Chapter 2

Factor Oriented Requirement Coverage Based System Test Case Prioritization of New and Regression Test Cases

2.1 Introduction

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. In this chapter a new prioritization technique has been proposed for system level Test Case Prioritization (TCP) from software requirement specification, to improve user satisfaction with quality software that can also be cost effective with improved rate of severe fault detection.

In this proposed prioritization technique, the test cases are prioritized based on weights of the test cases. The procedure for computing the weights of the test cases is two fold. Firstly, to calculate the weights of the requirements, the factors that influence the requirements are identified as (1) Customer assigned priority of requirements (2) Developer-perceived code implementation complexity (3) Changes in requirements (4) Fault impact (5) Completeness and (6) Traceability. Subsequently the impacts of these factors on the requirements are quantified by assigning values in a ten point scale. With these factor values the weight of each requirement is computed. Secondly, the test cases are mapped towards corresponding requirements by establishing a knowledge based mapping between them. Also two new validation techniques are devised to validate the proposed prioritization technique. The first validation technique is based on the analysis of the faults detected and the second is based on the analysis of the
number of test cases executed. The experiments and results of the proposed prioritization technique show improvement in the rate of severe fault detection.

2.2 Related work

The related work found in the literature for test case prioritization, validation metrics and traceability are presented in this section.

2.2.1 Test case prioritization: coverage based

Software testing and retesting (Regression testing) occurs continuously during the software development life cycle to detect the errors as early as possible. M. Harrold has proved in [Har00] that in software development, 50% of the cost is spent in testing activities. So, to optimize the time and cost spent on testing, prioritization of the test cases can be beneficial and this has been tested and proved by Elbaum et al. in [Elb00].

In recent years, several researchers have addressed the test case prioritization problem and presented techniques to solve them for both new and regression testing. These techniques include various coverage based prioritizations. In these prioritizations the test cases are sorted based on the number of lines covered by the test case. Rothermel et al. proposed a code coverage based prioritization technique in [Rot94], that involves ranking the test cases based on the number of code it covers. The coverage-based prioritization strategies presented by Elbaum et al. in [Elb00, Elb02] and Rothermel et al. in [Rot01, Rot99] are summarized hereunder.

2.2.1.1 Comparator techniques

- Random prioritization: In this strategy test cases are ordered in random for execution.
CHAPTER 2. TCP BASED ON FACTOR VALUE

- Optimal prioritization: In this strategy the ordering of test cases is based on the number of faults they expose. This approach assumes that the program faults are given as input and use this information to iteratively select the test case that exposes the largest number of faults unexposed hitherto by already-selected test cases, until, test cases that expose all faults have been chosen. This approach is theoretical as it is not possible to have prior knowledge of all faults present in a program.

2.2.1.2 Statement level techniques

This strategy involves, ranking test cases based on the number of statements covered by the test case such that the test case covering the maximum number of statements would be executed first.

- Total statement coverage prioritization: The total statement coverage prioritization schedules test cases based on the total statement each test case executes. It is likely that the prioritization would allow the same set of statements to be covered by multiple test cases. This ignores the fact that more coverage is feasible by focusing on statements that have not yet been executed by any of the test case selected so far.

- Additional statement coverage prioritization: In this strategy the ordering of test cases is based on the coverage feedback attained in the testing process. This approach first selects the test case with the maximum statement coverage, adjusts the coverage information on the remaining test cases to reflect the statements not covered by that test case, and then iteratively selects a test case that provides the largest additional statement coverage until all program statements have been covered by at least one test case.

- Total branch coverage prioritization: This strategy sorts-out the test cases based on the program branches covered. For testing purposes, branch
coverage is defined as the coverage of each possible outcome of a condition. For example, for each if and while statements, both true and false conditions must be evaluated at least once by the test cases for the statement to have 100% branch coverage.

- Additional branch coverage prioritization: In this strategy ordering of test cases is similar to additional statement coverage prioritization, except that it measures coverage based on program branch and not statements.

2.2.1.3 Function level techniques

- Total function coverage prioritization: In this strategy the ordering of test cases is similar to total statement coverage, except that it measures coverage based on the total number of functions they execute [Elb00, Elb02]. Function coverage measures the number of functions invoked by a test case.

- Additional function coverage prioritization: In this strategy the ordering of the test cases is analogous to additional statement coverage prioritization, except that prioritization takes place at the function level and not at the statement level [Elb00, Elb02].

In [Rot99], Rothermel et al. have pointed out that the potential goal of prioritization is to increase a test suite’s rate of fault detection earlier in the software process. It has been reported in [Che02, Bei90], that the coverage techniques, such as statement or branch coverage, are applicable at the code level. Wei-Tek et. al. reported that these coverage based techniques are also applied to improve regression testing in [Wei01]. Regression testing is an attempt to ensure that changes to a program does not negate the correctness of the software. As the software is often developed and modified in isolated components, testing the software as a single entity is imperative. So, James A Jones et al. [Jam03] have presented an algorithm for test suite prioritization that
incorporates the complexities of modified condition or decision coverage. In [Beo03a] Boehm, proposes a value based approach to software engineering, that measures the value, the system provides to the prospective customers. Since value based software engineering practices are believed to improve user-perceived software quality [Beo03b], a value-driven approach to software testing is presented in this thesis.

