International Journal of u- and e- Service, Science and Technology
Vol.8, No. 6 (2015), pp.1-08
http://dx.doi.org/10.14257/ijunesst.2015.8.6.01

Improving Test Efficiency through Prioritization Based On Testing Dependency

Na Zhang[1], Yangqiu Liu[1], Xiaoan Bao[1], Xiaoming Xie[1], Zhongtao Ren[1] and Hui Lin[1]

1. Zhejiang Sci-Tech University, Hangzhou 310018, China
zhangna@zstu.edu.cn

Abstract

During the various iterations of software development, test case prioritization generally schedules test cases in order to increase test efficiency as soon as possible without reducing the scale of the test. It is noted that currently, many prioritization techniques are usually based on the assumption that the test cases are independent so as to reduce testing effort. However in the actual test, the dependencies and relations exist between the test cases. In this paper, we propose a new type of prioritization technique based on the relationship of dependencies between test cases. We gather the dependency information of a test suite and then put forward a weighted depth-first search algorithm to determine the most effective order. To assess our approach, an academic feasibility study and three industrial experiments are conducted. Evaluation results indicate that our proposed method can be used to improve the fault detection effectiveness when compared with random ordering and some existed prioritization techniques which are based on function coverage.

Keywords: *software testing and reliability; test case prioritization; dependency-based testing.*

1. Introduction

With the scale and complexity of software system becoming larger and larger, the problem of software reliability which attracts developers, is a typically significant research. Software testing, aiming at finding errors and establishing the confidence of software quality, is a strenuous and expensive process consuming at least 50% of the total software cost [1]. To increase the effective of test case maintenance within limited time and resources, test case prioritization can be performed.

To date, a large number of prioritization techniques have been put forward by research institutions. In order to provide earlier feedback and earlier defect fixing to testers, test case prioritization techniques [2-4] reorder tests cases, scheduling tests cases with the highest priority to achieve some performance goals earlier in the testing process. It is also reported that these studies can significantly increase the effectiveness of testing and improve the rate of fault detection as early as possible.

While this research has made considerable progress in software testing, one attractively crucial problem has been overlooked. The dependencies and relations between test cases, especially in functional test suites, should be taken into consideration for executing the test correctly and appropriately. It is likely that these dependencies are closely related to the coupling and interactions between the parts making up software systems. Rothermel, *et al.*, [5] provides a comprehensive hypothesis that testing the parts of the system which includes more complexity as soon as possible may improve the defect detection rate. Therefore, the method that

ISSN: 2005-4246 IJUNESST

assigning the test case with the more dependencies to the higher priority may increase the likelihood of discovering faults earlier in the software testing cycle.

We propose to address this lack by creating and empirically studying prioritization strategies based on the relationship of dependencies between test cases, which we call dependency-based prioritization (DBP). We collect some information about testing dependency, and adopt a weighted depth-first search algorithm to find out an order with the greatest rate of fault detection. Numerical results based on some experiments show that, whether previous test executing information is available or not, our proposed approach can effectively improve the effectiveness of testing without analyzing the source code.

The rest of the paper is structured as follows: Section 2 presents the brief summary of Background and related work. Next, our proposed approach is presented and discussed in Section 3. Some experiments will be performed and numerical results are analyzed in Section 4. Finally, a summary is provided in Section 5.

2. Related Work

In this section, we present an overview of background work of current test case prioritization techniques in Section 2.1 and using dependencies in the field of software testing to improve test efficiency in Section 2.2.

2.1. Test Case Prioritization

Numerous diversity approaches of test case prioritization have been widely studied by many researchers so far. Coverage-based prioritization techniques prioritize test cases according to the quantity of statements or functions executed by the test cases. Jeffrey and Gupta [6] assign the priority of the test case in the light of relevant slices during test runs. Test cases are ranked based on the number of program code blocks written by Li [7]. In addition, Li, *et al.*, use this prioritization technique to automatically generate execution sequence of test suites in later work [8]. It is commonly believed that a greedy algorithm, which is divided into Total strategy and Additional strategy, is generally used for solving coverage-based prioritization problems, including the above-mentioned two literatures. Total strategy sorts the test with the highest coverage, but additional strategy chooses the one not covered by all tests which have already been sorted. Empirical study in Section 4 selects both greedy prioritization strategies for comparison, because they are similar to our proposed approach prioritized by executing parts of the software that interacts with other parts.

