ISSN:0975-9646

Sanjukta Mohanty et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (3) , 2011, 1042-1047

A SURVEY ON MODEL BASED TEST CASE PRIORITIZATION

Sanjukta Mohanty[#1], Arup Abhinna Acharya[#2], Durga Prasad Mohapatra[*3]

[#]School of Computer Engineering, KIIT University,

Bhubaneswar, India

*Department of Computer Science Engineering
National Institute of Technology
Rourkela, India

***Abstract*— Regression testing is the process of validating modifications introduced in a system during software maintenance. As the test suite size is very large, system retesting consumes large amount of time and computing resources. This issue of retesting of software systems can be handled using a good test case prioritization technique. A prioritization technique schedules the test cases for execution so that the test cases with higher priority executed before lower priority. The objective of test case prioritization is to detect fault as early as possible. Test case prioritization becomes a challenge in Component-based Software System (CBSS) which facilitates development of complex systems by integrating the reusable components. CBSS has emerged as an approach that offers rapid development of system using fewer resources and effort. The core idea of reuse and reducing the development costs can be achieved if the components offer reliable services. Thus, integration of components and testing become an important phase in CBSS. Integration of components involves understanding communication and coordination between the components. Developers do not provide the sufficient information on these components. As a result of this, understanding of component interactions while integrating these components becomes a challenge. Testing components is a challenging area of research. There have been troubles integrating the components. This in turn affects the quality and reliability of the software. Our research aims at analysing the existing test case prioritization techniques in code based, requirement based and model based prioritization techniques and it' s implementation in CBSS . The systematic literature survey is based on nine articles collected from multiple-stage selection process.**

Keywords*— *Regression Testing, Test Case Prioritization, CBSS.

I. INTRODUCTION

Regression testing is the process of testing a modified system using the old test suites. Developers need to make sure that modifications are correct and do not adversely affect the unchanged portion of the system. During regression testing the modified parts of the system are first tested. Then the whole system needs to be retested using the old test suite to have confidence that the modifications did not introduce new faults into the system. Because of the large size of a test suite, system retesting tends to consume a large amount of time and computing resources; it may last for hours, or even days. So one of the issues developers face during retesting of the system is ordering test cases for execution. Test case prioritization tries to address this issue. Test case prioritization orders tests for execution so that the test cases with the higher priority, based on some criterion, are executed before lower priority test cases . Several test prioritization criteria are there. For example, tests can be ordered to achieve selected code coverage at the fastest rate[10]. There exist different types of test case prioritization methods: code-based test case prioritization and model-based test case prioritization [5]. In code-based test case prioritization, source code of the system is used to prioritize the test cases. Most of the test case prioritization methods are code based. In model-based test case prioritization a system's model is used to prioritize the test cases. System models are used to capture some aspects of the system behaviour. The model based test case prioritization may improve the early fault detection as compared to the code-based test case prioritization. Model-based test prioritization may be an inexpensive alternative to the existing code-based test prioritization methods. However, model-based test case prioritization may be sensitive to the correct/incorrect information provided by the testers/developers. Hence model-based test case prioritization is the best one compared to code-based test case prioritization [6].

In the recent past, Component Based Software System (CBSS) has gained a very high importance. This is attributed to the reduction of cost and time in building the software using reusable components. A component is a executable software having a published interface. The identified advantages of CBSS: Reduced lead time, enhanced quality. Developers are not provided with sufficient information on these components.

Reliance of components introduces problems of testing. Testing components is a challenging area of research. Overall our research objective is to find and scrutinize the current techniques and issues in model based test case prioritization techniques in CBSS. In this research paper we aim at conducting a systematic literature survey of code based test case prioritization, model based test case prioritization, requirement based test case prioritization, component interactions in component composition and model based test case prioritization in CBSS.

The rest of the paper is organized as follows: Section II describes the prioritization methodologies. Discussion containing the advantages and disadvantages of various prioritization are summarized in Section III. Section IV presents conclusion and future work.

