Asian Journal of Information Technology 11 (5): 169-180, 2012
ISSN: 1682-3915

An Efficient Method to Achieve Effective Test Case Prioritization in Regression Testing using Prioritization Factors

[1]S. Raju and [2]G.V. Uma
[1]Department of Computer Applications,
Sri Venkateswara College of Engineering, 602105 Sriperumbudur, India
[2]Department of Information Science and Technology,
College of Engineering, Anna University, 600025 Chennai, India

Abstract: Test case prioritization techniques have been shown to be beneficial for improving regression-testing activities. With prioritization, the rate of fault detection is improved, thus allowing testers to detect faults earlier in the system-testing phase. Most of the prioritization techniques to date have been code coverage-based. These techniques may treat all faults equally. Test case prioritization techniques schedule test cases for execution so that those with higher priority, according to some criterion are executed earlier than those with lower priority to meet some performance goal. In this study, researchers introduce a cluster-based test case prioritization technique. By clustering test cases, based on their dynamic runtime behaviour researchers can reduce the required number of pair-wise comparisons significantly. Researchers present a value-driven approach to system-level test case prioritization called the prioritization of requirements for test. In this approach, prioritization of test cases is based on four factors rate of fault detection, requirements volatility, fault impact and implementation complexity. The results show that this prioritization approach at the system level improves the rate of detection of severe faults.

Key words: Test case prioritization, regression testing, prioritization factors, agglomerative hierarchical clustering, India

INTRODUCTION

In today's changing business environment, time to market is a key factor to achieving project success. For a project to be most successful quality must be maximized while minimizing cost and keeping delivery time short. Quality can be measured by the customer satisfaction with the resulting system based on the requirements that are incorporated successfully in the system (Karlsson and Ryan, 1997). Many software engineering methodologies face complex cost and benefit tradeoffs that are influenced by a wide variety of factors. For example, software development processes are influenced by system size, personnel experience and requirements volatility and these contribute to determining the overall costs and benefits of particular processes (Do and Rothermel, 2008). The quality of services is a key issue for developing service-based software systems and testing is necessary for evaluating the functional correctness, performance and reliability of individual as well as composite services (Askaruinisa and Abirami, 2010). During development and testing, changes made to a system to repair a detected fault can often inject a new fault into the code base. New faults introduced during testing and maintenance can be isolated by impact analysis and regression testing techniques. Impact analysis and regression testing techniques exists that find changes in a system based upon a modification (Sherriff *et al*., 2007).

Every software product typically undergoes frequent changes in its lifetime. These changes are necessitated on account of various reasons such as fixing defects, enhancing or modifying existing functionalities or adapting to newer execution environments. Whenever a program is modified, it is necessary to carry out regression testing to ensure that no new errors have been introduced Swarnendu (Biswas *et al*., 2009). Test case prioritization techniques improve the cost-effectiveness of regression testing by ordering test cases such that those that are more important are run earlier in the testing process. Prioritization can provide earlier feedback to testers and management and allow engineers to begin debugging earlier. It can also increase the probability that if testing ends prematurely, important test cases have been run (Do *et al*., 2010). Regression testing is the de

Corresponding Author: S. Raju, Department of Computer Applications, Sri Venkateswara College of Engineering, 602105 Sriperumbudur, India

facto activity to address the testing problems due to software evolution. However, many existing regression testing techniques (Harrold *et al*., 1993; Kim and Porter, 2002) assume that the source code is available and use the coverage information of executable artifacts (e.g., statement coverage of test cases) to conduct regression testing. When such coverage information cannot be assumed to be available it is vital to consider alternative sources of information to facilitate effective regression testing (Mei *et al*., 2009). Although, efficient algorithm selection for detecting suitable test case prioritization is an undesirable problem, researchers still attempt to develop various methods and have made some progress. These means can be classified as dynamic selection of suitable algorithm. Dynamic methods include random selection of suitable algorithm. Dynamic methods include random selection of algorithms and extract suitable test case selection it's a kind of goal-oriented approach and evolutionary approach (Pravin and Srinivasan, 2012).

Prioritization involving human judgement is not new. The Operations Research community has developed techniques including the Analytic Hierarchy Process (AHP) algorithm that help decision makers to prioritize tasks. However, prioritization techniques that involve humans present scalability challenges. A human tester can provide consistent and meaningful answers to only a limited number of questions, before fatigue starts to degrade performance (Yoo *et al*., 2009). It is often desirable to filter a pool of test cases for a program in order to identify a subset that will actually be executed and audited (checked for conformance to requirements) at a particular time (Leon and Podgurski, 2003). For example: the suite of regression tests for a long-lived system may become so large that it is feasible to execute only a fraction of them when the program is modified (Harrold *et al*., 1993).

A deployed application may be instrumented to capture its own executions, so that developers can replay and audit them (Steven *et al*., 2000) or so that captured inputs can be used to refresh a regression test suite. The number of executions that are captured may exceed the number it is feasible to audit.

An automatic test data generator may be capable of producing many more tests than a tester is able to run and audit.

Naturally, it is desirable to select those test cases that are most likely to reveal defects in the program under test. To be worthwhile the sum of the cost of the filtering process and the costs of executing and auditing the selected tests should be less than the cost of executing and auditing all of the tests in the original pool. Some approaches to filtering or prioritizing regression tests employ profiles collected when testing previous versions of a program. These approaches emphasize economical reuse of test cases and hence they are not applicable to new tests cases (Leon and Podgurski, 2003). Recent studys have investigated several techniques for test case filtering and for the closely related problem of test-case prioritization that are based on analyzing profiles of test executions. These profiles characterize aspects of test executions that are thought to be relevant to whether the tests reveal defects.

