Int J Syst Assur Eng Manag (June 2021) 12(3):361–376
https://doi.org/10.1007/s13198-021-01056-4

ORIGINAL ARTICLE

Test scenario prioritization from user requirements for web-based software

Namita Panda[1] · **Durga Prasad Mohapatra**[2]

Received: 29 October 2019 / Revised: 5 August 2020 / Accepted: 9 January 2021 / Published online: 28 March 2021

Abstract User requirements are the building blocks for development of software applications. User requirements decide the width and breadth of any software. Nowadays, test scenarios are prepared from the user requirements, which give the test engineers ample scope to review the test plan thoroughly before doing software testing. Regression testing is carried out to know the effect of requirement changes on the functionalities and performance of the software. Test scenario prioritization, which is one of the techniques to perform regression testing, maximizes the ease of debugging for the system under test. The code based test scenario prioritization and model based test scenario prioritization have their own limitations. So, to achieve ease of debugging and to get more time for reviewing the test plans, researchers are now working on test scenario prioritization using requirements collected from the end users. In this paper, we propose an approach named Requirement based test scenario prioritization to prioritize test scenarios using requirements collected from end users for developing software applications. The user's functional requirements are collected and is assigned with some weight depending upon different factors like complexity of implementing the requirements, type of release of the requirements, requirement volatility, coupling between requirements etc. Test scenarios are generated from requirements collected from the end users. Then, the final priority weight of each test scenario is found out by considering the weight of each requirement covered by the corresponding test scenario and the percentage of requirements coverage made by each test scenario. The test scenarios are prioritized based on the final priority weight. The proposed approach is evaluated using average percentage of fault detection metric and is found to be very efficient in early test scenario prioritization and detection of faults.

Keywords Regression testing · Requirement volatility · User requirements · Test case prioritization · APFD metric

1 Introduction

Software testing is one of the indispensable phases of Software Development Life Cycle (SDLC) (Mall 2018; Chauhan 2018; Pressman and B.R.M. 2019). In software development life cycle, testing accounts for as much as 50% of total development efforts and cost. User requirements are the only input for the testing. Different phases of software testing, starting from test scenario generation, test execution and test report evaluation require the user requirements as input (Swain et al. 2014; Panthi et al. 2018; Panthi 2017; Mahali and Mohapatra 2018). The recent trends in software testing aims at generating test scenarios from the user requirements. As it is very difficult to formalize the user requirements, it is a challenge to generate test scenarios, so that the test scenarios generated should satisfy more number of requirement coverage criteria (Krishnamoorthi and Mary 2009). The challenges for

✉ Namita Panda
npandafcs@kiit.ac.in

Durga Prasad Mohapatra
durga@nitrkl.ac.in

[1] School of Computer Engineering, KIIT Deemed to be University, Bhubaneswar, India

[2] Department of Computer Science and Engineering, National Institute of Technology, Rourkela, India

generating test scenarios from the user requirements increases many folds, when the software gets modified.

The modifications in software includes addition of new functionalities, modifying existing requirement etc. After doing the required modifications, the system is again completely retested to ensure that the modifications do not affect the existing functionalities of the system. The retesting of the system is called *Regression Testing* (Chauhan 2018; Mathur 2008; Yoo and Harman 2007; Samuel and Joseph 2008; Jayant and Rana 2011). Retesting incurs more effort & time and at the same time induces doubt over the coverage of all possible faults within the product delivery time line. Test scenario prioritization is a efficient technique to address the limitations of retesting and it also increases the average percentage of fault detection (Rothermal et al. 2001; Wang and Zeng 2016; Rava and Wan-Kadir 2016).

Test scenarios are prioritized, so that major faults can be detected early during regression testing. The prioritization activity is done by taking inputs from system design or code. If test scenarios are prioritized prior to system design or code in the development life cycle, then test scenarios can be validated early. There exists different attributes of a user's requirement like type of release, end user assigned priority, developer assigned priority etc., which can be collected from the user specification documents. These parameters can be used to effectively prioritization the test scenarios. So, there is a need of developing a method to prioritize test scenarios using different attributes of user's requirements.

In the related research it is found that, first code based test scenario prioritization techniques and later model based test scenario prioritization had played a vital role in test scenario prioritization. The source code and the system design are taken as input for the test scenario prioritization respectively. The code based test scenario prioritization takes input as the source code and has the limitations like unavailability of source code for outsourced software, and delay in test scenario generation which is only possible after the code is ready. Similarly, model based test scenario prioritization is completely depended on system models, hence test scenarios can only be prioritized after the system design is ready and traceable with the user requirements. Though, model based test scenario prioritization addresses the limitations of code based test scenario prioritization, but still one has to wait till the design is completed to start reviewing the test plan and generating the test scenario prioritization. So, for early generation of test scenarios, through review of test plans before actual testing and early detection of faults during the regression testing, user requirement based test scenario prioritization will be an efficient approach. So, we have set the objective of this paper to test scenario prioritization using use requirements.

In this paper, we propose an approach to prioritize test scenarios using user's requirements. The approach is named Requirement based Test Scenario Prioritization (RTSP). We have only considered the functional requirements collected from the user. Each requirement is assigned with some weight depending upon different factors like complexity of implementation of the requirements, type of release of the requirements, requirement volatility, coupling between requirements etc. First, test scenarios are generated from user requirements. Then, the priority weight of each test scenario is calculated by considering the weight of each requirement covered by the corresponding test scenario and the percentage of requirements coverage made by each test scenario. The test scenarios are prioritized based on the priority weight. The proposed approach is evaluated using Average Percentage of Fault Detection (APFD) metric and is found to be very efficient in early detection of faults. As, we generate system test plan at requirement specification level, the proposed approach is applicable to system level testing only.

The remaining paper is arranged as follows: Sect. 2 discusses the related work. Section 3 discusses proposed approach and Sect. 4 discusses the working of the proposed approach using a case study of Online Shopping Cart(OSC). The implementation and results are described in Sect. 5. Section 6 discusses comparison of the proposed work with the existing related work. The limitation of the proposed approach is listed in Section 7. Section 8 discusses the conclusion and the possible future extension to our work.

2 Related work

In this section, we have briefly discussed the related research works proposed by several researchers. All the approaches referred in this paper are confined to test case prioritization from user requirements (Mukherjee and Patnaik 2018; Kumar and Sujata Kumar 2010; Srivastava et al. 2008; Wang and Zeng 2016; Roongruangsuwan and Daengdej 2010; Kavitha et al. 2010; Arafeen and D 2013; Abbas et al. 2019; Luo et al. 2019; Hettiarachchi and Do 2019; Hafeez 2019).

Mukherjee et al. (Mukherjee and Patnaik 2018) surveyed different approaches of test case prioritization. Their study covers requirement based test case prioritization, test case prioritization using historical information, and model based test case prioritization. Mogyorodi (Mogyorodi 2002) over viewed requirement based testing(RBT) techniques which design minimum number of test cases from requirements. RBT helps to conduct testing in parallel with development. The distribution of bug report indicates that 56% of all bugs are rooted in requirement phase while

design and coding phase yield 27% and 7% respectively (Mogyorodi 2002).

If all requirements are given equal importance, then a value neutral approach is created (Mukherjee and Patnaik 2018). To overcome this problem, Srikanth et al. (2013) have designed PORT (Prioritization Of Requirements for Testing) which was a value driven and system level prioritization technique. PORT considers four factors such as customer assigned priority, requirement implementation complexity, requirement volatility and fault proneness of requirements. These four factors help in computing the prioritization factor value of each requirement and finally test case prioritization is done by finding the weighted priority of each associated test case.

Zhang et al. (2007) presented a metric named "units-of-testing-requirement-priority-satisfied-per-unit-test-case-cost". The testing requirement priority changes frequently and testing cost also vary which makes this metric more important for prioritization.

