Factors Oriented Test Case Prioritization Technique in Regression Testing using Genetic Algorithm

S. Raju
Associate Professor, Department of Computer Applications
Sri Venkateswara College of Engineering, Sriperumbudur, India
Research Scholar, DIST, Anna University, Chennai - 600025
E-mail: srajuphd@gmail.com

G. V. Uma
Professor and Head, Department of Information Science and Technology
College of Engineering, Anna University, Chennai – 600025, India

Abstract

Test case prioritization involves scheduling test cases in an order that increases the effectiveness in achieving some performance goals. One of the most important performance goals is the rate of fault detection. Test cases should run in an order that increases the possibility of fault detection and also that detects the most severe faults at the earliest in its testing life cycle. In this paper, we develop and validate requirement based system level test case prioritization scheme to reveal more severe faults at an earlier stage and to improve customer-perceived software quality using Genetic Algorithm (GA). For this, we propose a set of prioritization factors to design the proposed system. In our proposed technique, we refer to these factors as Prioritization Factors (PF). These factors may be concrete, such as test case length, code coverage, data flow, and fault proneness, or abstract, such as perceived code complexity and severity of faults, which prioritizes the system test cases based on the six factors: customer priority, changes in requirement, implementation complexity, completeness, traceability and fault impact. The goodness of these orderings was measured using an evaluation metric called APFD and PTR that will also be calculated.

Keywords: Test Case Prioritization, Regression Testing, Genetic Algorithm (GA), Prioritization Factors, Requirement Factor Value (RFV), Test Case Weight (TCW).

1. Introduction

Regression testing, which intends to ensure that a software program works as specified after changes have been made to it, is an important phase in software development lifecycle [12]. Regression testing is the re-execution of some subset of test that has already been conducted. In regression testing as integration testing proceeds, number of regression tests increases and it is impractical and inefficient to re execute every test for every program function if once change occurs. It is an expensive testing process used to detect regression faults [10]. Regression testing has been used to support software-testing activities and assure acquiring an appropriate quality through several versions of a software
Factors Oriented Test Case Prioritization Technique in Regression Testing using Genetic Algorithm

product during its development and maintenance [1]. Regression testing is an important and yet time consuming software development activity. It executes an existing test suite on a changed program to assure that the program is not adversely affected by unintended amendments. Test suites can be large and conducting regression tests is tedious [6]. Regression testing assures the quality of modified service-oriented business applications against unintended changes.

The test case prioritization is important in regression testing. It schedules the test cases in a regression test suite with a view to maximizing certain objectives which help reduce the time and cost required to maintain service-oriented business applications. Existing regression testing techniques for such applications focus on testing individual services or workflow programs [7]. 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 [5]. Test suite prioritization is a regression testing technique where test cases are ordered such that faults can be detected early in the test execution cycle. This is useful because tests accumulate over multiple revisions and versions of the system and it is not feasible to execute all the tests in a limited amount of time [23].

Test case prioritization is used in regression testing, at the test suite level, with the goal of detecting faults as early as possible in the regression testing process, given a test suite inherited from previous versions of the system. There are many techniques for prioritizing test cases based on various forms of information such as code coverage or modification history [2]. In test case prioritization techniques two dimensions are considered. The first is granularity and the second dimension is the prioritization strategy [8]. Over the lifetime of a large software product, the number of test cases could drastically increase as new versions of software are released. Because the cost of repeatedly retesting all test cases may be too high, software testers tend to remove redundant or trivial test cases to construct a reduced test suite for regression testing at a reasonable cost [22]. After development and release, software undergo regress maintenance phase of ten to fifteen years. Modifications in software may be due to change in customer’s requirements or change in technology or platform. This leads to release of numerous versions or editions of the existing software.

