A Brief Empirical Survey on Test Case Prioritization Techniques in Regression Testing

Binoop Kumar C. A.[1] A. Venugopal[2]
[1]Research Scholar [2]Assistant Professor
[1,2]Department of Computer Science & Engineering
[1,2]Sree Narayana Guru College, Coimbatore - 641105, India

Abstract— Software testing plays the most significant mechanism in improving the software system's quality. The software can be modified and validated by analyzing the new faults that gets evolved in the preceding tested code. Prior to the implementation phase, the early identification of faults, anomalies and bugs are detected and finally the software is moved to the maintenance stage. The anomalies can be effectively identified by the usage of accurate methodology. Here, regression testing is one of the most important methodologies in software testing. Any enhancement or updation can be implemented on the software and the changes do not have impact on remaining sections of the software because the test cases present in the regression testing are prioritized to reprocess the already available and new test cases in regression testing. The approaches available in test case prioritization are used to organize the test cases to maximize the effectiveness to attain their performance. The performance measure is the fault identification rate i.e. the rate of quickly analyzing the faults during the testing phase and enhanced fault identification rate imparts feedback quickly. Previous reports have proved that the test prioritization efficiently improves the fault detection rate. But there arises few questions related to the reports: a) Will the prioritization techniques are effective at target specified versions? b) What is the exact trade-off value that occurs between coarse and fine granularity techniques? c) Can the effectiveness get improved by collaborating the fault proneness measures into prioritization mechanisms? To answer the above statements, several methods are analyzed to improve the overall fault detection test suite rate and small improvements are performed by both fine and coarse granularity. This survey brings some of the effective measures in accosting the above questions.

Key words: Regression Testing, Test Cases, Prioritization & Fault Detection

I. INTRODUCTION

The quality is defined as gathering the requirements in software engineering and testing gives the view about the quality as gratifying the requirement needs. The concept of testing has the objective of minimizing the variability regarding the software system quality. Software testing is the notable phase in software development life cycle [11]. The regression testing is the type of testing to validate the software that is altered to uncover the recent faults that have been evolved into the preceding tested code. The testing is comprised of application programs to identify the bugs and faults. Test cases i.e. the developed test suites are reserved by software engineers and they can be reused for later use.

The test case suites of regression testing is said to be more cost effective in implementing the entire test cases and testing engineers may categorize and emphasize the important cases. The ultimate aim of test case prioritization is the maximization of fault reduction rate of test suite. The fault reduction rate is defined as, "how the test suites are discovered quickly in testing". The improved fault discovery rate can be used to furnish the advanced feedback and debugging by maximizing the likelihood [5]. When the testing is untimely halted and the corresponding test cases are executed by providing the fault discovering ability at the greatest level in the allocated time.

By enhancing this approach, the paper raises few questions: first thing is attempting "general prioritization" by selecting the effectual average test cases over the progression of succeeding software versions. Regression testing is typically anxious about the software version and effective test cases are to be prioritized for the exact software version. This is called "version oriented prioritization" due to its effectiveness. The concept of version specific feature is applied to general specific category due to the difference in cost. The paper mainly focuses on version oriented prioritization, second thing is examining the fine granularity operations including the performance of the analysis, instrumenting and prioritizing the source code level [2]. The systems having infeasible instrumentation methods may become more expensive. The cost experience can be resolved by applying coarse granularity that makes efficient analysis and instrumentation at the function level. But the effectiveness of coarse granularity is poor compared to fine granularity and this leads to less efficiency. The third thing is examining the performance measure gap between the outputs obtained by prioritization methods and the obtained results. The gap can be solved by combining the factors for faults into the proposed work.

Initially the work has started by conducting the experiments on similar set of programs and it is extended to include three set of large programs such as two open source UNIX utilities and real time embedded systems that belongs to the RAID level 5 category. Combining all the works of prioritization methods the strength of each technique can be enhanced. The remaining sections are organized as follows: section 1 describes the introduction to test case prioritization, section 2 describes previous methods related to test case fault detection rate, section 3 describes test case prioritization, section 4 describes methods of prioritization, section 5 describes the control techniques used in test case prioritization, section 6 describes the results and discussion, section 7 and 8 describes the conclusion and the references

II. RELATED WORK

Rother et al in 1999 has given the test case prioritization definition and the methods for test case prioritization and formulated the outputs based on the prescribed techniques. Six categories of prioritization methods are analyzed based

on the statements or branch coverage programs [15]. The test suites generated by these methods are compared with other methods including random, optimal and untreated test cases. The theory has proved that fault detection is improved and least expensive.