Krishnamoorthi et al. (2009) proposed two more factors, i.e. completeness and traceability, as regression test case factors. A system level test case prioritization technique like PORT (Srikanth et al. 2013) was proposed. Yoon (2013) proposed a new technique to measure the risk exposure value of different risk items originated from each requirement and then prioritized the test cases.

Roongruangsuwan et al. (2010) proposed two new prioritization techniques to resolve the problem of assigning the same weight to many test cases, and to effectively prioritize multiple test suits while minimizing the prioritization time. The authors considered four factors such as cost factors, time factors, defect factors and complex factors. The cost factors include execution cost, cost of analysis, and cost of data preparation. The time factors include execution time, time consumed for data preparation, and time consumed for validation. The defect factors include defects occurred and defect severity. The complex factors include test case complexity, requirement coverage, dependency, and test impact.

Rothermel et al. (2001) classified the test case prioritization techniques into 4C categories i.e. customer requirement based technique, coverage based technique, cost effectiveness technique, and chronographic history based technique. Kavitha et al. (2010) proposed an approach to prioritize test cases using three different factors i.e. customer priority, changes in requirement, and implementation complexity. The proposed approach improved the severity of fault detections. The authors have highlighted the addition of fault severity as one of the factors in future.

Marchetto et al. (2016) proposed a test case prioritization technique, which maximizes the fault discovery that are both technical and business critical. The proposed approach is multi-objective in nature i.e. it focuses on early discovery of faults as well as reducing the execution cost of test cases. The author applies metrics and algorithms to identify critical and fault-prone portion of the software which is assigned with more priority value. The major component of the proposed approach is requirements, code, and test cases. The requirements deals with requirement traceability link where as the code and test cases facilitate the execution of test cases. The metrics used automatically assigns weights to the code as well as the requirements.

Nayak et al. (2017) proposed a test case prioritization technique to enhance the efficiency of the test case prioritization by improving the rate of fault detection. They considered four factors- rate of fault detection, number of fault detected, the ability of the test case for risk detection, and test case effectiveness, for test case prioritization. The author considered five levels of severity values for risk detection. All the values of the four factors are added to find out the weight of each test case and based on the value assigned, the test cases are prioritized.

Ouriques et al. (2018) presented a replicated study on test case prioritization for the model based systems. They surveyed that, model based testing is based on structural element of the model and the derived test cases. Previous related work on model based testing focuses on labeled transition system and the factors considered are amount of branches, joins, and loops etc. But, these factors have little influence on the effectiveness of test case prioritization. The characteristics of the test case that actually fails has definitely influence on this. The author concluded that, there is no best performer among the investigated techniques and characteristics of test cases that fails represents an important factor.

Singh et al. (2018) proposed object-oriented coupling based test case prioritization. The author considered class diagrams present in object-oriented systems. They first investigated the coupling oriented object-oriented metrics for each class. Then, the coupling weight was assigned to each class based on some threshold. They considered the dependency graphs to estimate the coupling measure. Then, the aggregated coupling weight is determined from classes covered by a test case. Finally, the test cases are ranked based on the aggregate weight.

Hettiarachchi et al. (2016) focused on the use of system requirements and their risks which enables software testers to identify more important test cases and the faults associate with it. Their approach made the requirements risk estimation process more systematic and precise by reducing subjectivity, impression, and inconsistency issues using a fuzzy expert system. The proposed approach increases the effectiveness of the test case prioritization and detects the related faults early for high-risk system components compared to the control techniques. The proposed

approach has main four steps i.e. estimate risks by correlating with requirements, calculate the risk exposure for the requirements, calculate the risk exposure for risk items, and prioritize requirements and test cases. The author considered four risk indicators previously used by other researchers for test case prioritization. The indicators are requirements complexity, requirements size, requirements modification status, and potential security threats. The authors empirically evaluated the new approach using two Java applications with multiple versions. The results of this study demonstrated that our new systematic, risk-based approach can detect faults earlier and is even better at finding faults in the risky components earlier than the control techniques. With the proposed approach, software companies can manage their testing and release schedules better by providing early feedback to testers and developers so that the development team can fix the problems as soon as possible. The proposed approach has few limitations also. The approach is determining the relationships between the requirements and risk items by human experts, but this can be improved by using some semi-automated approach. Secondly, the authors have only considered three triangular membership functions for both input and output variables as well as used centroid defuzzification method to defuzzify output variables. The use of these methods and functions may affect the result. So, different other defuzzificztion methods and different membership functions like trapezoidal, Gaussian may be used and final result may be analysed. The authors have only considered four risk indicators, but different other risk indicator like usage rate, dependency among requirements can be considered.

Hettiarachchi et al. (2019) proposed an improved approach over their own approach (Hettiarachchia et al. 2016). In this research, the authors introduced a fuzzy expert system that emulates human thinking to address the subjectivity related issues in the risk estimation process in a systematic so that the effectiveness of test case prioritization can be improved. Further, the authors gathered required data for our approach by employing a semi-automated process that made the risk estimation process less subjective. The empirical results indicate that the new prioritization approach can improve the rate of fault detection over several existing test case prioritization techniques, while reducing threats to subjective risk estimation. The authors have proposed a requirements risk-based test case prioritization approach, which uses the fuzzy expert system (FES) to estimate the risks in software requirements. The new approach has three major steps i.e. estimate requirements risks using FES, estimate requirements risks using the weighted sum model (WSM), and prioritize requirements and test cases. Compared to the previous approach of the author Hettiarachchia et al. (2016), this new approach simplified the risk estimation process by removing a major step, reducing the number of risk indicators, and eliminating the risk items from the risk estimation process. The results indicated that the technique with the trapezoidal wave type and four membership functions produces better results than the other techniques. Though in this research, authors considered three risk indicators: Requirement Modification Level (RML), Requirement Complexity (RC), and Potential Security Risk (PSR), however, more domain specific risk indicators may produce better risk estimations and thus yield better results.

3 Proposed approach

In this section, the proposed approach is discussed in detail. We have proposed an approach for test scenario prioritization using user requirements. We have named our approach: Requirement based Test Scenario Prioritization (RTSP). The detailed steps of our approach for test scenario prioritization using user requirements are explained below:

Step-1: **Maintain the Requirements Repository with weights**

In this step, different software requirements are collected and stored in a repository. Along with the requirements, different attributes are also maintained in the repository. Different metrics given in Eq.1 to Eq. 7 are proposed to calculate the final weight of the user requirements. The calculations are explained in the following steps:

- *Mention the type of each user requirement* The type of user requirements may be classified as *No Change (NC)*, *New Requirement (NR)* and *Modified Requirement (MR)*.
- *Calculate the value of the factor for Requirement Volatility (RV)*. Volatile requirements are those requirements which are more likely to change. The system requirements may change more frequently, but generally some requirements change more frequently than other requirements (IanSommerville 2010). The Requirement Volatility (RV) and Total Volatility (TV) for a requirement can be calculated by using Eq.1 and Eq.2, respectively. In Eq. 1, Total no. of modifications till the current version is the accumulated number of modifications for all requirements and sub-requirements after the first release of the software. The counting starts when the first modification is made to the software. Here, $R_{i.j}$ represents the sub requirement of R_i.