Also in case of the version or edition’s test only modified and affected parts are to be tested to impart confidence in the modified software which is the process of regression testing [20]. Test case prioritization is an aspect of regression testing that permutes a test suite with the view of maximizing a test goal. Test case prioritization does not discard any test cases, having the advantage of not impairing the fault detection ability of the test suite as a whole [13]. The main disadvantage in regression testing is that it is an expensive process used to validate modified software [18]. The test case prioritization in regression testing can be used in various system such as in Time Aware Test Case Prioritization where experiments are performed on two subject programs involving four techniques two techniques for an approach to time aware test case prioritization based on genetic algorithms, and four traditional techniques for test case prioritization using integer linear programming [4]. Web Applications Testing where several test suite prioritization strategies for web applications are performed and these strategies improved the rate of fault detection for web applications and their preexisting test suites [3]. For Audit Testing of Evolving Web Services where an approach to the prioritization of audits test cases using information retrieval is performed.

This approach matches a service change description with the code portions exercised by the relevant test cases [19]. For Audit Testing of Web service where a test case prioritization method specifically tailored for audit testing of services is performed. The method is based on the idea that the most important test cases are those that have the highest sensitivity to changes injected into the service responses [24]. In Black-Box Testing Environment an approach to test suite reduction for regression testing in black box environment has been proposed. The reduced regression test suite has the same bug finding capability and covers the same functionality as the original regression test suite [28]. In Recent Priority Algorithm where a model that achieves 100% code coverage optimally during version
specific regression testing is done. Here prioritization of test cases is done on the basis of priority value of the modified lines covered by the test case [14].

2. A Survey of Recent Research in the Field

A handful of researches have been presented in the literature for the prioritization of regression testing test cases. Recently, utilizing artificial intelligence techniques like Greedy Algorithm and Genetic Algorithm (GA), in Prioritization has received a great deal of attention among researchers. A brief review of some recent researches is presented here.

Yogesh et al. [9] have proposed an approach that variables were vital source of changes in the program and test cases should be prioritized according to the variables of any changed statement and variables computed from the variables of changed statements. In support of their prioritization approach they extended to validate the effectiveness of prioritized test cases with respect to data flow technique. The experimental study investigating the effectiveness of their prioritization approach by considering programs. The results obtained were encouraging and support their work to validate the prioritization technique with respect to DU or DC paths of data flow testing technique. Here the variable based prioritization of test cases for regression testing is in conformance with the data flow testing strategy.

R.Kavitha et al. [15] have proposed a prioritization technique to improve the rate of fault detection of severe faults for Regression testing. Here, two factors rate of fault detection and fault impact for prioritizing test cases are proposed. The proposed algorithm was validated by analyzing two sets of industrial projects. 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. Results indicate that the proposed technique lead to improved rate of detection of severe faults in comparison to random ordering of test cases. And also it was tested experimentally that the number of test cases runs to find the entire fault was less in case of proposed prioritization technique. The results prove that the proposed prioritization technique was effective.

Arup et al. [16] have proposed a method to prioritize the test cases for testing component dependency in a Component Based Software Development (CBSD) environment using Greedy Approach. An Object Interaction Graph (OIG) was being generated from the UML sequence diagrams for interdependent components. The OIG was traversed to calculate the total number of inter component object interactions and intra component object interactions. Depending upon the number of interactions the objective function was calculated and the test cases were ordered accordingly. This technique was applied to components developed in Java for a software system and found to be very effective in early fault detection as compared to non-prioritize approach.

Ruchika et al. [17] have proposed both regression test selection and prioritization technique. They implemented their regression test selection technique and demonstrated that their technique was effective regarding selecting and prioritizing test cases. The results show that their technique may significantly reduce the number of test cases and thus the cost and resources for performing regression testing on modified software. The proposed technique increases confidence in the correctness of the modified program. The test cases selected using the proposed technique will identify and locate errors in the modified program and it will help in preserving the quality and reliability of the software. Test cases selected by the proposed technique will ensure the software’s continued operation in reducing the test cases by a significant number.