Apart from code coverage techniques, test cases are also prioritized based on the requirement coverage. H. Srinath has tested experimentally in [Sri04, Sri05] that some of the major causes for project failures are lack of user input and changing requirements. Hence he has proposed an algorithm called Prioritization of Requirement Testing (PORT), based on the factors that influence the requirements.

2.2.2 Test case prioritization: validation metrics

The benefits of prioritization techniques are measured using various metrics. G. Rothermel et. al. proposed a validation metric, Average of the Percentage of Faults Detected (APFD) in [Rot01]. This is a measure of how quickly the faults are identified for a given test suite. Their APFD values range from 0 to 100 and are monitored during test suite execution. The APFD values represent the area under the curve by plotting percentage of faults detected on the y-axis, and percentage of test suite run on the x-axis of the graph. Similar to APFD metric, Zheng et. al. have proposed three different validation algorithms such as Average Percentage Block Coverage (APBC), Average Percentage Decision Coverage (APDC) and Average Percentage Statement Coverage (APSC) in [Zhe07]. H. Srinath has also proposed another similar validation metric Weighted Percentage of Fault Detected (WPFD) in [Sri05].
Since APFD metric works for equal severity faults, it will provide misleading information, if all the faults are not equally severe. So, Elbaum et al. [Elb01] proposed a new metric APFD\textsuperscript{c}. This metric incorporates fault severity and measures the unit-of-fault-severity-detected-per-unit-test-cost. In the validation of the APFD\textsuperscript{c} metric, Elbaum et al. use six levels of severity to assign severity values to the faults in the program.

### 2.2.3 Requirements traceability

Traceability relationships provide valuable insight into a system. They allow stakeholders to answer questions such as “which messages are related to this requirement?” or “which test cases cover this code module?”. In [Hua03] Huang et al. have described an event-based approach to traceability, where, dynamic links are maintained as publish/subscribe relationships and dependent objects subscribe to the requirements on which they depend. Here the emphasis is made on maintaining an accurate traceability scheme. Zisman et al. [Zis03] describe an approach that automates the discovery of traceability relationships between requirements artifacts. Gotel and Finkelin define requirements traceability as the ability to describe and follow the life of a requirement from its origin to development to deployment in an iterative way in [Got94]. Various researchers have pointed out the significance of the traceability. In [Tah01], L.Tahat et. al. discussed the traceability, as a mapping between requirements and test; if the test cases are not associated with individual requirements it could be difficult for testers to determine whether the requirement is adequately tested or not. Practitioners report that, lack of traceability has been a widely reported problem [Got94]. Also J.Cleland et al. [Cle03] state that poor traceability causes project over-runs and failures. In a case study, Heindl et al. [Hei05] pointed that, often, traceability is conducted in an ad-hoc manner rather than a systematic
process. It is reported in [Bal01] that requirements traceability is imperative to support critical software engineering tasks.

In [Ram96], Ramachandran has stated that there is often no formal way of validating the software process quality due to inadequate traceability between test and requirements. The importance of traceability is reflected by the fact that the U.S Department of Defense expends roughly 4% of its Information Technology (IT) costs on traceability. In [Ram95], Ramesh et al. state that many standards like Department of Defense (DoD) standard 2167A require traceability to be practiced in software development. The business value for the money spent for traceability is often not realized as traceability in organizations is done ineffectively. The need for traceability of non functional requirements (NFR) to functional requirements and design artifacts is discussed by Huang et al. [Hua05]. Studies have shown that various catastrophic software failures are partially due to lack of inadequate traceability and management of NFR. Since traceability is an important characteristic, an approach to traceability that allows traceability between requirements, test cases and their associated defects is presented in this chapter.

It is observed that most of the prioritization techniques found in the literature have considered only coverage information. Also requirements are not quantified and severity levels of faults are not assessed by these techniques. Hence in this chapter a new prioritization technique that incorporates priorities of requirements and severity of faults is proposed. In order to validate the proposed prioritization technique two new validation techniques based on severity of faults and number of test cases executed, are also proposed.

2.3 Proposed prioritization technique

The proposed set of prioritization factors and the prioritization technique are presented in this subsection.
2.3.1 Factors considered for prioritization

To determine the factors involved in the proposed prioritization algorithm, a postmortem analysis is conducted with two student projects and a detailed description of this process is presented in this section.

To determine the proposed prioritization factors, two application projects, Project-A and Project-B, of approximately 1600 LOC, developed by the UG students of Computer Science and Engineering at Bharathidasan Institute of Technology, Trichy, are analyzed and tested (Beta Test) using WINRUNNER testing tool. Test cases for these two projects are developed by two pairs of students from final year Master of Technology in Software Engineering of the same University. Test cases of Project-A are executed for their customer requirements, implementation complexity and requirement changes. Test cases of Project-B are executed for its usability, application flow and fault impact. As in the WINRUNNER testing tool, the faults are classified into four levels i.e. Severity1- Testing must be stopped until the defect is fixed, Severity2- Tester can work around failure but the faults have to be fixed before the product can be deployed, Severity3- Faults can be fixed in later version, and Severity4- Faults cannot be fixed at all. A total of 97 faults (37 faults of severity1, 25 faults of severity2, 20 faults of severity3, and 15 faults of severity4) in Project-A and 33 faults (15 faults of severity1, 10 faults of severity2, 8 faults of severity3) in Project-B are identified in the system test.

