TEST CASE PRIORITIZATION USING FUZZY LOGIC BASED ON REQUIREMENT PRIORITIZING

Usha Badhera¹ and Debarupa Biswas²

Computer Science Department, Banasthali University, India
¹ushas133@yahoo.com, ²biswas.debarupa@gmail.com

ABSTRACT

Boolean expressions are popularly used for modelling decisions or conditions in specifications or source programs and they are very much prone to introduction of faults. Even for a Boolean expression with few numbers of literals the possible number of test cases can be quite large. Boolean expressions with n variables require \(2^n\) test cases to distinguish from faulty expression. In practice, n can be quite large and there are examples of specification having Boolean expressions with 30 or more variables. To test a system based on Boolean specification in limited time, it is not possible to execute all test cases so prioritization is required which leads to early fault detection in testing life cycle. There are various testing strategies for generation of test cases for Boolean specifications like MUMCUT, which generate fewer test cases then \(2^n\) with high probabilities of finding errors but their prioritization are not considering the criteria from user’s perspective. We have proposed a new approach which prioritizes test cases based on requirement prioritization. Our aim is to find the severe faults from user’s perspective early in the testing process and hence to improve the quality of the software. This paper considers method for assigning weight value on the basis of factors which generates the criteria for test case prioritization for Boolean Specifications. These factors are: Business Value Measure (BVM), Project Change Volatility (PCV), and Development Complexity (DC). Priority is assigned to test cases based upon these factors using fuzzy logic model.

KEYWORDS

Regression testing, Test case prioritization, Fuzzy-Based Approach, Requirement prioritization

1. INTRODUCTION

Requirements prioritization has become a relevant research area in requirements engineering thus requiring more effective methods and techniques that enable to rank a whole set of requirements based on factors such as business goals or development cost and many more. Requirement prioritization plays a vital role in the requirement engineering process, identifying critical requirements, can affect software release planning and requirement negotiation. Prioritized requirements can be seen as the process of deriving an order relation on a given set of test cases derived from those requirements [10].

1.1. Prioritization

Rothermel at el. [3, 4, 5, 6] defines the test case prioritization as:

Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.
Problem: Find T’ \(\in\) PT such that (for all T") (T’” \(\in\) PT) (T’”\(\neq\) T") [f (T’) \(\geq\) f (T’”)]
Here, PT is the set of all possible ordering for prioritizations of T, and f is a function which when applied to any such ordering, yields an award value for that ordering. The definition assumes that higher award values are preferred over the lower ones. Requirement prioritization can have many goals, in our study it is done to prioritize test cases. The objective of this study is to develop a test case prioritization technique that prioritizes test cases for regression testing on the basis of prioritized Boolean specifications. For this purpose requirements are prioritized according to various factors which are important from business and implementation perspective.

2. PROPOSED WORK

This paper proposes a fuzzy logic based tool to prioritize requirements that will be considered for prioritizing test case of Boolean specification. In this analysis using fuzzy logic we identified important factors which lead to gathering of important requirements. From the literature survey, we found list of factors that leads to the gathering of important requirements which are not precisely specified. We constructed rule base using different rules as per expert’s guidance. MATLAB has been used to develop a graphical user interface tool according to the rule base. The tool considers factors from the requirement gathering phase as input and then generates output which shows the impact on the requirements.

2.1 Important Factors influencing the requirement prioritization

Following factors can be considered to assign the weighted prioritization (WP) to the requirements. Based on these following factors which are applied as inputs, weights are obtained as output:

- Business Value Measure (BVM)

Requirements are rated according to its importance from customer’s business point of view. A value to each requirement is assigned by the customer and this value can range from 1 to 100 where value 100 and value 1 represents the most critical and the least critical requirements respectively. If time is limited and the project is close to deadline then the test cases need to be prioritized according to the values assigned to these requirements. Requirements with lower priority should be tested later. This is helpful in increasing the customer confidence level as it increases the business value.

- Project Change Volatility (PCV)

PCV denotes the number of times project requirement undergoes changes in software development life cycle (SDLC). Thus testing effort increases as PCV increases, as it leads to delay in project completion which in turn may result in project which has high defect density. The requirements which underwent most frequent changes are prioritized higher than the one with least frequency of changes.

- Development Complexity (DC)

The requirements are analysed based on how complex is its implementation in terms of development efforts, technology, environmental constraints and requirement feasibility matrix. Highly complex requirements are prioritized higher [8].
2.2 Metric used for prioritization

All the prioritization factors get a fraction of the total assigned weight 1.0. These weights are calculated using fuzzy rules described below in section 2.3. Equation (1) is used to calculate a weighted prioritization (WP) factor for every requirement that measures the importance of testing a requirement earlier.

\[
WP = \sum_{j=1}^{n}(PF \text{ value } \times PF \text{ Weight})
\]  

WP represents the weighted prioritization for the requirements computed using Equation (1). The results will show the prioritization of test cases for the requirements. As factor-weights and factor-values changes, the weighted prioritization varies accordingly [8].

