Automatic Discovery of Dependency Structures for Test Case Prioritization

C. Prakasa Rao

Research Scholar, Dept. of Computer Science, S.V. University, Tirupati, India.

P.Govindarajulu

Retd. Professor , Tirupati, India Dept. of Computer Science, S.V. University,

Abstract

In software engineering "testing" is one of the phases in system development life cycle. Functional test suites are used to discover bugs in Software Under Test (SUT) and improve its quality. A good test suite uncovers more faults in the SUT. As test suite contains many test cases, the order of their execution plays an important role in increasing the rate of fault detection which can provide early feedback to development team so as to help them to improve the quality of the software. Therefore it is very useful to prioritize test cases that will lead to the increase in the rate of fault detection. However, prioritization of functional test suites is a challenging problem to be addressed. Recently Haidry and Miller proposed a family of test case prioritization techniques that use the dependency information from a test suite to prioritize that test suite. The nature of the techniques preserves the dependencies in the test ordering. Dependencies in test cases can have their impact on the discovery of faults in software. This hypothesis has been proved by these authors as their empirical results revealed it. However, they do not automate the extraction of dependency structures among the test suits that can help in effective prioritization of functional test suites. In this paper we propose a methodology that automates the process of extraction of dependency structures from the test cases that will result in the increase the rate of fault detection. Thus the number of bugs uncovered from the software under test is improved. This leads to the improvement of quality of the software.

Index Terms

Software engineering, testing, test case prioritization, dependency structures

1. Introduction

Test suites can help detect faults in SUT. Provided this goal is achieved, there are many issues with it. For instance test suites when executed in particular sequence can provide chances to unearth more faults. It does mean that test suite prioritization can be used to optimize testing results or to uncover more hidden faults. In order to achieve this, it is possible to find dependency structure that can be used to priorities test suites. The functional test suites when subjected to prioritization can give effective test results that can help developers to rectify problems in SUT. Haidry and Miller [1] focused on the process of test suit prioritization. They used a hypothesis "dependencies among test cases can have their influence on the rate of fault detection". Thus the test case prioritization is given importance. It is a process of ensuring that the test cases are executed in proper sequence in order to achieve high rate of fault detection. The rate of fault detection is measured using the number of faults detected. As some tests should occur before other tests, it is essential to prioritize test cases so as to achieve optimal results [1]. Sample dependency structure can be visualized in Figure 1.

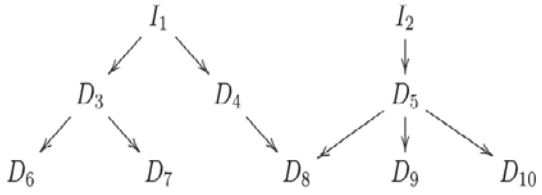

Fig.1 – Sample dependency structure

As seen in Figure 1, it is evident that the root nodes are independent of other nodes and they do not have dependencies. Dependencies are of two types namely direct and indirect. For example in Fig.2 D6 is a direct dependent of D3 and indirectly dependent on I1. Yet in another classification, dependency is of two types namely open dependency and closed dependency. Open dependency is the fact that a test case is executed before another one but need not be necessarily just immediately before the test case. The closed dependency is opposite to it where a test case needs to be executed immediately before the other test case. The combination of open and closed dependencies is also possible for optimal results. To measure dependencies, two measures are used. They are known as DSP height and DSP volume. Dependency Structure Prioritization volume refers to the count of dependencies while the DSP height indicates the depth in dependency levels. Direct and indirect dependencies are considered while computing DSP volume. On the other hand, the height of all test paths is considered for computing DSP height. The two graph measures are used for best ordering of test cases for optimal results. Experiments are made with open dependencies and closed dependencies. Many real time projects were considered for experiments. Out of them Bash is recorded to have highest dependencies and CRM1 and CRM2 recorded the lowest. Many SUTs were tested with prioritization of test cases.