$$RV(R_{i,j}) = \frac{No.\ of\ times\ R_{i,j}\ modified}{Total\ no.\ of\ modifications\ till\ the\ current\ version} \quad (1)$$

$$Total\ volatility\ (TV(R_i)) = \sum RV(R_{i,j}) \quad (2)$$

- *Set the weights for different factors*: The weights for the different factors are assigned as follows:
  - Assign *End User priority* $PE(R_{i,j})$ to each requirement, which is set by the end user as per their business requirements, in a scale of 1 to 3. A requirement assigned with 3 as the end user priority treated as the highest priority.
  - Assign *Developer assigned priority* ($CL\ R_{i,j}$) to the requirements depending upon the complexity of their implementation. We have only considered the technical complexity of the requirement as accessed by the developer.The complexity level $CL\ R_{i,j}$ will be categorised into High(3), Medium (2), and Low(1). The final end user priority of a sub requirement and the total end user priority of the requirement can be calculated by using the formula given in Eq.3 and Eq.4, respectively

$$Final\ priority\ of\ (R_{i,j}) = FP(R_{i,j}) = [PE(R_{i,j}) * CLR_{i,j}] \quad (3)$$

$$Total\ Priority\ of\ the\ Requirement\ TP(R_i) = \sum FP(R_{i,j}) \quad (4)$$

  - Assign weight to the *type of release* ($TR(R_i)$)and find the *updated priority* ($UP(R_i)$). Based on the type of release for the requirement, a value will be assigned in the range of 0 to 3, to calculate the updated priority. The types of release are - have no impact and never important (0), may be required after the current version (1), may be implemented in the current version(2), must be implemented in the current version (3).The updated priority can be calculated by using Eq. 5. The formula given in Eq. 5, calculates the final priority of the requirement by multiplying developer assigned complexity value with the end user assigned priority. So, higher the value of the developer assigned complexity, higher will be the final priority of the requirement.

$$Updated\ priorityUP(R_i) = TP(R_i) * TR(R_i) \quad (5)$$

  - Assign the error *proneness weight* ($EP\ (R_i)$)to each requirement depending upon the total number of faults identified in the respective requirements during the testing of previous version of the software. Zero will be assigned for the new requirements.
  - Assign weights to all the requirements depending upon the *level of coupling*. Coupling is defined as the degree of inter-dependency between any two requirements. A coupling factor will be decided depending on the type of coupling. There are five types of coupling i.e. content coupling, common coupling, control coupling, stamp coupling, and data coupling. If we consider the software requirement specification (SRS) document, one requirement may contain another requirement. This give rise to control coupling between requirements. Similarly, when two requirements share the same input data, it gives rise to data coupling. But, rest of the coupling like content, and common can't be identified from requirement specifications. These coupling factors can only be derived from system models or code. We have not considered stamp coupling in our proposed approach which can be one of the limitation of our approach. Here, we have only considered *data coupling* and *control coupling* Mall (2018). Data coupling is assigned with a coupling factor of 0.5, whereas control coupling has been assigned with a coupling factor of 1. The weight for the coupling factor of each requirement can be found out by using the metric proposed in Eq.6.

$$COUL(R_i) = Coupling\ Factor * No.\ of\ requirements\ onwhich\ R_i depends \quad (6)$$

The final weight of each requirement can be calculated using Eq.7

$$W(R_i) = TV(R_i) \times UP(R_i) \times EP(R_i) \times COUL(R_i) \quad (7)$$

The block diagram of the proposed approach for test scenario prioritization from user requirements is given in Fig. 1.

Step-2: Generate the test scenarios from the requirements repository

In this step, the test scenarios $TS_{i,j}$ are generated using the coverage based technique. The coverage based technique aims at 100% requirement coverage. The test scenarios generated are put in a traceability matrix. All the requirements covered by each test scenario can be found out from the traceability matrix. The mapping between test scenarios created, and the corresponding requirements can be represented as given in Eq. 8.

$$R_{i,j} \rightarrow TS_{i,j} \quad (8)$$

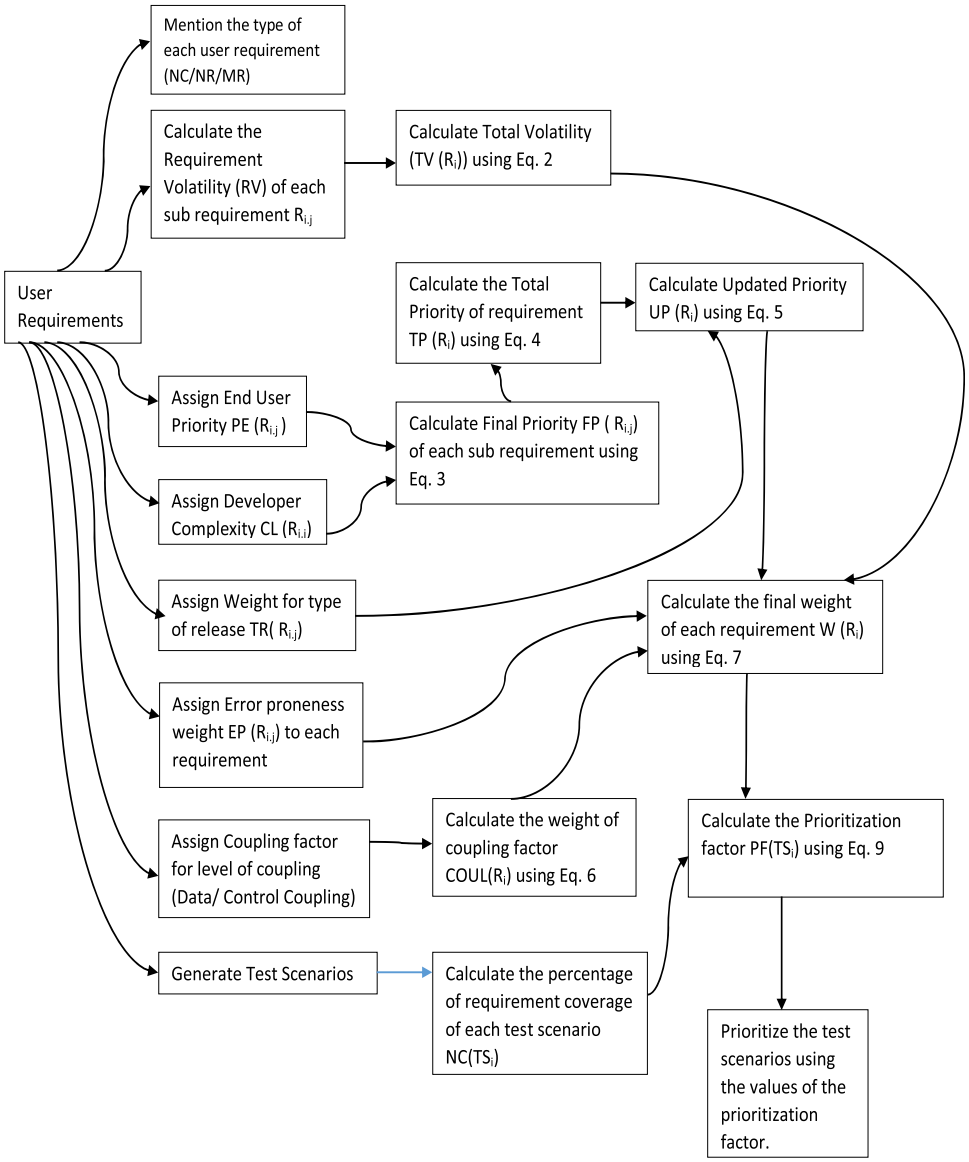

Fig. 1 Block Diagram of Test Scenario Prioritization from User Requirements

where $TS_{i,j} = \{T_1, T_2, T_3, \ldots\ldots\ldots\ldots T_n\}$ $T_1, T_2, T_3 \ldots T_n$ represents the test scenarios generated from the user's requirements.

Step-3: **Prioritize and evaluate the test scenarios**

The test scenarios are prioritized considering different factors and evaluated using the APFD metric. The different subtasks involved in this step are as follows:

- Find out the percentage of requirements covered by each test scenario. The percentage of requirement coverage by each test scenario is calculated using the formula given in Eq. 11.
  Here, we have considered all the sub-requirements also while taking into account the total number of requirements. So, the formula also addresses those cases where the requirement is partly validated.
- Calculate the value of *Prioritization Factor* (PF(TS_i)) of each test scenario present by using Eq.9.
- Arrange the test scenarios in the ascending order of the values of prioritization factor PF(TS_i). If two test scenarios have the same PF value, arrange them randomly.

$$PF(TS_i) = WR(TS_i) \times NC(TS_i) \tag{9}$$

where WR(TS_i)is the sum of final weight of each sub-requirements of a requirement covered by test scenario TS_i, and NC(TS_i) is the percentage of requirement coverage of each test scenario TS_i.

