International Journal of Current Engineering and Technology
ISSN 2277 - 4106

Available at http://inpressco.com/category/ijcet

Research Article

Test Case Prioritization using Clustering

Jastej Badwal[a*] and Himanshi Raperia[a]

[a]Computer Science and Engineering, Lovely Professional University, India

Accepted 26 April 2013, Available online 1June 2013, **Vol.3, No.2 (June 2013)**

Abstract

Regression Testing is a very influential activity for controlling the quality of a software product, and it accounts for a large percentage of the costs of software. Here, we are going to implement new prioritization techniques that incorporate a clustering approach and utilize code coverage, code complexity and history faults as well to increase the effectiveness of the prioritization. Preliminary come within reach to investigating the relationships in enormous data in software repositories. In this, we are making an allowance for clustering approach to help improve test case prioritization. Results show that test case prioritization that utilizes a clustering approach can improve the effectiveness of test case prioritization techniques. Clustering approach can increase the effectiveness of test case prioritization techniques. Our research will be an attempt to test its validity means to test the prioritization techniques with the increment of efficiency as well as effectiveness.

Keywords: *Regression Testing, Clustering Approach*

1. Introduction

Software testing is a process used to identify the accuracy, total, and quality of developed computer software. It includes a set of actions conducted with the aim of finding errors in software so that it could be corrected before the product is out to the end users.

In other words, **software testing is a movement or you can say an activity to check whether the actual results match the expected results and to ensure that the software system is defect free**.

Software testing is an exploration conducted to give stakeholders with information about the quality of the product or service beneath test. Software testing also gives an objective, self-regulating view of the software to permit the business to appreciate and recognize the risks of software implementation. Test techniques comprise, but are not restricted to the process of executing a program or application with the target of finding software bugs. Software testing can also be acknowledged as the process of validating and verifying that a software program/application/product:

1. Meets the commerce and scientific necessities that guided its design and development;
2. Works as expected; and
3. Can be executed with the same uniqueness.

Software testing (Gregory M), depending on the testing method engaged, can be implemented at any instant in the development process. Different software development models (R. Pressman et al,2009) will focus the test effort at different points in the development process. Beginner development models, such as responsive, frequently use test driven development and place an increased portion of the testing in the hands of the developer, before it reaches an approved team of testers. In a supplementary traditional model, most of the test effecting occurs after the requirements have been defined and the coding process has been accomplished. **Software Testing is a systematic activity but it also involves economics and human psychology.** - Glenford J. Myers

2. Test Case Prioritization

Test case prioritization techniques, (Alexey G. et al,2006), schedule test cases so that those with the maximum priority, are executed in progress in the regression testing process than lower priority cases regarding the main concern test cases. For example, testers may wish to plan test cases in an order that achieves code coverage at the greatest rate possible, exercise features in order of estimated frequency of use, or increases the probability of detecting faults early in testing. A potential improvement of these techniques is that unlike test case reduction and non-safe regression test selection techniques, they do not discard tests.

Empirical results advise that several simple prioritization techniques can significantly improve one testing performance purpose; that is, the rate at which test suites notice faults. An enhanced rate of fault discovery

*Corresponding author: Jastej Badwal

during regression testing provides earlier feedback on the system under test and lets software engineers begin addressing faults earlier than might or else be possible. These outcomes also suggest, on the other hand, that the relative cost-effectiveness of prioritization techniques varies across workloads (programs, test suites, and types of modifications).Many different prioritization techniques have been proposed.

Test Case Design Techniques

Table 1: Categories of Test case design Techniques

Black box(functional)	White box(structural)	Other
Specification derived tests Equivalence partitioning Boundary value analysis State-transition testing	Branch testing Condition testing Data definition use testing Internal boundary value testing	Error guessing

Test case prioritization is a strategy for improving regression testing, an expensive but necessary process to validate software systems. Yet despite its use by practitioners, to date, there has been little work regarding how to incorporate varying test costs and fault severities into this strategy. This is surprising because the assumption that test costs and fault severities are uniform is one that is often not met in practice for the rate of fault detection of severe faults.

