A Study on the Effect of Issue Handling Indicators on Test Code Quality

Hemanth Kumar.K
Student, Department of CSE
KMM Institute of Technology and Sciences
Tirupati, India

S. Sivarama Krishna
Head of the Department, CSE
KMM Institute of Technology and Sciences
Tirupati, India

Abstract: **Delivery of software depends on its method of development. Software can be delivered to the customer quickly if it is developed and tested in less time. Quality of the software can be assured if and only if effective and enough testing is performed. For which we need to follow a software development life cycle which assists the developers to deliver the project quickly. Agile development suits best in this scenario. It uses automated testing to resolve the defects in the software quickly. Its benefits include early defect detection, defect cause localization and removal of fear to apply changes to the code. While it comes to testing, which consumes most of the software's development effort, it must be effective. Therefore we need to maintain high quality test code for better results. So we need to follow a quality model which assesses test code quality by combining source metrics that reflect three main aspects of test code quality: completeness, effectiveness and maintainability. We think the test code quality has an effect on issue handling performance. To know the relation between test code quality and issue handling performance, some quality models applied on few open source projects are studied. The metrics like defect resolution speed, throughput and productivity are used to study the relation between test code quality and issue handling performance. Efforts and experiments are started to find the relation and the positive results are expecting through the experiments.**

Keywords: testing, quality, productivity, throughput, metrics, defects, issue tracking system.

I. INTRODUCTION

Quality of the software depends on the quality of the tests performed on it. So, software testing is well established as an essential part of the software development process and as a quality assurance technique widely used in industry. We need to put 30 to 50 percent of the project's effort on testing. In traditional testing, we are required to perform unit testing, after detecting and fixing the defects we need to perform regression testing. It is not possible to do a regression test for every change we made in the code. But in agile development, we follow a different testing framework such as: Developer testing is a critical testing technique, in which a developer has to write a codified unit or integration test. Developer testing has risen to be an efficient method to detect defects early in the development process. Unit testing is the other testing technique which is gaining more popularity as many programming languages are supported by unit testing frameworks.

The main goal of any testing is to find defects accurately. In developer testing, people from the development team are involved in the testing as they have clear idea of what they had written and they can easily detect where the defect occurred if the testing is performed. So, we will get quicker results thereby saving time. Another benefit of this kind of testing is faster implementation of new features or refactoring. But it depends on the development team's performance in fixing defects and implementing new features. This can happen only if we implement the high quality test code. Therefore, we can relate the quality of the test code and issue handling performance of the development team.

Here we miss something about, how can we measure the quality of the test code? And how the developed test code quality model acts as an indicator of issue handling performance? So, first we need to assess the test code quality. Monitoring the quality of a system's test code can provide valuable feedback to the developers' effort to maintain high quality assurance standards. Several test adequacy criteria have been suggested for this purpose. But practically some these criteria are computationally too expensive. So, we need to pick a combination of a these criteria that provides a model to measure test code quality. So we propose a test code quality model that is inspired by the Software Improvement Group (SIG) quality model. This proposed quality model is solely based on source code metrics. It has three dimensions, namely completeness, effectiveness, and maintainability.

After measuring the test code quality we need to measure the issue handling performance. To measure issue handling performance of software development teams, Issue Tracking Systems (ITSs) can be mined. In Issue Tracking System of a project has its previous work on defects it had faced, developers involved, and defect resolution time kind of information in it. In addition, further indicators of issue handling performance, such as throughput and productivity, can be derived from the data that is mined from the respective Issue Tracking Systems (ITSs).

As this paper progresses, we discussed and furnished information regarding test code quality and issue handling performance in Section 2. A brief about the quality models and the SIG quality model is furnished in Section 3. In Section 4, we describe about the issue handling performance and how could it affect test code quality. Section 5 explains the execution plan where as the Section 6 describes the scope of the future work and Section 7 concludes the discussion.

II. BACKGROUND

In this section let us know more about test code quality, issue handling and SIG quality model.

2.1 Test Code Quality

What makes a good test? How can we measure the quality of a test suite? Which are the indicators of test effectiveness? Answers to these questions remain complicated. Test adequacy criteria will be helpful to assess test quality. The main role of test adequacy criteria is to assist software testers to monitor the quality of software in a better way by ensuring that sufficient testing is performed. In addition, redundant and unnecessary tests should be avoided, thus contributing to controlling the cost of testing. That's the main aim of any organization to save time and money without affecting the quality of the software.