In Project-A, 38% of faults are of severity1, 25% of faults are of severity 2 and in Project-B, 45% of faults are of severity1, 30% of faults are of severity 2. So it is planned to identify the percentage of severe fault identification in beta testing. After repeating several tests by changing the order of test cases a conclusion is reached that suitable ordering of test cases can be beneficial in identifying the severe faults earlier in the development phase and can reduce the number of severe faults found in Beta test.
Then the whole system is segregated into six different modules with the first module comprising of changed coding, second module comprising of complex coding, third module comprising of major faults identified in the previous test, fourth, fifth and sixth modules comprising of the coding for customer priority, usability and application flow respectively. The percentages of faults are then mapped to these six different modules as presented in Figure 2.1.

![Figure 2.1 Mapping of Fault - factor - module](image)

In Project-A, 85% faults of severity1 and severity2 are found in module1, module2, module5 and module6. In Project-B, 95% faults of severity1 and severity2 are found in module1, module2, module5 and module6. It is found that modules1, 2, 5 and 6 are the most complex and volatile modules along with application flow, and usability. This result motivated to consider requirements volatility and requirement implementation complexity. Also it motivated to consider usability and application flow, since one of the goals is to produce a customer satisfied product. Since it is planned to use the same TCP technique for
regression testing too, Fault impact of the previous test is also considered as a factor.

Based on the result of the above mentioned postmortem analysis, the following factors are considered in the proposed prioritization technique: (1) Customer assigned priority of requirements, (2) Developer-perceived code implementation complexity, (3) Changes in requirements, (4) Fault impact of requirements, (5) Completeness and (6) Traceability, where the factors (1 to 3) are considered for the new test cases and the factors (4 to 6) are considered for the regression test cases. These factors and the reason for their selection in the proposed prioritization technique are presented hereunder.

2.3.1.1 New test case factors

In this proposed prioritization technique, following factors are considered for the new test cases as they influence much on new software.

i) Customer-assigned Priority (CP): It is a measure of the importance of a requirement to the customer. The customer assigns a value for each requirement that ranges from 1 to 10 where 10 indicates the highest customer priority.

*Reasoning:* A focus on customer requirements for development improves, customer perceived value and satisfaction. So, the requirements that would be of highest importance to the customer should be tested early and thoroughly to improve customer satisfaction.

ii) Implementation Complexity (IC): It is a subjective measure of how difficult the development team perceives the implementation of requirement to be. Each requirement is analyzed and a value ranging from 0 to 10 is assigned by the
developer based on its implementation complexity; a larger value indicates higher complexity.

Reasoning: Requirements with high implementation complexity is expected to have a higher number of faults.

**iii) Requirement changes (RC):** It is based on the number of times a requirement has been altered in the development cycle with respect to its origin date and a value ranging from 1 to 10 is assigned by the developer. If the requirement is altered more than 10 times, the volatility values for all the requirements are quantified on a 10-point scale. The Requirement Changes $RC_i$ for a requirement $i$ can be computed by dividing the number of changes for requirement $i$ to the highest number of changes for any requirement amongst all the project requirements. If the $i^{th}$ requirement is changed $N$ times and the maximum number of requirement changes amongst all the requirements is $M$ then the requirement change of $i$, $RC_i$ is proposed to be computed as

$$RC_i = (N / M) \times 10$$

(2.1)

Reasoning: Roughly 50% of all faults identified in the project are the errors introduced in the requirement phase. The significant factor that causes the failure of the project is attributed to changing requirements.

**2.3.1.2 Regression test case factors**

In this proposed work the following factors are considered as they influence much on reusable software.

**i) Fault Impact of requirements (FI):** It allows the development team to identify the requirement that have customer reported failures. As a system evolves to several versions, the developers can use the prior data collected from versions to
identify requirements that are likely to be error prone. FI is based on the number of field failures and in-house failures. FI is considered for those requirements that have already been in a released product.

In this research work, we propose to calculate Fault Impact of a requirement, based on the severity of the fault identified in the previous run. Faults are classified into 5 levels of severity such as severity1, severity2, severity3, severity4 and severity5 and their severity values are assigned as $2^5$, $2^4$, $2^3$, $2^2$ and $2^1$ respectively.

If a requirement $i$ with the set of $t$ test cases, discovers the set of $d$ faults with the set of $V$ severity values, then the severity $S_i$ of the requirement $i$ is proposed to be computed as

$$S_i = \sum_{x=1}^{t} \sum_{y=1}^{d} V_{x,y}$$

(2.2)

And if $S = \sum_{i=1}^{n} S_i$, where $n$ is the total number of requirements in a project, then the Fault Impact FI of a requirement $i$ is proposed to be computed as

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

(2.3)

Reasoning: Test efficiency can be improved by focusing on the function that is likely to contain higher number of faults.

ii) Completeness (CT): One element of "requirement completeness" is a test to determine that each requirement, individually, is complete for the conditions under which the function is to be performed. Each requirement is analyzed for its completeness and a value ranging from 0 to 10 is assigned by the customer, when that requirement is considered for reuse.
CHAPTER 2. TCP BASED ON FACTOR VALUE

Reasoning: Customer satisfaction such as the quickness of the software response to the user request can be improved by considering the completeness of the requirement.

iii) Traceability (TR): Requirements traceability refers to the “ability to follow the life of a requirement, in both forward and backward direction, i.e., from its origin, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases. Each requirement is analyzed for its traceability and a value ranging from 0 to 10 is assigned by the tester when that requirement is considered for reuse.

Reasoning: The quality of the software can be improved by considering the traceability of the requirement as reported in [Got94].