In this an algorithm is proposed to prioritize test cases based on rate of fault detection and fault impact. **The proposed algorithm identifies the severe fault at earlier stage of the testing process and the effectiveness of prioritized test case and comparison of it with unprioritized ones with the help of APFD.**

Test case prioritization techniques offer an alternative approach to improve regression testing cost-effectiveness. Test case prioritization techniques (G. Rothermel et al,2001)(W. E. Wong et al,1997) reorder test case to increase the chance of early fault detection using various types of information available from software artifacts. These techniques help engineers reveal faults early in testing, allowing them to begin debugging earlier than might otherwise be possible.

Various test case prioritization techniques can be utilized, and to date, numerous test case prioritization techniques have been proposed. Most research utilized code coverage information to implement prioritization techniques. (S. Elbaumet al,2002)(J. Kim al,2002)(Srivastava al,2002).

Software Testing Life Cycle

We follow test life cycle methodology. Figure represents how we proceed from one stage to another stage and finally gets desired result. This approach integrates our domain expertise with technical knowledge. As in test case design, we schedule the whole program code and apply break points (developer side). The tester will design the test cases for the respective code. Then select the test data and after this execute the respective test case. Before execution we apply clusters means grouping of four test cases in a single one.

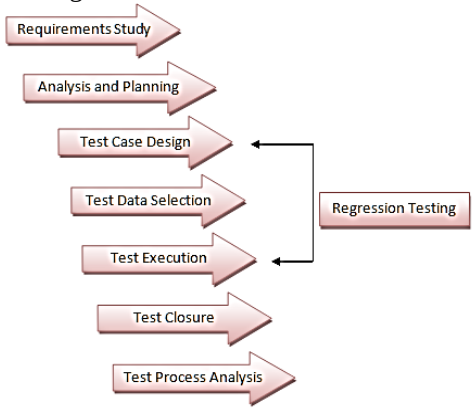


Figure 1 Software Testing Life Cycle

The actual outcome of this study is to improve the software quality, improve the effectiveness of prioritization. Results provide insights into usefulness of clustering in test case prioritization. A better understanding of the effects of time constraints could lead to improved testing processes.

Test case prioritization is a strategy for improving regression testing, an expensive but necessary process to validate software systems. Yet despite its use by practitioners, to date, there has been little work regarding how to incorporate varying test costs and fault severities into this strategy. This is surprising because the assumption that test costs and fault severities are uniform is one that is often not met in practice for the rate of fault detection of severe faults.

An algorithm is proposed to prioritize test cases based on rate of fault detection and fault impact. The proposed algorithm identifies the severe fault at earlier stage of the testing process and the effectiveness of prioritized test case and comparison of it with unprioritized ones with the help of APFD(Average Percentage of Fault Detection). (A. Malishevsky al,2002).

Clustering will increase the efficiency of the program code. Like, if 4 persons are going individually to same destination point from the same source point. They will reach but not in same time. Because each and every will come with different traffic or you can say congestion. But if four persons move together, they will reach definitely on time without any delay.

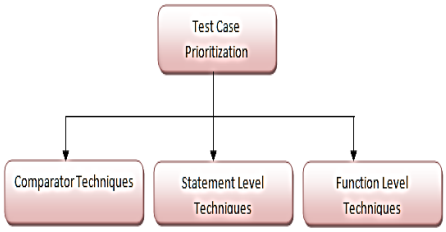


Figure 2 Categories of Test Case Prioritization

The purpose of this prioritization is to increase the likelihood that if the test cases are used for regression testing in the given order, they will more closely meet some objective than they would if they were executed in some other order. Test case prioritization can address a wide variety of objectives, including the following: Testers may wish to increase their coverage of coverable code in the system under test at a faster rate, Testers may wish to increase the rate of fault detection – that is, the likelihood of revealing faults earlier in a run of regression tests, Testers may wish to increase the rate of detection high-risk faults, locating those faults earlier in the testing process, Testers may wish to increase the likelihood of revealing regression errors related to specific code changes earlier in the regression testing process, Testers may wish to increase their confidence in reliability of the system under test at a faster rate.

