International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Analysis of Efficiency of Automated Software Testing Methods: Direction of Research

K. Valliammai[1], Dr. P. Sujatha[2]

[1]Research Scholar, Department Computer Science, Bharathiar University, Coiambatore, TN, India

[2]Associate Professor, Department Computer Science, Vels University, Chennai, TN, India

Abstract: ***Efficiency is an important property of software testing potentially even more important than effectiveness. Because complex software errors exist even in critical, widely distributed programs for many years, developers are looking for automated techniques to gain confidence in their programs correctness. The most effective way to inspire confidence in the program's correctness for all inputs is called program verification. However, due to state explosion and other problems, the applicability of verification remains limited to programs of a few hundred lines of code. Now, software testing trades this effectiveness for efficiency. It allows one to gain confidence in the program's correctness with every test input that is executed. So, automated testing is an efficient way to inspire confidence in the program's correctness for an increasing set of inputs. Yet, most research of software testing has mainly focused on effectiveness.***

Keywords: Automated Testing, Gain Confidence, Software Errors, Software Testing, State Explosion

1. Introduction

The most effective testing technique reveals a maximal number of errors and inspires a maximum degree of confidence in the correctness of a program.

We start working to the efficiency
The most efficient testing technique
i) Generates a sufficiently effective test suite in minimal time or
ii) Generates the most effective test suite in the given time budget.

Using a simple set of assumptions, we construct a general model of software testing, define testing strategies where each generated test input is subject to a cost, and cast our efficiency analysis as a problem in probability theory.

We model the testing problem as an exploration of error based input partitions. Suppose, for a program there exists a partitioning of its input space into homogeneous sub domains [4], [5]. For each sub domain, either all inputs reveal an error or none of the inputs reveal an error. The number and "size" of such error-based partitions can be arbitrary but must be bounded. Assuming that it is unknown a-priori whether or not a partition reveals an error, the problem of software testing is to sample each partition in a systematic fashion to gain confidence in the correctness of the program. A testing technique samples the program's input space. We say that a partition Di is discovered when Di is sampled for the first time. The sampled test input shows whether or not partition Di reveals an error. Effectively, the sampled test input becomes a witness for the error-revealing property of Di. A testing technique achieves the degree of confidence x when at least x% of the program inputs reside in discovered partitions. Hence, if none of the discovered partitions reveals an error, we can be certain that the program works correctly at least for x% of its input. For our efficiency analysis,
We consider two strategies:

1. Random testing that is oblivious of error-based partitions and
2. Systematic testing that samples each partition exactly once.

Random testing (R) samples the input space uniformly at random and might sample some partitions several times and some not at all. Specifically, we show that for R the number and size of partitions discovered decays exponentially over time.

Systematic testing samples each error-based partition exactly once and thus strictly increases the established degree of confidence. We model a systematic testing technique that chooses the order in which partitions are discovered uniformly at random and show that number and size of partitions discovered grows linearly over time.

2. Motivation

The software is a major component of organizations in computer engineering. They make their own projects and learn how to develop software. But it seems that while learn to develop software just by following the actual software development cycle, which should spend 50 percent or more on software testing, publish software on the market. Users only emphasize the software design and coding part. The proof of the software development lifecycle is informally filmed. Regardless of the software they develop, they do not have the system manual or use automated test tools to test the execution. In the software development stage, developers have to face some challenges, and consider a variety of situations.

Ultimately, it affects the reputation and profile of the organization. Since manual testing is very expensive, the use of automated test tools is essential to reduce software costs and enable us to compete with global markets. Automated testing tools provide wizard testing automation and their own commands, and provide recording functions and rework. It is self-controlled flow and self-motion.

Automated testing means that fewer people need to test the system and often can test with hundreds of people who manage the simulation of the network on the terminal. In addition, we note that users who use software testing tools are very confused. There is a number of software testing tools available online. The automation tools that master the integration of some important tools are also a great work that will take a long time. Even most of the powers are confused about providing the appropriate tools to study the selected student practice guidelines. Few companies can spend money for testing tools. As can be seen, most educational institutions using automated testing tools. We would like to provide guidance and research on automated testing tools with minimal effort. The results of our work will help users gain exposure to automated testing tools that can easily learn automated testing tools.