2.3.2 Proposed technique

In this proposed work, the values for all the six factors are assigned for each requirement during analysis phase. These values tend to evolve continuously during the software development. With \( n \) requirements and \( j \) factor values, a Requirement Factor Value for each requirement \( i \), \( RFV_i \), is proposed to be computed as follows.

\[
RFV_i = \left( \frac{\sum_{j=1}^{6} \text{FactorValue}_j}{6} \right)
\]  

The Requirement Factor Value for requirement \( i \), signifies the mean of factor value. Since RFV is a measure of importance of testing a requirement, it is used in the computation of Test Case Weight (TCW).

Since traceability is an important characteristic, several tools like traceability tree and RebaTe are used in testing software. In this proposed work test cases are mapped to its associated requirements by the testers using an end–
to-end Requirement Traceability tool, TBreq, which is unique in providing end-to-end traceability reports in a single view [Web02].

After having computed the requirement weights and the traceability mapping between the requirements and test cases, the TCW is computed, as a product of the following two elements:

(i) The average RFV of the requirements the test case maps

(ii) The Requirements-coverage a test case provides.

The second element, 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 TCW, is proposed to be computed as follows.

\[
TCW_t = \left[ \frac{\sum_{x=1}^{i} RFV_x}{\sum_{y=1}^{n} RFV_y} \right] \cdot \frac{i}{n}
\]  

(2.5)

TCWs are thus computed for all test cases and these test cases are sorted for execution, based on the descending order of TCW, such that the test case with the highest TCW runs first.

2.3.3 Steps involved in the proposed prioritization technique

The requirements and their corresponding proposed factor values and test cases serve as inputs to the proposed prioritization technique. The sorted test cases in descending order of its weights, is produced as output. The algorithm for the proposed prioritization technique is presented hereunder
Algorithm:

Input: The requirements and their corresponding proposed factor values and test cases.

Output: The sorted test cases in descending order of their weights.

Begin

1. Select the factors to be considered based on the prioritization goal.
2. Get the total number of requirements and total number of test cases planned for the project that is to be tested.
3. Get the factor values for all requirements from the person involved in software development.
4. Obtain each Requirement Weight using Equation (2.4).
5. Using an end-to-end traceability approach, analyze and map the test cases to the respective requirements.
6. Obtain each Test Case Weight using Equation (2.5).
7. Sort the test cases in descending order of its weights and the test cases with a higher weight are run first before other test cases.

End

2.4 Validation Techniques

In this section two new validation techniques are devised to validate the proposed prioritization technique. The first validation technique is based on the analysis of the faults detected for a product and the second validation technique is based on the analysis of the number of test cases executed to detect the faults. These validation techniques are presented in the following subsections.

2.4.1 Validation based on the analysis of the faults detected

In this proposed validation technique, for the purpose of analysis, each failure is assigned a Severity Value (SV), similar to the approach used in APFD©
metric by Elbaum et. al. [Elb01]. The proposed severity levels and their values are determined based on the impact of failures as explained below.

- Highly severe (Very_High (VH) – Severity1) : Severity1 is assigned to a failure that could cause loss of life or property and/or loss of a system. Severity1 failures are assigned a SV of 32 i.e. $2^5$.

- Severe (High (H) - Severity2) : Severity2 is assigned to a failure when a customer can no longer use the product and/or testing must cease until the defect causing the failure is fixed. Severity2 failures are assigned a SV of 16 i.e. $2^4$.

- Medium (Medium (M) - Severity3) : Severity3 is assigned to a failure when there is a work around for failure and the product can be used with the work around. Severity3 failures are assigned a SV of 8 i.e. $2^3$.

- Less severe (Low (L) - Severity4) : Severity4 is assigned to a failure for which a fix can be done in later versions. Severity4 failures are assigned a SV of 4 i.e. $2^2$.

- Least severe (Very_Low (VL) - Severity5) : Severity5 is assigned to a failure for which a fix can be done in later versions or not done at all. Severity5 failures are assigned a SV of 2 i.e. $2^1$.

The proposed severity levels and the values assigned to them are summarized in Table 2.1.
Table 2.1 Severity values

<table>
<thead>
<tr>
<th>Levels of Severity</th>
<th>Severity Notation</th>
<th>Severity Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Severity 1 - Highly severe</td>
<td>Very_High --- VH</td>
<td>32</td>
</tr>
<tr>
<td>Severity 2 - Severe</td>
<td>High --- H</td>
<td>16</td>
</tr>
<tr>
<td>Severity 3 – Medium</td>
<td>Medium---M</td>
<td>08</td>
</tr>
<tr>
<td>Severity 4 – Less severe</td>
<td>Low --- L</td>
<td>04</td>
</tr>
<tr>
<td>Severity 5 - Least severe</td>
<td>Very_Low---VL</td>
<td>02</td>
</tr>
</tbody>
</table>

In this research work the severity values are assigned to the identified faults to enable the computation of Total Severity of Faults Detected (TSFD) for a product, which in turn serves to compute the Average Severity of Faults Detected (ASFD) for each requirement.