II. PRIORITIZATION METHODOLOGY

A. Code Based Test Case Prioritization

Srivastava [1], suggested prioritizing test cases according to the criterion of increased APFD(Average percentage of Faults detected) value. He proposed a new technique which could be able to calculate the average number of faults found per minute by a test case and using this value sorts the test cases in decreasing order. APFD is calculated as:

$$\mathbf{APFD} = \frac{1-(TF_1 + TF_2 + \ldots\ldots + TF_m)}{nm} + \frac{1}{2n}$$

Where T = the test suite under evaluation,

m = the number of faults contained in the program under test P,

n = The total number of test cases and

TF_i = The position of the first test in T that exposes fault i.

An experiment was conducted in which the rate of fault detection for each test case is calculated and order of test suit is evaluated in decreasing order of the value of rate of fault detection. Than the APFD value is determined for both the prioritized and non prioritized test suite and it is shown that the APFD value of prioritized test suite is higher than the non prioritized test suite. As prioritized test suite is more effective due to higher APFD value than non prioritized test suite hence test case prioritization is necessary.

Rothermel et al. [2], have formally defined the test case prioritization problem and empirically investigated nine prioritization techniques. Four of the techniques are based on the coverage of either statements or branches for a program and two of the techniques are based on the estimated ability to reveal faults. Several experiments compared these with the use of no prioritization (untreated), random prioritization and optimal prioritization. The techniques are:

T1 : No prioritization.
T2 : Random prioritization.
T3: Optimal prioritization.
T4 : Total branch coverage prioritization.
T5 : Additional branch coverage prioritization.
T6:Total fault-exposing-potential (FEP) prioritization.
T7:Additional fault-exposing-potential(FEP) prioritization.
T8: Total statement coverage prioritization.
T9:Additional statement coverage prioritization.

The experimental results showed that the prioritization techniques can improve the rate of fault detection of test suites and empirically examined their relative abilities to improve how quickly faults can be detected by those suites. They investigated coverage-based prioritization by examining a wide range of prioritization techniques, and focused on general, rather than modified version-specific, prioritization.

Prashant et al. [3], discussed the regression test framework which orders test cases based on code coverage. First the test suite is selected then sequencing is done based on some criteria.
Tester marked the affected portion of code with help of business analyst and development team and select the test suite. The regression test prioritization implemented as Test Framework built over a standard test automation tool Rational Functional Tester. The prioritizer orders the test cases based on code coverage information like lines of code, methods, and blocks. The code coverage information is collected during actual execution by the framework and stored in repository. This data is analysed and used in prioritization during subsequent execution cycle(s). The framework is integrated with coverage analysis tool EMMA, an open source tool, which collects code coverage information of each test case during run time.
An experimental study has taken and the coverage effectiveness of prioritized and non prioritized test suite is analysed. The study shows that the prioritized test cases achieve greater coverage in earlier execution phase than the non prioritized test cases. The percentage code coverage decreases as the execution moves in case prioritized test suite. In non prioritized test suite execution, it varies period to period and depends on the test cases executed during that period.

Li et al. [4], focused on test case prioritization techniques for code coverage, including block coverage, decision (branch) coverage, and statement coverage. Five search techniques are studied: two metaheuristic search techniques (Hill Climbing and Genetic Algorithms), together with three greedy algorithms (Greedy, Additional Greedy, and 2-Optimal Greedy). An empirical study has been conducted and the result of the study compared the performance of the five search algorithms applied to six programs, ranging from 374 to 11,148 lines of code. For determining the effectiveness of different techniques different metrics are used. Depending on the coverage criterion, three metrics are considered :

1. **APBC** (Average Percentage Block Coverage). This measures the rate at which a prioritized test suite covers the blocks.
2. **APDC** (Average Percentage Decision Coverage). This measures the rate at which a prioritized test suite covers the decisions (branches).
3. **APSC** (Average Percentage Statement Coverage). This measures the rate at which a prioritized test suite covers the statements.