Apart from coverage-based prioritization techniques resemblance to our proposed method, knowledge-based prioritization and model-based prioritization all exist related works in great amount just like our technique. In terms of the former, Qu, *et al.*, [9] define a systematic technique called combinatorial interaction testing for selecting parts of test case interactions to solve the explosion problem. For the purposes of the latter, Kundu, *et al.*, [10] adopts UML sequence diagrams as system models and translates them into sequence graphs, in which the order of testing is acquired by picking all basic paths. In a word, the more interactions between different software components, the higher prioritization of that test case.

2.2. Software Dependency

To date, studies for the relationship of dependencies have already been spread all aspects of software development, such as the dependencies between scenarios [11], software components [12], system feature [13] and so on. Ryser and Glinz [11] introduce a new type of chart and notation, which is called dependency chart, to

model dependencies between scenarios and use dependency charts based on scenarios in testing to support testers to systematically develop test cases for system test. Abate, *et al.*, [12] introduce strong dependency as first step towards modeling semantic rather than syntactic relationships, and define a notion of component sensitivity derived from strong dependency. Both of these works demonstrate that the techniques based on software dependency are applicable for improving the efficiency of software development. What's more, Wang Pei, *et al.*, [14] implement a precise and scalable tool which extracts code dependencies and their utilization for large C/C++ software projects. The tool extracts both symbol-level and module-level dependencies of a software system and identifies potential underutilized and inconsistent dependencies. This work can help developers perform large-scale refactoring tasks.

3. Prioritization Technique Based on Dependencies between Test Cases

In this section we present the proposed prioritization technique and the corresponding algorithm for solving the problem of ordering the test cases.

3.1. Dependency Structure

A dependency structure is typically specified by a directed acyclic graph (DAG), G = (V, E), where V is a set of nodes and E is a group of the arcs between these nodes. In this paper, V represents a set of test cases, and E shows the dependencies between test cases.

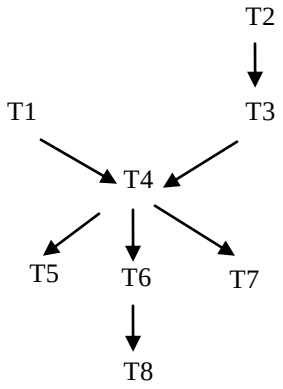

Figure 1. Example of Dependency

As shown in Figure 1, a dependency structure consists of several test cases correlated with each other, such as the tests from T1 to T8. Except for no dependencies between T1 and T2, the nodes T3 to T8 are all dependent on other nodes. Moreover, the relationship of dependency is divided into direct and indirect one. For instance, T8 depends on T6 directly, but dependent upon T4 indirectly.

3.2. Applying Dependency Structure to Prioritization Strategy

Taking dependencies between test cases into account, the process of testing can be regarded as searching for the routes of a dependency structure graph defined in Section 3.1, and the routes represent for the executing sequence of a test suite. Firstly, the metric for measuring the degree of dependency is defined in Section 3.2.1. In next section the priority of test cases is calculated by using a weighted

depth-first search algorithm in which the weighting function is just the metric defined in Section 3.2.1.

3.2.1. DBP Sum: In order to measure the level of dependency, we define a quantitative metric called DBP sum, which indicates that the more number of dependents of the test case included in a path, the higher priority assigned to this test case.

To calculate the DBP sum of a test case, we need to calculate the sum of all length of the paths started from the test case. It happens that Warshall's transitive closure algorithm is used to calculate all available paths of a directed graph expressed by the form of a adjacency matrix.

As shown in algorithm 1, where G[i , j] is a Boolean adjacency matrix of dependency structure, representing the direct dependency between test cases; D[i , j] represents indirect dependencies between test cases.

Algorithm 1. Route calculation based on transitive closure algorithm

1: $D_0[i, j] = G[i, j]$
2: **for** $k := 1$ *to* n
3: **for** $i := 1$ *to* n
4: **for** $j := 1$ *to* n
5: $D_k[i, j] = D_{k-1}[i, j]\ or(D_{k-1}[i,k] + D_{k-1}[k, j]);$
6: **return** D_n

Through the above-mentioned algorithm we can obtain the information that whether the path between any nodes is connected. The sum of the i th row in the matrix is the value of dependency metric called DBP sum and also is the weighted function w[t].

3.2.2. Prioritization Sorting Algorithm: Our proposed prioritization sorting algorithm is shown in Algorithm 2 and used to schedule the test cases in a test suite based on dependency structures. This algorithm uses a weighted depth-first search algorithm to order the test cases, in which the weighted function w[t] is just the value of DBP sum calculated by Algorithm 1.

