2020 6th Conference on Data Science and Machine Learning Applications (CDMA)

Empirical Study of Software Test Suite Evolution

Wajdi Aljedaani
Department of Computer Technology
Al-Kharj College of Technology
Al-Kharj, Saudi Arabia
waljedaani@tvtc.gov.sa

Yasir Javed
Department of Computer Science
Prince Sultan University
Riyadh, Saudi Arabia
yjaved@psu.edu.sa

***Abstract*—In a variety of market environments, open-source software plays a major role these days. Open-source systems have expanded to the research area from only academic projects. There are more than thousands of successful and effective open source projects to be checked and their level of performance requires to be calculated. The reliability of software systems can be measured in several respects. Essentially, the ability to detect and locate flaws in test cases is measured. This research aims to identify a good technique for evaluating the efficacy of test cases in open source systems to identify defects. This research study focused on six OSS projects (Open Source Software) publicly available. This study tends to find a relationship between software code suites and test code suites in terms of software evolution. It is seen from results that test suite is becoming enriched to have a better code coverage that directly relates to awareness about writing better test cases. The complexity of software as depicted in the result is still in-fancy as only a marginal change of less than 2 percent has occurred.**

***Index Terms*—Test suites, Code changes, Test suite evolution, Source code evolution, Code coverage, Software evolution**

I. Introduction

Software existence is mainly dependant on its evolution that addresses bug fixing, new functionalities or even graphics. These evolution may occur due to software requirements from the changes in the government rules, environment, and customer that in end result in new software code. These software changes because of software evolution make the older version usually obsolete thus requiring fixing in test cases to reflect new changes. Usually in open-source software these changes are very frequent making it challenging for tested to keep-up with valid test cases. Accordingly, the current test-suites becomes obsolete for the new version of the software. Therefore, the tester must revise all changes on the code to repair the corresponding test cases in the test suite that is time challenging and time consuming process. Unfortunately, only a few recent studies were investigated how the test suites evolve automatically. Most of these studies are preliminary studies, where the other studies are limited to a specific task.

The software development and maintenance process is not a static process where the software system has many versions during its lifetime, and each version has a set of test cases. Each new version of a software system is upgraded from the old version that is usually supported by code refactoring. This means the test cases must evolve according to code changes. Generally, if the test case failed on a new version of a software system, then there is a problem with the new code or in the test case itself. The developer either fix the problem in the code or fix the problem in the broken test case. If broken test case covers valid functionality, then broken test case must be repaired. Otherwise, the test case deleted from the test suite.

The main objective of this analysis is to consider the evolution of the test suite over time. Therefore, twelve (12) versions of the six open-source Java systems were used to explore different aspects of the development of the test-suites to achieve this objective. These systems have been chosen according to many popular systems size criteria, each system is equipped with 20 versions while each version have its own test-suite built in JUnit. All of selected system for this empirical study are available in GitHub[1]. Table II lists each used systems in this study along with the relative information.

Specifically, the study examined three research questions as follows:

RQ1: *What is the relationship between source code size and test suite size during software evolution?*

Usually, source code size and always higher than test suite size and over the period of time often it is taken as the code size will increase with time of evolution and so is the test suite size. This research addresses the relationship of test suite size and code size within different a version.

RQ2: *What is the relationship between source code complexity and test suite complexity during software evolution?*

Cyclomatic complexity is one of means of measuring software complexity whereas desire is to lower cyclomatic complexity to have easier testing that will also result in higher code coverage.

RQ3: *What is the effectiveness of test suite using code coverage during software evolution?*

In order to measure the code coverage effectiveness that is related to test code coverage, this study has measure the ratio of coverage between code suite and test suite in terms of interfaces coverage. This study is conducted to check whether the effectiveness of the test suite is evolved over different version or not.

The key contributions of the papers are:

1) This research performed an empirical study about test-suite compared to code suite evolution on the major

[1] GitHub: https://github.com/

978-1-7281-2746-0/20/$31.00 ©2020 IEEE
DOI 10.1109/CDMA47397.2020.00021

open-source system in order to address the effectiveness of test-suite in software development.
2) An extensive experimental study on 12-versions for each system has been conducted to formulate the key conclusion about the findings.
3) An extensive literature review is provided to support this research study.