This paper described five algorithms for the sequencing problem in test case prioritization for regression testing. It presented the results of an empirical study that investigated their relative effectiveness. The data analysis indicate that the Greedy Algorithm performs much worse than Additional Greedy, 2-Optimal, and Genetic Algorithms overall. Also, the 2-Optimal Algorithm overcomes the weakness of the Greedy Algorithm and Additional Greedy Algorithm. However, the experiments indicate that, in terms of effectiveness, there is no significant difference between the performance of the 2-Optimal and Additional Greedy Algorithms. This suggests that, where applicable, the cheaper-to-implement-and execute additional Greedy Algorithm should be used.

B. MODEL BASED TEST CASE PRIORITIZATION

Korel et al. [5], performed a small experimental study in order to compare simple code-based and model-based test prioritization methods. The objective of this study was to experimentally evaluate these methods with respect to the effectiveness of early fault detection in the modified system. The code based test prioritization techniques includes:

- Total Statement Coverage,
- Total Function Coverage,
- Additional Statement Coverage,
- Additional Function Coverage

In this paper the Additional Statement Coverage Prioritization was selected, referred to as *Heuristic #1*. The model-based test prioritization used a technique known as *Heuristic #2,* which is based on *marked transitions (w*hen the modifications to the source code are made, developers identify model elements that are related to these modifications and such transitions are referred as *marked transitions*.) criteria. The experimental study has shown that model based test prioritization may improve the early fault detection as compared to the code-based test prioritization because the execution of the model is very fast as compared to the execution of the actual system. Therefore, execution of the model for the whole test suite is relatively inexpensive, whereas execution of the system for the whole test suite, as required by some code-based test prioritization methods, may be expensive (both resource-wise and time-wise). Model-based test prioritization may be sensitive to the correct/incorrect information provided by the testers/developers.

Korel et al. [6], prioritized the test cases by using several model-based test prioritization heuristics. The existing model based test prioritization methods can only be used when models are modified during system maintenance. But they presented model-based prioritization for a class of modifications for which models are not modified (only the source code is modified). An experimental study has been conducted to investigate the effectiveness of those methods with respect to early fault detection. The results from the experiment suggested that system models may improve the effectiveness of test prioritization. Several model-based test prioritization heuristics are:

- selective prioritization,
- Heuristic #1 prioritization,
- Heuristic #2 prioritization,
- Heuristic #3 prioritization, and
- model dependence-based prioritization,

where Heuristics #1, #2 and #3 have been developed for modifications with multiple-marked transitions.

C. REQUIREMENT BASED TEST CASE PRIORITIZATION

Srikanth et al.[7], developed a prioritization scheme with three main goals: identifying the severe faults earlier, to improve the software field quality and to devise the minimal set of PORT(Prioritization of Requirements for Testing) PFs (Prioritization Factors) that can be used to effectively for test case prioritization.

Current Test Case Prioritization schemes are enhanced by incorporating additional knowledge gained through requirements engineering research:

(a) requirements with high complexity tend to have a higher number of faults ,

(b) requirements volatility, which results in re-design, addition or deletion of existing requirements, tend to increase project risk , and fault density , thus often times causing project failures ,

(c) roughly 20% of the system is responsible for about 80% of the faults .

Here the main criteria used is prioritization factors. For measuring the PFs a equation is used i.e

$$WP = \sum_{PF=1}^{n} (PF_{value} * PF_{weight}) \quad \text{.....(1)}$$

Based on the project and customer needs, the development team assigns weight to the PFs such that the assigned total weight (1.0) is divided amongst the PFs. For every requirement, the above equation is used to calculate a weighted prioritization (WP) factor that measures the importance of testing a requirement earlier. Test cases are then ordered such that the test cases for requirements with high WP are executed before others.