To optimize the time and cost spent on testing, prioritization of test cases in a test suite can be beneficial. Test Case Prioritization (TCP) involves the explicit planning of the execution of test cases in a specific order with the intention of increasing the effectiveness of software testing activities by improving the rate of fault detection earlier in the software process (Srikanth *et al*., 2005). In this study, one new approach to prioritize the test cases at system level for regression test cases is proposed. This technique identifies more severe faults at an earlier stage of the testing process. Factors proposed to design algorithm are rate of fault detection, requirements volatility, fault impact and implementation complexity. The objective of this research is to develop and validate a system-level test case prioritization scheme to reveal severe faults earlier and to improve customer-perceived software quality. In this study, severity value also is considered as one of the factors to prioritize the test cases where severity value ranging from 2-10 to the faults. Before applying the prioritization algorithm, here we use clustering technique. By clustering test cases, based on their dynamic runtime behaviour researchers can reduce the required number of pair-wise comparisons significantly. Then, researchers present a value-driven approach to system-level test case prioritization called the Prioritization of Requirements for Test (PORT). PORT can be used to prioritize system-level black box test when traceability between requirements, test case and test/field failures is maintained by the development team.

LITERATURE REVIEW

Many researchers have addressed the test case prioritization problem. A wide range of prioritization techniques have been proposed and studied. Only a few studies have considered issues related to the presence of time constraints during prioritization. Where prior studies of prioritization are concerned, the vast majority reported in the literature.

During development and testing, changes made to a system to repair a detected fault can often inject a new fault into the code base. These injected faults may not be

in the same files that were just changed since, the effects of a change in the code base can have ramifications in other parts of the system. Sherriff *et al.* (2007) proposed a methodology for determining the effect of a change and then prioritizing regression test cases by gathering software change records and analyzing them through singular value decomposition. That methodology generated clusters of files that historically tend to change together. Combining these clusters with test case information yields a matrix that can be multiplied by a vector representing a new system modification to create a prioritized list of test cases. They have performed a post hoc case study using that technique with three minor releases of a software product at IBM. They have found that their methodology suggested additional regression tests in 50% of test runs and that the highest-priority suggested test found an additional fault 60% of the time.

Software engineers frequently update COTS components integrated in component-based systems and can often chose among many candidates produced by different vendors. Mariani *et al.* (2007) presented a study which tackles both the problem of quickly identifying components that were syntactically compatible with the interface specifications but badly integrate in target systems and the problem of automatically generating regression test suites. The technique proposed in that study to automatically generate compatibility and prioritized test suites was based on behavioral models that represent component interactions and were automatically generated while executing the original test suites on previous versions of target systems.

Software engineering methodologies were subject to complex cost benefit tradeoffs. Economic models can help practitioners and researchers assess methodologies relative to these tradeoffs. Effective economic models, however, can be established only through an iterative process of refinement involving analytical and empirical methods. Sensitivity analysis provides one such method. By identifying the factors that are most important to models, sensitivity analysis can help simplify those models; it can also identify factors that must be measured with care, leading to guidelines for better test strategy definition and application. Do and Rothermel (2008) proposed a research which uses sensitivity analysis to examine the model analytically and assess the factors that were most important to the model. Based on the results of that analysis, they proposed two new models of increasing simplicity. They assessed those models empirically on data obtained by using regression testing techniques on several non-trivial software systems.

Srivastava (2008) proposed that test case prioritization techniques involve scheduling over test cases in an order that improves the performance of regression testing. It was inefficient to re execute every test cases for every program function if once change occurs. Test case prioritization techniques organize the test cases in a test suite by ordering such that the most beneficial are executed first thus allowing for an increase in the effectiveness of testing. One of the performance goals, i.e., the fault detection rate is a measure of how quickly faults are detected during the testing process. In that study they presented a new test case prioritization algorithm which calculates average faults found per min. They presented the results illustrating the effectiveness of algorithm with the help of APFD metric. The main aim of that study was to determine the effectiveness of prioritized and non-prioritized case with the help of APFD.

Jin *et al.* (2010) proposed a novel approach called Behavioral Regression Testing (BERT). They have given two versions of a program; BERT identifies behavioral differences between the two versions through dynamical analysis in three steps. First, it generated a large number of test inputs that focus on the changed parts of the code. Second, it ran the generated test inputs on the old and new versions of the code and identifies differences in the tests' behavior. Third, it analyzed the identified differences and presents them to the developers. By focusing on a subset of the code and leveraging differential behavior, BERT provided developers with more information than traditional regression testing approaches. To evaluate BERT they have implemented it as a plug in for Eclipse a popular Integrated Development Environment and used the plug-in to perform a preliminary study on two programs.

Regression test selection techniques for embedded programs have scarcely been reported in the literature. Biswas *et al.* (2009) proposed a model-based regression test selection technique for embedded programs. Their proposed model in addition to capturing the data and control dependence aspects also represents several additional program features that were important for regression test case selection of embedded programs. These features include control flow, exception handling, message paths, task priorities, state information and object relations. They selected a regression test suite based on slicing the proposed graph model. They also proposed a genetic algorithm based technique to select an optimal subset of test cases from the set of regression test cases selected after slicing the proposed model.