The remainder of the paper is structured as follows. Section II provides information about the context. Section III shows the adapted approach to case study, data gathering and processing. In Section IV research questions are presented and discussed. Section VI discusses related work, and in second last section threats to validity are explained. Eventually, in Section VIII, conclusions to this study has been presented.

TABLE I: Definition of some terminologies.

Name	Definition
Test case	One test in a suite of tests and executes as a unit It is whether executed or not
Test suite	Collection of test cases
Code coverage	Calculation of how much a software source code is executed when a specific test set is executed
Master suite	The entire test suite written by the software developers
Metric	Measurement of software quality characteristics and their development process
LOC	The Line Of Code used for size evaluation of a software or codebase.
NOM	The Number Of Methods is defined as the total number of methods within the project package
NOC	Number of classes is count the declared classes within the Project package

II. Background

This section explores the background information of test suite systems and six subject systems of this study. Table I, describes several terms that will be used throughout the paper before explaining the methodology in depth. Table I shows the basic definition of test cases and test suite that refers to the collection of test cases. This study focuses on code coverage that is coverage of source code whenever a test-case is executed. This study selects the average code coverage for all test cases.

A. Overview of Subject programs

This study was performed on six open-source projects. These projects have been selected based on the number of criteria. First selected criteria required projects be relatively large (ordering 100,000 SLOCs) written and actively built-in Java in order to ensure the novelty and generalizability of this research in terms of team members that refers to contributors that actively participating in terms of software development for a particular project. As it can be seen in Table II that most numbers of contributors are in PMD project and GSON that also make a close relationship of better test code coverage for these project. It also shows the number of issues and commits that have occurred for a project showing how active is the development community for solving any issues and this also guided selection of the project. In terms of releases and forks, Table II presents at least 40 releases have been made for each project while this study considered the last 12-versions to have a quick picture of improvement in terms of test-suites addition to software projects.

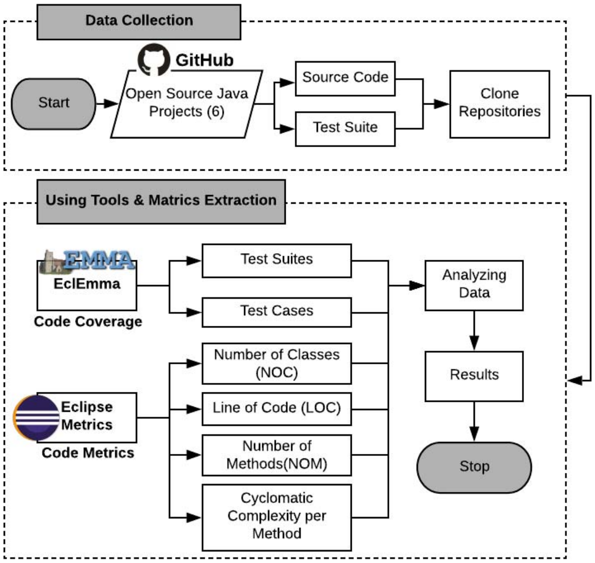

Fig. 1: Overview of Study Approach.

B. Overview of Test Suite

When the test suite generated it rarely remains the same, but it evolves with the software. The developers always reuse the test suite to save time and effort, however, need to repair it. Sometimes the developer needs to both correct test cases that are invalidated by the software changes, and add new ones to test new functionality. Repairing the test suite manually is expensive and time-consuming, which has motivated in development of techniques for test repair. Automation of test repair techniques understanding the software evolution as well as how the repairing of test-suites work.

III. Study Design

A. Study Approach

This section describes the design of the study on the test suite evolution on six well-known open-source Java projects. Figure 1 gives a description of the study approach. First source code is extracted and test suite data from GitHub repository for each case study system and used two different tools in the study. This research used EclEmma[8] for code coverage and Eclipse Metrics[9] for code metrics. For code coverage tool, This research used EclEmma to analyse the test suites and test cases files. In addition, code metrics are used to analyse

[8] EclEmma: https://www.eclemma.org/
[9] Eclipse Metrics: https://marketplace.eclipse.org/content/eclipse-metrics

