Chapter 4

Factor Oriented Requirement Coverage and Test Cost Based Test Case Prioritization and Its Metric

4.1 Introduction

Test case prioritization techniques and metrics, proposed in previous chapters, assumed that requirement weights and test case costs are uniform. In this chapter two new test case prioritization techniques have been proposed based on the Requirement Weight \((\text{ReqWeight})\) and Test Case Cost \((\text{TCCost})\). Also a new metric is proposed to measure the effectiveness of the proposed prioritization technique. In these proposed prioritization techniques, test cases are prioritized based on Test Case Weights \((\text{TCW})\). The procedure for computing \(\text{TCW}\) is two fold. Firstly, the factors that influence the requirement are identified from the previous test information and are assigned values in a 10 point scale. The assigned Requirement Factor Values \((\text{ReqFactorValue})\) are used to evaluate Requirement Factor Weight \((\text{ReqFactorWeight})\). Using \(\text{ReqFactorValue}\) and \(\text{ReqFactorWeight}\), \(\text{ReqWeight}\) of each requirement is computed. Secondly, \(\text{TCCost}\) for each test case is obtained from the previous test information. Based on \(\text{ReqWeight}\) and \(\text{TCCost}\), \(\text{TCW}\) of each test case is computed and sorted, in two different ways viz. based on Additional requirement coverage and Total requirement coverage. Experiments are conducted to evaluate the effectiveness of the proposed prioritization techniques and the proposed metric.

4.2 Related work

Regression testing is an expensive process used to validate new features and detect regression faults, which occurs continuously during the software development lifecycle. Software engineers often save the test cases and reuse
them later during regression testing. However as stated in [Leu89, Ono98] due to time and resource constraints, it may not be possible to execute all the test cases on every testing iteration. Therefore testers may want to sort out the test cases such that those with higher priorities, according to some criterion, are run earlier than those with lower priorities. Various techniques and empirical studies have been reported on prioritizing test cases in recent years. In [Won97] Wong et al. proposed a hybrid technique that combines modification, minimization and prioritization-based selection using source code and test history. Elbaum and Rothermel studied different prioritization techniques based on source code and test history, and indicated that prioritization techniques can significantly improve rate of fault detection [Elb04a, Rot01]. In [Kim02] Kim et al. modeled, regression testing as an ordered sequence of testing sessions and presented a history based prioritization technique utilizing the information from previous testing. Time budget is also incorporated in regression testing. In [Wal06] Walcott et al., presented a time aware prioritization approach. To improve the rate detection of faults, test cases are prioritized using relevant slices by Jeffery et al. in [Jef06]. Search algorithms are also used to prioritize the test cases. Li et al. discussed these search algorithms in [Li07].

To illustrate how rapidly a prioritized test suite detects faults, Rothermel proposed a metric called APFD in [Rot99]. APFD value ranges from 0 to 100%. An ordered test suite with higher APFD value has better fault detection rate than those with lower APFD values. Elbaum et al. proposed an improved metric APFD$^S$ for measuring the fault detection that allows for varying test costs and fault severities.

Most of the previous prioritization techniques relied on the assumption that testing requirement weights and test case costs are uniform. But, in practice, requirement weights and test case cost can vary widely. Therefore, improvisation is made in this research as follows: (1) to consider varying test case cost along with varying testing requirement weights; (2) to propose a suitable metric that
where $k$ represents the total number of requirement factors. Higher requirement weight implies a higher importance for the requirement to be tested.

Having computed $\text{Req Weight}$ of each requirement, $\text{TCW}$ of each test case has to be computed. To compute $\text{TCW}$ for each test case, $\text{TCCost}$ for each test case is obtained from the previous test information. The $\text{TCW}$ of each test case is obtained by means of $\text{Req Weight}$ of each requirement and $\text{TCCost}$ of each test case. Then the test cases are sorted based on its $\text{TCW}$. To obtain $\text{TCW}$ and to sort the test cases, two types of test case prioritization techniques are proposed viz.

1. Prioritization based on Additional Requirement Coverage ($\text{Addtl_TCP}$)
2. Prioritization based on Total Requirement Coverage ($\text{Total_TCP}$)

These two prioritization techniques are presented in the following subsections.