A web service may evolve autonomously, making peer web services in the same service composition uncertain as to whether the evolved behaviors may still be

compatible to its originally collaborative agreement. Although, peer services may wish to conduct regression testing to verify the original collaboration the source code of the former service can be inaccessible to them. Traditional code-based regression testing strategies are inapplicable. The rich interface specifications of a web service, however, provide peer services with a means to formulate black-box testing strategies. Mei *et al.* (2009) formulated the new test case prioritization strategies using tags embedded in XML messages to reorder regression test cases and reveal how the test cases use the interface specifications of services. They have evaluated experimentally their effectiveness on revealing regression faults in modified WS-BPEL programs. The results shown that the new technique that have a high probability of outperforming random ordering.

Software developers use testing to gain and maintain confidence in the correctness of a Software System. Automated reduction and prioritization techniques attempt to decrease the time required to detect faults during test suite execution. Smith and Kapfhammer (2009) proposed a study which used the Harrold Gupta Soffa, delayed greedy, traditional greedy and 2-optimal greedy algorithms for both test suite reduction and prioritization. Even though reducing and reordering a test suite was primarily done to ensure that testing was cost-effective those algorithms were normally configured to make greedy choices with coverage information alone. That study extended those algorithms to greedily reduce and prioritize the tests by using both test cost and the ratio of code coverage to test cost. An empirical study with eight real world case study applications shown that the ratio greedy choice metric aids a test suite reduction method in identifying a smaller and faster test suite.

Askaruinisa and Abirami (2010) proposed an approach for generating web service test cases using WSDL-S and Object Constraint Language (OCL) while the test case generation technique is Orthogonal Array Testing (OAT). They have developed a prototype namely Semantic Web Services Test Case Generator (SWSTCG). They have generated WSDL of Web Service to be tested using NetBeans IDE and converted into WSDL-S by giving OCL references where pre and post conditions are defined. Test data using OAT with different factors, levels and strengths were generated and documented in XML based test files called Web Service Test Specifications (WSTS) and executed. The proposed method is compared with the Pair-Wise Testing (PWT) Method. They have conducted testing on various web service applications and the results have shown that the method was effective in generating minimal test cases with maximum test case effectiveness.

Genetic algorithms have been successfully applied in the area of software testing. The demand for automation of test case generation in object oriented software testing is increasing. Genetic algorithms are well applied in procedural software testing but a little has been done in testing of object oriented software. Pravin and Srinivasan (2012) proposed a study which discusses genetic algorithms that can automatically select an efficient algorithm which was suitable for test cases selection. That algorithm takes a selected path as a target and executes sequences of operators iteratively for efficient algorithm selection to evolve. The evolved efficient algorithm selection can lead the program execution to achieve the target path. An automatic path-oriented test data generation was not only a crucial problem but also a hot issue in the research area of software testing today. They also proposed genetic algorithm for the selection of the suitable algorithm which perform much better than the existing methods and can provide very good solutions.

PRIORITIZATION FACTORS

In the research, researchers refer to these factors as Prioritization Factors (PF). These factors may be concrete such as test case length, code coverage, severity value and data flow. Researchers will describe these factors as rate of fault detection, requirements volatility, fault impact and implementation complexity.

Rate of Fault detection (RF): The average number of faults per min by a test case is called rate of fault detection. The rate of fault detection of test case i have been calculated using the number of faults detected and the time taken to find out those faults for each test case of test suite:

$$RFTi = \frac{\text{No. of faults}}{\text{time}} \times 10 \quad (1)$$

Every factor is converted into 1-10 point scale. The technique presented in this study implemented a new test case prioritization technique that prioritize the test cases with the goal of giving importance of test case which have higher value for rate of fault detection and severity value.

Requirements Volatility (RV): Requirements Volatility (RV) is based on the number of times a requirement has been changed during the development cycle. If a requirement has changed more than ten times the volatility values for all requirements are quantified on a 10 point scale. Requirements volatility is an assessment of the requirements change once the implementation begins (Malaiya and Denton, 1999).

Roughly 25% of the requirements for an average project change before project completion and volatile requirements tend to make the testing activities difficult and cause the software to contain high defect density. Changing requirements result in re-design, the addition or deletion of existing functions and often an increase in the fault density in the program. Severe defects that escape into the field can cost 100 times more to fix after delivery than correcting the problem in the requirements phase. For non-severe defects the ratio is 2:1 for fixing the defect in the field as opposed to pre-delivery.

Fault Impact (FI): Testing efficiency can be improved by focusing on the test case that is likely to contain high number of severe faults. So for each fault severity value was assigned based on impact of the fault on the product. Severity value has been assigned based on a 10 point scale as given below:

- Very High Severe: SV of 10
- High Severe: SV of 8
- Medium Severe: SV of 6
- Less Severe: SV of 4
- Least Severe: SV of 2

Equation 2 shows that the severity value of test case, i where t represent number of faults identified by the ith test case:

$$Si = \sum_{j=1}^{t} SV \quad (2)$$

If Max (S) is the high severity value of test case among all the test cases then fault impact of ithtest case is shown below:

$$Fli = \frac{Si}{Max(S)} \times 10 \quad (3)$$

Implementation Complexity (IC): Implementation complexity (IC) is a subjective measure of how difficult the development team perceives the implementation of requirement to be. Each requirement is analyzed to assess the anticipated implementation complexity and is assigned a value ranging from 1-10; the larger value indicates higher complexity. IC is a prioritization factor for requirements being implemented for the first time or for all requirements in the first release.

Requirements with high implementation complexity tend to have a higher number of faults. Amland (1999) conducted a case study to find that the functions with high number of faults were the functions with higher McCabe Complexity. About 20% of the modules result in 80% of the faults and roughly 50% of the modules are defect free.