Program	Description	#Releases	#Forks	#Stars	#Issues	#Commits	#Contributors
Gson[2]	It is a Java library open source for serializing and deserializing Java objects to JSON	40	3.3K	16,766	1,211	1,474	98
IzPack[3]	It is commonly used as a cross platform installer for packaging applications on the Java platform	75	243	221	0	5,603	77
JodaTime[4]	It is the widely used Java date and time class replacement before Java SE 8	76	842	4,249	369	2,139	74
PMD[5]	It is the analyzer of the source code. It finds typical programming weaknesses such as unused variables	71	890	2,539	282	15,485	168
OGNL[6]	It is uses simpler expressions than the full range of Java language-supported expressions.	62	105	50	996	432	18
Struts[7]	It is an open source web application platform for Java EE Software applications development	115	651	973	0	5,697	43

TABLE II: Software Systems with their contribution, releases, issues and no of commits.

four software quality metrics which are number of classes (NOC), line of code (LOC), number of methods (NOM), and cyclomatic complexity per method. Selected metrics were evaluated in order to identify possible differences between the test suites and metrics of all software versions for each project. In the approach subsections of each research question, the specific metrics analyzed in this paper are described in detail (see Section IV).

IV. Study Results

This section describes the findings of the research and explains them. First the question is presented, the approach to the study, the results of the analysis for each research question, and then discuss the findings made during the study.

RQ1: *What is the relationship between source code size and test suite size during software evolution??*

Motivation. The basic motivation is to find the relationship of code evolution over the different version that can help in getting knowledge about code evolution as well as how test cases evolve with different versions.

Approach. In order to measure the relationship between source code size and test suite size this study considered NOM, Classes, LOC and NOC for all mentioned project by finding their evolution for both actual project and test suite over different version.

Results and Discussion. It can be seen from results that usually there is not much of code added into code suite or test suites and in even in some cases the code may be removed as can be seen for PMO project. Only for GSON the code size has been increased but there is no effect on test suite size as shown in Fig 2. This study also looked into the relationship of classes with methods shown in Fig 4, as well as LOC over methods with classes and LOC as an average and found test suite code has higher average as compared to code per class except in case of OGNL and IzPack.

Summary. Thus it can be stated that usually, the test suite remains the same as compared code that may vary in some cases but mostly it is only coded that increase over a period of time compared to NOM that usually remains same.

RQ2: *What is the relationship between source code complexity and test suite complexity during software evolution?*

Motivation. The basic motivation about the relationship between both complexities is to check the effect of increased software principals recognition in terms of different versions.

Approach. In order to measure the relationship of test suite this study considered cyclomatic complexity of methods for both code and test suite.

Results and Discussion. It is evident from Fig 3, that mostly the cyclomatic complexity is changed minor and has little effect with the time of software evolution. But this effect is better direction as cyclomatic complexity is reduced in minor that is around 1.1 percent in case of Struts and OGNL. In two cases the complexity has increased over different versions such as GSON and IzPack where the complexity increased 1.9 and 1.6 percent respectively.

Summary. Thus usually the cyclomatic complexity changes very minimally that + - 2 percent over different versions.

RQ3: *What is the effectiveness of test suite using code coverage during software evolution?*

Motivation. The basic motivation about coverage of code during software evolution is to check whether over a period of time focus has shifted to test case suite. It will also highlight the actions needs to be taken in order to have a decision on software test case suite.

Approach. In order to measure the effectiveness of test suite this study considered interfaces and it has evolved over different versions. **Results and Discussion.** It is shown in Table III, that mostly the interfaces have not changed over different versions of software and even in some cases they have reduced such as GSON. The ratio shows maximum coverage is for PMD that is around 87.94 percent compared to worst case that is JodaTime that has only 0.36 percent interface ratio. Another version selected also provide a between coverage thus a proper coverage is provided within test cases.