**Definition 1:** TSFD is the summation of severity values of all faults identified for a product. TSFD of \( f \) number of faults identified in a product is obtained with the following equation.

\[
TSFD = \sum_{i=1}^{f} SV_i
\]  

(2.6)

**Definition 2:** ASFD is the ratio of summation of severity values of all faults identified for a requirement to the TSFD. Thus the Average Severity of Faults Detected (ASFD) for each requirement with \( f \) defects is obtained with the following relation.

\[
ASFD = \left( \frac{\sum_{j=1}^{f} SV_j}{TSFD} \right)
\]  

(2.7)
CHAPTER 2. TCP BASED ON FACTOR VALUE

The computation of ASFD needs knowledge of the mapping of the faults to the requirements. This mapping is obtained using the proposed end-to-end traceability. ASFD is used to analyze the effectiveness of the prioritization technique and it is intended to experimentally test whether the requirement with a higher computed RFV actually has higher ASFD when the product is system tested.

2.4.2 Validation based on the analysis of execution of test cases

Two terms are proposed to analyze the number of test cases executed to detect the faults: Total Test Effort Index (TTEI) and Average Test Effort Index (ATEI). Total Test Effort Index (TTEI) is defined as the summation of test cases run to detect fault in all faulty versions. In, p number of faulty programs, if $TE_j$ is the number of test cases executed to detect the faults in each faulty program, then TTEI of all faulty programs of a project is computed as follows.

$$\text{TTEI} = \sum_{j=1}^{p} TE_j$$

(2.8)

where $\sum_{j=1}^{p} TE_j$ is the total number of test cases executed in each faulty program from 1 to $p$. TTEI thus computed is used to obtain ATEI. If there are $T$ numbers of total test cases in each faulty program then Average Test Effort Index (ATEI) for each program is defined as:

$$\text{ATEI} = \left( \frac{TTEI}{\sum_{j=1}^{p} T_j} \right)$$

(2.9)
where TTEI is the Total Test Effort Index of each faulty program from 1 to $p$ and 
$$\sum_{j=1}^{p} T_j$$ is the total number of test cases in each faulty program from 1 to $p$.

2.5 Experiments and results

Experiments have been conducted in three categories to measure the effectiveness of the proposed prioritization technique and validation techniques. Category I includes five student projects, Category II includes two industrial projects and Category III includes two industrial case studies. The testing and results of projects in these three Categories are presented in the following subsections.

2.5.1 Category I

To measure the effectiveness of the proposed prioritization technique, five J2EE application projects of size approximately 6000 LOC, developed by the students of Master of Technology in Software Engineering, at Bharathidasan Institute of Technology, are system tested using Quick Test Professional (QTP) as the testing tool.

Each project is given the same set of 25 requirements for development. For each project, the students have been asked to assign values for the factors RC and IC. The role of customer is played by an individual and values are assigned for the factor CP. From the same class, five pairs of students, who possess vast knowledge in software testing, are assigned as testers. These testers have been asked to write test cases for the execution of each requirement and they have also been asked to assign the values for requirement factors CT, TR and FI prior to the start of testing every time. The requirement factor values of the first project are presented in Table 2.2.
### Table 2.2 Factor values for the first student project

<table>
<thead>
<tr>
<th>Req-ID</th>
<th>CP</th>
<th>IC</th>
<th>RC</th>
<th>FI</th>
<th>CT</th>
<th>TR</th>
</tr>
</thead>
<tbody>
<tr>
<td>R001</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>R002</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>R003</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>R004</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>R005</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>R006</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>R007</td>
<td>6</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>R008</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>R009</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>R010</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>R011</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>R012</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>R013</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>R014</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>R015</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>R016</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>R017</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>R018</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>R019</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>R020</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>R021</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>R022</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>R023</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>R024</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>R025</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
For each requirement, RFV is computed using Equation (2.4) as described in subsection 2.3.2. A snapshot of the mapping of the test cases to its associated requirements by the testers is shown in Figure 2.2.

![Software Test Case Prioritizer](image)

Figure 2.2 Mapping between Test Case and Requirement

After traceability mapping, the TCW of each test case is computed as shown in equation (2.5). The test cases are executed based on the descending order of TCW and the faults are detected. Screen shot for the defect view is presented in Figure 2.3.
Based on the faults detected, the student projects are analyzed in two parts. In the first part, we evaluate the faults identified and compute ASFD as given in equation (2.7) and the ASFD for each requirement is mapped to their respective RFV. The results obtained are presented in subsection 2.5.1.1. In the second part, the student projects are tested by applying the proposed technique and in random order of the test cases. The effectiveness of the proposed technique is measured by comparing these two tests. The results obtained are presented in subsection 2.5.1.2.

2.5.1.1 Comparison of ASFD and RFV

Upon completion of the project, the testing group, tests the projects using the testing tool they are familiar with. When a failure is identified by the testers a severity level and its value are assigned to the failure, based on the discussion with developers. For each project, an average of ten faults are identified. These faults are analyzed to determine the requirements it maps. The screen shot for mapping the faults to requirements is presented in Figure 2.4.
The RFV and ASFD for each of the 25 requirements and the TSFD as explained in equation (2.6) for each project is computed. Based on the values RFV is divided into five ranges. The requirements are grouped into one of these five ranges. For each project the mean ASFD value is computed for each range of RFV using equation (2.7). The RFV ranges and mean ASFD values obtained by the proposed technique for all the five projects are presented in Table 2.3. A lower RFV value indicates a lower priority for the particular requirement to be tested.