Acharya et al. [8], generated test cases for testing components in component composition technology. They modelled component interactions using a Component Interaction Graph (CIG) which depicts interaction scenarios among components. A new algorithm is applied on state chart diagrams to construct CIG. An example Vending Machine which contains a component Dispenser has taken for better understanding of

component interactions. Dispenser provides an interface that is used by Vending Machine. Vending Machine uses the services provided by Dispenser to manage credits inserted into the vending machine, validate selections, and check for availability of requested items.

Wu et al. [9], presented a test model that depicts a generic infrastructure of component based systems and suggested key test elements. The test model is realized using a Component Interaction Graph (CIG) in which the interactions and the dependence relationships among components are illustrated. By utilizing the **CIG,** the authors proposed a family of test adequacy criteria which are as follows:

- All-interfaces coverage
- All-events coverage
- All-context-dependence coverage
- All context/some-content-dependence coverage
- All-content-dependence coverage

To explore the potential of test model a case study has been conducted to investigate the possible usages of these elements in testing and to compare their relative strengths. The methodology proposed is efficient and effective, as demonstrated by promising results obtained from a case study.

III. DISCUSSION

The disadvantage of Srivastava [1], is calculation of APFD is only possible when prior knowledge of faults is available. APFD calculations therefore are only used for evaluation.

Rothermel et al. [2], discussed the effectiveness of different techniques. To measure the effectiveness of these techniques, an experiment was conducted where 7 different programs were taken, for each subject program , the prioritization techniques T2 through T9 were applied to each of the 1000 sample test suites, yielding 8000 prioritized test suites. The original test suite (not reordered) was retained as a control; for analysis this was considered "prioritized" by technique T1 . The APFD values of these 63000 prioritized test suites were calculated and used as the statistical data set. It was shown that additional FEP prioritization outperformed all prioritization techniques based on coverage. Furthermore, total FEP prioritization outperformed all coverage-based techniques other than total branch coverage prioritization. However, these results did vary across individual programs and, where FEP-based techniques did outperform coverage-based techniques, the total gain in APFD was not great. These results suggested that FEP-based prioritization may not be as cost-effective as coverage-based techniques. Again considering overall results, total branch coverage prioritization outperformed additional branch coverage prioritization and that total statement coverage prioritization outperformed additional statement coverage prioritization. These effects, too, vary across the individual programs. Nevertheless, the worst-case costs of total branch and statement coverage prioritization are much less than the worst-case costs of additional branch and statement coverage prioritization; this suggests that the less expensive total-coverage prioritization schemes may be more cost-effective than additional-coverage schemes. Another effect worth noting is that generally (on five of the seven programs) randomly prioritized test suites outperformed untreated test suites. We conjecture that this difference is due to the type of test suites and faults used in the study. Random prioritization essentially redistributes test cases that reach and expose these faults throughout the test suites, causing the faults to be detected more quickly. However the disadvantages of all the techniques are based on execution of the number of marked transition like the test case which covered maximum marked transition are given higher priority. So considering only one criteria may not significantly improve the early fault detection.

The experimental results indicate that some types of information about models may not improve the effectiveness of early fault detection. In addition, the results have shown that some simple heuristic methods can be as effective in early fault detection as more complex ones. More importance is given to coverage based prioritization like statement coverage, branch coverage etc.

The advantage of Prashant et al. [3], is that a test prioritizer(test frame work) as extension to Test Automation tool is taken which is practically implemented. This paper also presents the results of empirical studies that evaluate test prioritizer. However the disadvantage is that the results may vary if prioritization is done using different criteria like methods, block, classes or its combination. If different tool is used for coverage analysis, result of test prioritization may vary as different techniques are used by the tool for measuring coverage. The performance of Automated test execution is reduced as testing is done on instrumented version of application. It is an expensive one as this technique is totally based upon code coverage.