Summary. In most of the software suites, code coverage to test suite coverage has evolved over time

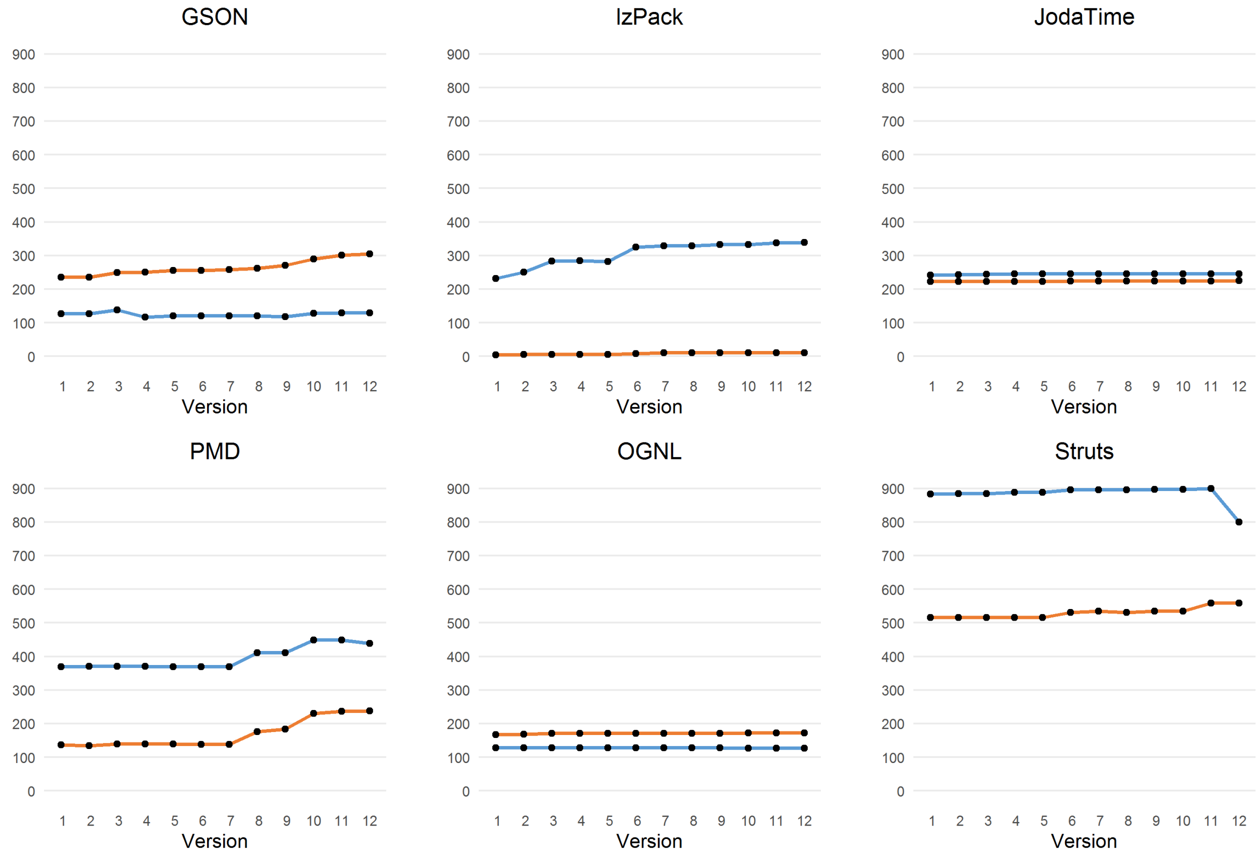

Fig. 2: Test suite code coverage compared to source code coverage with last 12 versions.

TABLE III: Statistical Data of RQ3.

Projects	Ratio Between Code & Test	Avg. Code Coverage
GSON	5:3	82.87%
IZPack	1:0	3.22%
Joda Time	1:0	0.36%
PMD	11:3.58	87.94%
OGNL	19:15	68.52%
Struts	1:17	48.68%

V. Discussion and Observations

There are numbers of studies conducted in the area of software evolution while awareness about test suites and test code coverage is a hot research topic where number of studies reveals the need for higher test code coverage. Designing the test suite is a challenging job as it requires high coverage and lowers complexity in order to mitigate the feeling of burden on development. It is observed from results that test cases does not evolve with software evolution in terms of methods. But checking the code size for both test suite and code coverage evolves over time but usually, they are directly relational in terms of code addition that is shown through code coverage that almost remains constant. It is also seen that a small effort has started in terms of reducing software complexity but not more than 2 percent. It is also observed that usually, coverage in terms of the test suite is quite significant that in terms of GSON and PMD but other selected systems are quite low in terms of IzPack but still every system coverage improved. This shows the awareness about writing test code suites that should provide better coverage.