Table 2.3 Comparison of ASFD and RFV Range

<table>
<thead>
<tr>
<th>RFV range</th>
<th>0-2</th>
<th>2.01-4</th>
<th>4.01-6</th>
<th>6.01-8</th>
<th>8.01-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projects</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project 1</td>
<td>0</td>
<td>2.7</td>
<td>4</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>Project 2</td>
<td>0</td>
<td>2.5</td>
<td>3.3</td>
<td>4.5</td>
<td>7.9</td>
</tr>
<tr>
<td>Project 3</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>7.7</td>
<td>12</td>
</tr>
<tr>
<td>Project 4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3.9</td>
<td>6.5</td>
</tr>
<tr>
<td>Project 5</td>
<td>0</td>
<td>1.8</td>
<td>3.7</td>
<td>5.9</td>
<td>8.4</td>
</tr>
</tbody>
</table>

Figure 2.4 Mapping between Faults and Requirements
The graphical representation of Table 2.3 is plotted as graph and presented in Figure 2.5. The results obtained indicate that ASFD is highest for the RFV range of 8-10. So, the requirements with higher RFV range have higher ASFD values and vice versa. As a result, a higher RFV value indicates a higher priority for the particular requirement to be tested.

2.5.1.2 Effectiveness of the proposed technique

The effectiveness of the proposed prioritization technique is proved by the rate of detection of severe faults. This is done by comparing the execution results of the test cases in the order given by the proposed technique and in random order. Seven faults for the severity levels VH and H (indicating Very High and High) and three faults for the severity levels M, L, and VL, (indicating Medium, Low and Very Low) giving a total of 10 faults, are injected in each of the projects. To test the projects, 50 test cases are written for each of the projects. With the same 25 requirements and their RFV values, TCW for each test case is computed. The test cases are run in the order given by the proposed technique. The faults and their severity are recorded. The faults are mapped to their respective requirements.
The test cases are then run in 20 different random orders and the failures are recorded with their severity levels and severity values. For each of the five projects the following two factors are determined for both prioritizations based on random orders:

- Rate of detection of severe faults
- Contribution of factors towards the effectiveness of the proposed prioritization technique

The experimental results for each of these two factors are discussed below:

The five faulty applications are tested by running the system test cases to find the induced faults in the application. The faults found are mapped to its respective requirements. The TSFD is computed for all the five projects using the severity of the identified faults. After executing each test case, the status is noted as pass or fail. If a test case fails, the severity of the identified fault is noted. Once all the test cases are executed and all the induced faults are identified, their severities are noted. To analyze the rate of fault detection a graph is plotted with the fraction of requirement on the X-axis, and percentage of TSFD on the Y-axis. The rate of detection of fault is computed using a metric called Total Percentage of Faults Detected (TPFD), which is the area under the curve in graph. Twenty random prioritization sets for each of the five projects are generated to allow for statistical comparison. The mean TPFD values for the 20 random orderings are compared against the TPFD achieved for the proposed prioritization technique. Statistical sign test is applied to determine the effectiveness of the proposed prioritization technique in comparison to 20 different sets of random ordered test cases. For each team, percentage of TSFD is determined at different stages of test suite execution after executing one tenth, two tenth, three tenth up to all the test cases.
The results of the TPFD for both random order and proposed prioritized order for all the five projects are presented in the form of a graph in Figure 2.6. For the purpose of depicting the results graphically, the mean TPFD for 20 random permutations is compared with the TPFD for the prioritized order. It is evident from Figure 2.6 that TPFD achieved with the proposed technique for all five projects is higher than the mean TPFD for 20 random permutations. TPFD for 20 different random order sets and one set of proposed prioritized order for all the five projects are presented in Table 2.4. The mean TPFD values for 20 random orders are also listed in the Table 2.4.

To measure the effectiveness of the proposed prioritization scheme, the following null and alternative hypotheses are considered:

\[ H_0: \text{TPFD for proposed prioritization scheme} = \text{Mean TPFD for Random set}. \]
\[ H_a: \text{TPFD for proposed prioritization scheme} > \text{Mean TPFD for Random set}. \]

It can be observed from Table 2.4 that TPFD value with the proposed prioritization technique is greater than that of random set, for all the five projects. So it is evident that the statistical results are in favor of \( H_a \) and the proposed prioritization scheme is better than random prioritization (\( p < 0.001 \)). The results indicate that the proposed prioritization scheme leads to improved rate of failure detection for all the five projects. The difference between proposed prioritization scheme and mean for twenty permutations are also found to be significant (\( p < 0.001 \)).
Figure 2.6 Comparison of rate of fault detection of Random testing and proposed prioritized testing
### Table 2.4 TPFD for proposed prioritization scheme and Random order

<table>
<thead>
<tr>
<th>Random Permutations</th>
<th>TPFD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Project1</td>
</tr>
<tr>
<td>Random Permutation 1</td>
<td>85.57</td>
</tr>
<tr>
<td>Random Permutation 2</td>
<td>68.25</td>
</tr>
<tr>
<td>Random Permutation 3</td>
<td>43.81</td>
</tr>
<tr>
<td>Random Permutation 4</td>
<td>40.29</td>
</tr>
<tr>
<td>Random Permutation 5</td>
<td>43.29</td>
</tr>
<tr>
<td>Random Permutation 6</td>
<td>52.21</td>
</tr>
<tr>
<td>Random Permutation 7</td>
<td>40.06</td>
</tr>
<tr>
<td>Random Permutation 8</td>
<td>35.88</td>
</tr>
<tr>
<td>Random Permutation 9</td>
<td>36.77</td>
</tr>
<tr>
<td>Random Permutation 10</td>
<td>41.77</td>
</tr>
<tr>
<td>Random Permutation 11</td>
<td>38.97</td>
</tr>
<tr>
<td>Random Permutation 12</td>
<td>53.85</td>
</tr>
<tr>
<td>Random Permutation 13</td>
<td>36.77</td>
</tr>
<tr>
<td>Random Permutation 14</td>
<td>49.93</td>
</tr>
<tr>
<td>Random Permutation 15</td>
<td>36.97</td>
</tr>
<tr>
<td>Random Permutation 16</td>
<td>18.43</td>
</tr>
<tr>
<td>Random Permutation 17</td>
<td>36.83</td>
</tr>
<tr>
<td>Random Permutation 18</td>
<td>30.38</td>
</tr>
<tr>
<td>Random Permutation 19</td>
<td>33.97</td>
</tr>
<tr>
<td>Random Permutation 20</td>
<td>45.79</td>
</tr>
</tbody>
</table>