Li et al. [4], studied metaheuristic and evolutionary algorithms empirically for test case prioritization which are the most efficient for regression testing. Which depicts the advantage of work done.

The disadvantage of Korel et al. [5], is no extensive study has performed. The experimental study presented in this paper was limited to two test prioritization heuristics only.

Korel et al. [6], found out the effectiveness of different techniques. To compare different test prioritization methods, the concept of the *most likely relative position*, $RP(d)$, is used. $RP(d)$ is a metric that is used during an experimental study and represents an average (most likely) relative position of the first failed test that detects d for a test prioritization method. The results from the experimental study indicate that some model-based test prioritization methods may improve on average the effectiveness of early fault detection as compared to random prioritization. The effectiveness of different Heuristics are measured by $RP(d)$ metric and it is shown by box plots. The best performance is shown by the model-based test prioritization (IP) and Heuristic #3. Effectiveness of

Heuristic #1 and Heuristic #2 prioritization methods is comparable with the selective prioritization because Heuristic #1 and #2 prioritization did not perform well compared to the selective prioritization. These results may suggest that considering only the number of executions of marked transitions may not have a significant influence on the improvement of the early fault detection. For modifications that involve one marked transition, model dependence-based prioritization outperforms Heuristic #3. For modifications with one marked transition, performance of Heuristic #3 is equivalent to the selective prioritization. For modifications with multiple-marked transitions, Heuristic #3 slightly outperforms the model dependence based prioritization. However, since Heuristic #3 is much simpler it may be more preferable for these types of modifications.

The advantage of Srikanth et al. [7], is PORT could improve the effectiveness of testing activities as it (1) reduces the effort utilized for TCP in comparison to coverage-based techniques that prioritize based on the number of statements or branches covered, and (2) focuses on functionalities that are of highest value to the customer, and (3) improves the rate of detection of severe faults. Rectifying severe faults earlier is believed to improve perceived software quality.

Acharya et al. [8], discussed the following advantages. Although much work has been proposed for building component-based systems, techniques for testing component-based systems have not been well developed. In this article, they have presented a new approach for testing component based software and the empirical studies show that testing component based software is necessary yet expensive. The technique they proposed includes several criteria for determining test adequacy. However the disadvantage is after the composition a new test case set for the composed component no prioritization technique is developed to prioritize the newly generated test cases.

Wu et al. [9], discussed about the different criteria capable detecting the faults. Test case selections based on the *all-interfaces, all-events* criteria are simple and efficient, however they can provide only a certain level of reliability. To further improve the quality of the system, *allcontext/some-content-dependence* criterion is necessary**.** The *all-context/ some-content-dependence* criterion is capable of detecting the majority of these faults, though the *all-content-paths* criterion can be used to detect more faults. The result demonstrates that after enforcing the *allcontext/some-content-dependences* criterion, it is not only detected 84% of the faults but also found 3 new faults in the system. Moreover the test effort they spent is only 41% of effort required by the functional testing approaches. The disadvantage of Wu et al. [9], is to investigate the efficiency of the technique more experiments are needed to make it useful in the real world. However the advantage of this article is, the technique proposed in this article, includes several criteria for determining test adequacy. The all-context/ some-content-dependence criterion used only 41% of test cases yet detected 84% of the faults; even the weakest criterion, the all-interface, used 26% of test cases and detected 26% of the faults. Therefore, the strengths of the technique can be expected. This method, which can be applied to all types of component-based systems, does not rely on the knowledge of source code.