VI. Related Work

The industry has steadily emphasized application testing and represents now half the total software development expense. As one of the major problems of software testing, many researchers have studied extensively how to reduce the sizes of the test suite while still satisfying the original test specifications. Most studies relate to the test case; the majority of papers relate to the level of automated test case [2] [3], generating test suite [16], effectiveness of test suite [1], assessment of test case effectiveness [21], and test suite reduction Approaches [12]. There are a few studies with the same goal of the research but in a different area of study. For example, the evolution of bug reports [18], keyword-driven test suites [19], and Android test suites [20]. In this section, related work are introduced on various aspects of test suite studies.

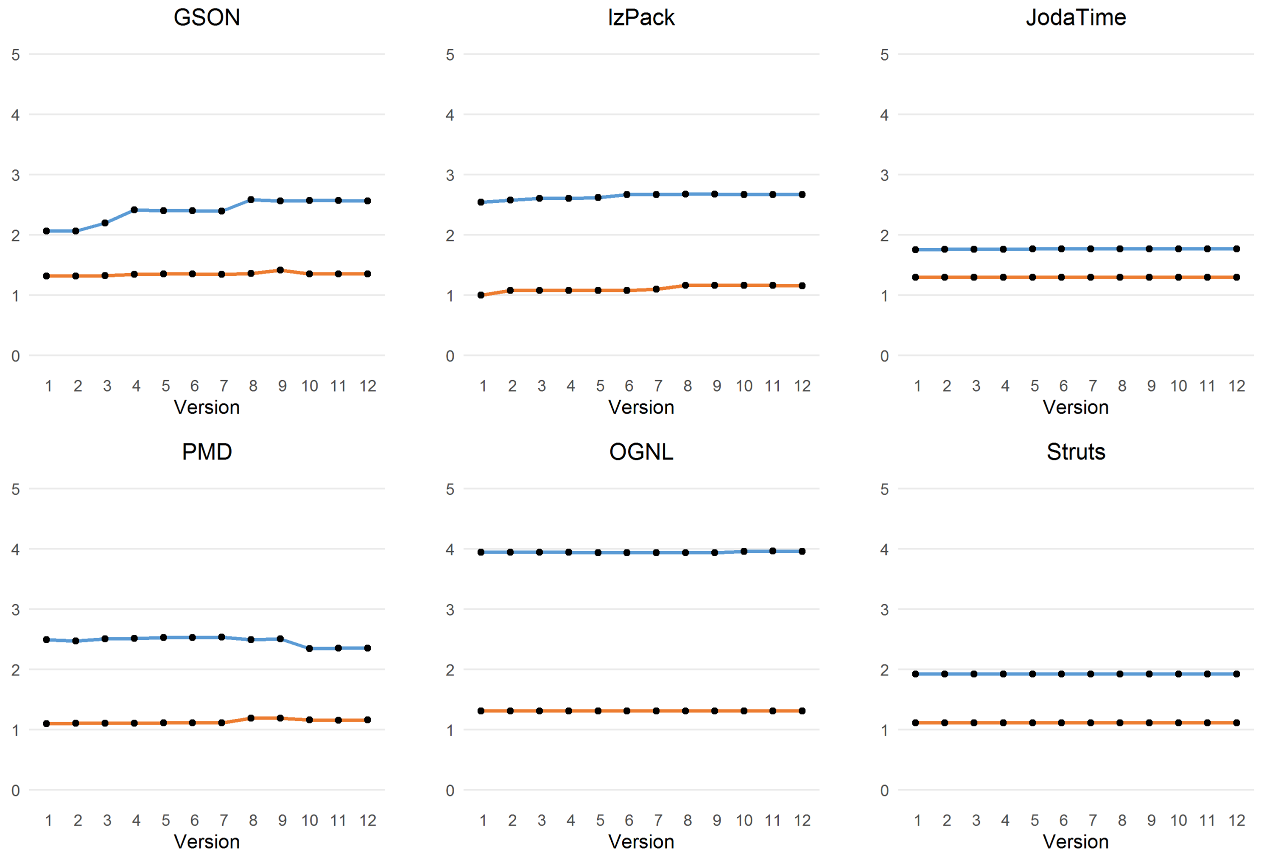

Fig. 3: Cyclometic Complexity of Methods for six systems and their evolution in differnet version.