Algorithm 2. Prioritization sorting algorithm

```
1.  Boolean visited[MAX];
2.  Status (*VisitFunc)(int v);
3.  void DFSTraverse(Graph G,Status (*Visit)(int v)){
4.        VisitFunc = Visit;
5.        for(v = 0; v < G.vexnum; + + v) visited[v] = FALSE;
6.        for(v = 0; v < G.vexnum; + + v)
7.           if(!visited[v]) DFS(G,v);   }
8.  void DFS(Graph G,int v){
9.        p = 0; if(p == 0)v = 0; else v = y; p + +;
10.       visited[v] = TRUE; VisitFunc(v); x = 0;
11.       for(t = FirstAdjVex(G, v); t ≥ 0 ; t = NextAdjVex(G, v, w))   {
12.            if (!visited[t])z=1;else z=-1;
13.            if(w[t]* z > x){x = w[t]; y = t;}            }
14.       for(t = FirstAdjVex(G, y); t ≥ 0 ; t = NextAdjVex(G, y, t))
15.            if (!visited[t]) DFS(G,t);       }
```


As can be seen from Algorithm 2, the test case obtained by the highest weighted value is executed firstly, followed by the test case dependent on the first one in which its weighted value is highest. If the weighted value of two test cases is the same, the choice is arbitrary. If the test cases which are dependent on the one with the highest value, have all been executed, the following one should be the highest weighted value among remained test cases.

The obvious advantage of depth-first search is that the parts of software system with more interactions are tested early for the purpose of increasing the rate of defect detection. The reason is that the more dependencies and relations contained by the parts, the more potential faults discovered by testers.

4. Empirical Comparisons and Evaluation

4.1. Experiment Setup

First of all, dependent on the requirements or design information of the system and the tests themselves we can extract dependent relationship between test cases to draw the dependency structure graph. Correspondingly, the test dependency matrix can be created for the dependencies of every test case of every system according to its dependency graph.

Secondly, the test fault matrix should be built for calculating the value of APFD, which is defined by Rothermel, *et al.,* [5] and measures how quickly a test suite detects faults are. Collecting this fault information from defect reports is available.

In the end, generate stochastically hundreds of dependency graph and calculate statistically the APFD and test cost per unit for each prioritization technique.

4.1.1. Selected Systems: The characteristics of the selected systems used in the experiment are listed in Table 1. As can be seen from the column named lines of code, these three systems not only contain the small-scale program like GSM1, but also include medium-sized programs like CZT and Bash. It is well-known that GSM is a file system with security permissions for mobile devices. The CZT system is the parser and type checker for the Community Z tools package, obtained from Sourceforge. Bash is a Unix system downloaded from the Software-artifact Infrastructure Repository (URL:http://sir.unl.edu). The dependencies and relations can be extracted indirectly from the functional information and documents for requirements design or directly from the developers.

Table 1. Characteristics of the Selected Systems

	Lines of code	Faults	Tests	Dependencies
GSM1	385	15	51	65
GSM2	975	14	51	65
CZT	27246	27	548	314
Bash	59800	6	1061	461

4.1.2. Effectiveness Comparison and Discussion: In order to evaluate the performance of our proposed technique, this paper picks up some other test case prioritization techniques for comparison. The features about these techniques are listed in Table 2. Some descriptions about these techniques are given as follow:

Table 2. Descriptions of Selected Prioritization Techniques

Abbreviation	Descriptions
Random	Run test cases randomly
Untreated	Order in an original sequence
Optical	Order with the most effectiveness for the prioritization problem
Total-fun	Executing test cases according to function coverage
Add-fun	Executing test cases according to updated function coverage
BFS	Prioritize by breadth-first search algorithm instead of depth-first search algorithm
DBP	Our proposed prioritization technique

1. Random prioritization (Random): Random prioritization orders randomly the test cases in the existing test suite.

2. Untreated ordering (Untreated): The ordering is the original sequence of the test cases not executed previously.

3. Optical prioritization (Optical): Prioritizing the test cases according to fault-finding ability except for function coverage proposed by Rothermel, *et al.*, [15].

4. Total function coverage prioritization (Total-fun): Order the test cases according to the function coverage. The more functions contained in the test case, the higher priority given to this test case [16].

5. Additional function coverage prioritization (Add-fun): It is similar to total function coverage except for executing test cases based on the functions not covered yet. Every time when calculating the priority of a test suite, we need recalculate function coverage for all tests after each greedy selection [17].

6. Breadth-first search (BFS): Take the place of depth-first search algorithm to sort test cases with unchanged weighted function compared with the empirical effect of our proposed technique.