- Evaluate the proposed prioritization approach using APFD metric. APFD metric can be evaluated using the formula and given in Eq. 10:

$$APFD = 1 - \frac{TF1 + TF2 + - - - - - + TFi}{mn} + \frac{1}{2n} \quad (10)$$

where, T → Test suite under evaluation **m** → *No. of faults contained in the program* **n** → *Total number of test cases* **TFi** → *Position of first test case in T that exposes fault i*

The proposed approach is represented through Algorithm 1 and Algorithm 2. First, The weight for each requirement is calculated by using Algorithm 1 and finally, the test scenarios are prioritized using Algorithm 2.

Algorithm 1 Requirement Weight Calculation Algorithm

Input: Software Requirement Specification document containing the Requirements (R_i)
Output: Final weight of each Requirement (R_i)

Identify each requirement (R_i) from the specification document, where i=1, 2, 3.....,n and n is the total number of requirements in the specification document.
Store the requirements description and type of requirements from the set No Change(NC), New Requirement(NR), Modified Requirement(MR)
for $i = 1$ **to** n **do**
    Identify and store for each requirement (R_i), the corresponding sub-requirements ($R_{i,j}$), where j=1, 2, 3,.....,m, m is the number of sub-requirements for each requirement
end for
for $i = 1$ **to** n **do**
    for $j = 1$ **to** m **do**
        //Calculate and store the requirement volatility $RV(R_{i.j})$ for each sub requirement.
        $RV(R_{i,j}) = \frac{No.\ of\ times\ R_{i.j}\ modified}{Total\ no.\ of\ modifications\ till\ the\ current\ version}$
    end for
end for
for $i = 1$ **to** n **do**
    for $j = 1$ **to** m **do**
        //Calculate the Total Volatility $TV(R_i)$.
        $TV(R_i) = \sum RV(R_{i,j})$
    end for
end for
for $i = 1$ **to** n **do**
    for $j = 1$ **to** m **do**
        //Assign weights for different factors
        Set $TP(R_i)$ to 0.
        Input End User Priority $PE(R_{i,j})$ for each sub-requirement $PE(R_{i,j})$
        Input Developer Assigned Priority ()$CL(R_{i,j})$ for each sub-requirement
        //Calculate and store the final priority (FP($R_{i.j}$)) of each requirements
        $FP(R_{i.j}) = [PE(R_{i.j}) * CLR_{i.j}]$
        //Calculate Total priority $TP(R_i)$
        $TP(R_i) += FP(R_{i.j})$
    end for
end for
for $i = 1$ **to** n **do**
    //Identify the Type of Release $TR(R_i)$ and calculate the Updated Priority ($UP(R_{i,j})$) of each requirement (R_i)
    Calculate Updated priority $UP(R_i) = TP(R_i) * TR(R_i)$
    Determine input Error Proneness weight ($EP(R_i)$) for each requirement
end for
for $i = 1$ **to** n **do**
    Input the type of coupling, number of function the requirement depends on and calculate the dependency weight through coupling factor ($COUL(R_i)$)
    $COUL(R_i) = Coupling\ Factor * No.\ of\ requirements\ onwhich\ R_i depends$
end for
for $i = 1$ **to** n **do**
    //Calculate and return the final weight $W(R_i)$ of each requirement
    $W(R_i) = TV(R_i) \times UP(R_i) \times EP(R_i) \times COUL(R_i)$
end for

Algorithm 2 Test Scenario Prioritization Algorithm

Input: Test Scenario Suite $TS = TS_1, TS_2, TS_3, ..., TS_n$ and Requirement-Test Scenario Traceability Matrix
Output: Prioritized Test Scenario Sequence
1: Read the test scenarios TS_i
2: Call Requirement Weight Calculation Algorithm to get the weight of each requirement $W(R_i)$
3: **for** $i = 1$ **to** n **do**
4: //Calculate the weight of each test scenarios $WR(TS_i)$
5: Read Requirement-Test Scenario Traceability Matrix
6: Intitialize $WR(TS_i)$ to Zero
7: **for** $j = 1$ **to** m **do**
8: // m=Number of requirements covered by Test Scenario TS_i
9: $WR(TS_i) + = W(R_j)$
10: **end for**
11: **end for**
12: //Calculate the requirement coverage (NC) of each test scenario TS_i
13: **for** $j = 1$ **to** m **do**
14: $NC(TS_i) = \frac{No.\ of\ requirements\ R_i\ covered\ by\ TS_i}{Total\ no.\ of\ requirements}$
15: **end for**
16: //Calculate the priority value $PF(TS_i)$ of each scenario TS_i
17: **for** $j = 1$ **to** n **do**
18: $PF(TS_i) = WR(TS_i) \times NC(TS_i)$
19: **end for**
20: Arrange and Print the Test Scenarios (TS) in descending order of their priority value $PF(TS_i)$

4 Case study: online shopping cart

In this section, we have considered a case study of **Online Shopping Cart**. The major functionalities of the shopping cart include *login*, *search item*, *view shopping cart*, *manage shopping cart*, *make payment*, and *checkout*. The sub-functionalities of each functionality of the Online Shopping Cart are also identified and represented in Table 1.

The detailed description of the proposed approach w.r.t. the online shopping cart case study is discussed below:

- *Maintain the requirement repository with weights* In this step, the weight of each requirement is calculated using Eq. 1 to Eq. 7. The type of requirement and the calculated requirement volatility factor for each sub-requirement listed in Table 1, are given in Table 2. In the case study of OSC, the type of all the requirements is ***MR*** except requirement R_6 which is of type ***NO***, and the total number of modification in the OSC application is 16.
The total requirement volatility of each requirement (R_i) is calculated using Eq. 2 and given in the Table3. The updated priority of each user requirement is calculated using Eq. 5 and given in Table 3. Different factors are quantified for the calculation of updated priority. The end users depending upon their organizational business need, must prioritize the requirements. The end user priority and developer complexity, for each requirement have been shown in Table 3. The final priority is calculated by multiplying the end user priority with the complexity level assigned by the

Table 1 Functionalities and Sub-functionalities of Online Shopping Cart Casestudy

Sl No.	Name of the Functionality	Name of Sub Functionalities (*if any*)
1	Login	–
2	Search Item	Browse Item
		Choose Item
3	View Shopping Cart	–
4	Manage Shopping Cart	Add item to Shopping Cart
		Remove Item from Shopping Cart
		Update Shopping Cart
5	Make Payment	Select Payment Mode
		Enter Payment Credentials
		Generate payment receipt
6	Checkout	–

Table 2 Requirement volatility of each sub requirements

Requirement ID	Name of sub requirement	No. of Times Modified	RV ($R_i.j$)
R_1	Login	1	0.06
$R_{2.1}$	Choose Item	1	0.06
$R_{2.2}$	Browse Item	2	0.12
R_3	View Shopping Cart	2	0.12
$R_{4.1}$	Add item to Cart	1	0.06
$R_{4.2}$	Remove item from Cart	1	0.06
$R_{4.3}$	Update Cart	2	0.12
$R_{5.1}$	Select Payment Mode	2	0.12
$R_{5.2}$	Enter Payment Detail	2	0.12
$R_{5.3}$	Generate Receipt of Payment	2	0.12
R_6	Check Out	0	0

Table 3 Total volatility of requirements

Requirement ID	TV (R_i)
R_1	0.06
R_2	0.18
R_3	0.12
R_4	0.24
R_5	0.36
R_6	0

developer. Further, the total priority of any requirement R_i is calculated by adding the final priority of all the respective sub requirements. Then, the type of release is identified as per the discussion in Section 3 and the release weight is assigned. Finally, the updated priority is calculated.