PROPOSED WORK

Test case prioritization seeks to find an efficient ordering of test case execution for regression testing. The most ideal ordering of test case execution is one that reveals faults earliest. Since, the nature and location of actual faults are generally not known in advance, test case prioritization techniques have to rely on available surrogates for prioritization criteria. Structural coverage, requirement priority and mutation score have all previously been utilized as criteria for test case prioritization. However, there is no single prioritization criterion whose results dominate the others. The proposed research is shown in Fig. 1.

This study aims to reduce the number of comparisons required for the pair-wise comparison approach through the use of clustering. Instead of prioritizing individual test cases, clusters of test cases are prioritized using techniques such as prioritization of requirements for test algorithm. From the prioritized clusters, the ordering between individual test cases is then generated.

Test case clustering: The clustering process partitions objects into different subsets so that objects in each group share common properties. The clustering criterion determines which properties are used to measure the commonality. When considering test case prioritization, the ideal clustering criterion would be the similarity between the faults detected by each test case. However, this information is inherently unavailable before the testing task is finished. Therefore, it is necessary to find

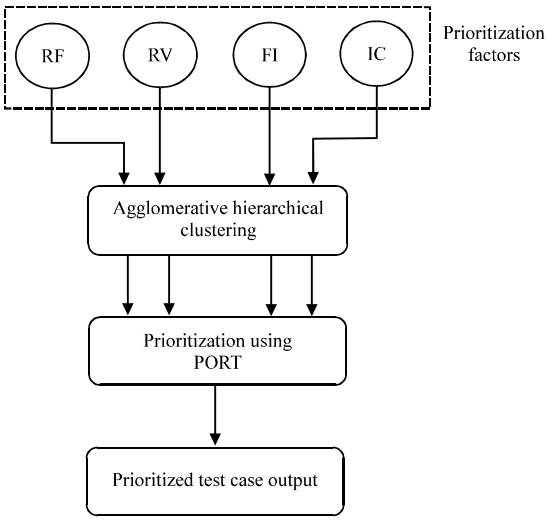

Fig. 1: Flow diagram of the proposed research

a surrogate for this in the same way as existing coverage based prioritization techniques turn to surrogates for fault-detection capabilities.

Here, researchers use a simple agglomerative hierarchical clustering technique (Yoo *et al*., 2009). Its pseudo-code is described:

```
Input- A set of n test cases, T
Output- A dendrogram, D , representing the clusters
Algorithm
Form n clusters each with one test case
C←{}
Add clusters to C
Insert n clusters as leaf node into D
while there is more than one cluster
  Find a pair of clusters with
  minimum distance
  Merge the pair into a new
  cluster, C_new
  Remove the pair of test cases
  from C
  Add C_new to C
  Insert C_new as a parent node of
  the pair into D
retum D
```

The resulting dendrogram is a tree structure that represents the arrangement of clusters. Figure 2 shows an example dendrogram from agglomerative hierarchical clustering. Cutting the tree at different height produces different number of clusters. It is possible to generate k clusters for any k in [1, n] by cutting the tree at different heights.

Prioritization technique: In the research, researchers are using prioritization of requirements for test to prioritize the test cases. After clustering the test cases, prioritization of requirements for test algorithm is applied to each cluster to prioritize the cluster of test cases. In this study, researchers can find a brief account about Prioritization of Requirements for Test algorithm.

Prioritization of Requirements for Test Algorithm (PORT): Based on the project and customer needs, the development team assigns weight to the prioritization factor such that the assigned total weight (1.0) is divided amongst the four factors. Factor weight which is unique for each project, allows the PORT user to customize the priority of each factor for a particular project. For example if the requirements for a project have been very stable then the development team might assign RV a relatively smaller portion of the total weight. A default value can be assigned, giving each factor equal weight.

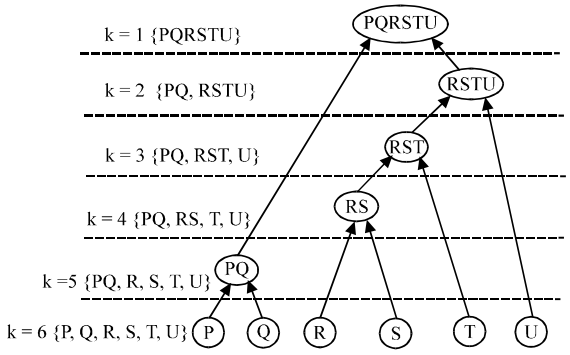

Fig. 2: Sample dendrogram from agglomerative hierarchical clustering

For every requirement, Eq. 4 is used to calculate a Prioritization Factor Value (PFV) which can be given as:

$$PFV_i = \sum_{j=1}^{4} (\text{Factor value}_{ij} \times \text{Factor weight}_j) \quad (4)$$

PFV_i represents prioritization factor value for requirement i which is the summation of the product of factor value and the assigned factor weight for each of the factors. Factor $value_{ij}$ represents the value for factor j for requirement i and Factor $weight_j$ represents the factor weight for jth factor for a particular product. PFV is a measure of the importance of testing a requirement.