Weyuker in 1988 suggested that the considerable level of testing needs to be performed for the component reuse by developers to certify that the software works accurately in the other environments [18]. The outputs are analyzed from the survey taken from the developers from different organizations on the new methods of reusing the software while testing. Some of the code coverage features are used in testing that involves the group of program elements and the program is treated as fully tested when all the elements are executed properly. A few of the test coverage tools are available to compute the program efficiently.

Like code reusability, the source code helps to measure the reused code test coverage for developers. The difficulty arises when reusing the tested software with traditional methods and it may produce ineffective data. Recent survey is proposed to enhance the test coverage features to code reusability. Testers need the information of how much the test coverage has scope in attaining reusability [6]. The calculation can be taken by considering the ratio of reused code utilized in the new code instead of covered elements to the total elements. In scope elements are introduced to the elements in the code that is reusing in the new scope. The other sections are out of scope elements and they are called as feasible elements.

Ravichandran and Rothenberger in 2003, analyzed the three types of reusing techniques called as white box and black box reusable internal components. In black box reusability, the modification procedures are not permitted and the reusability without these changes is considered. But, in the white box testing the changes can be applied and considered by having code and context change [19]. The concept of scope aided testing can be applied for satisfying the above criteria.

Hence in order to answer for the above mentioned questions, various studies have been analyzed and the results are determined.

III. TEST CASE PRIORITIZATION

The test case prioritization is described by considering the following terms:

- T → Test case suite
- PT → The set of permutation functions of T
- F → function obtained from T (real numbers)
- The problem is formulated by finding the T' ϵ PT where (∀T") (T" ϵ PT) (T"≠ T') | f(T')≥ f(T")|. Here,
- PT → Possible prioritization of the order T
- F → Used for the ordering to obtain award value

There are primary goals in attaining this type of prioritization. They are described below:

- The software engineers who involved in testing process may tend to maximize the fault detection rate which is called as likelihood of the test cases to disclose the faults while performing regression testing by the usage of those test suites.
- They may tend to maximize the code coverage rate at the faster level in the system which permits the code coverage measure to meet the needs priory in testing.
- The confidence level can be maximized while testing under reliable rate at the faster level.
- The likelihood of disclosing the faults can be maximized associated to the alterations of the particular change in the testing.

The goals are clearly stated and the prioritization techniques success can be assessed by meeting these goals. The goals are said to be qualitatively mentioned. In the given statement of test case prioritization, f is said to be quantifier. The survey is focused on maximizing the likelihood of disclosing the faults at the beginning of the testing process itself [8]. The goal is defined to enhance the fault detection rate of a test suite by furnishing the quantifiers.

The test case prioritization becomes unmanageable and unpredictable based on the value of f. Consider the following example, where the quantifier f is given to determine whether the test suite attains code coverage is at the faster level and the solutions are effectively produced to the test case prioritization problem and the same solution can provide better solutions to the knapsack problem [10]. In the same way, f is the quantifier function in detecting whether the faults of the test suites are detected at a faster rate and the explicit solutions generated to solve the test case prioritization problem may solve and furnish the better methods to the halting problem. In these categories, it is termed as heuristic test case prioritization.

The test case prioritizations are categorized as general and version oriented test case prioritization. According to general test case prioritization, the program is given as P and T is the corresponding test suite and the test cases T are prioritized with the aim of identifying the test case ordering that must be applicable for the following altered versions of the program. Hence the prioritized results may obtain success than the given suite in meeting the prioritization goals. Next is the version oriented test case prioritization where the program function and test suites are given as P and T and the test cases given in T can be prioritized with the aim of identifying the test case ordering which will be applicable for the particular versions P' of P. The version oriented prioritization can be executed after performing the alterations to P and before obtaining P'. The test case that is prioritized will be more effectual in satisfying the goals for P' than the general test case prioritization [14]. In this survey, the prioritization test cases of regression testing are analyzed and how it is helpful in initial stage of testing.

IV. METHODS OF TEST CASE PRIORITIZATION

The prioritization goals are stated previously and these goals can be satisfied by performing several test case prioritization techniques. Consider the example, for maximizing the fault detection rate of the test suite, the test cases can be prioritized to the extent where the executed modules appear to fail previously. Another technique is the test cases can be prioritized by maximizing the cost of the code coverage elements or by maximizing cost of the parameters provided in specifying the requirements [4]. The main attempt of this

test case prioritization is to maximize the likelihood by meeting the goal efficiently than ad hoc, random ordering test cases, etc.