The error proneness value of each sub requirement is identified depending upon the number of faults identified, and listed in Table 5. The average of the error proneness of sub requirements is calculated to get the error proneness of requirement R_i. Then, the coupling factor for each sub requirement is identified as per the discussion in Section 3 and the average of it is calculated to get the coupling factor for R_i. Then, the final weight of each requirement is calculated using the metric given in Eq. 7 and given in Table 5.

- *Generate test scenarios from the requirements repository*

  In this step, the test scenarios are identified keeping in mind to achieve maximum requirement coverage. In the process, nine number of test scenarios are generated and we have achieved 100% requirement coverage. The test scenarios are given in Table 6. S and E, in Table 6, denote the start & end of a test scenario respectively. The extend of requirement coverage can be observed from Table 7. " $\surd$ " in Table 7, represents that corresponding test scenario covers the respective user requirement.
- *Prioritize and evaluate the test scenarios*

  In this step, the test scenarios identified in the previous step, are prioritized according to the decreasing value of the Priority Factor (PF(TS_i)) listed in Table 7. First, the weight of all the requirements covered by a test scenario are summed up to find the weight of a test

Table 4 Updated priority of each requirement

Requirement ID	End User Priority	Developer Priority	Final Priority	Total Priority of R_i	Release Weight(TR (R_i))	Updated Priority
R_1	3	2	6	$R_1(6)$	3	18
$R_{2.1}$	2	1	2	4	3	12
$R_{2.2}$	2	1	2		3	
R_3	2	1	2	$R_3(2)$	3	6
$R_{4.1}$	2	2	4	12	3	36
$R_{4.2}$	2	2	4		3	
$R_{4.3}$	2	2	4		3	
$R_{5.1}$	2	2	4	22	3	66
$R_{5.2}$	3	3	9		3	
$R_{5.3}$	3	3	9		3	
R_6	2	1	2	R_6 (2)	3	6

Table 5 Total weight of each requirement R_i

Req. ID	EP	Average EP(R_i) Priority	Coupling Weight (COUL)	Avg. (R_i) COUL	TV(R_i)	UP(R_i)	WR_i
R_1	3	3	Control (1)	1	0.06	18	3.24
$R_{2.1}$	1	1.5	Control (1)	1	0.18	12	3.24
$R_{2.2}$	2	1.5	Control (1)	1	0.18	12	
R_3	1	1	Control (1)	1	0.12	6	0.72
$R_{4.1}$	1	1.3	Control (1)	0.66	0.24	36	7.41
$R_{4.2}$	1	1.3	Data (0.5)	0.66	0.24	36	
$R_{4.3}$	2	1.3	Data (0.5)	0.66	0.24	36	
$R_{5.1}$	3	3	Control (1)	0.83	0.36	66	59.16
$R_{5.2}$	3	3	Control (1)	0.83	0.36	66	
$R_{5.3}$	3	3	Data (0.5)	0.83	0.36	66	
R_6	1	1	Control (1)	1	0	6	0

Table 6 Test scenarios

Test scenario ID	Test scenario sequence
TS_1	S $\longrightarrow$ $R_1[Unsuccessful]$ $\longrightarrow$ E
TS_2	S $\longrightarrow$ $R_1[Successful]$ $\longrightarrow$ R_2 $\longrightarrow$ E
TS_3	S $\longrightarrow$ $R_1[Successful]$ $\longrightarrow$ R_2 $\longrightarrow$ R_6 $\longrightarrow$ E
TS_4	S $\longrightarrow$ $R_1[Successful]$ $\longrightarrow$ R_2 $\longrightarrow$ $R_{4.1}$ $\longrightarrow$ R_6 $\longrightarrow$ E
TS_5	S $\longrightarrow$ $R_1[Successful]$ $\longrightarrow$ R_2 $\longrightarrow$ R_3 $\longrightarrow$ $R_{4.2}$ $\longrightarrow$ R_6 $\longrightarrow$ E
TS_6	S $\longrightarrow$ $R_1[Successful]$ $\longrightarrow$ R_2 $\longrightarrow$ R_3 $\longrightarrow$ $R_{4.3}$ $\longrightarrow$ R_6 $\longrightarrow$ E
TS_7	S $\longrightarrow$ $R_1[Successful]$ $\longrightarrow$ R_2 $\longrightarrow$ R_3 $\longrightarrow$ R_4 $\longrightarrow$ R_6 $\longrightarrow$ $R_5[Successful]$ $\longrightarrow$ E
TS_8	S $\longrightarrow$ $R_1[Successful]$ $\longrightarrow$ R_2 $\longrightarrow$ R_3 $\longrightarrow$ R_4 $\longrightarrow$ R_6 $\longrightarrow$ $R_5[Unsuccessful]$ $\longrightarrow$ E
TS_9	S $\longrightarrow$ $R_1[Successful]$ $\longrightarrow$ R_2 $\longrightarrow$ R_4 $\longrightarrow$ R_3 $\longrightarrow$ R_6 $\longrightarrow$ E

Table 7 Traceability matrix and final priority calculation of test scenarios

	R_1	R_2	R_3	$R_{4.1}$	$R_{4.2}$	$R_{4.3}$	R_5	R_6	$WR(TS_i)$	$NC(TS_i)$	$PF(TS_i)$
TS_1	√								3.24	0.09	0.29
TS_2	√	√							6.48	0.18	1.16
TS_3	√	√						√	6.48	0.27	1.74
TS_4	√	√		√				√	12.72	0.36	4.5
TS_5	√	√	√		√			√	11.52	0.45	5.1
TS_6	√	√	√			√		√	15.84	0.45	7.1
TS_7	√	√	√	√	√	√	√	√	73.77	0.72	53.11
TS_8	√	√	√	√	√	√	√	√	73.77	0.72	53.11
TS_9	√	√	√	√	√	√		√	14.61	0.63	9.20

scenario (WR(TS_i)). Then, the requirement coverage percentage of each test scenario i.e. $NC(TS_i)$ is calculated. The more the requirement coverage is, the more important the test scenario will be. So, the $NC(TS_i)$ is multiplied with WR(TS_i) to generate the PF(TS_i), which is given in Table 7.

The prioritized test scenarios are given by:

PTS = { TS_7, TS_8, TS_9, TS_6, TS_5, TS_4, TS_3, TS_2, TS_1 }

where WR(TS_i)is the sum of final weight of each sub-requirements of a requirement covered by test scenario TS_i, and NC(TS_i) is the percentage of requirement coverage of each test scenario TS_i. PF(TS_i) is the priority factor of each scenario. The formula to

calculate percenatge of requirement coverage, PF(TS_i), is given in Eq. 11.

$$PF(TS_i) = \frac{Number\ of\ requirement\ covered\ by\ the\ test\ scenario}{Total\ number\ of\ requirements} \quad (11)$$

Average Percentage of Fault Detection (APFD) metric is used to check the efficiency of the prioritized test suite Chauhan (2018); Mathur (2008); Yoo and Harman (2007). APFD metric is applied on different faults given in Table 8.

For the non-prioritized test scenarios, APFD is calculated as follows:

APFD = $1-\frac{1+2+6+7+3}{5\times9} + \frac{1}{2\times9}$ = 1 - 0.42 + 0.05 = **0.63**

For the prioritized test scenarios, APFD is calculated as follows:

APFD = 1 - $\frac{1+1+1+1+1}{5\times9} + \frac{1}{2\times9}$ = 1 - 0.11 + 0.05 = **0.94**

The APFD value for the given prioritized test scenarios and non-prioritized test scenarios are calculated to be 0.94 and 0.63 respectively. We have also considered another six case studies to evaluate efficiency of the proposed approach. In all the cases, it is found that, the proposed approach is efficient enough for early fault detection. The APFD values for the case studies are given in Table 9. The graphical representation of the APFD values of different case studies are shown in Fig. 2.