A value-matrix representation of PFV for requirements is shown in Eq. 5 where PFV (P) is the product of Value (V) and Weight (w):

$$P = Vw \quad (5)$$

Equation 5 can be expanded with respect to the prioritization factors as given in the equation:

$$\begin{pmatrix} PFV_1 \\ . \\ . \\ . \\ PFV_n \end{pmatrix}_{(n\times 1)} = \begin{pmatrix} R_1^{RF} & R_1^{IC} & R_1^{RV} & R_1^{FI} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ R_n^{RF} & R_n^{IC} & R_n^{RV} & R_1^{FI} \end{pmatrix}_{(n\times 4)} \begin{pmatrix} w_{RF} \\ w_{IC} \\ w_{RV} \\ w_{FI} \end{pmatrix}_{(4\times 1)} \quad (6)$$

The computation of PFV_i for a requirement is used in computing the Weighted Priority (WP) of its associated test cases. WP of the test case is the product of two elements:

- The average PFVof the requirement (s) the test case maps
- The requirements-coverage a test case provides
- Requirements coverage is the fraction of the total project requirements exercised by a test case
- Let there be n total requirements for a product/release and test case j maps to i requirements. WP_j is an indication of the priority of running a particular test case. WP_j is represented by the Eq. 7 which is given:

$$WP_j = \left[\frac{\sum_{x=1}^{i} PFV_x}{\sum_{y=1}^{n} PFV_y}\right] \times \left(\frac{i}{n}\right) \quad (7)$$

The test cases are ordered for execution based on the descending order of WP values such that the test case with the highest WP value is run first and so on.

Validation algorithm: Refinement and validation of Prioritization of Requirements for Test (PORT) will proceed via the analysis of the severity of faults detected for a product. For analysis purposes, each failure is assigned a Severity Value (SV) on a 10 point scale as shown:

- Highly severe (Severity 1): SV value of 10
- Medium severe (Severity 2), SV of 6
- Less severe (Severity 3), SV of 4
- Least severe (Severity 4) SV of 2

Total Severity of Faults Detected (TSFD) is the summation of severity values of all faults identified for a release. Equation 8 shows TSFD for a product/release where t represents total number of faults identified for the product/release:

$$TSFD = \sum_{t=1}^{t} SV_t \quad (8)$$

The Average Severity of Faults Detected (ASFD) is computed for each requirement to analyze if the requirement with a higher computed PSFV actually had higher average severe faults when the product was system tested or used by the customer. The ASFD for requirement i ($ASFD_i$) is the ratio of the summation of severity values of faults identified for that requirement divided by the TSFD. The computation of ASFD is shown in Eq. 9 where m is the number of faults mapped to requirement i:

$$ASFD_i = \left[\frac{\sum_{j=1}^{m} SV_j}{TSFD}\right] \quad (9)$$

ASFD is used to analyze the effectiveness of PORT technique by mapping the total percentage of severe faults identified for a requirement to the PFV for that requirement.

RESULTS AND DISCUSSION

The test case prioritization system, proposed in this study was implemented in the working platform of JAVA (Version JDK 1.6). Researchers can use the bank application project for regression testing.

Intermediate step results: In the proposed method, researchers are using a banking application was created to find the test cases. In this study, researchers can find some of the screen shorts for the intermediate results of the proposed prioritization technique. Figure 3 shows the initial screen of the proposed prioritization technique.

After the initial screen appears, the data must be saved in the database regarding the accounts. Figure 4 the screen to shows the sample output before regression. Figure 5 shows create new account. Figure 6 shows the sample screen for depositing amount. Figure 7 shows the display for withdrawing the amount from the account. Figure 8 shows the screen for the mini statement for the particular account.

The earlier mentioned banking application is used to create the test cases for the proposed prioritization technique. After completing all the applications shown earlier the test cases are generated for the errors occurred during the operation of the banking application. Figure 9 shows some of the sample test cases generated during the banking operation.

After generating test cases by operating the banking application, the test cases must be clustered together to reduce the total number of test cases to be analyzed. This step reduces the time to prioritize the test cases. The sample test case after clustering is given in Fig. 10.

After clustering the PORT algorithm is applied to the clustered test cases to prioritize the test cases. The output

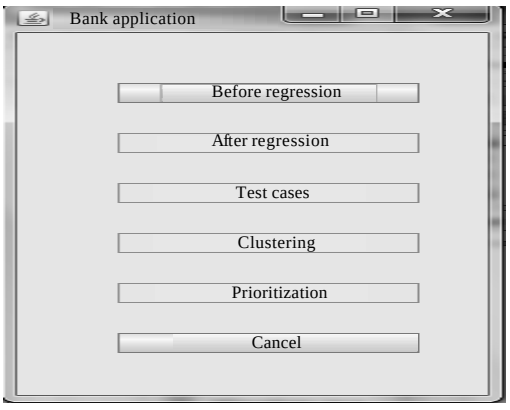

Fig. 3: Initial screen of the banking application

Fig. 4: Sample screen beforer regression

Fig. 5: Sample screen for creating new account

obtained after applying PORT is the final prioritized test cases. Figure 11 shows the prioritized output of the test cases. Rate of fault detection is one of the factors considered in this proposed method to prioritize the test cases. This factor depends upon the time and the number of faults occurred during the operation. Figure 12 gives the error count per sec for the test cases.

Figure 13 shows the average factor values of the prioritization factors considered in the proposed technique.