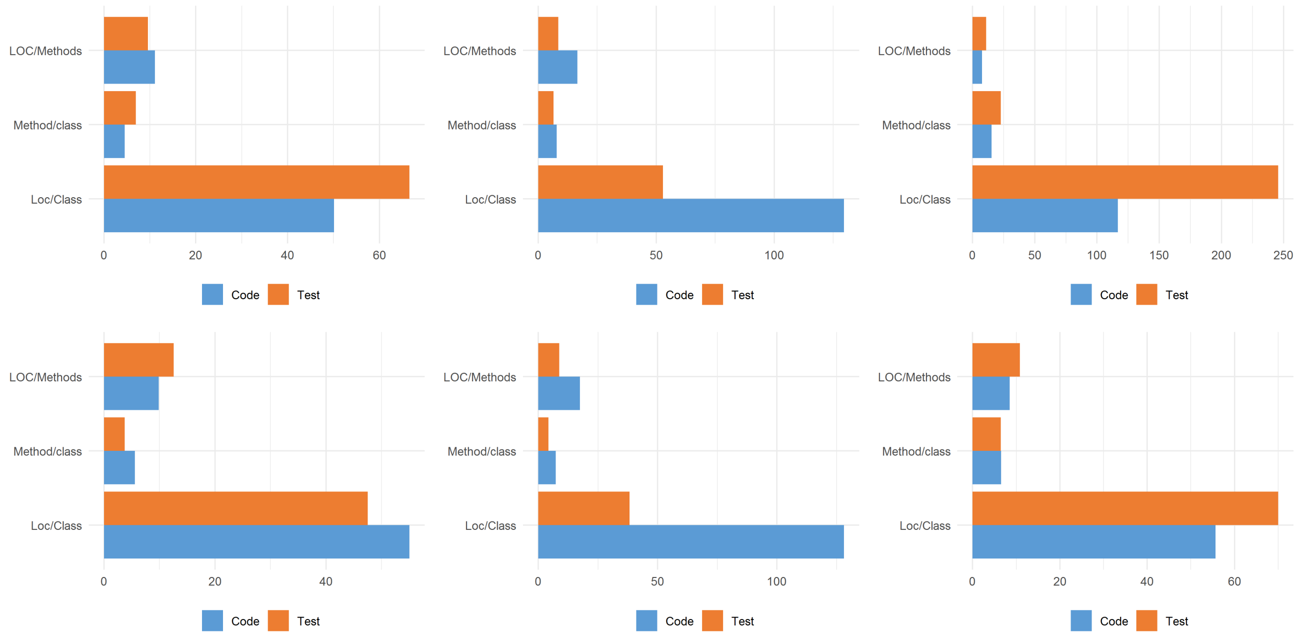

Fig. 4: Comparison of test suite and source code coverage in terms of LOC, NOM and NOC.

Elbaum et al. [4] studied the code coverage effect on different version of software's. Their study shows that small changes in software code can provide in-depth information about code usage, and this effect grows quickly with increasing levels of change. The great change indicates a greater impact on the quality of the coverage information. The impact of changes in coverage information can be difficult to predict.

Kochhar et al. [8] studied over 300 large open-source projects to measuring code coverage of their associated test cases. The findings show that the majority of projects were low coverage levels, with an average of 41.96%. However, with the increasing size and complexity, the coverage rate reduces, however, at the file level it increases in both the size and the complexity of the file. Many open source developers do not test their code sufficiently. There is a need for additional tools and techniques to help developers increase coverage.

Mishra et al. [11] compared various coverage model and presented the best practices for unit testing. Authors also examine the factors that affect code coverage and concluded that lines of code have a weak linear correlation to coverage.

Fraser et al. [5] [6] used a novel approach which called "whole test suite generation" and covered all coverage goal at the same time with small test suite as possible. Start with randomly generated population test suite then used a Genetic algorithm to optimize toward satisfying a chosen coverage criterion, the fitness of individuals is measured with respect to an entire coverage criterion. Implemented this novel approach in the EvoSuite tool [7], and compared it to the common approach of addressing one coverage goal at a time. The papers show EvoSuite tool is excelled over a traditional approach that covers a single goal at a time.