5 Implementation and results

The proposed approach is implemented using C++. There are two classes defined in the program for representing the *Requirements* and the corresponding *sub-requirements* along with their attributes. The Requirements and the corresponding Sub-requirements are kept in a linked list node. Two functions i.e. *weightofRequirement*()and $tc_p rioritize()$ are defined to calculate the weight of each requirement and prioritize the test scenario. The implementation result for Online Shopping Cart(OSC) case study is shown in Fig. 3.

Table 8 Test scenario-fault mapping table

	TS1	TS2	TS3	TS4	TS5	TS6	TS7	TS8	TS9
F1	√	√	√	√	√	√	√	√	√
F2		√	√	√	√	√	√	√	√
F3						√	√	√	√
F4							√	√	
F5			√	√	√	√	√	√	√

6 Comparison with related work

In this section, we compare our work with some of the existing related work. Looking at the importance of test case prioritization, researchers have proposed several approaches for test case/ test scenario prioritization. To minimize the cost of testing, requirement based test case prioritization has been given more priority. Some Mogyorodi (2002), Srikanth et al. (2013), Zhang et al. (2007), Krishnamoorthi and SahaayaArul (2009), Roongruangsuwan and Daengdej (2010), Kavitha et al. (2010) have identified different factors like requirement volatility, development complexity, risk exposure capability of requirements, time and cost factors etc. for test case prioritization from user's requirements. In addition to this, we have identified several critical factors like error proneness, release type of the requirement, coupling factor of the requirements which give an idea about the requirement dependencies, and the percentage of requirement covered by each test scenario. We have also decomposed the requirements into sub-requirements and proposed metrics for calculation of requirement-weight at sub-requirement level. We generate system test plan at requirements specification level. The proposed approach is applicable to system level testing. Specification-based testing has the advantage of generating and prioritizing test scenarios early in the software development life cycle. This gives the test engineers enough time for proper test planning and validating the test scenarios early. The APFD value of the prioritized test scenarios for the case studies depicts the efficiency of our approach in terms of early fault detection.

Srikanth et al. (2013) have designed PORT (Prioritization Of Requirements for Testing) which was a value driven and system level prioritization technique. PORT considers four factors such as customer assigned priority, requirement implementation complexity, requirement volatility and fault proneness of requirements. These four factors help in computing the prioritization factor value of each requirement and finally test case prioritization is done by finding the weighted priority of each associated test case.

Mogyorodi (2002) over viewed requirement based testing(RBT) techniques which design minimum number of test cases from requirements. RBT helps to conduct testing in parallel with development. The distribution of bug report indicates that 56% of all bugs are rooted in requirement phase while design and coding phase yield 27% and 7% respectively Mogyorodi (2002).

Table 9 APFD values for different case studies

Sl No	Name of Case study	Total No. of Reqs.	Total No. of Test Scenarios	Total No. of Faults	APFD Value (prioritized)	APFD Value (non-prioritized)	Percentage Increase in APFD value
1	Online Shopping Cart (OSC)	6	9	5	0.94	0.63	49.20
2	E-Banking System-ATM (EBS)	10	13	6	0.64	0.34	88.23
3	Online Ticket Vending Machine (OTVM)	14	14	7	0.90	0.66	36.36
4	E-Hospital Management System (EHMS)	11	15	7	0.86	0.47	82.97
5	Crime Management System (CMS)	15	14	7	0.92	0.59	55.93
6	Online Project Manager (OPM)	13	13	7	0.91	0.51	78.43
7	Digital Imaging & Communications in Medicine (DICOM)	12	12	5	0.70	0.54	29.62
Average Percentage Increase in APFD Value = 60.10							

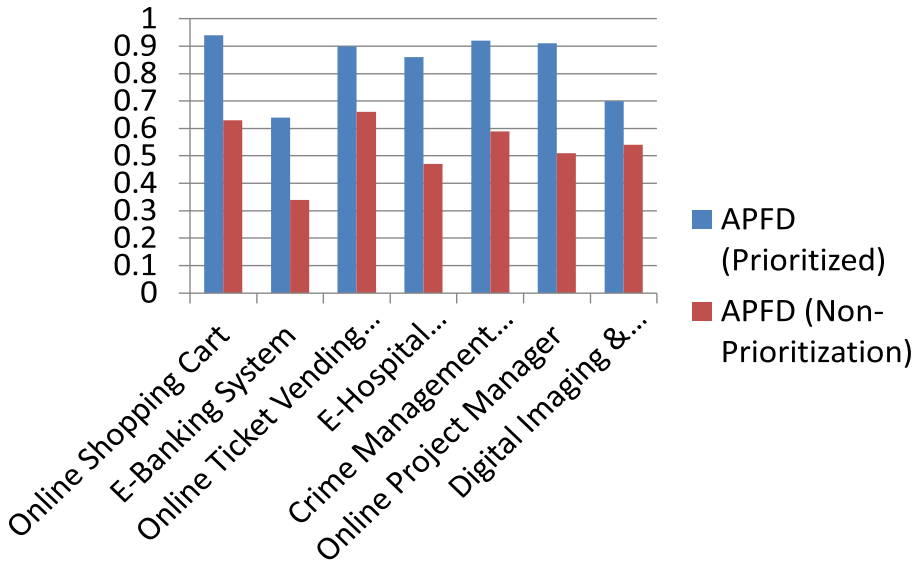

Fig. 2 APFD Values of Prioritized and Non-prioritized Test Scenarios for different case studies

Krishnamoorthi et al. (2009) proposed two more factors, i.e. completeness and traceability, as regression test case factors. A system level test case prioritization technique like PORT Srikanth et al. (2013) was proposed.

Table 10 summarizes the comparison of different approaches with our proposed approach. The different approaches compared in Table 10 have considered various factors like error proneness, requirement volatility, traceability, customer assigned priority etc., but they have not considered some important factors like type of release, dependencies factor in terms of coupling, and percentage of requirement coverage. We have considered these factors to increase the accuracy in average percentage of fault detection.

Kavitha et al. (2010) proposed an approach to prioritize test cases using three different factors i.e. customer priority, changes in requirement, and implementation complexity.

Marchetto et al. (2016) proposed a test case prioritization technique, which maximizes the fault discovery that are both technical and business critical. The proposed approach is multi-objective in nature i.e. it focuses on early discovery of faults as well as reducing the execution cost of test cases. The author applies metrics and algorithms to

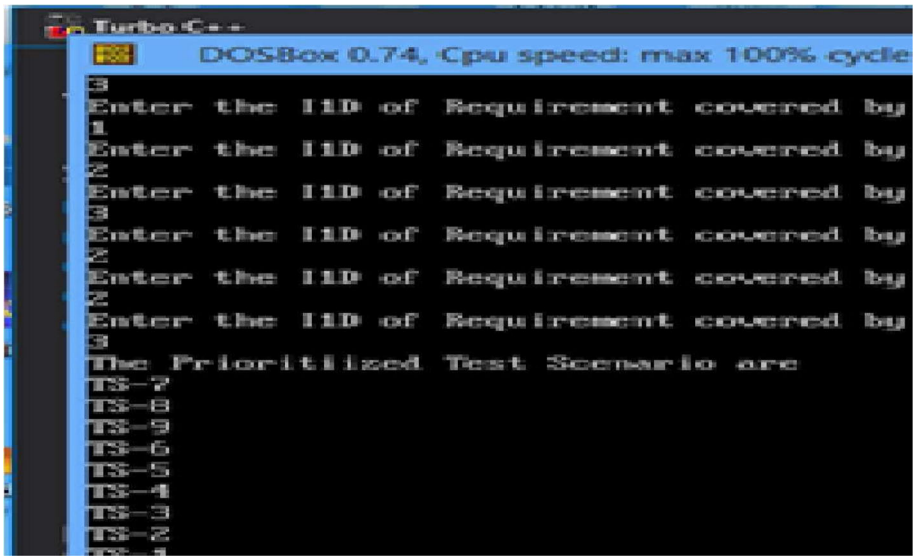

Fig. 3 Implementation Result of our approach Online Shopping Cart (OSC) Casestudy

identify critical and fault-prone portion of the software which is assigned with more priority value. The major component of the proposed approach is requirements, code, and test cases. The requirements deals with requirement traceability link where as the code and test cases facilitate the execution of test cases. The metrics used automatically assigns weights to the code as well as the requirements.