In future they can use any of the following metric or the combination of these for Test Case Prioritization:

1. **Compatibility:** This factor defines compatibility of the system with the other existing systems; prioritizing the test case related to this will help in proper test bed arrangements for testing.

2. **Performance:** This factor defines how each requirement is related to the performance of the system, in the customer point of view.

3. **Security:** This factor defines the inter relationship of each requirement with security aspects such as unauthorized Login, Memory leakage and system alerts for different kind of failure conditions.

To check the effectiveness of the prioritization techniques, we consider the control techniques that do not use clustering and their corresponding heuristics prioritization techniques. APFD provides with a value between 0 and 100 that indicates how successful the prioritization technique is. The closer the value is 100 the better prioritization technique is.

Table 2: Control Techniques

Group	Description
Control	Code coverage without clustering Code complexity without clustering Combination of both
Heuristics	Code coverage using clustering Code complexity using clustering Combination of both

3. Proposed Work

New Approach for Prioritization Using Clustering

3.1 Introduction

In previous factors have to wait for the test cases to execute the respective code and after that prioritized the test case by using software metrics method. In this, we are merging four test cases to execute. This will execute one by one but will not entertain any traffic.

The figure will elaborate that during action period here the clubbing of test cases or you can say the clustering is done of test cases. The closest two test cases in terms of code coverage similarity are combined into a cluster first. Then the table of pair-wise (S. Yoo al,2010) similarities is recomputed with the new cluster being considered as a single document.

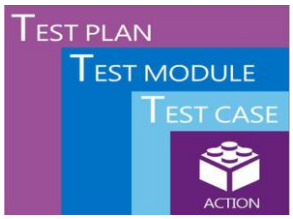


Figure 3 Clustering approach in Action

Having created clusters, we apply prioritization techniques to them (P. Tan al,2006). First, we reorder test cases within each cluster using specific software metrics. The first test case will be done according to the priority and rest will be ordered by using software metrics method. We considered four different test case prioritization techniques that usually use the three types of information written as below:

- Code coverage
- Complexity Metric
- Fault History Information

3.2 Factors of Clustering (Prioritization)

The purpose is to define the metric say new metric that form the basis of priority order according to which test cases in a suite are scheduled.

Refer to new metric named product symbolized as P is proposed. It comprises of two varying factors for a test case.

3.2.1 Code Coverage

Code coverage is calculated for each test case that is executed in a test suite. This usually based on just the count of the number of statements traversed by the particular cluster.

Mathematically, it is calculated as the number of statements covered upon the total number of clusters. **Statement coverage (St) = (No. of statements covered / total no. of statements) * 100**

3.2.2 Function Calls

Number of function or you can say job call (Fc) calculated; test case as it is looked into code coverage (statements covered) by the particular test case in a test suite.

No. of Function calls (Fc) are calculated for each test case as it is looked into statements covered by the particular test case in a test suite.

This variable is also a numeral as it includes the calls to a routine, procedure, method, function or subprogram. It is sum of both built in functions as well as user defined functions.

No. of Function calls (Fc) = Local functions + nested functions + private functions + overloaded methods (S. Elbaum al,2002)

After the test suite is executed first time for the software then it yields a result that when it is non – prioritized. Now as we will be able to keep an account for the re-test that is regression test that what is the statement coverage for each test case and how many function calls are occurring in the particular statement coverage for the particular test case.

As described above we will be calculating the product of the statement coverage and the no. of function calls. This becomes the software metric or the basis of the ordered set of test cases. As each test case has a value now so they will be ordered as the one with greatest value as the highest prioritized test case and followed in descending order. For this an algorithm is proposed with the assumption that the statement coverage and no. of function calls are known, such that this is given as the input to the algorithm and the output is the prioritized (in some order (either ascending or descending)) test suite.