### 4.3.1 Additional requirement coverage prioritization technique ($\text{Addtl_TCP}$)

The $\text{Addtl_TCP}$ adjusts the coverage information for remaining test cases, and iteratively selects a test case that yields the greatest coverage of requirements not yet covered. In $\text{Addtl_TCP}$, based on the computed value of $\text{Req Weight of each requirement}$, additional requirement weight is computed. For a test case $tc$, the Additional Requirement Weight denoted as $\text{AddtlReqWeight}(tc)$, is computed using the following relation.

$$\text{AddtlReqWeight}(tc) = \frac{\text{Sum of unsatisfied requirement weights covered by the test case}}{\text{Sum of requirement weights not yet covered}}$$  \hspace{1cm} (4.3)

Having obtained $\text{AddtlReqWeight}$ and $\text{TCCost}$ for each requirement, $\text{TCW}$ of each test case is computed. For a test case $tc$, the $\text{TCW}(tc)$ is computed using the following relation
Here, the larger \( Addtl\, Req\, Weight(tc) \) and the smaller \( TCCost(tc) \) give larger \( TCW(tc) \). Three different relations are obtained based on the requirement weight and test case cost. Theses relations are presented hereunder.

i. When the requirement weights are not considered, \( Addtl\, Req\, Weight(tc) \) is equal to the ratio of number of unsatisfied requirements covered by \( tc \) to the total number of requirements not yet covered, denoted as \( Addtl\, Req\, Covg(tc) \).

ii. When all the test costs are equal, \( TCW(tc) \) is decided by \( Addtl\, Req\, Weight(tc) \).

iii. When both requirement weights and test costs are identical, \( TCW(tc) \) only depends on \( Addtl\, Req\, Covg(tc) \), in this situation, \textit{Addtl_TCP} is equivalent to the conventional technique that is based on additional requirement coverage.

Having computed the weights of the test case, the test cases are sorted as follows.

i. Firstly, test cases are sorted in descending order of \( TCW \).

ii. When the \( TCW \) of test cases are equal, then the test cases are sorted in ascending order of total number of Requirements Satisfied by the Test Cases (\textit{RSTC}). The test case which covers the higher \textit{ReqWeight} with the less number of requirements should be selected first. In this way, the requirements with higher \textit{ReqWeight} will be satisfied.

The algorithm for the proposed prioritization technique \textit{Addtl_TCP} is presented hereunder:


Algorithm: Addtl_TCP

Input:
- Set of requirements
- Set of test cases

Output:
- Final_list: Prioritized test cases

Begin
1. Final_list := null;
2. Obtain Requirements Factors and TCCost from previous tests;
3. Assign ReqFactorValue for each factor;
4. Compute ReqFactorWeight using equation(4.1);
5. For each requirement repeat steps 6 to 14
6. Compute ReqWeight for each requirement using equation(4.2);
7. For each test case repeat steps 8 to 10
8. Compute RSTC;
9. Compute AddtlReqWeight(tc) using equation(4.3);
10. Compute TCW(tc) using equation(4.4);
11. Sort the test case in descending order of TCW
12. If there exists equal TCW test cases then Sort the test cases in ascending order of RSTC;
13. Select the first test case from the sorted list;
14. Add the selected test case to the final list, Final_list = Final_list + selected test case;

End

4.3.2 Total requirement coverage prioritization technique (Total_TCP).

The Total_TCP does not adjust the coverage information for remaining test cases. It merely prioritizes the test cases based on TCW and RSTC.
CHAPTER 4. TCP BASED ON TEST CASE COST

In Total_TCP, based on ReqWeight of each requirement, total requirement weight is computed. For a test case \( tc \), the Total Requirement Weight, denoted as \( \text{TotalReqWeight}(tc) \) is computed using the following relation.

\[
\text{TotalReqWeight}(tc) = \frac{\text{Sum of weights of requirements satisfied by the test case}}{\text{Sum of weights of all requirements}}
\] (4.5)

Having computed the ReqWeight for each requirement, the TCW is computed using the TotalReqWeight, and TCCost. For a test case \( tc \), the \( TCW(tc) \) is computed using the following relation.

\[
TCW(tc) = \frac{\text{TotalReqWeight}(tc)}{\text{TCCost}(tc)}
\] (4.6)

Similar to Addtl_TCP, the test cases are sorted in descending order of their weights. And if there occurs test cases with equal TCW then the test cases are sorted in descending order of RSTC.

