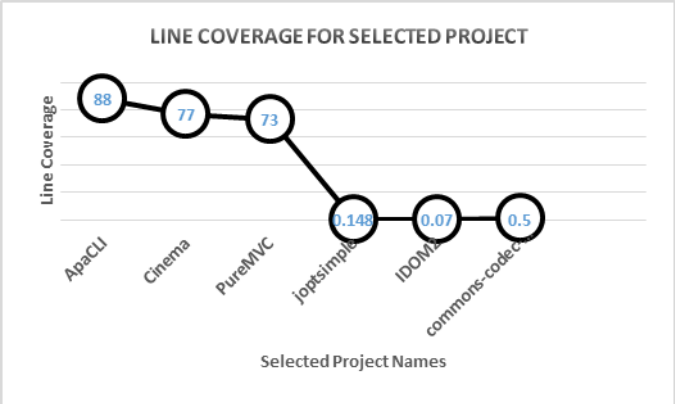

Figure 2: Showing Line Coverage for selected project

Lower the line coverage makes the test case suite bad quality, thus for the maximum line coverage it shows that test suites are of good quality. Like for the designed test cases of ApaCLI covers 88% of lines that are 770 lines out of 880, while Cinema covers 77% of lines. Joptsimple, IDOM2 and commons-codec are very poorly designed and thus are not providing any coverage.

Difference between line coverage and mutation coverage is that higher line coverage doesn't mean that the test cases are well defined and are capable of covering all the issues. When we inserted the mutant the line coverage of 77 for application cinema reduced to 43.

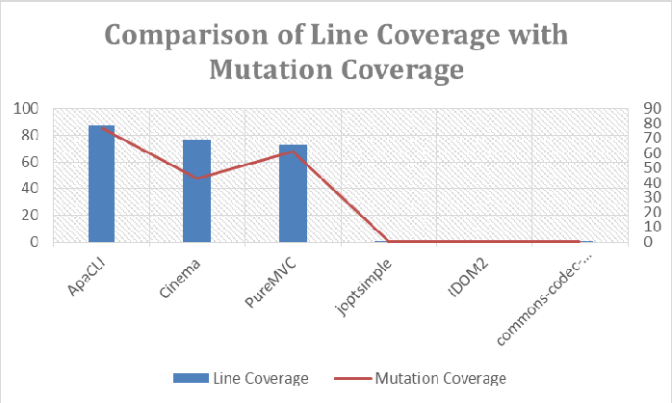

Figure 3: Showing comparison of Line Coverage and Mutation Coverage

As showing in Figure 3 the Better Line Coverage shown as line chart is not a comparative to mutation coverage. In terms of ApaCli the test are almost well designed as line coverage of 88% to 77%. But in case of Cinema the Line coverage of 77% is reduced to 48% that means of bad test suites designed. Thus mutation testing is a better way to design the test suites and it will allow better picture of test suite quality. Also in our opinion both testing techniques should be used as it can provide better picture in two directions. If the value of Line coverage and Mutation testing value are almost same with difference of 3 to 5%, the test suites and overall value of coverage exceeds 85% are well designed as can be seen in following equation

WD { Mutation Testing + 3% coverage >= Line Coverage }
where minimum value is above 85%.

WD is well designed test case suite and value is true or false, in our picked system the test fails so it is advised to design the test suites again to provide full coverage. Also the number of test cases doesn't refer to better line or mutation coverage as shown below in following table 3. As ApaCli have 88% line coverage and 77% mutation coverage but only no of test cases are 13.77%. While Joptsimple have so many test cases but the coverage is very low. It is seen that quality of test case is more important than more number of test cases.

Table 3: Showing total percentage of test cases with total line of code

No	Project Name	Test Case Comparison
1	ApaCLI	0.1375
2	Cinema	0.398496241
3	PureMVC	0.079027356
4	Joptsimple	0.442011354
5	IDOM2	0.186106984
6	commons-codec-1.10	0.287534626

VI. Conclusion

This research focuses on finding the best technique for finding the effectiveness of test case suite developed along with open source software. Two techniques Line code coverage and mutation testing were compared with each other; it is observed that better line coverage doesn't refer to better test suites or high quality of test suites. Mutation testing techniques allows inserting mutant in codes and comparing with old code, and can provide a better picture of coverage. We suggest that both techniques should be used in conjunction and as a comparative to each other as it can provide a comparative and actual

picture of where the test suites are lacking and for that area a new test case can be developed. And also it doesn't mean that if the numbers of test cases are more the better coverage in terms of mutation of line will be there. It depends on how test suites are developed thus an extreme care should be made while developing the test cases.

REFERENCES

[1] Ammann, P., & Offutt, J. (2008). Introduction to software testing. New York: Cambridge University Press.

[2] Delahaye, M., & Bousquet, L. (2013). A Comparison of Mutation Analysis Tools for Java. 2013 13th International Conference on Quality Software.