3.3. The Algorithm

Now we introduce a standard algorithm for test case prioritization which is framed in a set pattern but with few changes in metric is done. The algorithm calculates the Product P metric for every test case given that the statement coverage St and Number of function calls Fc is known for every test case in advance. Then any sorting algorithm can be implemented to sort the product P values in descending order. These can be quick sort, merge sort or heap sort.

Definition 1: The Test Case Prioritization Problem with clustering

Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.
Problem: Find T′ ∈ PT such that (∀T″) (T″ ∈ PT) (T″ != T′) [f(T′) ≥ f(T″)].

In this explanation, PT is the set of possible prioritized test case orderings of T, and f is an objective function that applied to any such order, yields an award value for that order.

There are many possible goals for prioritization. For example, engineers may wish to increase the coverage of code in the system under test at a faster rate, or increase the rate at which test suites detect faults in that system during regression testing. In Definition 1, f shows a quantification of such a goal. Given any prioritization goal, various test case prioritization techniques may be used to meet that goal.

For example, to increase the rate of fault detection of test suites, engineers might prioritize test cases in terms of the extent to which they execute modules that have tended to fail in the past. Alternatively, engineers might prioritize test cases in terms of their increasing cost-per-coverage of code components, or in terms of their increasing cost-per-coverage of features listed in a requirements specification. In any case, the intent behind the choice of a prioritization technique is to increase the likelihood that the prioritized test suite can better meet the goal than would an ad hoc or random order of test cases. For a given time constraint say T to complete testing, I prepared a large set of test cases that covered functionality. Without clustering in a given time T with available resources only N test cases can be executed. Prioritizing test cases helps in helps in covering critical functionality but code coverage will still be a concern: Here is an approximation of code coverage of the cases covered with and without clustering.

Prioritized test suite T'

1. **begin**
2. set T' empty
3. **for each** test case t € T **do**
4. **Cluster$_i$ =$_{i=1}\square^k$ T$_i$**
5. calculate Product P as St * Fc
6. **end for**
7. sort T in descending order based on the value of P for each test case
8. let T' be T
9. **end**

(Gregory M)

Now the test cases are arranged in descending order according to the value of P. But there can be a possibility that the product for the two or more test cases comes out to be same.

4 Analysis

Here's the example that make you more clear that, how clustering is helpful for testing.

TEST CASES	VALUES
T1	54
T2	86
T3	83
T4	59
T5	23
T6	43
T7	99
T8	87
T9	51
T10	25
T11	29
T12	35
T13	65
T14	61
T15	71
T16	72
T17	2
T18	10
T19	3
T20	24

Clusters	C1	C2	C3	C4	C5	C6
Test cases	T1 T2 T3 T4	T5 T6 T7	T8 T9 T10	T11 T12 T13	T14 T15 T16	T17 T18 T19 T20

Non-prioritized

C1=	T1+T2+T3+T4	(54+86+93+59)/4	=73
C2=	T5+T6+T7	(23+43+99)/3	= 55
C3=	T8+T9+T10	(87+51+25)/3	= 54.33
C4=	T11+T12+T13	(29+35+65)/3	= 43
C5=	T14+T15+T16	(61+ 71+ 72)/3	=68
C6=	T17+T18+T19+T20	(2+ 10+ 3+24)/4	=9.75

(C1+C2+C3+C4+C5+C6)/6

(73+55+54.33+43+68+9.75)/6 =50.51

APFD= 50.51/100 = 0.51

Formula $= 1 - \sum_{i=1}^{k} Ci$ **=1-0.51 =0.49**

Prioritized

C1=	T17+T19+T18+T5	(2+3+10+23)/4	=38
C2=	T20+T10+T11	(24+25+29)/3	=78
C3=	T12+T6+T9	(35+43+51)/3	=129
C4=	T1+T4+T14	(54+59+61)/3	=174
C5=	T13+T15+T16	(65+71+72)/3	=208
C6=	T2+T8+T3+T7	(86+87+93+99)/4	=365