The algorithm for the proposed prioritization technique Addtl_TCP is presented here under:

**Algorithm: Total_TCP**

**Input:** Set of requirements  
Set of test cases

**Output:** Prioritized test cases

**Begin**

1. Obtain Requirements Factors and TCCost from previous tests;
2. Assign ReqFactorValue for each factor;
3. Compute ReqFactorWeight using equation(4.1);
4. Compute ReqWeight for each requirement using equation(4.2);
5. For each test case repeat steps 6 to 8
6. Compute RSTC;
7. Compute TotalReqWeight(tc) using equation (4.5);
8. Compute TCW(tc) using equation (4.6);
9. Sort the test case in descending order of TCW;
10. If there exists equal TCW test cases then sort the test cases in ascending order of RSTC;

End

4.4 Proposed validation metric

A new metric $ReqSatTcc$ (Requirement Satisfied by Test case cost) based on test requirement weights and test case cost is devised to validate the proposed prioritization technique.

The proposed metric is represented in terms of graph. The horizontal axis of the graph is represented by “Percentage of total TCCost incurred” and the vertical axis is represented by “Percentage of total ReqWeight satisfied.” Under this interpretation, a test case’s contribution is weighted along the horizontal dimension in terms of test cost, and along the vertical dimension in terms of the cumulative requirement weights it satisfies. The proposed metric $ReqSatTcc$ can be quantitatively described as follows: Let $T$ be a test suite containing $q$ test cases with test costs $tcc_1, tcc_2, \ldots tcc_i, \ldots tcc_q$, Let $R$ be a set of requirements satisfied by $T$, and let $rep_1, rep_2, \ldots rep_j, \ldots rep_p$ be the weights of these requirements. Let $TRj$ be the first test case in an ordering $T'$ of $T$ that satisfies requirement, $re_j$. Then, the $ReqSatTcc$ value of prioritized test suite $T'$ is given by the relation:

$$ReqSatTcc = \frac{\sum_{j=1}^{p} rep_j \times \left( \frac{\sum_{i=TR_j}^{q} tcc_i - \frac{1}{2} tcc_{TR_j}}{\sum_{j=1}^{p} rep_j \times \sum_{i=1}^{q} tcc_i} \right)}{\sum_{j=1}^{p} rep_j \times \sum_{i=1}^{q} tcc_i} \quad (4.7)$$
ReqSatTcc values range from 0 to 100. The higher ReqSatTcc value indicates the better performance of $T$. Equation (4.7) remains applicable when either requirement weights or test costs are identical. Furthermore, when both requirement weights and test costs are identical, the formula can be simplified to:

\[
\text{ReqSatTcc} = 1 - \frac{TR_1 + TR_2 + \ldots + TR_p}{pq} + \frac{1}{2n}
\]  

(4.8)

From equation (4.7) and (4.8) it is evident that APFD$^C$ [Elb01] and ReqSatTcc evaluate the performance of prioritized test suite from different profiles respectively. An increased ReqSatTcc, improves the testing quality and customer satisfaction. It provides earlier feedback on the system under test, supporting faster strategic decisions about release schedules. Moreover, if the testing period is cut short, it can increase the possibility of executing those test cases that cover the higher testing requirement weights during the available testing time.

4.5 Experiments and results

The effectiveness of the proposed prioritization technique is measured by conducting experiments in two categories using the proposed validation metric ReqSatTcc and the validation metric ATEI, proposed in chapter 2. In Category I the research method is applied on an industrial case study. In Category II experiments are conducted on two industrial projects. The testing and results of the projects in these two categories are presented in the following subsections.