Manuscript received April 5, 2015
Manuscript revised April 20, 2015

The experiments are useful to know the fault detection rate when dependency structures are used for prioritization. A measure used in [6] is known as Average Percentage of Faults Detected (APFD) for fault detection. The more APFD value is the more in the rate of detection of faults in SUT. All SUTs are tested with APFD measures under open and closed dependencies. Many DSP prioritization methods were considered and some other methods that do not use DSP measures can also be used in the process. The experimental results showed that DSP prioritization methods achieved higher results while non-DSP prioritization methods could not achieve high rate of fault detection. The empirical results proved the fact that DSP measures were able to increase the rate of fault detection for any given SUT. As explored in [1] there are many test case prioritization techniques. They are model – based [12], history – based [11] and knowledge-based [2]. The first one uses model of the system, second one uses past execution cycles, and third one uses human know how of the task for the purpose of test case prioritization. Our contributions in this paper are as follows.

We proposed a methodology for automatic discovery of dependency structures from SUIT. This methodology guides the program to obtain dependency structures and help in prioritization of test cases.

We proposed an algorithm that makes use of discovered dependency structures and prioritizes test cases automatically.

We evaluate the functions such as automatic discovery of dependency structures and also the test case prioritization with empirical study using the tool built to demonstrate the proof of concept.

The remainder of the paper is structured as follows. Section II reviews literature on the prior works. Section III presents the proposed methodology for automatic discovery of dependency structures and algorithm for prioritizing test cases. Section IV presents evaluation of the proposed work while section V provides conclusions and recommendations for future work.

2. Related Works

This section provides review of literature on prior works. In 1997 Wong et al. [1] proposed a hybrid approach for regression testing which uses the combination of approaches like minimization, modification, and prioritization-based selection. The purpose of regression testing is to ensure that changes made to software, such as adding new features or modifying existing features, have not adversely affected features of the software that should not change. Regression testing is usually performed by running some, or all, of the test cases created to test modifications in previous versions of the software. Many techniques have been reported on how to select regression tests so that the number of test cases does not grow too large as the software evolves. Our proposed hybrid technique combines modification, minimization and prioritization-based selection using a list of source code changes and the execution traces from test cases run on previous versions. This technique seeks to identify a representative subset of all test cases that may result in different output behavior on the new software version [1].

Ryser and Glinz [6] discussed about scenarios or use cases that can be used to capture requirements. The modeling tools such as UML also do not have scenario based dependencies. They opined that verification and validation are important activities in software development process. It is true in the case of test case generation and execution as well. They proposed a new model to find dependencies between scenarios. In this paper we focused on the dependencies among methods while [6] explored dependencies among the scenarios. Dependency charts were built in order to help test engineers to test the SUT in systematic fashion so as to discover more bugs. Elbaum et al. [11] focused on test case prioritization by considering fault severities and varying test costs. The regression testing can take the help of prioritization results in order to improve the possibilities of finding and fixing bugs. APFD measure is used to know the rate of fault detection. The previous uses of APFD were made when severities and test costs are uniform. In [11] a new technique is proposed in order to assess the rate of fault detection with prioritized test cases. Thus priority based reuse of test suits save more time to software engineers besides helping them in discovering more bugs. The new technique was an improved form of APFD that is based on test costs and the severities.

Rothermel et al. [2] focused on cost-effectiveness of regression testing with respect to test suite granularity. Since regression testing is an expensive test process, the cost can be reduced with the methods that are cost-effective. Towards this prioritization of test cases play an important role in order to make it less costly besides being able to discover more bugs. The bottom line of the research is to reduce the cost and also increase the rate of fault detection. Elbaum et al. [12] made a empirical study on test case prioritization. The aim of their research is to reduce the cost of regression testing. The end result expected is the same “increasing the rate of fault detection”. One potential goal of test case prioritization is that of increasing a test suite’s rate of fault detection—a measure of how quickly a test suite detects faults during the testing process. An improved rate of fault detection can provide earlier feedback on the system under test, enable earlier debugging, and increase the likelihood that, if testing is prematurely halted, those test cases that offer the greatest fault detection ability in the available testing time will have been executed [12].