2.3 Design Methodology

The impact levels are categorized into three levels as the impact of the factors may vary according to different organization environments. The impact can be “Low”, “Medium”, and “High”. Inputs are represented by fuzzy sets to design fuzzy inference system. Membership functions are used to represent the fuzzy sets.

Some of the rules are given below:

1. If (BusinessValueMeasure (BVM) is Low) and (ProjectChangeVolatile (PCV) is Low) and (Development Complexity (DC) is Low) Then (Weight is Low) (1)
2. If (BusinessValueMeasure (BVM) is Low) and (ProjectChangeVolatile (PCV) is Low) and (Development Complexity (DC) is Medium) Then (Weight is Low) (1)
3. If (BusinessValueMeasure (BVM) is Low) and (ProjectChangeVolatile (PCV) is Low) and (Development Complexity (DC) is High) Then (Weight is High) (1)
4. If (BusinessValueMeasure (BVM) is Low) and (ProjectChangeVolatile (PCV) is Medium) and (Development Complexity (DC) is Low) Then (Weight is Low) (1)
5. If (BusinessValueMeasure (BVM) is Low) and (ProjectChangeVolatile (PCV) is Medium) and (Development Complexity (DC) is Medium) Then (Weight is Medium) (1)
6. If (BusinessValueMeasure (BVM) is Low) and (ProjectChangeVolatile (PCV) is Medium) and (Development Complexity (DC) is High) Then (Weight is High) (1)
7. If (BusinessValueMeasure (BVM) is Low) and (ProjectChangeVolatile (PCV) is High) and (Development Complexity (DC) is Low) Then (Weight is Low) (1)
8. If (BusinessValueMeasure (BVM) is Low) and (ProjectChangeVolatile (PCV) is High) and (Development Complexity (DC) is Medium) Then (Weight is Medium) (1)
9. If (BusinessValueMeasure (BVM) is Low) and (ProjectChangeVolatile (PCV) is High) and (Development Complexity (DC) is High) Then (Weight is High) (1)
10. If (BusinessValueMeasure (BVM) is Medium) and (ProjectChangeVolatile (PCV) is Low) and (Development Complexity (DC) is Low) Then (Weight is Low) (1)
11. If (BusinessValueMeasure (BVM) is Medium) and (ProjectChangeVolatile (PCV) is Low) and (Development Complexity (DC) is Medium) Then (Weight is Medium) (1)
12. If (BusinessValueMeasure (BVM) is Medium) and (ProjectChangeVolatile (PCV) is Low) and (Development Complexity (DC) is High) Then (Weight is High) (1)
13. If (BusinessValueMeasure (BVM) is Medium) and (ProjectChangeVolatile (PCV) is Medium) and (Development Complexity (DC) is Low) Then (Weight is Low) (1)
14. If (BusinessValueMeasure (BVM) is Medium) and (ProjectChangeVolatile (PCV) is Medium) and (Development Complexity (DC) is Medium) Then (Weight is Medium) (1)
15. If (BusinessValueMeasure (BVM) is Medium) and (ProjectChangeVolatile (PCV) is Medium) and (Development Complexity (DC) is High) Then (Weight is High) (1)
16. If (BusinessValueMeasure (BVM) is Medium) and (ProjectChangeVolatile (PCV) is High) and (Development Complexity (DC) is Low) Then (Weight is Low) (1)
17. If (BusinessValueMeasure (BVM) is Medium) and (ProjectChangeVolatile (PCV) is High) and (Development Complexity (DC) is Medium) Then (Weight is Medium) (1)
18. If (BusinessValueMeasure (BVM) is Medium) and (ProjectChangeVolatile (PCV) is High) and (Development Complexity (DC) is High) Then (Weight is High) (1)
19. If (BusinessValueMeasure (BVM) is High) and (ProjectChangeVolatile (PCV) is Low) and (Development Complexity (DC) is Low) Then (Weight is Low) (1)
20. If (BusinessValueMeasure (BVM) is High) and (ProjectChangeVolatile (PCV) is Low) and (Development Complexity (DC) is Medium) Then (Weight is Medium) (1)
21. If (BusinessValueMeasure (BVM) is High) and (ProjectChangeVolatile (PCV) is Low) and (Development Complexity (DC) is High) Then (Weight is High) (1)
22. If (BusinessValueMeasure (BVM) is High) and (ProjectChangeVolatile (PCV) is Medium) and (Development Complexity (DC) is Low) Then (Weight is Low) (1)
23. If (BusinessValueMeasure (BVM) is High) and (ProjectChangeVolatile (PCV) is Medium) and (Development Complexity (DC) is Medium) Then (Weight is Medium) (1)
24. If (BusinessValueMeasure (BVM) is High) and (ProjectChangeVolatile (PCV) is Medium) and (Development Complexity (DC) is High) Then (Weight is High) (1)
25. If (BusinessValueMeasure (BVM) is High) and (ProjectChangeVolatile (PCV) is High) and (Development Complexity (DC) is Low) Then (Weight is Low) (1)
26. If (BusinessValueMeasure (BVM) is High) and (ProjectChangeVolatile (PCV) is High) and (Development Complexity (DC) is Medium) Then (Weight is Medium) (1)
27. If (BusinessValueMeasure (BVM) is High) and (ProjectChangeVolatile (PCV) is High) and (Development Complexity (DC) is High) Then (Weight is High) (1)