4.5.1 Category I

To measure the effectiveness of the proposed prioritization technique, an industrial case study, that involves the analysis of employee self service portal written in JAVA language, and comprises of approximately 300,000 lines of
code, is 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. From the previous test information the requirement factors are identified as Requirement Changes (RC), Fault Impact of the requirement (FI) and Customer Priority (CP). To present the process of proposed prioritization technique and metric, a case study with a set of requirements, test cases and test case cost is considered and is presented hereunder.

A set of testing requirements $R=\{re_1, re_2, re_3, re_4, re_5, re_6\}$, test suite $T=\{tc_1, tc_2, tc_3, tc_4\}$, and requirements satisfied by the test cases are presented in Table 4.1. $ReqFactorValue$ for all the factors are assigned based on previous test information. Using these factor values, $ReqFactorWeight$ for each requirement is computed. Then, the weight of each requirement is obtained using equation (4.3) and it is recorded in $ReqWeight[6] = \{6, 9, 7, 6, 3, 3\}$. From the previous test information the test case costs of $T$ are assigned as $TCCost[4]=\{2, 3, 2, 3\}$ from the previous test information. The requirement factor values, $ReqFactorWeight$ and $ReqWeight$ are presented in Table 4.2.

### Table 4.1 The relation between test suite and requirements

<table>
<thead>
<tr>
<th>Requirements</th>
<th>$re_1$</th>
<th>$re_2$</th>
<th>$re_3$</th>
<th>$re_4$</th>
<th>$re_5$</th>
<th>$re_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test case</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Tc_1$</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Tc_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>$Tc_3$</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Tc_4$</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Table 4.2 Requirement Factor Value and Weight

<table>
<thead>
<tr>
<th>ReqFactor Weight</th>
<th>ReqFactor</th>
<th>re</th>
<th>re1</th>
<th>re2</th>
<th>re3</th>
<th>re4</th>
<th>re5</th>
<th>Re6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>RC</td>
<td>3</td>
<td>10</td>
<td>9</td>
<td>2</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>FI</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>CP</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

TCW for each test case is computed and the test cases are prioritized based on three different techniques:

1. Addtl_TCP
2. Total_TCP
3. Random-Ordering

Finally metrics for all the three techniques are computed using ReqSatTcc. The result of each technique is presented as follows.

(1) The results of Addtl_TCP

From $R=\{re_1, re_2, re_3, re_4, re_5, re_6\}$ and $T=\{tc_1, tc_2, tc_3, tc_4\}$, $re_1$ and $re_2$ are the requirements satisfied by $tc_1$ and the cost of $tc_1$ is 2. The additional requirement weight if $tc_1$ is computed as follows

$$Addtl\text{ReqWeight}(tc_1) = \frac{6+9}{6+9+7+6+3+3} = 0.441$$

$$TCW(tc_1) = 0.441/2 = 0.221$$

Similarly $TCW$ for all the test cases are computed and the results are presented in Table 4.3.
Table 4.3 Prioritization by Addtl_TCP

<table>
<thead>
<tr>
<th>R={re_1, re_2, re_3, re_4, re_5, re_6}</th>
<th>T={tc_1, tc_2, tc_3, tc_4}</th>
<th>( TCW(tc_1) )</th>
<th>R={re_3, re_5, re_6}</th>
<th>T={tc_2, tc_3, tc_4}</th>
<th>( TCW(tc_2) )</th>
<th>R={re_3}</th>
<th>T={tc_2, tc_4}</th>
<th>( TCW(tc_3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( TCW(tc_1) )</td>
<td>0.221</td>
<td>--</td>
<td>( TCW(tc_2) )</td>
<td>0.228</td>
<td>0.233</td>
<td>( TCW(tc_3) )</td>
<td>0.221</td>
<td>0.237</td>
</tr>
<tr>
<td>( TCW(tc_4) )</td>
<td>0.157</td>
<td>0.175</td>
<td>( TCW(tc_5) )</td>
<td>0.221</td>
<td>--</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SORTEDLIST</td>
<td>{tc_1}</td>
<td>{tc_1, tc_3}</td>
<td>{tc_1, tc_3, tc_4}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Initially from \( R={re_1, re_2, re_3, re_4, re_5, re_6} \) and \( T={tc_1, tc_2, tc_3, tc_4} \), \( TCW(tc_1) = TCW(tc_3) = 0.221 \),
  Requirements satisfied by \( tc_1 = 2 < \) Requirements satisfied by \( tc_3 = 3 \),
  So, \( tc_1 \) is selected first.
- Secondly, while \( R={re_3, re_4, re_5, re_6} \), \( T={tc_2, tc_3, tc_4} \),
  \( TCW(tc_3) > TCW(tc_2) > TCW(tc_4) \),
  So, \( tc_3 \) is selected next
- Thirdly, while \( R={re_3, re_6} \), \( T={tc_2, tc_4} \),
  \( TCW(tc_4) > TCW(tc_2) \)
  So, \( tc_4 \) is selected next.