Pinto et al. [14] examined a total of 88 versions of program and proposed a technique to study test suites evolution. Their study is based on above 14K tests as well as test changes.. Their study provides initial insight on how test cases are added, removed, and modified in practice. The study helped in finding relationship of software evolution with addition or deletion of test cases, that they referred as TestEvol [13].This tool enables researchers and software programmers to study the test-suites evolution for Java based programs. Studies shows that non-repair test modification are quite higher than word test repairing that is almost 4:1. Second finding of this research was that usually test cases are not deleted and often renamed or moved to other places.

Mirzaaghaei et al. [9] [10] studies the test cases adoptability by software testers. Their proposed approach can do automatic test repairing and generating new test cases based on newer version of softwares. Number of algoirthm for creation and reapring of test cases was implemented in tool called as TestCareAssistant (TCA). It was found that TCA fix almost 90% of the compilation errors.

Skoglund et al. [15] conducted a case study on an evolving system with three updated versions by using three different change strategies. These strategies vary in their impact on the test suite, one of the more changes are made in the test suite while another no need change in the test suite. The paper concludes the Role Splitting strategy was in this case study superior the others in terms of effort required to make the regression tests run.

Zhang et al. [17] implemented automated testing in all the architecture layers in Bikube 2011 and the results are verified. The study found that automated testing is worthwhile because it can easily offer high code coverage, fast execution speed and improve the software quality. This gives more confidence at each stage of the product development life-cycle. Automated tests can run in less than one minute without monitoring which reduces the development iterations. The study concludes automated testing can be used in development phases means has been regarded as a useful strategy to detect defects in an earlier stage. Finding errors early is a lot more cost-effective than finding them later. Automated testing is used to achieve these requirements, thus it can save a lot of time and money.

VII. Threats and Validity

This study only focuses java project thus the results cannot be generalized for all kind of open source systems developed in a different language. Secondly only GitHub version submitted are considered for this study.

VIII. CONCLUSIONS

Open-source software systems are considered as one of a major role player in terms of software development. There are numbers of projects that are adapted from open-source and are running the full operations of the organization. One of the criteria for selecting an open-source software solution is to a proper versioning system and a concrete test suite that can test all its functionality often referred to as coverage. This study tends to find the relationship of the test suite to code suite in terms of software evolution. For this purpose, six java open-source systems having at least 12-versions were selected and was browsed for finding the relationship between code and test suites. It is seen from results that little improvement has happened over a period of time in terms of test suite coverage that shows the awareness among developers and secondly, all-new kind of version has their relevant test suites. Still, the question holds about not a complete coverage of test suites that can be solved as literature shows through automation of test-case generation process. Code complexity is often highlighted to be addressed in the literature to have an easier adoption of a software system but still, the result shows a little progress in terms of addressing the complexity that is only less than 2 percent. In future, this study will be extended to other languages system to have a general understanding of the results achieved.

References

[1] Alenezi, Mamdouh, Mohammed Akour, Alaa Hussien, and Mohammad Z. Al-Saad. "Test Suite Effectiveness: An Indicator for Open Source Software Quality." In 2016 2nd International Conference on Open Source Software Computing (OSSCOM), 1–5, 2016. https://doi.org/10.1109/OSSCOM.2016.7863677.

[2] Baresi, Luciano, and Matteo Miraz. "TestFul: Automatic Unit-Test Generation for Java Classes." In 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2:281–84, 2010. https://doi.org/10.1145/1810295.1810353.

[3] Du, Yunqi, Ya Pan, Haiyang Ao, Nimako Ottinah Alexander, and Yong Fan. "Automatic Test Case Generation and Optimization Based on Mutation Testing." In 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 522–23, 2019. https://doi.org/10.1109/QRS-C.2019.00105.

[4] Elbaum, S., D. Gable, and G. Rothermel. "The Impact of Software Evolution on Code Coverage Information." In Proceedings IEEE International Conference on Software Maintenance. ICSM 2001, 170–79, 2001. https://doi.org/10.1109/ICSM.2001.972727.