Peirce's criterion is a rigorous method based on probability theory that can be used to eliminate data "outliers" or spurious data in a rational way. Currently, another method called Chauvenet's criterion is used in many educational institutions and laboratories to perform this function. Although Chauvenet's criterion is well established, it makes an arbitrary assumption concerning the rejection of the data. Peirce's criterion does not make this arbitrary assumption . In addition, Chauvenet's criterion makes no distinction between the case of one or several suspicious data values whereas Peirce's criterion is a rigorous theory that can be easily applied in the case of several suspicious data values. In this paper, an example is given showing that Peirce's and Chauvenet's criterion give different results for the particular set of data presented.[13]
Code prioritization for testing promises to achieve the maximum testing coverage with the least cost. This paper presents an innovative method to provide hints on which part of code should be tested first to achieve best code coverage. This method claims two major contributions. First it takes into account a "global view" of the execution of a program being tested, by considering the impact of calling relationship among methods/functions of complex software. It then relaxes the "guaranteed" condition of traditional dominator analysis to be "at least" relationship among dominating nodes, which makes dominator calculation much simpler without losing its accuracy. It also then expands this modified dominator analysis to include global impact of code coverage, i.e. the coverage of the entire software other than just the current function. We implemented two versions of code prioritization methods, one based on original dominator analysis and the other on relaxed dominator analysis with global view.[4].

Software engineers often save the test suites they develop so that they can reuse those test suites later as their software evolves. Such test suite reuse, in the form of regression testing, is pervasive in the software industry. Running all of the test cases in a test suite, however, can require a large amount of effort: for example, one of our industrial collaborators reports that for one of its products of about 20,000 lines of code, the entire test suite requires seven weeks to run. In such cases, testers may want to order their test cases so that those with the highest priority, according to some criterion, are run earlier than those with lower priority.[10].
Test case prioritization techniques have been shown to be beneficial for improving regression-testing activities. With prioritization, the rate of fault detection is improved, thus allowing testers to detect faults earlier in the system-testing phase. Most of the prioritization techniques to date have been code coverage-based. These techniques may treat all faults equally. We build upon prior test case prioritization research with two main goals: (1) to improve user perceived software quality in a cost effective way by considering potential defect severity and (2) to improve the rate of detection of severe faults during system level testing of new code and regression testing of existing code. We present a value-driven approach to system-level test case prioritization called the Prioritization of Requirements for Test (PORT). PORT prioritizes system test cases based upon four factors: requirements volatility, customer priority, implementation complexity, and fault proneness of the requirements. We conducted a PORT case study on four projects developed by students in advanced graduate software testing class. Our results show that PORT prioritization at the system level improves the rate of detection of severe faults. Additionally, customer priority was shown to be one of the most important prioritization factors contributing to the improved rate of fault detection [3].
Test engineers often possess relevant knowledge about the relative priority of the test cases. However, this knowledge can be hardly expressed in the form of a global ranking or scoring. In this paper, we propose a test case prioritization technique that takes advantage of user knowledge through a machine learning algorithm, Case-Based Ranking (CBR). CBR elicits just relative priority information from the user, in the form of pair wise test case comparisons. User input is integrated with multiple prioritization indexes, in an iterative process that successively refines the test case ordering. Preliminary results on a case study indicate that CBR overcomes previous approaches and, for moderate suite size, gets very close to the optimal solution [7].
Regression testing is an expensive part of the software maintenance process. Effective regression testing techniques select and order (or prioritize) test cases between successive releases of a program. However, selection and prioritization are dependent on the quality of the initial test suite. An effective and cost efficient test generation technique is combinatorial interaction testing, CIT, which systematically samples all t-way combinations of input parameters. Research on CIT, to date, has focused on single version software systems. There has been little work that empirically assesses the use of CIT test generation as the basis for selection or prioritization. In this paper we examine the effectiveness of CIT across multiple versions of two software subjects. Our results show that CIT performs well in finding seeded faults when compared with an exhaustive test set. We examine several CIT prioritization techniques and compare them with a re-generation/prioritization technique [14].
Test case prioritization techniques have been empirically proved to be effective in improving the rate of fault detection in regression testing. However, most of previous techniques assume that all the faults have equal severity, which dose not meets the practice. In addition, because most of the existing techniques rely on the information gained from previous execution of test cases or source code changes, few of them can be directly applied to non-

regression testing. In this paper, aiming to improve the rate of severe faults detection for both regression testing and non-regression testing, we propose a novel test case prioritization approach based on the analysis of program structure. The key idea of our approach is the evaluation of testing-importance for each module (e.g., method) covered by test cases. As a proof of concept, we implement A pros, a test case prioritization tool, and perform an empirical study on two real, non-trivial Java programs. The experimental result represents that our approach could be a promising solution to improve the rate of severe faults detection.[15]