All of the requirements have been satisfied by the test cases \( \{tc_1, tc_3, tc_4\} \) at this stage. Finally, the remaining test case \( tc_2 \) is also added and at the end, the proposed prioritized list is \( \{tc_1, tc_3, tc_4, tc_2\} \).
- Using equation (4.7), the metric is computed as \( ReqSatTcc = 71.5\% \) and it is presented in Figure 4.1.
(2) The results of Total_TCP

- From R={re_1, re_2, re_3, re_4, re_5, re_6} and T={tc_1, tc_2, tc_3, tc_4},
  \[ TCW(tc_1) = TCW(tc_3) = 0.221, \]
  Requirements satisfied by \( tc_1 = 2 < \) Requirements satisfied by \( tc_3 = 3, \)
  So, \( tc_1 \) is selected first.
- Secondly
  \[ TCW(tc_3) > TCW(tc_2) > TCW(tc_4), \]
  So, \( tc_3 \) is selected
- Thirdly
  \[ TCW(tc_2) > TCW(tc_4), \]
  So, \( tc_2 \) is selected

Finally, the remaining test case \( tc_4 \) is also added and at the end, the obtained prioritized list is \{tc_1, tc_3, tc_2, tc_4\}.

- Using equation(4.7), the metric is computed as \( ReqSatTcc = 68.8\% \) and it is presented in Figure 4.2.
(3) The results of **Random-Ordering**

Test cases are selected and executed in the original sequence of T. Thus, the prioritized list is \{tc_1, tc_2, tc_3, tc_4\}.

- Using equation (4.7), the metric is computed as $\text{ReqSatTcc} = 69.4\%$ and it is presented in Figure 4.3.
Comparison of the metric results of the three prioritization techniques Addtl_TCP, Total_TCP and Random-ordering are presented in Figure 4.4. On this comparison, the performance of Addtl_TCP (71.5%) with the sorted list \{tc_1, tc_3, tc_4, tc_2\} is greater than the performance of Random-Ordering (69.4%) with the sorted list \{tc_1, tc_2, tc_3, tc_4\} and the performance of Random-Ordering (69.4%) with the sorted list \{tc_1, tc_2, tc_3, tc_4\} is greater than the performance of Total_TCP (68.8%) with the sorted list \{tc_1, tc_3, tc_2, tc_4\}. Finally on comparison, the prioritized performance of Addtl_TCP is best in ReqSatTcc Metric.

![Comparison of three prioritization techniques](image)

Figure 4.4 Comparison of three techniques Addtl_TCP, Total_TCP and Random-ordering

4.5.2 Category II

In category II, the effectiveness of the proposed prioritization technique is validated based on the number of test cases executed to detect the induced faults. For validation purpose, the same set of projects (VB project-Project-1 and PHP project-Project-2) each with 20 requirements and 50 test cases, used in section 2.5.2 of chapter 2 is considered. After thorough testing of the programs using rational test suite, the overall coordinator of the projects is allowed to create 10 faulty programs (5 faulty programs from each project) by seeding one fault in
each, invariant of the severity. The test cases are prioritized based on the proposed technique. On the entire faulty programs, the test cases that are prioritized based on the proposed technique are run and the total numbers of test cases executed, to identify the faults are noted. The test cases are then executed in 20 different random orders and the total numbers of test cases run to find faults are detected. The mean value of these 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. Test cases for the five faulty programs of Project-1 and five faulty programs of Project-2 are executed both in proposed Addtl_TCP technique and in random order.