Seifedine Kadry. [21] proposed a technique to improve the cost-effectiveness of the regression testing. The regression test process to test and verify the changes made on software. The main goal to use regression test selection based on risk analysis or automation test is to improve the cost-effectiveness of the test. The developed technique use the automation test based on decision tree and test selection process in order to reduce the testing cost was given. The developed technique was applied to a practical case and the result shows its improvement. They recommended using the proposed technique as a compromise technique between TSRA and ATVM.
Kaur et al. [25] proposed an algorithm to prioritize test cases using Genetic Algorithm. The genetic algorithm was introduced that will prioritize regression test suite within a time constrained environment on the basis of total fault coverage. The proposed algorithm has been automated and the results are analyzed. Here, total fault coverage with in time constrained environment on different examples is used for prioritization of test cases and their finite solution was obtained. Through Genetic Algorithm technique, an approach has been identified to find a suitable population, which was further formulated by GA operations to make it more flexible and efficient. The elaborations of results were shown with the help of APFD metrics. The APFD has been calculated to evaluate the usefulness of the proposed algorithm.

Sanjukta Mohanty et al.[26] have proposed a test case prioritization techniques in code based, requirement based and model based prioritization techniques and it’s implementation in CBSS. A prioritization technique schedules the test cases for execution so that the test cases with higher priority executed before lower priority. The systematic literature survey was based on nine articles collected from multiple-stage selection process. For determining the effectiveness of prioritization techniques two metrics (APFD and RPd) are used. There was good coverage in terms of research in understanding the concepts of different code based techniques and behavior of components, interactions and compatibility of components. CBSS has emerged as an approach that offers rapid development of system using fewer resources and effort. The core idea of reuse and reducing the development costs could be achieved if the components offer reliable services.

Jayant et al. [27] have proposed a study on test case prioritization based on cost, time and process aspects. Prioritization concept increases the rate of fault detection or code in time and cost constraints. Test case, test suite prioritization’s approaches are used in regression testing. It is because testing plays an important role in software development life cycle. Test case prioritization (TSP) is an effective and practical technique in regression testing to reduce it. It schedules test cases in order of precedence that increases their ability to meet some performance goals, such as code coverage, rate of fault detection. They have concluded that prioritization of test case or a test suit has different aspects of fault detection. On the basis of prioritization techniques, functionality of regression testing can improved in minimum time and recourses. This can support to make a better software product.

3. Test Case Prioritization in Regression Testing

As the name suggests, regression testing involves saving and reusing test suites which have been created for earlier version releases of the software. By reusing these already created test cases, the costs of the designing and creating of test cases can be amortized across the lifetime of a system. Empirical studies have allowed researchers to compare regression testing techniques earlier; however these studies suffer from severe limitations in their abilities to assess cost-benefit tradeoffs relative to practical testing situations. The main limitations include: Context factors, Lifetime factors and Cost-benefit models. These limitations make it difficult to empirically compare the various regression testing techniques or when evaluated could lead to improper assessment of costs and benefits in practical situations. This ultimately leads to inaccurate conclusions about the cost-effectiveness of techniques and decisions of engineers relying on such conclusions to select techniques for regression testing.

Regression testing is the process of validating modifications introduced in a system during software maintenance. Regression testing is an expensive process used to validate modified software. As the test suite size is very large, system retesting consumes large amount of time and computing resources. This issue of retesting of software systems can be handled using a good test case prioritization technique. A prioritization technique schedules the test cases for execution so that the test cases with higher priority executed before lower priority. The objective of test case prioritization is to detect fault as early as possible.
An improved rate of fault detection during regression testing can let software engineers begin their debugging activities earlier than might otherwise be possible, speeding the release of the software. An improved rate of fault detection can also provide faster feedback on the system under test and provide earlier evidence when quality goals have not been met, thus allowing strategic decisions about release schedules to be made earlier than might otherwise be possible. 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.

4. Test Case Prioritization Technique in Regression Test Cases Through GA

Regression testing may take minutes to weeks to months of time depending on the size of the test suite and how long each test case takes to run. However by using effective prioritization technique test cases can be reorder by the tasters to obtain fault detection at increased rate in the system. It should be ensure that the Software has been tasted very carefully. A prioritized test is more effective if execution needs to stop after some time period, it can also be achieved using a random ordering. However, if the desired time for the execution of the test case which is known in advance then a better a better test case ordering may be possible. The first preference of this research work is to prioritize the regression testing test cases.