Nayak et al. (2017) proposed a test case prioritization technique to enhance the efficiency of the test case prioritization by improving the rate of fault detection. They considered four factors- rate of fault detection, number of fault detected, the ability of the test case for risk detection, and test case effectiveness, for test case prioritization. The author considered five levels of severity values for risk detection. All the values of the four factors are added to find out the weight of each test case and based on the value assigned, the test cases are prioritized.

Singh et al. (2018) proposed object-oriented coupling based test case prioritization. The author considered class diagrams present in object-oriented systems. They first investigated the coupling oriented object-oriented metrics for each class. Then, the coupling weight was assigned to each class based on some threshold. They considered the dependency graphs to estimate the coupling measure. Then, the aggregated coupling weight is determined from classes covered by a test case. Finally, the test cases are ranked based on the aggregate weight.

The proposed approach is compared with the existing related work and it is observed that factors like type of release, coupling between the requirement, and percentage of requirements, coverage are not considered in any of the existing related work. In all these works, APFD metric can be considered to show the effectiveness of their approaches. The APFD metric is only used to calculate the average percentage of fault detection by the prioritized test scenarios and non-prioritized test scenarios for different case studies. We have also computed the APFD values for the case studies using our proposed technique and some existing techniques. Our average increase in APFD value for the prioritized test scenario is 60.10. From the experiments, we have observed that, the average increase in APFD value for our approach is better than that of the existing approaches. This can be inferred from Tables 9 and 11. The percentage increase in APFD value for our approach is given in last column of Table 9. So, it is observed that there is an increase in APFD value for prioritized test scenarios over non-prioritized test scenarios, in all the case studies considered. At the same time the APFD values for prioritized and non-prioritized test scenarios are also compared for the case studies by applying the approached proposed by other related papers i.e. listed in Table 10. The different APFD values and the corespondding percentage increase in APFD values for different approaches are given in Table 11. The proposed approach is capable of identifying and prioritizing the critical test cases present in the test suite, much early in the software development life cycle, which helps in regression testing.

The title alphabets in Table 11 are - P : Prioritized N: Non-Prioritized A: Percenatge Incresase in APFD Value.

The standard deviation of the APFD values for the prioritized test scenarios is calculated using the formula given in Eq. 12. and the values are summarized in Table 12. It can be inferred from Table 12 that, the variance is very low for all the case studies. Also, the APFD values are very close to the mean and the deviation in all the case studies are minimal. This proves the effectiveness of the proposed work.

$$\sigma = \sqrt{\frac{\sum((x_i - \mu)^2)}{N}} \tag{12}$$

Table 10 Comparison with related work

Sl. no.	Author name	Year	Prioritization technique used	Key factors considered
1	Mogyaoradi (2002)	2002	Author proposed Requirement Based Testing (RBT)	Conducting testing in parallel to minimize test scenarios
2	Krishnamoorthi et al. (2009)	2009	A new system level test case Prioritization Technique	Completeness Traceability
3	Kavitha et al. (2010)	2010	A new system level test case Prioritization (TCP) Technique to improve the rate of fault severity detection	Customer priority Changes in requirement Implementation complexity
4	Srikanth et al. (2013)	2013	Author designed Prioritization of Requirements (PORT)	Customer Assigned Priority Error Proneness Requirement volatility Implementation complexity
5	Marchetto et al. (2016)	2016	A multi objective technique to prioritize the test case	Critical and fault-proneness portion of software
6	Hettiarachchi et al. (2016)	2016	Risk-based test case prioritization using a fuzzy expert system	Requirement Complexity Requirement Size Requirement Modification Status Potential Security Threats Triangular Membership Function for Fuzzy expert system
7	Nayak et al. (2017)	2017	Enhancing Efficiency of the Test Case Prioritization prioritize the Technique by Improving the Rate of Fault Detection	Rate of fault detection Number of faults detected Ability of test case to detect risk Test case effectiveness
8	Singh et al. (2018)	2018	Object Oriented Coupling based Test Case Prioritization	Coupling between classes
9	Hettiarachchi et al. (2019)	2019	A systematic requirements risk-based test case prioritization using a fuzzy expert system	Requirement Complexity Requirement Modification Status Potential Security Threats Trapizoidal wave type membership function for Fuzzy expert system Semi-automated systems
10	Our Proposed Approach (RTSP)	2019	A new test scenario Prioritization Technique (RTSP) to prioritize multiple test suit by assigning requirement weight	Type of release Coupling factor Percentage of requirement coverage

σ = population of standard deviation
N = size of the population
x_i = each volume from the population
μ = the population mean

7 Threats to validity

The proposed approach has the following threats:

- Here the non-functional aspects of the system are not taken into account while evaluating different factors.
- It is very difficult to identify the functional dependencies from the requirement specification document only. A model based approach will be more appropriate to identify the functional dependencies.
- Now, we have semi-automated our proposed approach with ordinary GUI, using C++. Still, some of the activities/ steps are carried out manually. So, the time and effort required for executing our approach with large and complex systems may increase rapidly. The time and effort, may be reduced, by fully automating all the activities/ steps with better GUI, and keeping track of errors detected in different versions of the requirements in a database, so that all the calculations as per the mentioned equations in the proposed approach, can be done automatically.

Table 11 APFD values of existing approaches of different case studies

Case study/ Author name	Srikanth et al.			Krishnamoorthi et al.			Kavitha et al.			Magyaoradi et al.			Our approach RTSP		
APFD Value	P	N	A	P	N	A	P	N	A	P	N	A	P	N	A
Online Shopping Cart	0.88	0.63	39.68	0.83	0.63	31.74	0.91	0.63	44.44	0.92	0.63	46.03	0.94	0.63	49.20
E-Banking System ATM	0.59	0.34	73.52	0.61	0.34	79.41	0.61	0.34	79.41	0.61	0.34	79.41	0.64	0.34	88.23
Online Ticket Vending Machine	0.87	0.66	31.81	0.92	0.66	39.39	0.85	0.66	28.78	0.83	0.66	25.75	0.90	0.66	36.36
E-Hospital Management System	0.81	0.47	72.34	0.79	0.47	68.08	0.82	0.47	74.46	0.84	0.47	78.72	0.86	0.47	82.97
Crime Management System	0.95	0.59	61.01	0.90	0.59	52.54	0.89	0.59	50.84	0.94	0.59	59.32	0.92	0.59	55.93
Online Project Manager	0.90	0.51	76.47	0.87	0.51	70.58	0.92	0.51	80.39	0.92	0.51	80.39	0.91	0.51	78.43
Digital Imaging and Communication in Medicine	0.68	0.54	25.92	0.73	0.54	35.18	0.71	0.54	31.48	0.71	0.54	31.48	0.70	0.54	29.62
Average Percentage Increase APFD	54.39			53.84			55.68			57.3			60.10		

Table 12 Standard deviation of APFD values for different case studies

Case Study	Standard Deviation of APFD Values (Prioritized test scenarios)
Online Shopping Cart	0.038
E-Banking System ATM	0.016
Online Ticket Vending Machine	0.032
E-Hospital Management System	0.024
Crime Management System	0.022
Online Project Manager	0.018
Digital Imaging and Communication in Medicine	0.016

8 Conclusion and future work

In this section, we summarize the important contributions of our work as well as the possible extensions. First, the major functional requirements as well as the sub-functional requirements are identified for the system under test. Then, the priority is calculated for each functionality using the proposed metrics. Different factors like requirement volatility, priority set by the end user, priority set by the developer, and coupling factor are considered to finalize the priority of each functionality. Test scenarios are generated using the coverage based approach to achieve maximum path coverage. Then, test case prioritization is carried out for the identified critical requirements by considering the factors like requirement coverage by each test case. The proposed approach has been evaluated using the APFD metric.