During execution of test cases of all faulty programs of Project-1 in 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 8, 7, 10 and 5 test cases are executed respectively to detect the faults. TTEI and ATEI are computed using equations (2.8) and (2.9):

\[
TTEI_{Prioritized} = 10 + 8 + 7 + 10 + 5 = 40 \\
ATEI_{Prioritized} = \frac{40}{250} = 0.160
\]

During execution of test cases of all faulty programs of Project-1 in Random order, for the first faulty program, the fault is detected after running 31 test cases. For second, third, fourth and fifth faulty programs 33, 27, 25 and 37 test cases are executed respectively to detect the faults. TTEI and ATEI are computed:

\[
TTEI_{Random} = 31 + 33 + 27 + 25 + 37 = 153 \\
ATEI_{Random} = \frac{153}{250} = 0.612
\]

Similarly, for the faulty programs of Project-2, ATEI_{Prioritized} and ATEI_{Random} are computed. The values obtained for both the projects are presented in Table 4.4.
Table 4.4 ATEI of Project-1 and Project-2 for TCP based on test case cost

<table>
<thead>
<tr>
<th>Project</th>
<th>ATEI Prioritized</th>
<th>ATEI Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project-1</td>
<td>0.160</td>
<td>0.612</td>
</tr>
<tr>
<td>Project-2</td>
<td>0.171</td>
<td>0.710</td>
</tr>
</tbody>
</table>

On comparing ATEI\textsubscript{Prioritized} and ATEI\textsubscript{Random} of Project-1, to detect all the induced faults, 16% of test cases are run in ATEI\textsubscript{Prioritized} and 61% of test cases in ATEI\textsubscript{Random}. Similarly to detect all the induced faults in Project-2, 17% of test cases are run in ATEI\textsubscript{Prioritized} and 71% of the test cases are run in ATEI\textsubscript{Random}. So the number of test cases executed to find all the faults, is less in case of the proposed prioritized technique and the reduces the cost of testing is reduced.

The proposed technique reduces the number of test cases executed by approximately 2% (for Project-1 18% to 16% and for Project-2 19% to 17.1%) when compared with the factor weight based prioritization technique proposed in chapter 3 and 7% (for Project-1 23% to 16% and for Project-2 24% to 17.1%) when compared to the factor value based (without factor weight) technique proposed in chapter 2. Comparisons between the proposed techniques are presented in Table 4.5.

Table 4.5 Comparison of the proposed TCP techniques with factor values, factor weights and test case cost

<table>
<thead>
<tr>
<th>Project</th>
<th>Proposed Prioritization Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Chapter 2 (With factor values)</td>
</tr>
<tr>
<td></td>
<td>ATEI\textsubscript{Prioritized}</td>
</tr>
<tr>
<td>Project-1</td>
<td>0.23</td>
</tr>
<tr>
<td>Project-2</td>
<td>0.240</td>
</tr>
</tbody>
</table>
4.6 Conclusion

In this chapter a new prioritization technique is proposed with requirement weight and test case cost for requirement based system level test cases to improve requirement coverage at minimal cost and to improve the customer satisfaction.

The proposed prioritization algorithm is validated through two validation algorithms with an industrial case study and two industrial projects respectively. The test cases involved in industrial case study are sorted in three different prioritization techniques viz. Addtl_TCP, Total_TCP and Random ordering. Metrics for requirement coverage and cost incurred for all the three techniques are computed by $ReqSatTcc$ and are compared. With the two industrial projects faulty programs are created for each project by seeding faults in each program. These injected faults are detected both in Addtl_TCP prioritized execution of test cases and random execution of test cases. ATEI is computed for each project and it is proved that the cost of testing is less in case of proposed prioritized execution of test cases. The proposed technique increases the requirement coverage at minimal cost and also improves customer satisfaction. However the mathematical approaches used to prioritize the test cases can be made more accurate using soft computing techniques such as fuzzy logic and genetic algorithm and the same are presented in the subsequent chapters.