In this method, we consider 14 test case prioritization approaches are analyzed and they are categorized into three levels of groups. The approaches are listed in the next section. The first level of group is the control group comprised of two approaches that is utilized for controlling the experimentation. The second level of group is the statement level approach comprised of four fine granularity methods and these approaches were used in the previous analysis and the same approach can be taken for observing the version-oriented prioritization context. The third level of function is the function level group comprised of eight coarse granularity approaches by which four of the statement oriented comparisons and the other four approaches provide supplementary information on the rate of observing the available faults and those information that are not employed by the statement level approach [9]. Next section and the table no 1 briefly determines the all the above techniques in detail.

S.No	Label	Techniques	Method	Description
1	L1	Control Oriented Functions	Random ordering	Randomized ordering
2	L2		Optimal ordering	Ordered to optimize rate of fault detection
3	L3	Statement Oriented Functions	Total statement coverage prioritization	Prioritize on coverage of statements
4	L4		Additional statement coverage prioritization	Prioritize on coverage of statements not yet covered
5	L5		Total Fault Exposing Potential (FEP) prioritization	Prioritize on probability of exposing faults
6	L6		Additional FEP Prioritization	Prioritize on probability of faults, adjusted to consider previous test cases
7	L7	Function Level	Total function coverage Prioritization	Prioritize on coverage of functions
8	L8		Additional function coverage Prioritization	Prioritize on coverage of functions not yet covered
9	L9		Total FEP (function level) Prioritization	Prioritize on probability of exposing faults
10	L10		Additional FEP (function level) Prioritization	Prioritize on probability of faults, adjusted to consider previous test cases
11	L11		Total Fault Index (FI) Prioritization	Prioritize on probability of fault existence
12	L12		Additional Fault-index (FI) Prioritization	Prioritize on probability of fault existence, adjusted to consider previous test cases
13	L13		Total FI with FEP Coverage Prioritization	Prioritize on combined probabilities of fault existence and fault exposure
14	L14		Additional FI with FEP Coverage Prioritization	Prioritize on combined probabilities of fault existence/exposure, adjusted on previous coverage

Table 1: Techniques in Test Case Prioritization

A. Control Oriented Functions in Test Case Prioritization

1) L1: Random ordering

The prioritization approach which comes under the category of experimentation group is random testing of the test case ordering test case suite.

2) L2: Optimal ordering

The optimal ordering is the second experimentation group test cases available in the test suite. We can obtain Such type of an ordering can be attained in the experimentation group to employ the given programs with predicted faults and can analyze which faults are to be exposed for each test case [1]. The test case ordering can be determined to maximize the fault detection rate of a test suite. But this approach is not practically approachable and it furnishes the efficacy of the heuristics of other approaches.

B. Statement Oriented Functions in Test Case Prioritization

1) L3: Total Statement Coverage Prioritization

The intermediary functions or programs can be used to verify the total number of functions or statements, the number of statements in that program that were utilized by any test case. The test cases are prioritized depending on the amount of statements covered by arranging orderly in attaining total coverage of statements [17].

2) L4: Additional Statement Coverage Prioritization

Total statement coverage prioritization helps to organize the test cases by attaining the entire coverage. The test cases connected with covered statements are executed in achieving the succeeding coverage statements of testing. This type of additional statement coverage prioritization covetously chooses the test case which helps to acquiesces the most statement coverage and alters the data coverage about succeeding test cases to point out their statement coverage i.e. the statements that are not covered and finally the process is repeated till all covered statements are covered for the minimum test [19]. When While performing statement covering the rest of the test cases must be ordered and this can be done recursively by recreating all the uncovered statements and apply the same additional coverage statements on the test of the test cases.

3) L5: Total Fault Exposing Potential (FEP) Prioritization

The test cases will be able to expose the faults and the fault exposed ability is based on two factors such as the test cases trying to implement a faulty element and the probability ratio of identifying the test case failures in executing faulty statement. Practically this type of probability assumption may or may not be superior in detecting the fault rate with respect to code coverage.

The fault-exposing-potential (FEP) approximation is performed by mutation analysis for the test cases. The mutation score ms(s, t) of t on s is defined as the mutation ratio of s that is uncovered by t to entire mutant's s is determined for the program and the test case are given as P and T, t ϵ T for each test case and each statement in the program P. For the given test case t_k in T the award value is computed for t_k by adding up all ms(s, t_k) values. The test cases present in the test suite is ordered by the total fault-exposing-potential prioritization by considering the award values [3].