We use the quality standards and functional standards. Common quality standards are tested for functionality, reliability, ease of use, efficiency, maintainability, portability, vendor support, licensing and pricing.

3. Objective

Our goal in this survey is to help professional's select efficient tools that suit their needs and environments, as well as give some indication of the state of the technology in hedging tools. This work is also of artistic value to those who are new to practice and software testing coverage, as well as those who want to understand the gap between industry and academia.

First, we note that the code coverage tool has been growing in interest to our software development organizations for several years. In the project evaluation [5], the internal and informal discussion of the forum, the following found several important issues.

1) Developers and testers do not need to know how much of our code has been included in a good way of testing.
2) The development manager knows how well he is interested in testing through development milestones through code completion such as unit testing, integration testing and system testing.
3) The cycles of change, manufacture and testing are generally inefficient and ineffective. There are too many manual steps in the loop and too many flaws crept through the fields.
4) Developers are so entangled in the fire that they have no time to search for tools to automate their processes [6].

Therefore, we decided to introduce better construction and use of automated testing processes. Due to an element in our approach, we focus on code coverage, which is very easy to learn by developers and administrators, and has an intuitive appealing approach, even though its effectiveness is limited by the basic theory of defect detection. Note to introduce other techniques, such as building evaluation [7], but we have limited the coverage of the tool code for the tools and the range of functionality they provide, as these tools are a great attraction for us to work with software developers and their managers. This approach also provides an overview of how these tools can improve our research, thereby extending and improving coverage tools.

4. Literature Review

Muhammad Shahid et al. [1] He explains that test Coverage is an important indicator of software quality and an essential part of software maintenance. It helps in evaluating the effectiveness of testing by providing data on different coverage items. Although much research effort has been put on how to get coverage information by either code based testing or requirement based testing, not much has been paid to measure and analyze the coverage by covering maximum number of coverage items. The systematic process was described in terms of the research questions defined, searching keywords used, the exclusion and inclusion criteria. Most of the research papers are from conference and paper proceedings, which indicate that more work needs to be done in order to improve the current state of research in test coverage measurement and analysis.

Pranali Prakash Mahadik et al. [2] He conveys that there are several methods which are capable of generating test input automatically based on the source code of the program under test. Survey paper mentioned the description in brief about test data generation technique like Random selection, Search-based techniques and Symbolic execution based techniques. Survey focuses on the problem of how to choose the most appropriate tool that will fulfill developer requirements consisting of level of automation, cost requirement, language support, etc. The idea to develop new efficient and effective tool by merging properties of some tools of similar kind that can find more range of errors and improve the code coverage for object oriented code by considering features of object oriented languages.

Chayanika Sharma et al. [3] He explains that testing ensures that software meets user specifications and requirements. However, the field of software testing has a number of underlying issues like effective generation of test cases, prioritisation of test cases etc which need to be tackled. These issues demand on effort, time and cost of the testing. Different techniques and methodologies have been proposed for taking care of these issues. Use of evolutionary algorithms for automatic test generation has been an area of interest for many researchers. Genetic Algorithm (GA) is one such form of evolutionary algorithms. The GA is also used with fuzzy as well as in the neural networks in different types of testing. It is found that by using GA, the results and the performance of testing can be improved.

Adnan Causevic et al. [4] He conveys that contemporary aspect, such as the introduction of a more lightweight process, trends towards distributed development, and the rapid increase of software in embedded and safety-critical systems, challenge the testing process in unexpected manners. To our knowledge, there are very few studies focusing on these aspects in relation to testing as perceived by different contributors in the software development process. One of the noteworthy testing research directions from an industrial perspective seems to be test driven development as indicated by the results of the survey. The survey has unique features such as strategic embedding of

multi-purpose questions and categorisation of respondents on contemporary aspects which enable us to gain qualitative insights.