The test adequacy criteria plays a key role in this process. It further classified into subtopics as: program-based criteria, which assess testing of the production code, and specification-based criteria, which assess testing of the specifications of a software project. But specification-based criteria are not related to this topic. Program-based test adequacy criteria can be subdivided into categories as: structural testing, fault-based testing and error-based testing.

2.1.1 Structural Testing Adequacy Criteria

This category consists of test criteria that focus on measuring the coverage of the test suite upon the structural elements of the program. These criteria can be split into control-flow criteria and data-flow criteria which are mostly based on analysis of the flow graph model of program structure.

Control-flow criteria are concerned with increasing the coverage of the elements of the graph as much as possible. Different criteria assess coverage in a different scope: statement coverage, branch coverage or path coverage. Based on these criteria, metrics can be derived to measure the quality of the test code.

Data-flow criteria are concerned with analyzing whether paths associating definitions of variables to their uses are tested.

2.1.2 Fault-Based Testing Adequacy Criteria

Criteria that fall inside this category focus on measuring the defect detection ability of a test suite. Error seeding and mutation analysis are the main approaches to fault-based test adequacy criteria. These techniques can be applied to acquire test effectiveness indicators. Error seeding is the technique of planting artificial errors in a software system and subsequently testing the system against artificial errors and counting the successfully detected ones. Mutation analysis is a more systematic way of performing error seeding.

2.1.3 Error-Based Testing Adequacy Criteria

In this test criteria we focus on measuring to what extent the error-prone points of a program are tested. In order to find error-prone points, a domain analysis of a program's input space is necessary. But this error-based testing criteria is limited when the complexity of the input space is high or non-numerical.

2.1.4 Assertions and Maintainability

Assertions are the notable points regarding test code. It can be defined in assertion density as the number of assertions per thousand lines of code and showed that there is a negative correlation between assertion and fault density. This means assertions can increase the test effectiveness.

Maintainability is important to ensure that it is clear to read and understand, to ease its modification. It is necessary to have maintenance to detect possible points of low quality in the test code. For this purpose we require test code refactoring.

2.2 Issue Handling

2.2.1 Issue Tracking Systems and the Life-Cycle of an Issue

ITSs are software systems used to track defects as well as enhancements or other types of issues, such as patches or tasks. ITSs are commonly used and they enable developers to organize the issues of their projects.

When defects are discovered or new features are requested, the developers typically reports to the ITS. Issues that are reported follow a specific life-cycle. Even though there are a variety of implementations of ITSs such as BugZilla, Jira, GitHub, they follow the same general process. If a developer found any issue or defect while working with some software, he can report to the respective software developers through ITSs. Firstly, he had register with the ITS of the corresponding software (for example: BugZilla). Then select the ITS of the software in which he found an issue. Then only he can write about the issue and possible enhancements that can be done to resolve the defect. After that the developers of the software tries fix it. The same procedure has to be followed with all ITSs.

We now briefly describe about the life-cycle of an issue report. Initially, the report is formed and submitted as an unconfirmed issue. After it is checked whether the issue has already been reported or the report is not valid, the issue status is changed to new. The next step is to assign the issue to an appropriate developer, an action which results in the issue state assigned. The possible resolutions are:

- Invalid: The issue is not valid.
- Duplicate: The issue has already been reported.
- Fixed: The issue is fixed.
- Won't fix: The issue will not be fixed.
- Works for me: The issue could not be reproduced.

The issue is marked as resolved and then it is closed, unless it was a fixed issue.

2.2.2 Defect Resolution Time

Defect resolution time is an indicator of the time that is needed to resolve a defect. There are a lot of stages involved in a defect resolution. So, the defect resolution time is defined as the interval between the moment when the defect was assigned to a developer and the moment when it was marked resolved. Every issue had to go through the above said life-cycle of an issue at any cost. And it is recommended that reported issue must be given to the appropriate developer.

2.2.3 Throughput and Productivity

Throughput and Productivity are the other metrics we need to know after defect resolution speed. They can be defined as follows:

Throughput measures the total productivity of team working on a system in terms of issue resolution.

$$Throughput=\frac{\#\,resolved\ \ issues\ \ per\ \ month}{KLOC}$$

Productivity measures how productive the whole team that works on a system is.

$$Productivity=\frac{\#\,resolved\ \ issues\ \ per\ \ month}{\#\,developers}$$

III. SOFTWARE QUALITY MODELS

Before knowing about the proposed test code quality model in detail we need to know about the quality models and their use in our study. Many quality models are available these days for the development of the software. Every model has a different scope and way of implementation. And every model intends to improve the quality the software. Each model has its own factors and characteristics. In late 1970's the quality models emerges and results in McCall model. After this model, the other models came into the picture with enhancements and new features. Few of them are the FURPS, Boehm, Dromey, ISO and so on. Each of these models was introduced at different time by different organizations for the same need, Quality.