7. Dependency-based prioritization (DBP): This is our proposed prioritization technique in this paper.

4.2. Selected Prioritization Techniques in Comparison

In this section, we present the results of experiments outlined in Section 4.1.

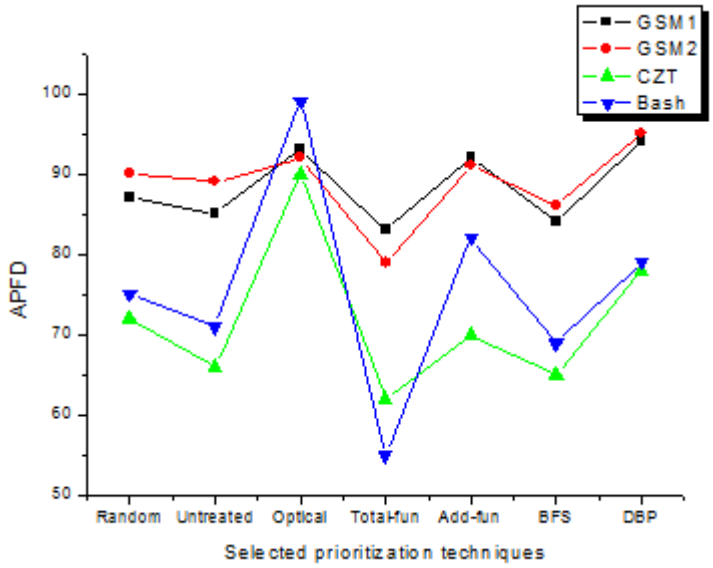

Figure 2. The Change of APFD Values through All Systems

Let's consider the mean APFD values of the prioritization techniques in the experiment. As seen from Figure 2, comparing DBP with random, untreated, BFS and Total-fun techniques, DBP has a significant improvement on mean APFD in all experiments. BFS performs worse than Random because executing the whole test scenarios seems to probably increase the rate of defect detection. For GSM1 and GSM2, DBP shows a little higher APFD value in comparison with Optical. However, Optical is better than DBP for the CZT and Bash systems, which is due to the small number of faults. Add-fun sometimes achieves a high rate of fault detection in contrast with the DBP technique.

Next we compare and discuss the efficiency of testing from the perspective of the cost of test resources per unit. From Figure 3, we find that DBP has a lower cost than Untreated, BFS and Random. In contrast with coverage-based prioritization techniques such as Total-fun and Add-fun, DBP shows a higher performance on mean test cost per unit. Optical performs well due to the same reason that the number of faults obtained by the CZT and Bash systems relatively is small.

Taken both metrics into account, our proposed prioritization technique shows a higher rate of defect detection with a lower test cost in contrast with prioritization techniques based on function coverage.

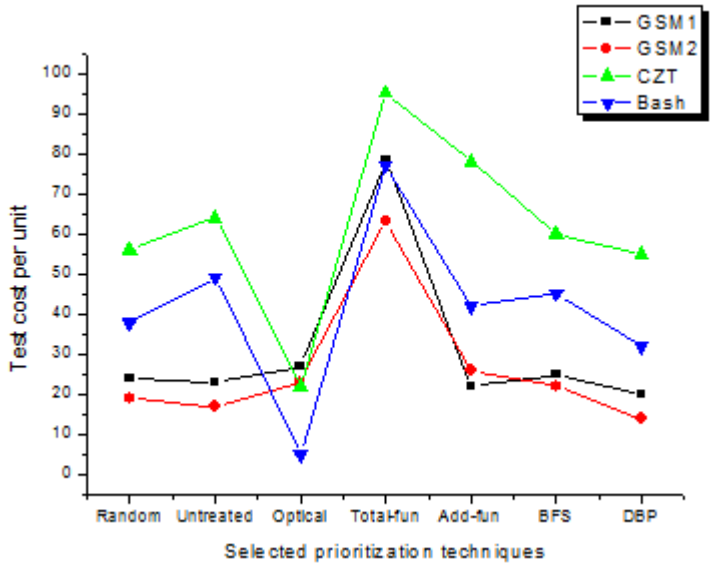

Figure 3. The Change of Test Cost per Unit through All Systems

5. Conclusion

In this paper, we have investigated a new prioritization technique based on the dependencies and relations between test cases to improve test efficiency. Compared with other coverage-based and knowledge-based prioritization techniques, the most significant advantage of our proposed method is that it can reduce the effort of analyzing the source code and obtain the information of previous test executing results. The proposed prioritization technique is validated through three industrial systems. Results demonstrate that our proposed approach leads to improve the rate of defect detection in comparison to randomly ordering of test cases and some prioritization techniques based on function coverage.