FEP prioritization turns out to be expensive in this approximation method than code coverage techniques because of the expenditure of mutation analysis. But when this FEP approximation becomes suitable method then this method becomes a accurate cost effective in fault exposing potential.

4) L6: Additional FEP Prioritization

To attain the extra statement coverage prioritization, the similar extension work is embedded into an entire statement coverage prioritization; the total FEP prioritization is extended to produce added fault-exposing-potential (FEP) prioritization. After opting the test case t, the test case t is opted to minimize the award values all other test cases that implement the statements used by t to reproduce the increased level of confidence in correcting the statements in the added FEP prioritization; the test cases are to be order by opting another test case and replicating the same process [12]. But this type of added statement implementation This approach lets us account for the fact that additional executions of a statement may be priceless than original implementations.

C. Function Level Prioritization Techniques

1) L7: Total Function Coverage Prioritization

This type of prioritization method is equivalent to the total statement coverage prioritization but this method works at the functional level and depending on the total functions implemented the test cases are prioritized.

2) L8: Additional Function Coverage Prioritization

This type of prioritization is equivalent to additional statement coverage prioritization but this method works at functional level and depending on the total functions implemented the test cases are prioritized.

3) L9: Total FEP (Function Level) Prioritization

At the statement level this method is equivalent to total FEP prioritization. A fault exposing potential is done by functional level to convert the statement level to functional level. A mutation analysis is used to compute, analyze each test case and function denoted by t and f; the amount of mutants in function f uncovered by t to mutants of function f implemented by t. The award values for the given test cases are added by these values. The same type of prioritization algorithm is applied for total FEP (statement level) prioritization and by replacing functions for the given statements [18].

4) L10: Additional FEP (Function Level) Prioritization

In the same way, this technique expands the total FEP (function level) technique in the same mode where the total FEP (statement level) prioritization technique is expanded.

5) L11: Total Fault Index (FI) Prioritization

Faults are not evenly probable to stay in each function but few functions are more expected to enclose to faults. This level of faults can be connected with quantifiable features of software attributes. These connections have the advantage by analyzing the test case priority depending on their records of implementing fault level functions.

The fault level can be represented by utilizing the fault index depending on the principal component analysis. The fault indexes are produced and this needs each function measure in the updated version and comparing the new indexes against the baseline versions of calculated indexes [7]. An absolute fault index is allocated for each function on behalf of the fault level and it also depends on complexity changes that happen in the function.

The total fault index prioritization is executed by the given fault indexes in a similar method associated to total function coverage. The sum of the fault indexes are computed for each test case and every function for executing the test case. The test cases are arranged in the decreasing way by the order of these sums.

6) L12: Additional Fault-Index (FI) Prioritization

The added fault index coverage prioritization technique is proficient in a way that is equivalent to extra function coverage. The preceding test cases are preserved and maintained by covering the set of functions [16]. The function set can be reset to ∅ when this set includes all functions more specifically and no test case embeds anything to this coverage. The another suitable test cases can be calculated by adding all the fault indexes for every function that is implemented by the test case excluding the functions in the covered set. The result turns out be perfect by summing up the test cases. The test cases are said to be prioritize by repeating this procedure.

7) L13: Total FI with FEP Coverage Prioritization

By using the approximation of both fault exposing potential and the fault detection rate can be achieved [13]. Hence, the total fault index prioritization is applied to all test cases and total FEP prioritization and fault index award values are applied as a secondary ordering for all test cases.

8) L14: Additional FI with FEP Coverage Prioritization

The previous technique can be expanded by this additional prioritization technique. In this method, the enhanced fault index prioritization is employed to attain an initial test case ordering and FEP prioritization is applied to grade all the test cases containing equal fault-index-based award values [15].

V. RESULTS & DISCUSSION

In regression testing, the survey mainly focuses on the utilization of test case prioritization techniques. The results are accomplished to improve the rate of fault and give the answers to the previously asked questions. a) Will the prioritization techniques are effective at target specified versions? b) What is the exact tradeoff value that occurs

between coarse and fine granularity techniques? c) Can the effectiveness get improved by collaborating the fault proneness measures into prioritization mechanisms? To answer the given query, several techniques are analyzed for the test case prioritization. First, the technique shows that version-oriented test case prioritization can generate considerable enhancements in the test suite fault detection rate. The techniques are similar in functional and statement level techniques. The functional prioritization level techniques use coarse oriented technique because of low cost than statement level techniques. Al though, the coarse oriented technique is quite low in the effectiveness and efficiency. The analysis states that the techniques in functional level are effective as in statement level and it could be more benefited.