Dudekula Mohammad Rafi et al. [5] He conveys that the academic views are studied with a systematic literature review while the practitioners views are assessed with a survey, where we received responses from 115 software professionals. The results of the systematic literature review show that the source of evidence regarding benefits and limitations is quite shallow as only 25 papers provide the evidence. Furthermore, it was found that benefits often originated from stronger sources of evidence (experiments and case studies), while limitations often originated from experience. The limitations were high initial invests in automation setup, tool selection and training. Additionally, 45% of the respondents agreed that available tools in the market offer a poor fit for their needs. Finally, it was found that 80% of the practitioners disagreed with the vision that automated testing would fully replace manual testing.

5. Automated Testing Measures

Software measures can help to improve the process of automated test organization and track its status. These measures and techniques have been successfully applied through our test equipment software. Just as the quote at the beginning of this study means that if we can measure something, then we have something to quantify. If we can quantify things, then we can explain in more detail and learn more about it. If we can explain it, then we have a better chance to try to improve it, and so on.

Over time, software projects have become more complex due to increased functionality, bug fixes, etc. It also requires that the task be done with fewer people and less time. Over time complexity will tend to reduce test coverage and, ultimately, product quality. The other factors involved in the time are the total cost of the product and the time that the software is provided. Software measures can provide insight into the state of automated test work.

5.1 Percent Automatable

At the beginning of the automated test work, the project automatically has an existing manual test program, a new automated effort from scratch, or some combination of the two. In either case, it can be determined as a percentage that can be automated. The proportion of automation can be defined as a given set of test cases, how many of them can be automated? This may be represented by the following formula:

$$\text{PA (\%)} = \frac{\text{ATC}}{\text{TC}} = \left[\frac{\#\text{ of test cases automatable}}{\#\text{ of total test cases}}\right]$$

PA = Percent Automatable
ATC = # of test cases automatable
TC = # of total test cases

5.2 Automation Progress

Automation Progress means that the proportion of automated test cases, how many have been fully automated at a given moment? Basically, how do we automate the test for what is the goal? The goal is to automate 100% of "automated" test cases. This measure is useful for monitoring at different stages of automated testing.

$$\text{AP (\%)} = \frac{\text{AA}}{\text{ATC}} = \left[\frac{\#\text{ of actual test cases automated}}{\#\text{ of test cases automatable}}\right]$$

AP = Automation Progress
AA = # of actual test cases automated
ATC = # of test cases automatable

5.3 Test Progress

The progress of automation is closely related, but not the only common indicator of automation is the progress of the trial. Test progress can simply be defined as the number of test cases that are attempted (or completed) over time.

$$\text{TP (\%)} = \frac{\text{TC}}{\text{T}} = \left[\frac{\#\text{ of test cases (attempted or completed)}}{\text{time (days \weeks \months ,etc)}}\right]$$

TP = Test Progress

TC = # of test cases (either attempted or completed)

T = some unit of time (days / weeks / months, etc)

5.4 Percent of Automated Testing Test Coverage

We want to consider the automatic measurement software is Percent of automated testing test coverage. This is a measure to determine which test coverage is being automated to test really long headlines? It is a measure of the integrity of the test. This measure is not as much of a measure of how much automation runs, but rather how much of the product functionality is covered. For example, running the same or similar data line may require a considerable amount of time and effort to run 2000 test cases, does not mean a large percentage of test coverage. The percentage of automated test coverage does not specify anything about the effectiveness of the ongoing trial; it is a measure of its size.

$$\text{PTC (\%)} = \frac{\text{AC}}{\text{C}} = \left[\frac{\text{automation coverage}}{\text{total coverage}}\right]$$

PTC = Percent of Automatable testing coverage
AC = Automation coverage
C = Total Coverage (KLOC, FP, etc)

5.5 Percent of Testing Coverage

The Percent Automated Test Coverage measure can be used in conjunction with the standard software testing measure called Test Coverage.

$$\text{TC} = \frac{\text{TTP}}{\text{TTR}} = \left[\frac{\text{total \# of TP}}{\text{total \# of test requirements}}\right]$$

TC = Percent of Testing Coverage
TTP = Total # of Test Procedures developed
TTR = Total # of defined Test Requirements