[5] Fraser, Gordon, and Andrea Arcuri. "Whole Test Suite Generation." IEEE Transactions on Software Engineering 39, no. 2 (February 2013): 276–91. https://doi.org/10.1109/TSE.2012.14.

[6] Fraser, Gordon, and Andrea Arcuri. "Evolutionary Generation of Whole Test Suites." In 2011 11th International Conference on Quality Software, 31–40. IEEE, 2011.

[7] Fraser, Gordon, and Andrea Arcuri. "EvoSuite: Automatic Test Suite Generation for Object-Oriented Software." In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, 416–419. ACM, 2011.

[8] Kochhar, Pavneet Singh, Ferdian Thung, David Lo, and Julia Lawall. "An Empirical Study on the Adequacy of Testing in Open Source Projects." In 2014 21st Asia-Pacific Software Engineering Conference, 1:215–22, 2014. https://doi.org/10.1109/APSEC.2014.42.

[9] Mirzaaghaei, Mehdi. "Automatic Test Suite Evolution." In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, 396–399. ESEC/FSE '11. New York, NY, USA: ACM, 2011. https://doi.org/10.1145/2025113.2025172.

[10] Mirzaaghaei, Mehdi, Fabrizio Pastore, and Mauro Pezze. "Supporting Test Suite Evolution through Test Case Adaptation." In Verification and Validation 2012 IEEE Fifth International Conference on Software Testing, 231–40, 2012. https://doi.org/10.1109/ICST.2012.103.

[11] Mishra, Shashank. Analysis of Test Coverage Metrics in a Business Critical Setup, 2017. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-213698.

[12] Coviello, Carmen, Simone Romano, and Giuseppe Scanniello. "An Empirical Study of Inadequate and Adequate Test Suite Reduction Approaches." In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 12:1–12:10. ESEM '18. New York, NY, USA: ACM, 2018. https://doi.org/10.1145/3239235.3240497.

[13] Pinto, Leandro Sales, Saurabh Sinha, and Alessandro Orso. "TestEvol: A Tool for Analyzing Test-Suite Evolution." In 2013 35th International Conference on Software Engineering (ICSE), 1303–6, 2013. https://doi.org/10.1109/ICSE.2013.6606703.

[14] Pinto, Leandro Sales, Saurabh Sinha, and Alessandro Orso. "Understanding Myths and Realities of Test-Suite Evolution." In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, 33:1–33:11. FSE '12. New York, NY, USA: ACM, 2012. https://doi.org/10.1145/2393596.2393634.

[15] Skoglund, M., and P. Runeson. "A Case Study on Regression Test Suite Maintenance in System Evolution." In 20th IEEE International Conference on Software Maintenance, 2004. Proceedings., 438–42, 2004. https://doi.org/10.1109/ICSM.2004.1357831.

[16] Xie, Tao. "Augmenting Automatically Generated Unit-Test Suites with Regression Oracle Checking." In ECOOP 2006 – Object-Oriented Programming, edited by Dave Thomas, 380–403. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006. https://doi.org/10.1007/11785477_23.

[17] Zhang, Lin. "Software Testing: A Case Study of a Small Norwegian Software Team." 74 p., 2012. https://uia.brage.unit.no/uia-xmlui/handle/11250/2441330.

[18] Aljedaani, Wajdi, and Yasir Javed. "Bug Reports Evolution in Open Source Systems." In 5th International Symposium on Data Mining Applications, edited by Mamdouh Alenezi and Basit Qureshi, 63–73. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2018. https://doi.org/10.1007/978-3-319-78753-4_6.

[19] Rwemalika, Renaud, Marinos Kintis, Mike Papadakis, Yves Le Traon, and Pierre Lorrach. "On the Evolution of Keyword-Driven Test Suites." In 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), 335–45, 2019. https://doi.org/10.1109/ICST.2019.00040.

[20] Coppola, Riccardo. "Fragility and Evolution of Android Test Suites." In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), 405–8, 2017. https://doi.org/10.1109/ICSE-C.2017.22.

[21] Giovanni Grano, Fabio Palomba, and Harald C. Gall. Lightweight Assessment of Test-Case Effectiveness Using Source-Code-Quality Indicators - Replication Package. Zenodo, 2019. https://doi.org/10.5281/zenodo.2571468.