ISO/IEC 9126 quality model was introduced as a part of ISO 9000 standard. It is one of the most popular quality models today because it inherits hierarchical tree structure of characteristics and sub- characteristics. So it is easy to understand the ISO 9126 quality model. The main characteristics include Functionality, Reliability, Usability, Efficiency, Maintainability and Portability; which are further divided into 21 sub characteristics. So, each and every quality measure is taken into the account. Therefore this model can be applicable to any kind of software. The SIG quality model is also to ISO 9126 model.

3.1 The SIG Quality Model

The Software Improvement Group (SIG) is an Amsterdam-based consultancy firm specialized in quantitative assessments of software portfolios. It has developed a model for assessing maintainability of software. The SIG quality model defines source metrics and maps the metrics to the quality characteristics of ISO/IEC 9126 that are related to maintainability.

In the first step, source code metrics are used to collect facts about a software system. The source code metrics that are used to express volume, duplication, unit complexity, unit size, unit interfacing and module coupling are also helpful. The measured values are combined and aggregated to provide information on properties at the level of the entire system. These system-level properties are then mapped onto the ISO/IEC 9126 standard quality characteristics that relate to maintainability, which are: analyzability, changeability, stability and testability.

In the second step, after the measurements are obtained from the source code, the system-level properties are converted from metric-values into ratings. This conversion is performed through benchmarking and relies on the database that SIG possesses and curates; the database contains hundreds of systems that were built using various technologies. After this step, the metric values can be converted into quality ratings which reflect the system's performance. The rating is given on a scale of 1 to 5. The system performs best is rated 5 and the worse system is rated 1.

3.2 Proposed Test Code Quality Model

The proposed test code quality model was inspired from Software Improvement Group's (SIG) quality model. In order to know more about this model, we need to know the answers to the following questions:

- How can we evaluate the quality of the test code?
- How completely is the system tested?
- How much of the code is covered by the tests?
- How effectively the system is tested?
- How able is the tested code to detect defects in the production code that it covers?
- How able is the test code to locate the cause of a defect after test detected it?

Source code metrics are the indicators to formulate this model (as we said early in the discussion). So, based on SIG quality model we have taken some metrics which are useful to know the answer to the above questions. The metrics that were selected as indicators of test code quality are Code Coverage, Assertions- McCabe Ratio, Assertion Density, Directness and Maintainability. Maintainability was the most important one among all these metrics that is why we are following SIG quality model.

3.2.1 Code Coverage

Code coverage is the most frequently used metric for testing code quality. There are many tools for dynamic code coverage estimation (e.g., Clover and Cobertura for Java, Testwell CTC++ for C++, NCover for C#). These tools use a dynamic analysis approach to estimate code coverage. But dynamic analysis has two disadvantages. First, the analyzer must be able to compile the source code. This is an important drawback both in the context of this study and in the intended context of application. Second, dynamic analysis requires execution of the test suite, a task that is time consuming. Another method for code coverage is static analysis of the source code in which code coverage percentage is calculated by measuring the percentage of covered lines of code, where it is assumed that in a covered method all of its lines are covered.

3.2.2 Assertions-McCabe Ratio

The Assertions-McCabe ratio metric indicates the ratio between the number of the actual points of testing in the test code and of the decision points in the production code. The metric is inspired by the Cyclomatic-Number test adequacy criteria and is defined as follows:

$$\textit{Assertions-McCabe Ratio} = \frac{\#assertions}{cyclomatic\ \ complexity},$$

Where #assertions is the number of assertion statements in the test code and cyclomatic complexity is McCabe's cyclomatic complexity for the whole production code.

3.2.3 Assertion Density

Assertion density aims at measuring the ability of the test code to detect defects in the parts of the production code that it covers.

$$\textit{Assertion Density} = \frac{\#assertions}{LOC_{test}},$$

Where #assertions is the number of assertion statements in the test code and LOC_{test} is lines of code.

3.2.4 Directness

An effective test should provide the developers with the location of the defect to facilitate the fixing process. When each unit is tested individually by the test code, a broken test that corresponds to a single unit immediately pinpoints the defect. Directness measures the extent to which the production code is covered directly, i.e. the percentage of code that is called directly by the test code.