VI. CONCLUSION

The combination of fault level and prioritization develops the efficacy of prioritization, but this enhancement was very small when compared to other techniques. This method tells that the advantages in combining such technique is not clear as perception and preceding accomplishment with fault level approximate in some of the application areas may work with higher efficiency. The survey also tells that efficacy of prioritization methods may be different in few programs. The result shows that, the outcomes may be similar when they do not break random and the outcomes may be different while breaking the random. First, the query can be answered by generalizing the results and checking the technique for further utilization. The various prioritization techniques performance differences are analyzed and further techniques are examined to improve their effectiveness. In order to satisfy these requirements, extra programs are gathered and the test suites are developed test suites for utilization. The added outcome of this technique is it provides effective prioritization techniques. Another type of prioritization goals and measures are acquired for its effectiveness. Additionally, alternate methods can be combined with feedback for employing fault-index-based techniques. This can be extended for its savings for the increasing rate of fault detection.

REFERENCES

[1] I. S. Association. Software Engineering Standards,volume 3 of Std. 1061: Standard for Software Quality Methodology. Institute of Electrical and Electronics Engineers, 1999 edition, 1992.
[2] A. Avritzer and E. J. Weyuker. The automatic generation of load test suites and the assessment of the resulting software. IEEE Trans. on Softw. Eng., 21(9):705-716, Sept. 1995.
[3] A. L. Baker, J. M. Bieman, N. Fenton, D. A. Gustafson, A. Melton, and R. Whitty. Philosophy for software measurement. J. Sys. Softw., 12(3):277-281, 1990.
[4] M. Balcer, W. Hasling, and T. Ostrand. Automatic generation of test scripts from formal test specications. In Proc. of the 3rd Symp. on Softw. Testing, Analysis, and Verication, pages 210-218,Dec. 1989.
[5] L. C. Briand, J. Wust, S. Ikonomovski, and H. Lounis.Investigating quality factors in object oriented designs: an industrial case study. In Proc. Int'l. Conf. on Softw. Eng., pages 345-354, May 1999.
[6] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. Technical Report 00-60-03, Oregon State University, Feb. 2000.
[7] S. G. Elbaum and J. C. Munson. Software evolution and the code fault introduction process. Emp. Softw. Eng. J., 4(3):241-262, Sept. 1999.
[8] T. Goradia. Dynamic impact analysis: A cost-effective technique to enforce error-propagation. In ACM Int'l. Symp. on Softw. Testing and Analysis, pages 171-181, June 1993.
[9] R. G. Hamlet. Testing programs with the aid of a compiler. IEEE Trans. Softw. Eng., SE-3(4):279-290, July 1977.
[10] M. Harrold and G. Rothermel. Aristotle: A system for research on and development of program analysis based tools. Technical Report OSU-CISRC- 3/97-TR17, Ohio State University, Mar 1997.
[11] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow and control flow-based test adequacy criteria. In Proc. Int'l. Conf. on Softw. Eng., pages 191-200, May 1994.
[12] T. M. Khoshgoftaar and J. C. Munson. Predicting software development errors using complexity metrics.
[13] J. C. Munson. Software measurement: Problems and practice. Annals of Softw. Eng., 1(1):255{285, 1995.
[14] J. C. Munson, S. G. Elbaum, R. M. Karcich, and J. P. Wilcox. Software risk assessment through software measurement and modeling. In Proc. IEEE Aerospace Conf., pages 137-147, Mar. 1998.
[15] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold, "Test Case Prioritization: An Empirical Study," Proc. Int'l Conf. Software Maintenence, pp. 179–188, Aug. 1999.
[16] P. Nikora and J. C. Munson. Software evolution and the fault process. In Proc. Twenty Third Annual Softw. Eng. Workshop, NASA/Goddard Space Flight Center, 1998.
[17] K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma. Regression testing in an industrial environment. Comm. ACM, 41(5):81-86, May 1988.
[18] E. Weyuker. Testing component-based software: a cautionary tale. Software, IEEE, 15(5):54–59, Sep 1998.
[19] T. Ravichandran and M. A. Rothenberger. Software reuse strategies and component markets. Commun. ACM, 46(8):109–114, Aug. 2003.