Regression testing assures changed programs against unintended amendments. Rearranging the execution order of test cases is a key idea to improve their effectiveness. Paradoxically, many test case prioritization techniques resolve tie cases using the random selection approach, and yet random ordering of test cases has been considered as ineffective. Existing unit testing research unveils that adaptive random testing (ART) is a promising candidate that may replace random testing (RT). In this paper, we not only propose a new family of coverage-based ART techniques, but also show empirically that they are statistically superior to the RT-based technique in detecting faults [5]. Pair-wise comparison has been successfully utilized in order to priorities test cases by exploiting the rich, valuable and unique knowledge of the tester. However, the prohibitively large cost of the pair wise comparison method prevents it from being applied to large test suites. In this paper, we introduce a cluster-based test case prioritization technique. By clustering test cases, based on their dynamic runtime behavior, we can reduce the required number of pair-wise comparisons significantly. The approach is evaluated on seven test suites ranging in size from 154 to 1,061 test cases. We present an empirical study that shows that the resulting prioritization is more effective than the existing coverage-based prioritization techniques in terms of rate of fault detection [8].

3 . Methodology For Automatic Discovery Of Dependency Structures

The research on test case prioritization focused on various approaches as found in the previous section. For instance, they are based on execution traces [1], dependency charts that are derived through scenario-based testing [6], test costs and fault severities [11], test suite granularity and its impact on cost-effectiveness on regression testing [2], comparator techniques, statement level techniques and function level techniques [12], cost prioritization [4], fine granularity and coarse granularity [9], Prioritization of Requirements for Test (PORT) which is a value-driven approach [3], use case based ranking methodology [7], combinatorial interaction testing [14], analysis of program structure [15], adaptive random test case prioritization [5] and clustering test cases [9]. More recently Haidry and Miller [15] used dependency structures for test case prioritization. In this paper, we improve the approach used in [15] by discovering dependency structures automatically. The architecture of the proposed methodology is as shown in Figure 2.

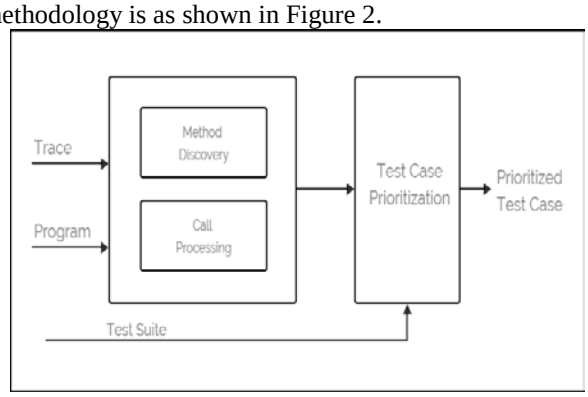

Figure 2. Proposed methodology

As can be seen in Figure 2, it is evident that the proposed methodology depends on program execution traces and the actual program. The method discovery process makes a list of all methods available and in fact the methods are discovered using reflection API. The call processing component is responsible to use traces and have some meta data associated with calls. This meta data is used later for test case prioritization. The test case prioritization component is responsible to understand the meta data associated with all calls and also considers test suite. It makes the final and best ordering of test cases. The prioritized test cases are thus produced by the proposed approach.

TCP (Test Case Prioritization) Algorithm

Input : Execution traces (***ET***) and program (***P***), Test Suite (***TS***)
Output: Prioritized test cases (***PT***)

1. Initialize a vector (***M***)
2. Initialize another vector (***MM***)
3. Discover methods from ***P*** and populate *M*
4. **for each** method ***m*** in ***M***
   a. scan TS
   b. associate meta data with calls
   c. add method ***m*** to vector MM
5. **end for**
6. for each ***mm*** in ***MM***
   a. analyze ***TS***
   b. correlate with ***mm***
   c. add corresponding ***m*** to ***PT***
7. return ***PT***

Algorithm for test case prioritization

As can be seen in listing 1, it is evident that the proposed method takes traces, program and test suite as input. It performs discovery of methods and automatic discovery of dependencies in the form of methods associated with meta data and finally performs prioritization of test cases in the given test suite.