[3] Just, R., Jalali, D., Inozemtseva, L., Ernst, M., Holmes, R., & Fraser, G. (2014). Are mutants a valid substitute for real faults in software testing?.Proceedings Of The 22Nd ACM SIGSOFT International Symposium On Foundations Of Software Engineering - FSE 2014. doi:10.1145/2635868.2635929

[4] Kochhar, P., Thung, F., & Lo, D. (2015). Code coverage and test suite effectiveness: Empirical study with real bugs in large systems. 2015 IEEE 22Nd International Conference On Software Analysis, Evolution, And Reengineering (SANER). doi:10.1109/saner.2015.7081877

[5] Nayyar, Z., Rafique, N., Hashmi, N., Rashid, N., & Awan, S. (2015, July). Analyzing test case quality with mutation testing approach. In Science and Information Conference (SAI), 2015 (pp. 902-905). IEEE.

[6] Tengeri, D., Beszedes, A., Gergely, T., Vidacs, L., Havas, D., & Gyimothy, T. (2015, April). Beyond code coverage—An approach for test suite assessment and improvement. In Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on (pp. 1-7). IEEE.

[7] Young, M. (2008). Software testing and analysis: process, principles, and techniques. John Wiley & Sons.

[8] Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012). Free/Libre open-source software development. CSUR ACM Comput. Surv. ACM Computing Surveys, 1-35.

[9] Mathur, A. (2013). Chapter 7: Test Adequacy Assessment Using Control Flow and Data Flow. In Foundations of Software Testing. Delhi: Pearson Education.

[10] Mathur, A. (2013). Chapter 8: Test Adequacy Assessment Using Program Mutation. In Foundations of Software Testing. Delhi: Pearson Education.

[11] Just, R., Schweiggert, F., & Kapfhammer, G. M. (2011, November). MAJOR: An efficient and extensible tool for mutation analysis in a Java compiler. InProceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (pp. 612-615). IEEE Computer Society.

[12] DeKoenigsberg, G. (2008, April). How successful open source projects work, and how and why to introduce students to the open source world. In Software Engineering Education and Training, 2008. CSEET'08. IEEE 21st Conference on (pp. 274-276). IEEE.

[13] Khanjani, A., & Sulaiman, R. (2011, March). The process of quality assurance under open source software development. In Computers & Informatics (ISCI), 2011 IEEE Symposium on (pp. 548-552). IEEE

[14] Ghani, K. (2009). Searching for test data (Doctoral dissertation, Doctoral dissertation, University of York).

[15] Schuler, D., & Zeller, A. (2013). Checked coverage: an indicator for oracle quality. Software Testing, Verification and Reliability, 23(7), 531-551

[16] Segura, S., Hierons, R. M., Benavides, D., & Ruiz-Cortés, A. (2011). Mutation testing on an object-oriented framework: An experience report. Information and Software Technology, 53(10), 1124-1136

[17] Inozemtseva, L. M. M. (2012). Predicting test suite effectiveness for java programs.

[18] Untch, R. H., Offutt, A. J., & Harrold, M. J. (1993, August). Mutation analysis using mutant schemata. In ACM SIGSOFT Software Engineering Notes (Vol. 18, No. 3, pp. 139-148). ACM.

[19] Ma, Y. S., Offutt, J., & Kwon, Y. R. (2005). MuJava: an automated class mutation system. Software Testing, Verification and Reliability, 15(2), 97-133.

[20] Gligoric, M., Groce, A., Zhang, C., Sharma, R., Alipour, M. A., & Marinov, D. (2013, July). Comparing non-adequate test suites using coverage criteria. In Proceedings of the 2013 International Symposium on Software Testing and Analysis (pp. 302-313). ACM.

[21] Hall, Clifford. "PureMVC Framework Goals & Benefits." (2009).

[22] Nicol, Gavin, et al. "Document Object Model (DOM) level 3 core specification." W3C Working Draft 13 (2001): 1-146.

[23] Fraser, Gordon, and Andreas Zeller. "Mutation-driven generation of unit tests and oracles." IEEE Transactions on Software Engineering 38.2 (2012): 278-292.

[24] Schuler, David, and Andreas Zeller. "Checked coverage: an indicator for oracle quality." Software Testing, Verification and Reliability 23, no. 7 (2013): 531-551.

[25] Mark Aberdour, Achieving Quality in Open Source Software, IEEE software 2007

[26] Saifan, Ahmad A., Mohammed Akour, Iyad Alazzam, and Feras Hanandeh. "Regression Test-Selection Technique Using Component Model Based Modification: Code to Test Traceability." International Journal of Advanced Computer Science & Applications 1, no. 7 (2016): 88-92.

[27] Kochhar, Pavneet Singh, Ferdian Thung, and David Lo. "Code coverage and test suite effectiveness: Empirical study with real bugs in large systems." In2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 560-564. IEEE, 2015.