Here, we will introduce a new regression test suite prioritization algorithm and genetic algorithm prioritizes this test suite with the aim of maximizing the number of faults, during time-constraint execution that are expected to be found. It may be possible that methodology presented in the newly prioritized test sub suites is more than a selection algorithm but the preference will be given to the prioritization of test case. We have done an experiment and analyze that the genetic algorithm with regard to effectiveness and time/space overhead and to prioritize test cases we utilizing structurally-based test adequacy criteria and this criteria will be referred to as test prioritization indices. We observed the recording of GA for test cases, without any idea of fault and using stabilized criteria. The ordering was identified using test adequacy criteria, which determine how likely errors are to be found instead of how many errors the test actually finds in other sense this method is used to identify how many faults exists in the code. These were used to identify the fitness of the all possible ordering of the test cases.

4.1. Prioritization Factors

Computation of certain factors such as (1) customer assigned priority of requirements, (2) implementation complexity, (3) changes in requirements, (4) fault impact of requirements, (5) completeness (6) traceability and (7) Execution time, is essential for prioritizing the test cases because they are used in the prioritization algorithm. According to these factors weights are defined for each test case in the software. Based on prioritization the evaluation cost and time could be decreased by focusing only on particular test cases.

4.1.1. Customer-Assigned Priority (CP)

CP is the measurement of importance to the customer’s need. Customer’s need vary from 1 to 10 and assigned by customer itself, where 10 is used to identify the highest customer priority. It is essential to improve the customer perceived value for the development of the customer. Software functions never, infrequently used 45%, occasionally used 19% and always used 36%, all are approximate idea. Maximum effort should be utilized for the faults identification which takes place on regular interval otherwise these faults results in continuous failures. It was proved that customer-perceived value and satisfaction can be improved by focusing on customers want and development.
4.1.2. Implementation Complexity (IC)
Implementation complexity is the subjective measure of the complexity anticipated by the development team in implementing the need and it is evaluated initially. Value from 0 to 10 assigned by the developer on the basis of its application complexity and by using larger value higher complexity is denoted. No of faults increases as the requirement is become high in implementation complexity.

4.1.3. Changes in Requirements (RC)
Developer assigned a measure and the range varies from 1 to 10 which used to indicate the number of times a need is changed in the development cycle by taking its origin date as a reference is called RC. Need changes to 10 times if the volatility values for all the needs are represents on 10 point scale. The number of changes for any requirement \( i \) divided to the highest number of changes for any requirement among all the project requirements yields the change in requirement \( R_i \) of that requirement. If the \( i^{th} \) requirement is changed \( M \) times and \( N \) is the maximum number of requirements then the requirement change of \( I \), \( R_i \) can be calculated as

\[
R_i = \left( \frac{M}{N} \right) \times 10
\]

4.1.4. Fault Impact of Requirements (FI)
FI allows the development team to identify the requirement which has had customer reported failures. As a system evolves to various versions, the developer can use the previous data which is collected from versions to identify requirements that are likely to be error prone. Fault impact is based on the number of field failure and in-house failures and it is consider to those requirements which have already been launched product.

In this work, we propose to calculate fault impact of a requirement, based on the severity of the fault identified in the previous run. Fault severity 1, 2, 3, 4 and 5 correspond to the severity values \( 2^5 \), \( 2^4 \), \( 2^3 \), \( 2^2 \) and \( 2^1 \). If a requirement \( i \) with the set of \( t \) test cases, discovered the set of \( d \) faults of the set of severity \( V \), then the severity \( i \) of the requirement \( i \) is computed as

\[
S_i = \sum_{x=1}^{t} \sum_{y=1}^{d} V(x, y)
\]

and if \( S = \{S_i \forall i = 1\ldots n \} \) where \( n \) is the total number of requirements, is the set of all severities of each requirement \( i \), then the fault impact \( FI_i \) of a requirement \( i \) is computed as

\[
FI_i = \left( \frac{S_i}{\text{Max}(S)} \right) \times 10
\]