IV. CONCLUSION AND FUTURE WORK

In our study of exiting testing techniques research is focused on techniques based on codes, models and UML state chart diagrams. For determining the effectiveness of prioritization techniques two metrics (APFD and RPd) are used. There is good coverage in terms of research in understanding the concepts of different code based techniques and behaviour of components, interactions and compatibility of components. In future more numbers of criteria may be included like number of state changes by a test case during component interactions ,and during the state changes , the number of times ,the test case is going to access the data base, as whether it is going to access single attribute or multiple attributes from a database schema. Further as optimization technique like Genetic Algorithm (GA) may be applied over our proposed technique to explore a more effective prioritized test suite.
The proposed frame work is diagrammatically represented in Fig.1

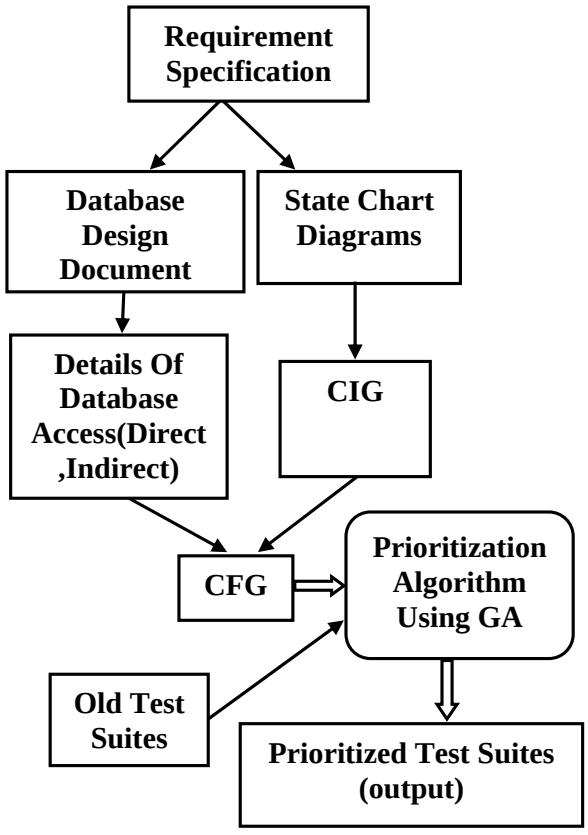

Fig.1. A frame work for prioritization of test cases

REFERENCES

[1] P. R. Srivastava, "Test Case Prioritization," Journal of Theoritical And Applied Information Technology, JATIT 2008 .

[2] G. Rothermel, R.H.Untch, C.Chu ,M.J.Harrold, "Test Case Prioritization:An Emperical Study," in Proceedings of the 24th IEEE International Conference Software Maintenance (ICSM '1999) Oxford, U.K, September 1999 .

[3] P. Malangave, D. B. Kulkarni, "Efficient Test case Prioritization in Regression Testinng".

[4] Z.Li, M. Harman, and R. M. Hierons," Search Algorithms for Regression Test Case Prioritization," IEEE Transactions On Software Engineering, Vol. 33, No. 4, April 2007.

[5] B. Korel, G. Koutsogiannakis, "Experimental Comparsion of Code Based and Model model Based Test prioritization," IEEE 2009.

[6] B. Korel, G. Koutsogiannakis, and L.H.Tahat, "Application of System Models in Regression Test Suite Prioritization," in Proceedings of the 24th IEEE International Conference Software Maintenance (ICSM '08) pp.247-256, 2008.

[7] Hema Srikanth and Laurie Williams, "Requirements-Based Test Case Prioritization," North Carolina State University, ACM SIGSOFT Software Engineering, pages 1-3, 2005.

[8] A. A. Acharya, S. K. Jena, "Component Interaction Graph: A new approach to test component composition," Journal of Computer Science and Engineering, Volume 1, Issue 1, May 2010.

[9] Y. Wu, D. Pan and M. Chen. "Techniques for testing component-based software," In Proceedings of the 7th IEEE International Conference on Engineering of Complex Computer Systems. pp. 222-232, 2001 .

[10] G. Rothermel, R. Untch, M. Harrol, "Prioritizing Test Cases For Regression Testing," IEEE Transactions on Software Engineering, volume 27, No. 10, pp. 929-948, 2001.