#### Mean TPFD-Random

| Mean TPFD-Random | 43.49 | 51.00 | 49.67 | 43.49 | 51.14 |

#### Prioritized TPFD

| Prioritized TPFD | 66.71 | 83.5  | 54.43 | 66.71 | 64.12 |

#### Number of times proposed scheme better than Random order

| Number of times proposed scheme better than Random order | 18 | 20 | 17 | 18 | 20 |

#### Test Statistic

| Test Statistic | 4.99 | 8.48 | 7.49 | 4.99 | 8.84 |

#### P value

| P value | <0.001 | <0.001 | <0.001 | <0.001 | <0.001 |

#### Sign Statistic

| Sign Statistic | 18 | 20 | 17 | 18 | 20 |

#### sign-test p-value

| sign-test p-value | =0.0002 | <0.0001 | =0.0013 | =0.0002 | <0.0001 |
Alternatively with the sign test, it may be concluded with the null hypothesis that the TPFD for proposed prioritization scheme is no better than that of a randomly-chosen prioritization. The sign test is conducted as simple nonparametric procedure that makes no assumption about the distribution of TPFD. In the 20 randomly-chosen prioritizations for the projects, the proposed prioritization scheme is observed to have a better TPFD for 18 random orders. Instead of sign test, experiment with binomial distribution is conducted and it is observed that the probability of observing 18 or more successes under the null hypothesis of equivalence is $p=0.0002$. This is a highly significant result and indicates that the TPFD with the proposed prioritization scheme is higher than the median of all permutations. The last four rows of Table 2.4 present the Test Statistic, P value, Sign Statistic and sign-test p-value for all five projects.

### 2.5.2 Category II

In order to validate the effectiveness of the proposed prioritization technique, one Visual Basic (VB) project \(\text{(Project-1)}\) of approximately 5000 LOC and one Personal Home Page (PHP) project \(\text{(Project-2)}\) of approximately 6000 LOC, developed by TECHZONE™, Software Development and Testing, Trichy, India, are considered. During the development of projects, 20 requirements are chosen for each of the projects. For analysis and design, the requirement analysts are requested to use Rational Requisite pro and Rational Rose software from IBM. Developers are required to write 50 test cases from the requirements and the design diagrams. Using the customer requirements, developers are required to give values for CP, IC, RC, CT and TR. During regression testing, developers are required to give FI values from the previous tests and the values are normalized to 10. During coding, RFV is calculated for all the requirements of both the projects as given in equation 2.4. The programs are then thoroughly tested by the testers using rational test suite. The project
coordinator creates 10 faulty programs (5 faulty programs from each project) by seeding one fault in each, invariant of the severity. On the entire faulty programs, prioritized test cases are run, and the execution of the total number of test cases to identify the fault is computed. Then 20 different random orders of test are generated using random number generation in ‘C’ language. The test cases are executed in these 20 different random orders and the total numbers of test cases run to detect the faults are found. The mean value of 20 different results is computed for each of the 10 faulty programs. The results of fault detection in both the cases are compared to strengthen the effectiveness of the proposed prioritization technique. The test cases for the five faulty programs of Project-1 and five faulty programs of Project-2 project are executed both with the proposed technique and with 20 different random orders.

During the execution of test cases of all faulty programs of Project-1 with the proposed prioritized order, for the first faulty program, the fault is detected after running 10 test cases. For second, third, fourth and fifth faulty programs, 25, 13, 6 and 4 test cases are executed respectively to detect the faults. TTEI and ATEI are computed as defined in equations (2.8) and (2.9) as follows.

\[
TTEI_{\text{Prioritized}} = 10 + 25 + 13 + 6 + 4 = 58
\]
\[
ATEI_{\text{Prioritized}} = \frac{58}{250} = 0.23
\]

During the execution of test cases of all faulty programs of Project-1 in 20 different Random orders, for the first faulty program, the fault is detected after running an average of 20 test cases. For second, third, fourth and fifth faulty programs, an average of 45, 15, 30 and 25 test cases are executed respectively to detect the faults. TTEI and ATEI are computed as follows.

\[
TTEI_{\text{Random}} = 20 + 45 + 15 + 30 + 25 = 135
\]
\[
ATEI_{\text{Random}} = \frac{135}{250} = 0.54
\]
On comparing ATEI$_{\text{Prioritized}}$ and ATEI$_{\text{Random}}$ to detect all the induced faults, 23% of test cases are run in ATEI$_{\text{Prioritized}}$ and 54% of test cases are run in ATEI$_{\text{Random}}$. So the number of test cases to be executed to find all the faults is less in case of the proposed prioritized technique and in turn it reduces the cost of testing. Also lower the value of Average Test Effort Index, the better is the prioritization technique.