4.1.5. Completeness (CT)
Completeness expresses a requirement as measured by a function to be executed, the degree of success, the restrictions under which the function is to be executed, and any limitations, which influence the likely solution, for example an interface constraint. Customer assigned a value in between 0 and 10, when the requirement is selected for reuse after analyzing the completeness of each requirement. Customer satisfaction like fastness of the software response to the user request can be improved, by taking completeness of the requirement into consideration.
4.1.6. Traceability (TR)

Traceability calculates the mapping that exists between requirements and test. It helps in determining whether a requirement is enough tested is difficult for the testers if the test cases are not relevant to particular requirements. A problem which is very common is insufficiency of traceability. If traceability is not good or traceability is poor that may cause project over runs and failures. In place of systematic process this process is performed by following a specialized way. It is well known that inappropriate traceability is somewhere a cause for several software failures. Ability of monitoring the life of a requirement, in both forward as well as backward direction i.e. from commencement up to its construction and specification, to its later application and use, through stages of continuous improvement and repetition in any of these stages is termed as requirements traceability. The tester defines the value between 0 to 10 after analyzing each requirement for its traceability. Consider the traceability of the requirement which helps in improvement in software quality.

4.1.7. Execution Time (ET)

The time required for regression testing is depends up on the test suite. However by using effective prioritization technique test cases can be reorder by the tasters to obtain fault detection at increased rate in the system. It should be ensure that the Software has been tasted very carefully. A prioritized test is more effective if execution needs to stop after some time period, it can also be achieved using a random ordering. However, if the desired time for the execution of the test case which is known in advance then a better test case ordering may be possible. The first preference of this research work is to prioritize the regression testing cases.

4.2. Prioritization Technique

Values for all the six factors are assigned for each requirement during the design analysis phase and evolve continually during the software development process as the project evolves. Requirement factor value for each requirement \( i \), \( R_{fv_i} \) is computed as follows:

\[
R_{fv_i} = \frac{\sum_{j=1}^{7} \text{factor}_j}{7}
\]  

(4)

Significantly it represents the requirement factor value for requirement \( i \), which is the mean of factor value. RFV is a measure of importance of testing a requirement and it is used in the computation of test case weight (TCW). After traceability, mapping the TCW of a test case is considered as the product of two elements as follows: (i) The average RFV of the requirements the test case maps to; (ii) The requirements-coverage a test case provides. Requirements-coverage is the fraction of the total project requirements exercised by a test case. With the total of \( n \) requirements, if test case \( t \) maps to \( i \) number of requirements then the test case weight \( T_{cw_i} \) is computed as follows.

\[
T_{cw_i} = \left( \frac{\sum_{a=1}^{i} R_{fv_a}}{n} \cdot \frac{\sum_{b=1}^{n} R_{fv_b}}{n} \right) \times i/n
\]  

(5)

The test cases are sorted for execution based on the descending order of TCW, such that the test case with the highest TCW runs first. The TCW and RFV are given as the input to the GA for the prioritization of test cases. The fitness can be calculated by using the factor values and weight age assigned. The test case with maximum fitness value will be elected as the high priority test case. In this research, the GA reordered the test cases using established criteria without knowledge of the faults in the system. The ordering was determined using test adequacy criteria, which predict how likely errors are to be found rather than how many errors the test actually finds or how many faults exist in the code. Focusing only on the particular test cases based on the prioritization will reduce the computation cost and time.
4.3. Genetic Algorithm

We know that each chromosome consists of genes in GA, a population $P = (c_1, ..., c_m)$ is formed from a set of chromosomes. The GA increases the population of chromosomes by continuously replacing one with another population and it based on fitness function assigned to each chromosome. The strong one go further and the weak chromosome eliminated generation by generation. Crossover and mutation these are two main concepts in genetic algorithm.

4.3.1. Selection

There is a selection pattern to find which are chosen for mating and it is based on fitness and capability of an individual to survive and reproduce in an environment. Selection generates the new one from the old one, thus starting a new generation. Each chromosome is examines in present generation to determine its fitness value. From all that we can say that fitness value of chromosome used to select for the next generation.