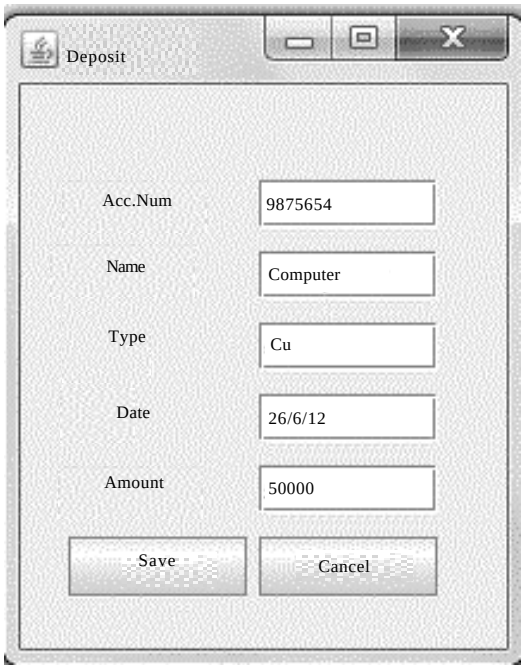

Fig. 6: Sample screen for depositing amount

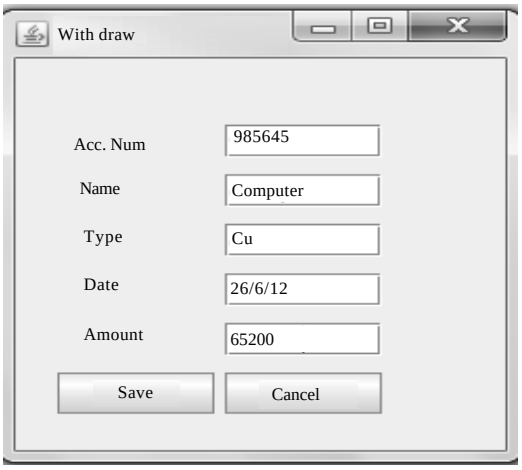

Fig. 7: Sample screen to withdraw the amount

Performance analysis

APFD: To quantify the goal of increasing a subset of the test suite's rate of fault detection researchers use a metric called APFD developed by Elbaum that measures the rate of fault detection per percentage of test suite execution. The APFD is calculated by taking the weighted average of the number of faults detected during the run of the test suite. APFD can be calculated as follows:

$$\text{Apfd} = 1 - \frac{(\text{Tf}_1 + \text{Tf}_2 + \ldots + \text{Tf}_m)}{nm} + \frac{1}{2n} \qquad (10)$$