5.6 Defect Density

The defect density is another well known because it is not specifically automated. It is a measure of all well-known defects measured by the size of the software entity. For example, if there is a high density of defects in a particular function, it is important to conduct a causal analysis. This is a very complex function, so it is expected that the defect density is high? Is there a problem with the design /

implementation functionality? Is the resource (or not enough) functioning allocated incorrectly because it has been assigned an inaccurate risk? We can also conclude that the development of this particular function requires more training.

$$DD = \frac{D}{SS} = \left[\frac{\#\text{ of known defects}}{\text{total size of system}}\right]$$

DD = Defect Density
D = # of known defects
SS = Total Size of system

5.7 Defect Trend Analysis

Closely related measure to Defect Density is Defect Trend Analysis. Defect Trend Analysis is calculated as

$$DTA = \frac{D}{TPE} = \left[\frac{\#\text{ of known defects}}{\#\text{ of test procedures executed}}\right]$$

DTA = Defect Trend Analysis
D = # of known Defects
TPE = # of Test Procedures Executed over time
Cost to locate defect = Cost of testing / the number of defects located
Defects detected in testing = Defects detected in testing / total system defects
Defects detected in production = Defects detected in production/system size

5.8 Actual Impact on Quality

One of the most popular measures to measure quality across tests (if the number of defects is used as a quality measure) is Defect Removal Efficiency (DREs), which are unspecific automation, but are very useful when used with automation work. DRE is a measure used to determine their efforts to eliminate the effectiveness of defects. This is also an indirect measure of product quality. The value is calculated as a percentage of DRE. The higher the percentage, the greater the positive impact on product quality. This is because it represents the absence of any particular phase in time for identification and elimination.

$$DRE(\%) = \frac{DT}{DT+DA}\left[\frac{\#\text{ of defects found during testing}}{\#\text{ of defects found during testing} + \#\text{ of defects found after delivery}}\right]$$

DRE = Defect Removal Efficiency
DT = # of defects found during testing
DA = # of defects acceptance defects found after delivery

5.9 Other Software Testing Measures

Along with the measures mentioned in the previous section, there are some common detection methods. Such measures do not necessarily apply to automation, but can also be, and often are, associated with general software testing. These measures fall into three categories:

- **Coverage:** Significant parameters are used to measure the range of tests and successes.
- **Progress:** The parameter helps identify the success criteria of the test to match the progress. Progress measures are collected on iterations. They can be used to draw the process itself (e.g. time to fix defects, time to test, etc).
- **Quality:** Excellence, worth, value and other significant measures to test the product. Quality is difficult to measure directly; however, the impact of measured mass is much easier and possible.

6. Conclusion

Nowadays, the testing in the software development has played an important role. It can be seen that the main is amount of money and the total cost of software development invests in software testing. The survey is based on practical and functional standards that some automated testing methods have to illustrate that users can try to get software to do it easily in an application. The software measures automated software testing important indicators of hygiene, quality and schedule. This can also be used for past performance, current status and future trends. Good measures are objective, measurable, meaningful, simple, and have ready-made data. Software quality engineering using traditional software test methods can be applied for automated software testing. Whether the automated assessment is meaningful or not in the test case reflects the automation of its work. Given an automated set of test cases, it is determined to provide the greatest return on investment. Just think, just because test automation does not mean it should be automated. Our study included three characteristics of comparison: (i) code coverage, (ii) coverage criteria and (iii) automation and reporting. Overall, there has been a lot of investigation and production of industrial software in software test coverage areas that have been used. We hope that our work will help to increase the use of tools to improve software testing.