3.2.5 Maintainability

As it is the key feature of SIG quality model having maintainability of production code, it uses various metrics in cooperation with ISO/IEC 9126. But test code is different from production code. So, it is needed to make changes to the SIG quality model which in turn results into the test code maintainability model. The SIG model has four sub characteristics which are analyzability, changeability, stability, and testability.

Analyzability is the capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified. Changeability is the capability of the software product to enable a specified modification to be implemented. Stability is the capability of the software product to avoid unexpected effects from modifications in the software. Testability is the capability of the software product to enable modified software to be validated.

Now we have clearer view of the metrics we are using and it is time to know the quality of the test code. At the beginning of the discussion we said that test code quality can be measured by: completeness, effectiveness and maintainability. All these metrics are checked against the metrics we've defined earlier, i.e. Code Coverage, Assertions-McCabe ratio, Assertion Density and Directness.

We will calculate the overall rating of the test code quality by aggregating sub-characteristics completeness, effectiveness and maintainability. All these three sub-characteristics must be high to get positive results in testing code quality. We can call it as the geometric mean:

TestCodeQuality=

$$\sqrt[3]{Compltness.Effectiveness.Maintainability}$$

IV. ISSUE HANDLING PERFORMANCE

To relate the test code quality with issue handling performance, we also need to compare the test code quality with the metrics of issue handling performance like:

- Is there a relation between the test code quality ratings and the defect resolution time?
- Is there a relation between the test code quality ratings and the throughput of issue handling?
- Is there a relation between the test code quality and the productivity of issue handling?

For this purpose we need to extract the issue handling measurements from the ITSs of some open source java projects. To get higher test code quality, we must have shorter defect resolution time and higher throughput and productivity. The measurements throughput and productivity can be calculated with the formulas mentioned earlier. But to decide whether the defect resolution speed is low or high we have to follow the following readings. If the defect is fixed in less than four weeks then it has shorter defect resolution time and if it takes more than ten weeks then is has higher defect resolution time.

4.1 Other Factors Affecting Issue Handling

Not only has the factor test code quality affect issue handling. There are other confounding factors which could also affect the issue handling. They are:

- Production code maintainability: While issues are being resolved, the maintainer analyses and modifies both the test code and the production code. Therefore, issue handling is affected by the maintainability of the production code.
- Team size: The number of developers working on a project can have a positive or negative effect on the issue handling efficiency.
- Maintainer's experience: The experience of the person or persons who work on an issue is critical for their performance on resolving it.
- Issue granularity: The issues that are reported in an ITS can be of different granularity. Because the scope can vary from issue to issue.
- System's popularity: High popularity of a project may lead to a larger active community that reports many issues.

These factors can be controlled only if these factors are measured. So that, we can reduce the impact of these factors on issue handling.

V. EXECUTION PLAN

Here comes the proposed execution plan to show that the test code quality has an effect on issue handling performance. Before proceeding into the action we need to collect and preprocess the data on which we are performing our research (e.g. open source projects). For this, first we need to collect source code and issue tracking data of systems. It is easy to get the source code of open source project as it is available publicly. But only few of them allow accessing their issue tracking data publicly. So we need to select those open source projects for which both the source code and the issue tracking data (ITS) are available. For example: Apache Ivy, Apache Ant, Apache Tomcat, ArgoUML, HSQLDB, JabRef, jMol, OmegaT etc.

Till now we are discussing the relation between test code quality and issue handling performance in terms of theoretical knowledge. But practically it is a time taking process. Because we need to take snapshots periodically such as if a defect occurred in a system, firstly it has to be found and then it needs to go through the life-cycle of the issue which we have said earlier and then it needs to get fixed. We have to take a snapshot when an issue first reported to its ITS and another snapshot after resolution of the issue. After collecting the data about all these issues on a snapshot basis, we have to prepare those data for the experiment. For an easy organization we are using a database tool to store these data. This is all about the issue tracking data and its preprocessing. Source code also can be preprocessed based on the snapshots. As we need to ignore the patches, irrelevant code and consider the code at the time when an issue is raised. Because these patches may be the resolutions to the defects reported. In each and every snapshot we will notice that the number of defects raised and number of defects resolved may vary.

After preprocessing the source code and issue tracking data we need to go to the statistical part of the experiment where we will calculate and measure the values for test code quality and issue handling indicators. That is, we will have to measure values for completeness, effectiveness and maintainability for test code quality and defect resolution speed, throughput and productivity for issue handling performance. After finding all these values now we can show that there is an effect of issue handling indicators on test code quality which is our ultimate aim of this research.