Similarly, for the faulty programs of Project-2, ATEI$_{\text{Prioritized}}$ and ATEI$_{\text{Random}}$ are computed. The values obtained for both the projects are presented in Table 2.5.

<table>
<thead>
<tr>
<th>Project</th>
<th>ATEI$_{\text{Prioritized}}$</th>
<th>ATEI$_{\text{Random}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project-1</td>
<td>0.230</td>
<td>0.540</td>
</tr>
<tr>
<td>Project-2</td>
<td>0.240</td>
<td>0.640</td>
</tr>
</tbody>
</table>

### 2.5.3 Category III

Two industrial case studies are analyzed by conducting a post hoc analysis of the test efforts at Cosmosoft Technologies limited, software technology parties of India, a leading developer of ERP projects and Tech Zone, a leading developer of JAVA projects in India. The first case study (Cosmosoft Technologies) involves, analyzing employee self service portal written in JAVA language. It comprises of 73,525 changed lines of code, built upon approximately a 300,000 lines of code base; 25 modified requirements, and 100 system level test cases. The second case study (Tech Zone) involves in analyzing BSNL token sending program, written in JAVA, comprises of 100,000 lines of code, 20 new requirements and 100 system level test cases.

To measure the rate of failure detection, the proposed prioritization technique is compared with three different test case prioritization techniques viz.
(i) Untreated, (ii) Total Statement Coverage, (iii) Total Method Coverage. Descriptions of these techniques are as follows:

(i) No prioritization (untreated): One control that we consider is simply the application of no technique; this allows to consider “untreated” test suites.

(ii) Total statement coverage prioritization (statement-total): For any test case, the number of statements in a program that are exercised by this test case can be determined by instrumenting the program. These test cases can be prioritized according to the total number of statements they cover.

(iii) Total method coverage prioritization (method-total): Total method coverage prioritization is the same as total statement coverage prioritization.

In this research work coverage information is obtained by running the test cases over instrumented object programs using the EMMA [Vla05] - an open source toolkit for measuring and reporting Java code coverage. EMMA distinguishes itself from other tools by going after a unique feature combination: support for large-scale enterprise software development while keeping individual developer’s work fast and iterative. This information lists the test cases exercised with the statements and methods that suits the requirement. A previous version’s coverage information is used to prioritize the current set of test cases.

The following steps are performed for the two industrial case studies.

1. Computation of RFV for the requirements by applying equation (2.4)
2. Computation of TCW for the test cases by applying equation (2.5)
3. Prioritize the test cases according to the proposed prioritization technique.
4. Computation of coverage information for all test cases.
5. Sort the test cases based on statement coverage and function coverage.
6. Execute the test cases in proposed prioritization order, original order, statement coverage order and method coverage order.
7. Computation of TSFD for both the projects which is a summation of the severities for all project failures by applying equation (2.6).

For both case studies, the engineering teams provide the research team with project requirements, test cases, test failures, mapping of failures to test cases, and mapping of test cases to requirements, for conducting this analysis. Also the engineering teams provide the factor values for the requirements. RC is assigned based on the following two factors:

1) New requirements added to the release
2) Existing requirements that were modified for the release.

CP is assigned based on the needs of customers. IC is assigned by the engineering team based on the implementation difficulties perceived for the requirements. FI is assigned based on the failures achieved from previous release. FI is applicable to Cosmosoft - ESSP - case study and not applicable for TechZone - JAVA - case study as the project is going through initial release and has no prior field data. CT and TR are assigned by the engineering team based on the user friendliness of the product.

After executing each test case, the result is noted as pass or fail, the severity values are obtained for the faults and TSFD is computed for the projects under test. The test results of the prioritized order, original order, statement coverage order and function coverage order are analyzed and the rate of failure detection is determined using the metric TPFD as described in the subsection 2.5.1.2. TPFD values for 25%, 50% 75% and 100% of execution of requirements is computed and a graph is plotted with the fraction of requirement executed on
X-axis and the corresponding TPFD on Y-axis for both the projects and the same is presented in Figure 2.7.

![Cosmosoft - ESSP](image)

![TechZone - JAVA](image)

Figure 2.7 Comparison of the proposed prioritization technique with other prioritization schemes

From Figure 2.7(a) it is evident that the rate of detection of severe faults of the proposed prioritization technique is 27% higher than original order, 5% higher than statement coverage order and 4% higher than method coverage order.
Also from Figure 2.7(b) it is evident that the TPFD of the proposed prioritization technique is 30%, 3% and 2% higher than original order, statement coverage order and method coverage order respectively.

2.6 Conclusion

In this chapter, a new prioritization technique for requirement based system level test cases is proposed, to improve the rate of fault detection of severe faults both in the new test and in the regression test. Here, three factors are proposed for new test cases and three factors for regression test cases. The proposed prioritization technique is validated through three categories of experiments with five student projects, two industrial projects, and two industrial case studies. Results indicate that the proposed technique leads to improved rate of detection of severe faults in comparison to random ordering of test cases. 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. Sign test is conducted to investigate both the null hypothesis and the alternate hypothesis and in the TPFD for the proposed prioritization technique is no better than that for a randomly chosen technique and for the latter the TPFD for the proposed prioritization technique is greater than that for a randomly chosen order. Another prioritization technique with two new factors and factor weight is proposed and presented in the next chapter.