References

[1] Muhammad Shahid, Suhaimi Ibrahim and Mohd Naz'ri Mahrin by "A Study on Test Coverage in Software Testing", International Conference on Telecommunication Technology and Applications,2011.
[2] Pranali Prakash Mahadik, Prof.Dr. D. M. Thakore by "Survey on Automatic Test Data Generation Tools and Techniques for Object Oriented Code", International Journal of Innovative Research in Computer and Communication Engineering, Vol. 4, Issue 1, January 2016.
[3] Chayanika Sharma , Sangeeta Sabharwal , Ritu Sibal by "A Survey on Software Testing Techniques using Genetic Algorithm", International Journal of Computer Science Issues, Vol. 10, Issue 1, No 1, January 2013.
[4] Adnan Causevic, Daniel Sundmark, Sasikumar Punnekkat by "An Industrial Survey on Contemporary Aspects of Software Testing",.
[5] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Kai Petersen Mika V. Mantyla by "Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey",
[6] Hitesh Tahbildar, Plabita Borbora,G. P. Khataniar by "Teaching Automated Test Data Generation Tools For C, C++ , And Java Programs", International Journal of Computer Science & Information Technology (IJCSIT) Vol 5, No 1, February 2013.
[7] Saswat Anand, Tsong Yueh Chen, John Clarkby "An Orchestrated Survey on Automated Software Test Case Generation".

[8] J. Lee, S. Kang, D. Lee by "Survey on software testing practices", The Institution of Engineering and Technology 2012.
[9] Pradeep Kumar Singh, Om Prakash Sangwan by "A Study and Review on the Development of Mutation Testing Tools for Java and Aspect-J Programs", I.J. Modern Education and Computer Science, 2014.
[10] Akalanka Mailewa and Jayantha Herath Susantha Herath by "A Survey of Effective and Efficient Software Testing".
[11] Abdullah Saad AL-Malaise AL-Ghamdi by "A Survey on Software Security Testing Techniques", International Journal of Computer Science and Telecommunications, Volume 4, Issue 4, April 2013.
[12] Vishal Sangave, Vaishali Nandedkar by "A review on Automating Test Automation", International Journal of Advance Research in Computer Sciece and Management Studies, Volume 1, Issue 12, December 2014.
[13] Manjit Kaur and Raj Kumari : "Comparative Study Automated Testing Tools: Test Complete and QuickTest Pro", International Journal of Computer Application , Volume 24 No.1 June 2011.
[14] Hitesh Tahbildar and Bichitra Kalita : "Automated software test data Generation : Direction of Research " International Journal of Computer Science and Engineering Survey , Volume 2 No. 1, PP 99 - 120, March 2011.
[15] Hitesh Tahbildar and Bichitra Kalita : "Automated test data generation based on individual constraints and boundary value" IJCSI International Journal of Computer Science Issues , vol. 7, pp. 350-359, September 2010.
[16] Sakamoto, K., H. Washizaki, et al. (2010). "Open Code Coverage Framework: A Consistent and Flexible Framework for Measuring Test Coverage Supporting Multiple Programming Languages", In the 10th International Conference on Quality Software, QSIC,2010, pp. 262-269
[17] Singh M, Gupta P.K., Mishra S., Automated Test Data Generation for Mutation Using AspectJ Programs‖, *In Proceedings of ICIIP-2011,* IEEE, 2011.
[18] Ferrari F.C., Nakagawa E.Y., Maldonado J.C., Rashid A. Proteum/AJ: a mutation system for AspectJ programs‖, *in Proceedings of AOSD-11,* ACM, 2010.
[19] Singh M., Mishra S. and Mall R., ―Accessing and Evaluating AspectJ based Mutation Testing Tools, *published in Iinternational Journal of Computer Application,* pp. 33-38, 2011.
[20] Singh, P.K, Sangwan O.P. and Sharma A., ―A Systematic Review on Fault Based Mutation Testing Techniques and Tools for Aspect-J Programs‖, published in proceedings of 3rd IEEE International Advance Computing Conference, IACC-2013, India, 22-23 Feb. 2013.
[21] Saurabh Sinha, Nimit Singhania, and Satish Chandra by "Automating Test Automation" Suresh Thummalapenta, IBM Research – India (2012)
[22] Fei Wang, Wencai Du,china by "A Test Automation Framework Based on WEB" , (2012)
[23] Jingfan Tang by "Towards adaptive framework of keyword driven automation testing" (2008)
[24] Prof.(Dr.)V. N.Maurya, Er.Rajender Kumar "Analytical Study on Manual vs. Automated Testing Using with Simplistic Cost Model" (2012)
[25] A. Sinha, S. Sutton, and A. Paradkar by "Text2Test: Automated inspection of natural language use cases" (2010).