2.4 Implementation Procedure

Fuzzy inference system as given in figure shows the mapping of inputs and outputs. A case study is used to explain the implementation method:

2.4.1 Case study

Let us consider the Boolean specification represented in DNF form “ab+cd+ef” which has three requirements ab, cd and ef. The objective is to prioritize these requirements which can be further used to prioritize test suite.

2.4.2 Input

Table-1: Input

<table>
<thead>
<tr>
<th>Factors</th>
<th>ab</th>
<th>cd</th>
<th>ef</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Value Measure(BVM)</td>
<td>40</td>
<td>80</td>
<td>60</td>
</tr>
<tr>
<td>Project Change Volatility(PCV)</td>
<td>70</td>
<td>60</td>
<td>90</td>
</tr>
<tr>
<td>Development Complexity(DC)</td>
<td>80</td>
<td>70</td>
<td>90</td>
</tr>
</tbody>
</table>
2.4.3 Fuzzy Inference System

Fuzzy inference system as given in figure shows the mapping of inputs and outputs. Here the inputs are Business Value Measure, Project Change Volatility, Development Complexity and the output is Weights.

![Figure 1. Fuzzy Inference System](image)

2.4.4 Membership Function Editor

Membership function editor as given in figure below shows input and output variables along with their range. The ranges for input and output can vary.

![Figure 2. Membership Function of an Input](image)

![Figure 3. Membership Function of Output](image)

2.4.5 Rule Editor

Rule editor helps to add, change and delete rules. In this rule base as shown in figure below, there are 27 rules.
2.4.6 Defuzzification

The defuzzified value is obtained from the rule viewer. This value is the required weight for a particular factor provided as input.

2.4.7 Output

In the figure below, the weights are obtained from the fuzzy rules and WP is calculated from Eq(1) which in turn gives us the prioritization order. Requirements are prioritized according to the decreasing WP values. If for more than one requirement, we have the same WP value then we can prioritize them in any order.

Table 2. Output

<table>
<thead>
<tr>
<th>Factors</th>
<th>ab</th>
<th>cd</th>
<th>ef</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusinessValue Measure(BVM)</td>
<td>40</td>
<td>80</td>
<td>60</td>
<td>0.400</td>
</tr>
<tr>
<td>ProjectChange Volatility(PCV)</td>
<td>70</td>
<td>60</td>
<td>90</td>
<td>0.627</td>
</tr>
<tr>
<td>Development Complexity(DC)</td>
<td>80</td>
<td>70</td>
<td>90</td>
<td>0.749</td>
</tr>
<tr>
<td>WP</td>
<td>119.81</td>
<td>122.05</td>
<td>147.84</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Here the weights are calculated for each factor by applying the values for each requirement for a particular factor as input to the fuzzy rules. For example:

- Weight for BVM: is obtained after applying [40 80 60] as input.
- Weight for PCV: input is [70 60 90]
- Weight for DC: input is [80 70 90]
The WP factor is calculated below based on the Eq (1) (refer section 2.2)

- WP for ab: \((40 \times 0.4) + (70 \times 0.627) + (80 \times 0.749) = 119.81\)
- WP for cd: \((80 \times 0.4) + (60 \times 0.627) + (70 \times 0.749) = 122.05\)
- WP for ef: \((60 \times 0.4) + (90 \times 0.627) + (90 \times 0.749) = 147.84\)

Thus the requirements “ab”, “cd”, and “ef” should be prioritized in the following order based on the descending values of WP.

Prioritized order is:
- ef
- cd
- ab

So the test cases of “ef” should be executed first followed by “cd” and “ab”

3. CONCLUSION

This paper applies fuzzy logic to prioritize the test cases based on prioritization of the requirements of Boolean specification which will lead to early fault detection based on the criticality of the requirements according to Business Value Measure (BVM), Project Change Volatility (PCV) and Development Complexity (DC). The effect of the rules has been observed by simulating the model using fuzzy logic toolbox of MATLAB. In future, more factors will be considered which in turn will help in getting more precise requirements from customers who can further enhance the prioritization of testing.

REFERENCES

Authors

Usha Badhera is an active researcher in the field of software testing, currently working as Assistant Professor in Department of Computer Science at Banasthali University (Rajasthan), India. She has done MCA from Rajasthan University and her PhD is in progress from Banasthali University (Rajasthan), India.

Debarupa Biswas is a student of M.Tech at Banasthali University (Rajasthan). She received her MCS from Assam University, silchar(a central university) in 2010. Her area of interest is Genetic Algorithms.