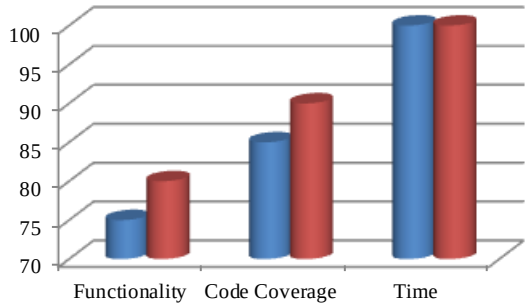
(C1+C2+C3+C4+C5+C6)/6
(38+78+129+174+208+365)/6 =48.9

APFD= 48.9/100= 0.48

Formula $= 1 - \sum_{i=1}^{k} Ci$ **=1-0.48 =0.52**

Here comparison among the results of prioritized and non-prioritized suite is done based on the results of the APFD metric. This is average percentage of faults detected. APFD is a standardized metric that is used to find the degree of faults detected.

Thus the prioritized test cases yield better fault detection than the non – prioritized test cases.



The above figure shows the difference between non-prioritized and prioritized. This will clearly examine the difference in code coverage as well as in functionality.

For e.g. Suppose few persons assigned for some target. They have to move from source A to destination B. if they move individually, than they reach the destination on same time but they all met with different congestions and all opt different ways to reach at destination, maximum consumption takes place. Whereas, if they move in group, then they will reach on same time with the same congestion and same way to be at destination.

5. Conclusion

This paper proposed such approach for test case prioritization using clustering in order to improve or to increase the efficiency in code coverage and in functionality. This will make the task easy. Analysis is done in clustering prioritized and non-prioritized cases with the help of APFD(average percentage fault detection). Prioritized cases always gives the best result rather than non-prioritized. In future test case prioritization with clustering should be done in such way that will increase more efficiency in time as well.

Acknowledgement

Working on my paper has been a milestone in my academic career. It has provided me an opportunity in learning new concepts, terminologies, and theories and expanding my academic knowledge beyond my classroom learning. I am grateful to that person who guided and supported me throughout the research process and provided assistance for my venture. Research papers suggestions and ideas helped me greatly in conceptualizing, designing, and structuring my research project. recommendations and instructions have enabled me to collect data, review literature, and plan for my dissertation. I also extend my thanks to the faculty members for their time in reviewing my proposal.

References

Gregory M. Kapfhammer Software Testing

R. Pressman (2009), Software Engineering: A Practitioner's Approach. Boston: *McGraw Hill*.

Alexey G. Malishevsky, Joseph R. Ruthruff, Gregg Rothermel, Sebastian Elbaum, Costcognizant (2006) Test Case Prioritization

G. Rothermel, R. Untch, C. Chu, and M. J. Harrold(2001), Prioritizing test cases for regression testing, *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948.

W. E. Wong, J. R. Horgan, S. London, and H. Agrawal(1997), A study of effective regression testing in practice, in *Proceedings of the International Symposium on software Reliability Engineering*.pp. 230-238

S. Elbaum, A. G. Malishevsky, and G. Rothermel(2002), Test case prioritization: A family of empirical studies, *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182

J. Kim and A. Porter(2002), A history-based test prioritization technique for regression testing in resource constrained environments, in Proceedings of the *International Conference on Software Engineering* Srivastava and J. Thiagarajan(2002), Effectively prioritizing tests in development environment, in *Proceedings of the International Symposium on Software Testing and Analysis*

A. Malishevsky, G. Rothermel, and S. Elbaum(2002), Modelig the costbenefits tradeoffs for regression testing techniques, in *Conf. Softw. Maint* pp. 204-213.

S. Yoo, M. Harman, P. Tonella, and A. Susi (2010), Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge, in Proceedings of the *International Conference on Software Testing and Analysis*.

P. Tan, M. Steinbach, and V. Kumar(2006), Introduction to Data Mining. *Addison-Wesley*.

S. Elbaum, A. G. Malishevsky, and G. Rothermel(2002), Test case prioritization: A family of empirical studies, *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182