4.3.2. Crossover or Recombination

In crossover process, exchange of segments occurs between a pair of chromosomes and crossover is applied on a particular by switching one of its allele with another from another individual in the population. The resultant is very different from its parents. The code below suggests an implementation of individual using crossover:

$$
Child_1 = c \cdot parent_1 + (1 - c) \cdot parent_2 \\
Child_2 = (1 - c) \cdot parent_1 + c \cdot parent_2
$$

4.3.3. Mutation

Mutation is a process wherein one allele of gene is randomly replaced by another to yield new structure. It alters an individual in the population. It can regenerate all or a single allele in the selected individual. To maintain integrity, operations must be secure or the type of information an allele holds should be taken into consideration. Mutation must be aware of binary operations, or it must be able to deal with missing values. A simple piece of code is mentioned below.

$$
child = generateNewChild();
$$

4.3.4. Algorithm

Figure 1 shows the process flow of genetic algorithm. The optimization problems are solved by GA’s recombination and replacement operators, where recombination is key operator and frequently used, whereas, replacement is optional and applied for solving optimization of problem.

Figure 1: Flowchart of Genetic Algorithm
Here, the initial population is automatically generated and the evaluation of the set of candidate solution has been done with the help of genetic algorithm. Here, test case weight (TCW) is used as the stopping criteria. The steps in the algorithm are given below.

**Step 1. Generation of initial population**

Generate ‘n’ number of chromosomes \(c_1, c_2, ..., c_n\)

**Step 2. Initialization of population**

Set Test Suite = No. of chromosomes (\(n\))

**Step 3. Fitness function criterion set**

Set fitness function = test case weight

**Step 4. Select suitable population on the basis of Fitness Function**

SELECT (Best 2 chromosomes based on fitness function)

**Step 5. Genetic Operators Applied**

Do for selected Chromosome(s)

While (all conditions are covered)

Do crossover

Do mutation

Remove Duplicacy

EndWhile

EndFor

**Step 6. Optimization of solution checked.**

If (solution != feasible)

Goto STEP 5

Else END.

The optimal solution is searched in GA on the basis of desired population which further can be replaced with the new set of population. Depend upon the problem, the generation and initialization of test cases (population) is done. Requirement factor value (RFV) and test case weight (TCW) are chosen as the fitness criterion. Henceforth, this fitness function will help in selecting suitable population for problem. Further, the genetic operations are performed. In the beginning, crossover recombines the two individual. Then mutation randomly swaps the individuals. Thirdly, the redundant individuals are removed. Finally, the solution is checked for optimization. If solution is not optimized, then, the new population is reproduced and genetic operators are applied.

### 4.4. Prioritized Test Suite Effectiveness

To gauge the performance of the prioritization technique used in this research, it is must to assess effectiveness of the sequence of the test suite. Effectiveness will be measured by the rate of faults detected. Some metrics were used to calculate the level of effectiveness these are as follows.

#### 4.4.1. APFD Metric

To quantify the goal of increasing a subset of the test suite's rate of fault detection, we use a metric called APFD 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:

\[
apfd = 1 - \frac{(T_{f_1} + T_{f_2} + \ldots + T_{f_m})}{nm} + \frac{1}{2n}
\]

Where,

- \(n\) is the no. of test cases.
- \(m\) is the no. of faults.
- \((T_{f_1}, ..., T_{f_m})\) are the position of first test \(T\) that exposes the fault.
Factors Oriented Test Case Prioritization Technique in Regression Testing using Genetic Algorithm

4.4.2. 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. It would be beneficial to calculate the percentage of test cases that must be run before all faults have been revealed. PTR is calculated as follows:

Let $t$ be the test suite under evaluation, $n$ the total number of test cases in $t$, and $n_d$ the total number of test cases needed to detect all faults in the program under test $p$.

$$\text{Ptr}(t, p) = \frac{n_d}{n}$$ (10)

However, note that the numerator of the PTR equation requires the knowledge of the minimal number of test cases needed to detect all faults. While it is easy to calculate the maximum number of tests needed, test set size minimization is equivalent to the NP-complete minimal set covering problem.