We outline the following possible extensions to our work.

- Our approach may be extended to include the non-functional aspects of the system under test.
- A model representation of the system will give a better understanding of the dependencies among the functional requirements. This may lead to better prioritization factors.
- The proposed approach is mainly applicable for Application Software. The proposed approach is implemented in C++ environment with an ordinary Graphical User Interface (GUI). All input and output for the implementation are through command prompt only. So, development of a better GUI, which may be obtained by using the advanced features of Java such as Applets, Swings etc., will be more appealing and can increase the ease of using the proposed approach.
- Maintaining a historical data store of faults associated with user requirements during the implementation will be an add-on to early fault detection. Usually Software goes through different modifications in different versions. These modifications give rise to many new faults in the systems by affecting some modules in the software. These modifications and their corresponding affected modules can be stored in a repository. Data mining algorithms like association rule mining can be

applied to the repository to generate frequent pattern of affected modules in the modified software. This will help in prioritizing the test scenarios

References

Chauhan N (2018) Software Testing Principles: Practices, 4th, edition. Oxford University Press, New Delhi

Rothermel G, Untch R. H, C. C, Harrold M.J (2001) Prioritizing test cases for regression testing. IEEE Trans Softw Eng

Srikanth H, Banerjee SWLOJ (2013) Towards the priritization of system test cases. J Softw Test Verif Reliab, pp 320–337

Hafeez SAY (2019) Enabling test case prioritization for component based software development. In: International conference on frontiers of information technology (FIT), pp 105–109

Hettiarachchi C, Do H (2019) A systematic requirements and risks-based test case prioritization using a fuzzy expert system. In: 2019 IEEE 19th international conference on software quality, reliability and security (QRS), pp 374–385

Hettiarachchia C, Do H, Choi B (2016) Risk-based testcas eprioritization using a fuzzy expert system. J Inf Softw Technol 69:1–15

IanSommerville PS (2010) Requirements engineering: a good practice guide, student. Wiley, London

Jayant K, Rana A (2011) Prioritization based test case generation in regression testing. Int J Adv Eng Res (IJAER) 1

Krishnamoorthi R, Mary SASA (2009) Requirement based system test case prioritization of new and regression test cases. Int J Softw Eng Knowl Eng 19(3):453

Kumar V, Sujata Kumar M (2010) Test case prioritization using fault severity. Int J Comput Sci Technol 1:67–71

Mahali P, Mohapatra DP (2018) Model based test case prioritization using uml behavioural diagrams and association rule mining. Int J Syst Assur Eng Manag 9:1063–1079

Mall R (2009) Fundamental of software engineering. PHI Learning Private Limited, 3rd edition

Marchetto A, Islam M, Asghar W, Susi A, Scanniello G (2016) A multi-objective technique to prioritize test cases. IEEE Trans Softw Eng 42(10):918–940. https://doi.org/10.1109/TSE.2015.2510633

Mathur AP (2008) Foundations of software testing, 1st edn. Addison-Wesley Professional, New York

Arafeen Md. JDH (2013) Test case prioritization using requirement-based clustering. In: IEEE sixth international conference on software testing, verification and validation, pp 312–321

Mogyorodi G (2002) Requirement-based testing: an overview. In: Proceedings 39th international conference and exhibition on technology of object-oriented languages and systems. TOOLS 39, IEEE, pp 286–295

Muhammad A, Inayat Irum SMJN (2019) Requirement dependencies-based test case prioritization for extra-functional properties. In: IEEE international conference on software testing, verification and validation workshops (ICSTW), pp 159–163

Mukherjee R, Patnaik K.S (2018) A survey on different approaches for software test case prioritization. Journal of King Saud University-Computer and Infromation Sciences, ScienceDirect

Nayak S, Kumar C, Tripathi S (2017) Enhancing efficiency of the test case prioritization technique by improving the rate of fault detection. Arab J Sci Eng 42:3307–3323

Ouriques JFS, Cartaxo EG, Machado PDL (2018) Test case prioritization techniques for model-based testing: a replicated study. Softw Qual J26:1451–1482

Panthi V (2017) Generation and prioritization of test scenarios for object-oriented software using uml state machines. Ph.D. thesis, National Institute of Technology Rourkela India

Panthi V, Gardizy A, Mohapatra RK, Mohapatra DP (2018) Functionality testing of object-oriented software using uml state machine diagram. In: 2018 IEEE international conference on circuits and systems in digital enterprise technology, pp 1–7

Luo Qi, Moran Kevin, L Z, D P (2019) How do static and dynamic tes case prioritization techniques perform on modern software systems? an extensive study on github projects. IEEE Trans Softw Eng 45:1054–1980

Kavitha R, Kavitha VRKN (2010) Requirement based test case prioritization. In: International conference on communication control and computing technologies (ICCCCT 2010), pp 826–829

Kavitha RVRK, Kumar NS (2010) Requirement based test case prioritization. In: International conference on communication, control and computing technologies, pp 826–829

Krishnamoorthi R, Sahaaya Arul SM (2009) Test case prioritization using requirement-basedclustering. Inf Softw Technol 51(799): 808

Rava M, Wan-Kadir WM (2016) A review on prioritization techniques in regression testing. Int J Softw Eng Appl 10(1):221–232

Pressman Roger SBRM (2019) Software engineering: a practitioner's approach, 9th edn. McGraw-Hill, New York

Roongruangsuwan S, Daengdej J (2010) Test case prioritization techniques. J Theor Appl Inf Technol 18:45–60

Rothermal G, Untch RH, Chu C, HarRold MJ (2001) Prioritizing test cases for regression testing. IEEE Trans Softw Eng 27(10):929–948

Samuel P, Joseph A.T (2008) Test sequence generation from UML sequence diagrams. In: Proceedings of 9th international conference on software engineering, artificial intelligence, networking, and parallel/distributed computing, vol 2008, pp 879–887

Singh A, Bhatia RK, Singhrova A (2018) Object oriented coupling based test case prioritization. Int J Comput Sci Eng 6:747–754

Srivastava PR, Kumar K, Raghurama G (2008) Test case prioritization based on requirements and risk factors. ACMSIGSOFT Softw Eng Notes 33(4):1–5

Swain RK, Panthi V, Behera PK, Mohapatra DP (2014) Slicing-based test case generation using uml 2.0 sequence diagram. Int J Comput Intell Stud, p 3

Swain R.K, Panthi V, Mohapatra D.P, Behera P.K (2014) Prioritizing test scenarios from uml communication and activity diagrams. Innov Syst Softw Eng, ACM, p 10

Wang X, Zeng H (2016) History-based dynamic test case prioritization for requirement properties in regression testing. In: Proceedings of international workshop on continuous software evolution and delivery, Austin, TX, USA, pp 41–47

Zhang X, Nie C, X B, Q B (2007)Test case prioritization based on varying testing requirement priorities and test case cost. Int Conf Qual Softw, pp 15–24

Yoo S, Harman M (2007) Regression testing. A survey, software testing, verification and reliability, minimisation, selection and prioritisation, pp 1–60

Yoon M (2013) A test case prioritization through correlation of requirement and risk. J Soft Eng 05:823–836