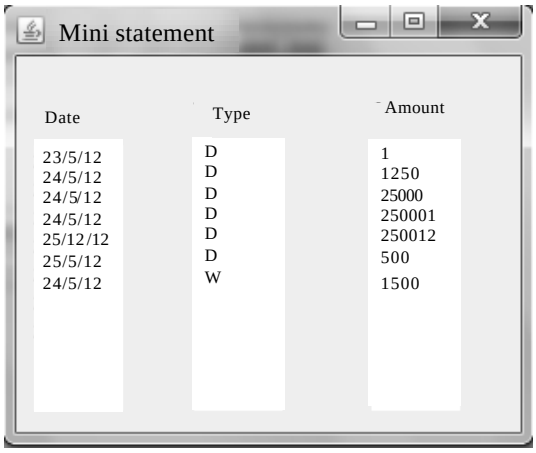

Fig. 8: Sample screen of mini statement of a particular account

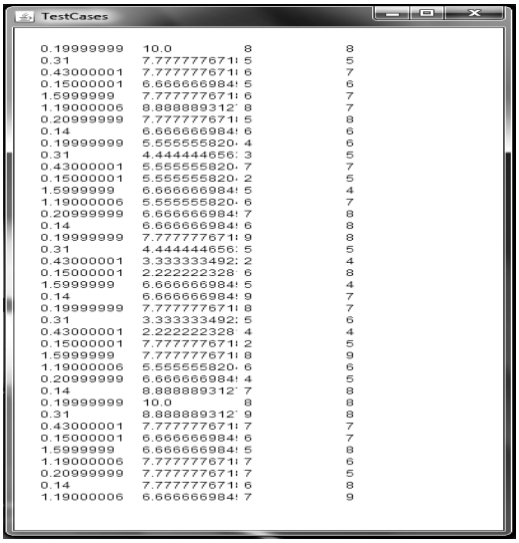

Fig. 9: Test cases generated during banking operation

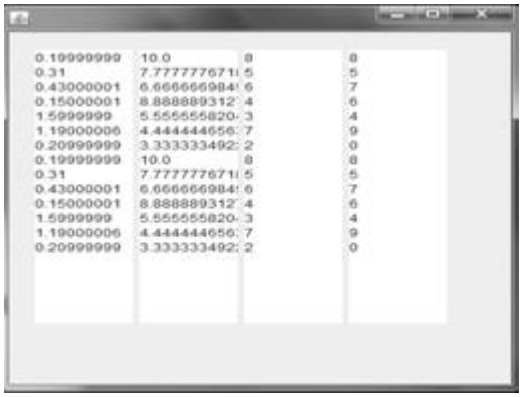

Fig. 10: Test cases after clustering

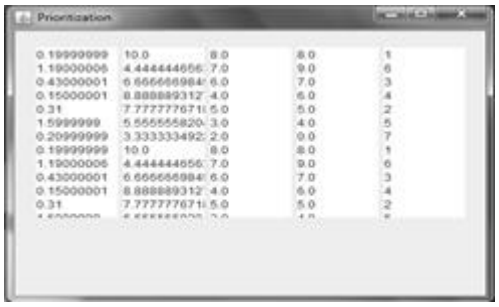

Fig. 11: Prioritized test cases

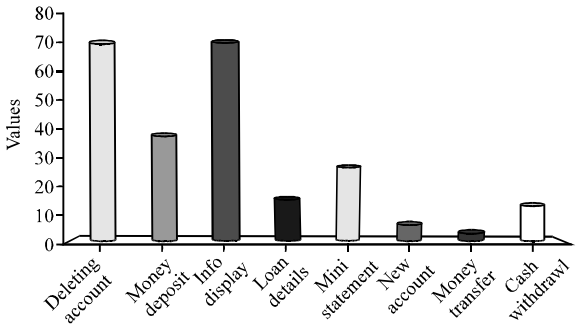

Fig. 12: Error count per sec for the test cases

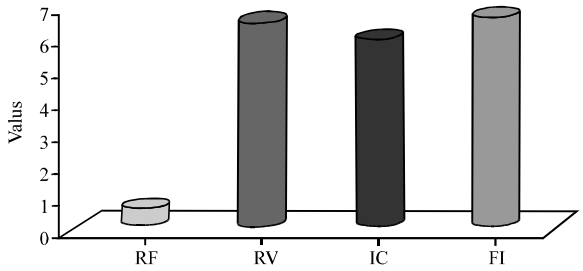

Fig. 13: Average factor values

Table 1: The faults detected by the test suites in bank project

	Test cases				
Faults	T1	T2	T3	T4	T5
F1	X	-	-	-	X
F2	-	X	-	-	X
F3	-	-	-	X	-
F4	X	-	X	-	-
F5	-	-	-	-	X

where, n be the no. of test cases and m be the no. of faults. $Tf_1...Tf_m$ are the position of first test t hat exposes the fault.

For the project the APFD metric is calculated as follows. Number of test cases m = 5 and the number of faults n = 5. This can be represented in following Table 1.

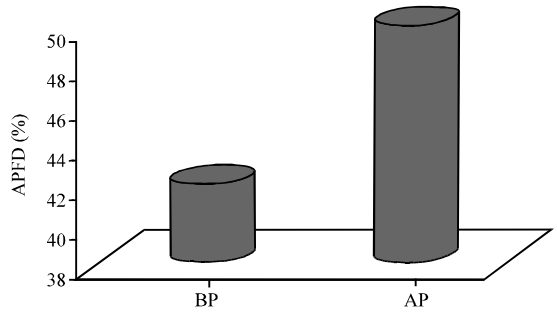

Fig. 14: APFD metric comparison

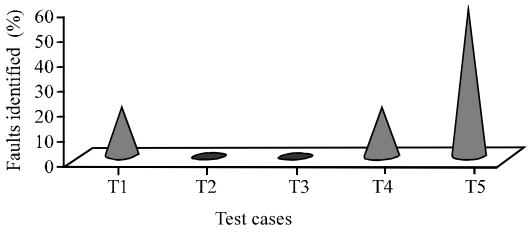

Fig. 15: Fault detection performance of test case

Here, the number of test cases is 5, i.e., {T1, T2, T3, T4, T5} and the number of faults occur during the regression testing is 5, i.e., {F1, F2, F3, F4, F5} The prioritized test suits with test sequence {T5, T1, T3, T4, T2} then the APFD metric after prioritization is:

$$\text{Apfd}(T,P) = 1 - \frac{(3+5+2+4+1)}{5\times 5} + \frac{1}{2\times 5} = 0.50$$

The APFD metric before prioritization is The APFD comparison before and after regression is represented in Fig. 14:

$$\text{Apfd}\,(T,P) = 1 - \frac{5+4+2+5+1}{5\times 5} + \frac{1}{2\times 5} = 0.42$$

PTR metric: The PTR metric is another way that the effectiveness of a test prioritization may be analyzed. Recall that an effective prioritization technique would place test cases that are most likely to detect faults at the beginning of the test sequence. Let t be the test suite under evaluation, n be the total number of test cases in t, and n_d be the total number of test cases needed to detect all faults in the program under test p. The PTR metric is shown in Fig. 15:

$$\text{Ptr}(t,p) = \frac{n_d}{n} \qquad (11)$$

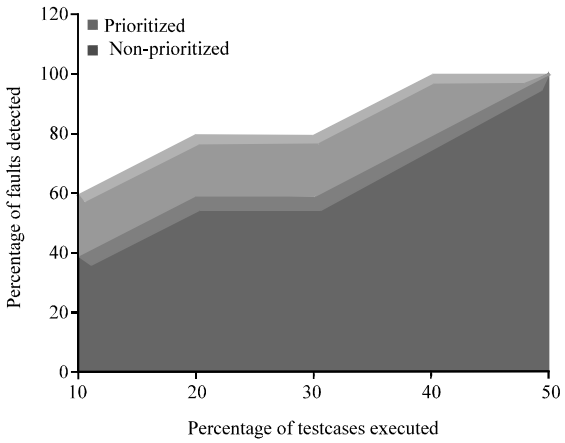

Fig. 16: Comparison of prioritized and non-prioritized test cases

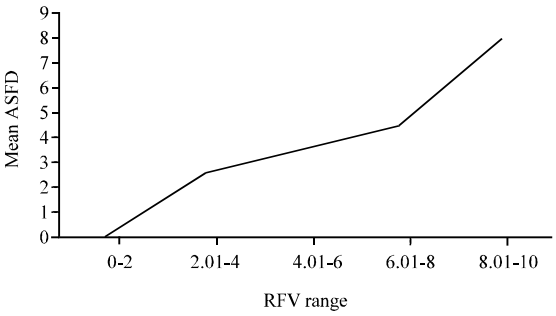

Fig. 17: Comparison of ASFD and RFV

The comparison is drawn between prioritized and non prioritized test case which shows that number of test cases needed to find out all faults are less in the case of prioritized test case compared to non prioritized test case.