5. Results and Discussion

The Test case prioritization system, which is proposed in this paper, was implemented in the working platform of JAVA (version JDK 1.6). Two types of application projects are used for regression testing. The gradual results obtained during the process are described as follows.

Performance Analysis

The performance of the prioritization technique used in this paper. It is necessary to assess effectiveness of the ordering of the test suite. Effectiveness will be measured by the rate of faults detected. The following metric is used to calculate the level of effectiveness.

Average of the Percentage of Faults Detected (APFD)

Here the performance of the proposed system can be evaluated by using the APFD metric. The APFD value is a measure that shows how quickly the faults are identified for a given test suite set. Let $T$ be the test suite under evaluation, $F$ the number of faults contained in the program under test $P$, $n$ the total number of test cases, and $reveal(i, T)$ the position of the first test in $T$ that exposes fault $i$. The formula for calculating the APFD metric is given below.

$$\text{APFD}(T, P) = 1 - \frac{\sum_{i=1}^{F} \text{reveal}(i, T)}{nf} + \frac{1}{2n}$$ (11)

In project1, the number of test cases $n=8$ and the number of faults $f=6$. This can be represented in the following table:

<table>
<thead>
<tr>
<th>Faults</th>
<th>Test Cases</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: The Faults detected by the test suites in Project1
Number of test cases is 8, i.e., \{T1, T2, T3, T4, T5, T6, T7, T8\}, 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, T6, T1, T4, T2, T3, T7, T8\}, then the APFD metric after prioritization is

\[
Apfd(T, P) = 1 - \frac{(3+3+1+1+1)}{8*6} + \frac{1}{2*8} = 0.8375
\]

The APFD metric before prioritization is

\[
Apfd(T, P) = 1 - \frac{(1+1+3+4+5)}{8*6} + \frac{1}{2*8} = 0.7125
\]

**Figure 2:** APFD metric for the proposed application project

![Figure 2](image_url)

From the Table 1 and Figure 2, it is observed that the prioritized test cases identify the faults at an early stage. The APFD measure of prioritized test cases are higher than the non prioritized order for both projects. In Figure 3 & Figure 4, the Fault identified during each test cases is listed. In Figure 8, Test case 5 can identify more number of faults when compared to others. Thus Test case 5 will be first executed. In Figure 4 also shows that, the Test case 5 identified most number of faults. From Figure 3 and Figure 4, we observed that the proposed method identifies the severe fault in the early stage. So it will reduce the computational time.

**Figure 3:** Fault detection performance of each test case for Project 1

![Figure 3](image_url)
In Figure 3, the test case 5 identifies most number of severe faults. The test case 5 is top in the prioritized order and it is executed first. Thus the proposed prioritization technique will identify most number of severe faults at an early stage.

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 from Figure 4, 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 Figure 4, plotting percentage of test cases executed against percentage of faults detected.

\textbf{Figure 4:} Comparison of Prioritized and Non-Prioritized test cases in Project1

The Figure 4 describes that the Prioritized order of test cases detect all the severe faults in early stage when compared to the non prioritized order of test cases. Thus our proposed method of test case prioritization process will reduce the re-execution time of the project by prioritizing the most important test cases.

6. Conclusion

In this paper the regression testing based test suite prioritization technique is illustrated. A new prioritization technique is proposed for requirement based System level test cases to improve the rate of fault detection of severe faults. This paper recognizes and assesses the challenges coupled with regression testing test case prioritization. The proposed method uses most efficient factors to prioritize the test cases. The beneficiary thing in the proposed method is the trace events process. This factor identifies the important test cases in the project. The effectiveness of the proposed prioritization technique can be evaluated by using the APFD metric. In the proposed method, two types of application projects are used. The proposed method gave the better rate for severe faults detection through results. Also it is tested experimentally that the numbers of test cases run to find the injected fault is less in case of proposed prioritized execution of test cases. Based on the performance measure obtained, the proposed method is effectively prioritizing the test cases in both projects based on the factors and weightage assigned. This will reduce the cost of time of executing the entire project.
References


Factors Oriented Test Case Prioritization Technique in Regression Testing using Genetic Algorithm