The ongoing research includes the following works. Metric expansion: adopting other metrics to measure the level of dependency such as dependency depth represented for the longest route in a dependency graph. Combination with other prioritization techniques: applying our proposed method to the test cases linked with

each other while using other prioritization strategies between unconnected test cases.

Acknowledgments

This work was supported by National Natural Science Foundation of China (61202050/61379036), Zhejiang Provincial Natural Science Foundation of China (Y13F020175), Zhejiang Talent Project (2013R10005), the New-shoot Talents Program of Zhejiang province(2014R406073), Zhejiang High-skilled Talent Project(2013R30001) and 521 talent project of ZSTU.

References

[1] G. Rothermel, R. H.Unteh, C. Chu and M. J. Harrold, "Prioritizing Test Cases For Regression Testing", IEEE Transaetions on Sonware Engineering, **(2001)**.
[2] S. Elbaum, A. Malishevsky and G. Rothermel, "Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization", Proceedings of the 23rd International Conference on Software Engineering, **(2001)** May 12-19, Ontario, Canada.
[3] A. Srivastava and J. Thiagarajan, "Effectively prioritizing tests in development environment", Proceedings of the International Symposium on Software Testing and Analysis, **(2002)**.
[4] W. E. Wong, J. R. Horgan, S. London and H. Agrawal, "A study of effective regression testing in practice", Proceedings of the 8th International Symposium on Software Reliability Engineering, **(1997)** November 2-5, Albuquerque, NM.
[5] G. Rothermel, S. Elbaum, A. Malishevsky, P. Kallakuri and B. Davia, "The Impact of Test Suite Granularity on the Cost-Effectiveness of Regression Testing", Proceedings of the 24rd International Conference on Software Engineering, **(2002)** May 25-25, Orlando, FL, USA.
[6] D. Jeffrey and N. Gupta, "Experiments with Test Case Prioritization Using Relevant Slices", J. Systems and Software, vol. 81, no. 2, **(2008)**, pp. 196-221.
[7] J. Li, "Prioritize Code for Testing to Improve Code Coverage of Complex Software", Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, **(2005)** November, pp. 1-1, Chicago, IL.
[8] J. Li, D. Weiss and H. Yee, "Code-Coverage Guided Prioritized Test Generation", J. Information and Software Technology, vol. 48, no. 12, **(2006)**, pp. 1187-1198.
[9] X. Qu, M. B. Cohen and K. M. Woolf, "Combinatorial Interaction Regression Testing: A Study of Test Case Generation and Prioritization", Proceedings of IEEE International Conference on Software Maintenance, **(2007)** October 2-5, Paris.
[10] D. Kundu, M. Sarma, D. Samanta and R. Mall, "System Testing for Object-Oriented Systems with Test Case Prioritization", Software Testing, Verification, and Reliability, vol. 19, no. 4, **(2009)**, pp. 97-333.
[11] R. Johannes and M. Glinz, "Using dependency charts to improve scenario-based testing", Proceedings of the 17th international conference on testing computer software, **(2000)** June 8-12, Washington, D.C.
[12] A. Pietro, J. Boender, R. D. Cosmo and S. Zacchiroli, "Strong dependencies between software components", Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement, **(2009),** Paris, France.
[13] J. Kim and D. Bae, "An Approach to Feature Based Modelling by Dependency Alignment for the Maintenance of the Trustworthy System", Proceedings of the 28th Annual International on Computer Software and Applications Conference, **(2004)** September, pp. 28-30.
[14] W. Pei, J. Yang, L. Tan, R. Kroeger and D. Morgenthaler, "Generating precise dependencies for large software", Proceedings of the 4th International Workshop on Managing Technical Debt (MTD), **(2013)** May 20-20, San Francisco, CA.
[15] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "PrioritizingTest Cases for Regression Testing, IEEE Trans. Software Eng., vol. 27, no. 10, **(2001)** September, pp. 929-948.
[16] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Test CasePrioritization: A Family of Empirical Studies", IEEE Trans.Software Eng., vol. 28, no. 2, **(2008)** February, pp. 159-182.
[17] H. Do, G. Rothermel and A. Kinneer, "Empirical studies of test case prioritization in a J Unit testing environment", In Proceedings of the 15th International Symposium on Software Reliability Engineering, **(2004)** November 2-5.