It can be observed in Fig. 16 that the new prioritization technique needs only 30% of test cases to find out all the faults. But 50% of test cases are needed to find out all the faults in the case of non prioritization if test cases are executed in non prioritized order. APFD is the portion of area below the curve in Fig. 16 and 17 plotting percentage of test cases executed against percentage of faults detected.

Comparison of ASFD and RFV: The RFV and ASFD for each requirements and the TSFD of the project is computed. RFV is divided into five range of categories based on the values of RFV. The requirements are grouped into one of these five ranges. The mean ASFD value is computed for each range of RFV. A lower RFV

value indicates a lower priority for the particular requirement to be tested. Thus, the proposed method of test case prioritization process will reduce the re-execution time of the project by prioritizing the most important test cases.

CONCLUSION

In this study, researchers describe regression testing based test suite prioritization technique. Experimental results demonstrate that the approach can create time-aware prioritizations. This study identifies and evaluates the challenges associated with time-constraint prioritization. The proposed method uses most efficient factors to prioritize the test factors.

The effectiveness of the proposed prioritization technique can be evaluated by using the APFD and PTR metric. Based on the experimental results obtained researchers observe that the proposed method is effectively prioritize the test cases in the Bank application project by employing the clustering and port based prioritization. This will reduce the cost of executing the entire project.

REFERENCES

Amland, S., 1999. Risk based testing and metrics. Proceedings of the 5th International Conference on EuroSTAR, November 8-12, 1999, Barcelona, Spain, pp: 1-20.

Askaruinisa, A. and A.M. Abirami, 2010. Test case reduction technique for semantic based web services. Int. J. Comput. Sci. Eng., 2: 566-576.

Biswas, S., R. Mall, M. Satpathy and S. Sukumaran, 2009. A model-based regression test selection approach for embedded applications. Software Eng. Notes, Vol. 34. 10.1145/1543405. 1543413.

Do, H. and G. Rothermel, 2008. Using sensitivity analysis to create simplified economic models for regression testing. Proceedings of the 2008 ACM International Symposium on Software Testing and Analysis, July 20-24, 2008, New York, USA.

Do, H., S. Mirarab, L. Tahvildari and G. Rothermel, 2010. The effects of time constraints on test case prioritization: A series of controlled experiments. IEEE Trans. Software Eng., 36: 593-617.

Harrold, M.J., R. Gupta and M.L. Soffa, 1993. A methodology for controlling the size of a test suite. ACM Trans. Software Eng. Methodol., 2: 270-285.

Jin, W., A. Orso and T. Xie, 2010. Automated behavioral regression testing. Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST), April 7-9, 2010, Paris, France, pp: 137-146.

Karlsson, J. and K. Ryan, 1997. A cost-value approach for prioritizing requirements. IEEE Software, 14: 67-74.

Kim, J.M. and A. Porter, 2002. A history-based test prioritization technique for regression testing in resource constrained environments. Proceedings of the 24th International Conference on Software Engineering, May 19-25, ACM Press, pp: 119-129.

Leon, D. and A. Podgurski, 2003. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. Proceedings of the 14th International Symposium on Software Reliability Engineering, Nov. 17-21, IEEE Computer Society Washington, DC., USA., pp: 442-453.

Malaiya, Y.K. and J. Denton, 1999. Requirements volatility and defect density. Proceedings of the 10th international Symposium on Software Reliability Engineering, November 1-4, 1999, Boca Raton, Florida, pp: 285-298.

Mariani, L., S. Papagiannakis and M. Pezz, 2007. Compatibility and regression testing of COTS-component-based software. Proceedings of the 29th International Conference on Software Engineering, May 20-26, 2007, Washington, DC., USA.

Mei, L., W.K. Chan, T.H. Tse and R.G. Merkel, 2009. Tag-based techniques for black-box test case prioritization for service testing. Proceedings of the 9th International Conference on Quality Software, August 24-25, 2009, Jeju, South Korea, pp: 21-30.

Pravin, A. and S. Srinivasan, 2012. Efficient algorithm selection for detecting suitable test case prioritization. Proceedings of the International Conference on Recent Advances and Future Trends in Information Technology, March 21-23, 2012, Patiala, Punjab, India, pp: 28-31.

Sherriff, M., M. Lake and L. Williams, 2007. Prioritization of regression tests using singular value decomposition with empirical change records. Proceedings of the 18th IEEE International Symposium on Software Reliability, November 5-9, 2007, Trollhättan, Sweden, pp: 81-90.

Smith, A.M. and G.M. Kapfhammer, 2009. An empirical study of incorporating cost into test suite reduction and prioritization. Proceedings of the 2009 ACM symposium on Applied Computing, March 9-12, 2009, New York, USA.

Srikanth, H., L. Williams and J. Osborne, 2005. System test case prioritization of new and regression test cases. Proceedings of the 4th International Symposium on Empirical Software Engineering, Nov. 17-18, IEEE Computer Society, pp: 10-10.

Srivastava, P.R., 2008. Test case prioritization. J. Theoretical Applied Inform. Technol., 4: 178-181.

Steven, J., P. Chandra, B. Fleck and A. Podgurski, 2000. jRapture: A capture/replay tool for observation-based testing. Proceedings of the International Symposium on Software Testing and Analysis, August 21-24, 2000, Portland, OR., USA., pp: 158-167.

Yoo, S., M. Harman, P. Tonella and A. Susi, 2009. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. Proceedings of the 18th International Symposium on Software Testing and Analysis, July 19-23, 2009, Chicago, IL., USA., pp: 201-212.