4. Experimental Results

The tool implemented in our previous work has been extended to incorporate the functionality of the proposed methodology in this paper. The tool demonstrates the proof of concept and discovers dependency structures from given program. The tool can distinguish between open and closed dependencies as described earlier in this paper. The inputs and outputs are presented in this section besides the results of experiments. Open dependency related input program is as shown in Listing 2.

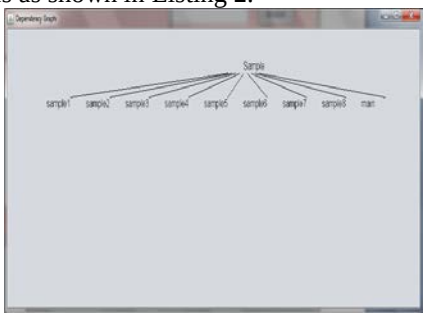

Figure3-Visualization of closed dependencies for given input file

As can be seen in Figure 3, it is evident that the closed dependencies are presented graphically. The closed dependencies as per the given input file are shown. The application can work for any input file so as to discover closed dependencies.

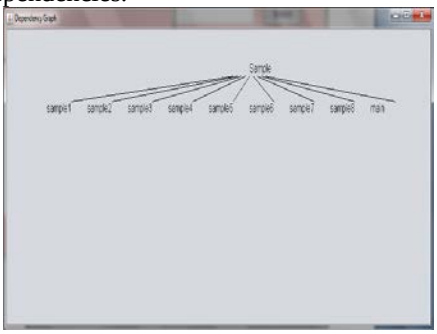

Figure 4 – Visualization of open dependencies for given input file

As can be seen in Figure 4, it is evident that the open dependencies are presented graphically. The open dependencies as per the given input file are shown. The application can work for any input file so as to discover open dependencies.

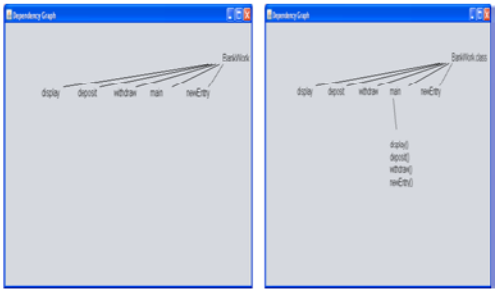

Figure 5 – Dependency discovery results

As can be seen in Figure 5, it is evident that the open and closed dependencies are presented graphically. The dependencies as per the given input file are shown. The application can work for any input file so as to discover open dependencies. The source code of these dependencies is found in appendix.

5. Evaluation

For evaluating our work specific procedure is followed as described here. First, the discovery of dependencies is done manually by human experts. The input file is shared with expert software engineers who have testing knowhow. The human experts studied the given inputs and provided their results which are done manually. Their results are saved and they reflect the ground truth. Later on our application is tested with same inputs. This process is continued for many Java applications to be tested. The results of manual discovery of dependencies (closed and open) are compared with the results discovered by our application. Around 100 times this evaluation of the application results by comparing with ground truth consistently resulted in the same. Thus 100% accuracy has been recorded by the application. When time is compared, human experts took 10 to 15 minutes to discovery dependencies in average while our application takes negligible time to show the dependencies.

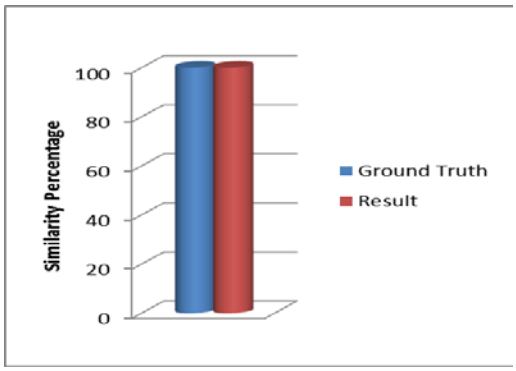

Figure 6– Performance comparison with ground truth

Many experiments proved that the automatic discovery of dependency structures do match with the ground truth and the tool has been extended to prioritize test cases automatically. In our previous paper we focused on test suite generation while this paper while this paper focused on automatic discovery of dependency structures for test case prioritization. More details on our tool will be presented in our next paper.