VI. FUTURE WORK

A sound theoretical framework and a design to get the relation between test code quality and issue handling performance is presented in this paper. The proper execution of our framework will guide the proposed work towards better and useful results. It is easy to say that there some relation between test code quality and issue handling performance.

We are monitoring the results and taking a snapshot of every observation of the selected open source projects. The results of this study will show the actual effect of the aforementioned things. So we will come up with results as soon as possible. If the proposed model works well, we can develop a tool based on this study which will minimize all the statistical calculations. It will be easy then to assess the quality of the test code with just a few mouse clicks.

We cannot limit the indicators which affects issue handling performance and test code quality. In, future we will definitely try to figure out the other factors which could possibly affect test code quality. But the limitation is this research is confined to Java projects only and we don't have any idea what could be the effect in case of other programming languages.

VII. CONCLUSION

Developer testing is an important part of software development in which the developer of the defect raised code handles the defects and fixes them. For an every change in the functionality must not be changed. So, traditionally we perform regression testing which is hectic to repeat the same testing many times. It will result in redundant and too much testing. To overcome this we need to automate the testing which is a key feature in agile development. These automated tests enable early detection of defects in software, and facilitate the comprehension of the system.

There is not a single model which is definite to assess the quality of the test code. But some quality models give us inspiration to find quality of the code. That is why we have chosen SIG quality model and added some adornments to the model and made it best suitable for assessing quality of the test code. But this quality model is inspired from ISO/IEC 9126.

Effective testing is one of the best ways to assure the quality of software. At the same time we must not sit on testing for ages. We must perform enough testing but effective testing. Effective testing means we have to implement effective test code. So, the quality of the test code matters here. We are not done with just finding defects but we have to assess the impact of the issue handling indicators on test code quality. So that's why we are trying to relate test code quality and issue handling performance.

The factors completeness, effectiveness and maintainability are the key indicators for test code quality. And defect resolution speed. Throughput and productivity are the key indicators for issue handling performance. So, first we have to measure these indicators and try to get relation between them.

REFERENCES

1. A. Zaidman, B. Van Rompaey, A. Van Deursen, and S. Demeyer, " Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining", Empir.softw.eng vol 16, no.3, pp. 325-364,2011.
2. X. Xiao, S. Thummalapenta, and T.Xie, "Advances on improving automation in developer testing", Adv.Comput., vol 85, pp. 165-212, 2012.
3. B. Luijten, J. Visser, and A. Zaidman, "Assessment of issue handling efficiency", in Proc. Working Conf. Mining softw. Repositories, 2010, pp 94-97.
4. C. Weiss, R.Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?", in proc. Int. Workshop Mining softw. eng. 2010, pp. 52-56.
5. D. Athanasiou, A.Nugroho, J.Visser and A. Zaidman, "Test code quality and its Relation to Issue Handling Performance", vol 40, pp. 1100-1125, 2014.
6. D. Bijlsma, "Indicators of issue handling efficiency and their relation to software maintainability", MSc Thesis, Univ. Amsterdam, The Netherlands, 2010.
7. S.C. Ntafos, "A comparison of some structural testing strategies", IEEE trans. Softw. Eng., vol 14, pp. 868-874, 1988.
8. A.B. AL-Badareen, M.H. Selemat, M.A. Jabar, J. Din and S. Turaev, "Software Quality Models: A Comparative Study".
9. H. Natarajan, Ram Kumar.S, Kalpana.L, "A Comparison between Past, Present and Future Models of Software Engineering", IJCSI, vol.10, 2013.
10. B. Luijten, "The influence of software maintainability on issue handling", M.S. Thesis, Delft Univ., Delft, The Netherlands.

11. D. Athanasiou, “Constructing a test code quality model and empirically assessing its relation to issue handling performance”, Delft Univ., Delft, The Netherlands.
12. J.Voas, “How assertions can increase test effectiveness”, IEEE soft.eng. vol.14, pp. 118-122, 1997.
13. J.A. McCall, P.K. Richards, and G.F. Walters, “Factors in software quality”, Nat. Tech. Serv., vol.1/2/3, 1997.
14. B.W Boehm, J.R. Brown, and M. Lipow,” Quantitative evaluation of software quality”, in Proc. Int. Conf. Softw. Eng., 1976, pp. 592-605.
15. R.G. Dromey, “A model for software product quality”, IEEE trans. Softw. Eng., vol. 21, no. 2, pp. 146-162, 1995.