6. Conclusions and Future Work

Test case prioritization has its utility in improving the rate of fault detection in SUT. As test suite contains many test cases, the order of their execution plays an important role in increasing the rate of fault detection which can provide early feedback to development team so as to help them to improve the quality of the software. Therefore it is very useful to prioritize test cases that will lead to the increase in the rate of fault detection. In this paper we proposed a novel mechanism to discover dependency structures from SUT automatically and use them for prioritization of test cases. This work is very closer to that of Haidry and Miller. However, they did not automate the discovery of dependency structures. Dependencies are of two types namely direct and indirect. Both types are considered in this paper. We built a prototype application that demonstrates the proof of concept. The empirical results reveal that the automatic discovery of dependency structures can help in complete automation of test case prioritization. In future we integrate the whole test suite generation and test suite prioritization into a single tool that will help software engineering domain for automatic test case generation and test case prioritization.

REFERENCES

[1] W. Eric Wong, J. R. Horgan, Saul London, Hira Agrawal. (1997). A Study of Effective Regression Testing in Practice. IEEE. 8 (1), p 264-274.

[2] Greegg Rothermel, Sebastian Elbaum, Alexey Malishevsky, Praveen Kallakrui and Brian Davia (2011), The impact of test suite granularity on the cost-effectiveness of Regression Testing, University of Nebraska – Lincoln, p1-12.

[3] Hema Srikanth, Laurie Williams, Jason Osborne. (2000). System Test Case Prioritization of New and Regression Test Cases. Department of Computer Science. 2 (4), p1-23.

[4] J. Jenny Li. (2001). Prioritize Code for Testing to Improve Code Coverage of Complex Software. CID. p1-18..

[5] Jiang, B; Zhang, Z; Chan, WK; Tse, TH. (2009). Adaptive random test case prioritization. International Conference On Automated Software Engineering. 4 (24), p233-244

[6] Johannes Ryser. (2000). Using Dependency Charts to Improve Scenario-Based Testing. International Conference on Testing Computer Software TCS. 18 (3), p1-10.

[7] Paolo Tonella, Paolo Avesani, Angelo Susi. (1997). Using the Case-Based Ranking Methodology for Test Case Prioritization. IEEE.p1-10.

[8] C. Prakasa Rao, P. Govindarajulu. (2015). Genetic Algorithm for Automatic Generation of Representative Test Suite for Mutation Testing. IJCSNS International Journal of Computer Science and Network Security. 15 (2), p11-17.

[9] Shin Yoo & Mark Harman. (2003). Clustering Test Cases to Achieve Effective & Scalable Prioritisation Incorporating Expert Knowledge.ISSTA. 19 (23), p1-20.

[10] Sebastian Elbaum. (2000). Prioritizing Test Cases for Regression Testing. International Symposium of Software Testing and Analysis. p102-112

[11] Sebastian Elbaum. (2001). Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization. Proceedings of the 23rd International Conference on Software Engineering. 23 (5), p1-10.

[12] Sebastian Elbaum. (2002). Test Case Prioritization: A Family of Empirical Studies. CSE Journal Articles. 2 (1), p157-182.

[13] Stephen M. Ross, Ph.D.. (2003). Peirce's criterion for the elimination of suspect experimental data. Journal of Engineering Technology, p1-23.

[14] Xiao Qu, Myra B. Cohen, Katherine M. Woolf. (2006). Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization. Department of Computer Science and Engineering, p149-170.

[15] Zengkai Ma. (2001). Test Case Prioritization based on Analysis of Program Structure. Department of Computer Science, p149-170.

Prakasa Rao Chapati received Master of Computer Applications degree from Madras University and Master of Technology degree in Computer Science & Engineering from Acharya Nagarjuna University. He is a research scholar in the department of Computer Science, Sri Venkateswara University. His research focus is on Software Testing to improve the Quality under Software Project Management perspective.

P.Govindarajulu, Professor at Sri Venkateswara University, Tirupathi, has completed M.Tech., from IIT Madras (Chennai),Ph.D from IIT Bombay (Mumbai), His area of research are Databases, Data Mining, Image processing and Software Engineering