Application of a Fuzzy Inference System to Measure Maintainability of Object-Oriented Software

Nasib Singh Gill and Meenakshi Sharma
Department of Computer Science & Applications
Maharshi Dayanand University, Rohtak-124001
Haryana, India

Abstract - In the software development life cycle, maintenance is the most costly activity because it requires more effort compared to other activities. To reduce the cost of software maintenance, it is essential to predict software maintainability during the early phases of software development. As a consequence of early estimation, further corrective and preventive actions can be performed more efficiently to improve the maintainability of the software. Predicting maintainability of software using fuzzy logic is gaining more attention among researchers due to its ability to deal with uncertain, imprecise and incomplete data. This paper develops a fuzzy logic-based model for predicting the maintainability of a class. The given model is based on the Mamdani's fuzzy inference system. Chidamber and Kemerer metrics are used as inputs to the model and the maintainability is computed as output. Maintainability can be used as an indicator of the quality of software at design time.

Keywords: Maintainability, Metric, Fuzzy Inference Engine

1. Introduction

“Software maintenance in software engineering is the modification of a software product after delivery to correct and to improve performance or other attributes” [1]. Software maintenance is the most important activity in the life of a software product. The total cost of the maintenance phase is much higher than the development cost of the software. It can consume 40% to 90% of the cost of the entire life cycle [2,3]. There are four types of software maintenance: Corrective, Adaptive, Preventive and Perfective. Software maintainability is the ease with which a software system can be understood, modified and adapted [4]. It is an important software quality attribute. Software maintenance is a post-development activity, but it is highly influenced by the way the software was developed. The object-oriented software development approach has the ability to produce highly maintainable software. The maintainability of a software system can significantly impact software costs. For effective management of software cost, software maintainability must be evaluated. Many metrics have been proposed by various researchers to evaluate the maintainability of software. Among these metrics, the Chidamber and Kemerer (CK) metrics suite [5] is the most widely used metric suite for assessing the maintainability of object-oriented software. Software maintainability is a difficult factor to quantify [6]. In object-oriented software, the maintainability of a class can be assessed in terms of its complexity using design-oriented metrics. However predicting the maintainability of a class using crisp logic is not appropriate in the presence of fuzzy input factors. Therefore, the fuzzy logic-based approach of measuring maintainability is being used by many researchers.

Fuzzy logic is a generalization of classical boolean logic. It provides a mechanism for representing linguistic constructs such as “low,” “medium,” “high,” “very high” etc. The conventional binary set theory describes crisp data, which is either fully true or false. Fuzzy logic operates on the concept of degree of membership. The degree of membership is continuous on an interval [0,1]. A fuzzy set captures vagueness using membership functions by assigning a degree of membership to each element of the set. The fuzzy logic also uses a fuzzy inference engine to manipulate imprecise, uncertain and conflicting data. The fuzzy inference engine maps the input fuzzy set to an output fuzzy set using a fuzzy rule base.

This paper evaluates the maintainability of object-oriented software using Mamdani’s Fuzzy Inference System (MFIS). Six CK metrics are used as inputs to the model. The value of each metric is fuzzified into one of three values: Low, Medium and High. The trapezoidal membership function is used for fuzzification. A total of 729 fuzzy rules were developed, which are used by the fuzzy inference engine. The maintainability computed by MFIS is correlated with the composite complexity, which is computed as a weighted sum of the CK metrics. The weight that will be multiplied to each CK metric is derived from the corresponding importance factor given by Jubair et al. [7].

The rest of the paper is divided into three sections. Section 2 describes the MFIS along with inputs, outputs, membership functions and rules. Section 3 applies the given model to 15 classes and correlates the composite complexity of each class with its maintainability. Finally, the conclusions of paper and the future directions are given in section 4.

2. Model Adopted

This paper uses the MFIS proposed in 1975 by Ebrahim Mamdani [8]. The main components of the MFIS are a fuzzy rule base, dictionary and reasoning mechanism. A fuzzy rule base is a collection of fuzzy rules obtained from experts. Fuzzy rules are linguistic statements, which describe how the fuzzy inference system should make a decision regarding classifying
an input or controlling an output [9]. The general format of a fuzzy rule is if (input-1 is membership function-1) and/or (input-2 is membership function-2) and/or …… (input-n is membership function-n) then (output is output membership-function). The dictionary defines all the membership functions used in the fuzzy rules. The fuzzy reasoning mechanism performs the inference procedure. The input to the FIS can be either fuzzy or crisp. If the input is a crisp value, it can be converted to a fuzzy value using a fuzzification process. Generally the output produced by MFIS is fuzzy; however if crisp output is required it can be converted into crisp form using a defuzzification process. This model uses the following steps to compute the output [10].

1. Design a set of fuzzy rules.
2. Convert the input into fuzzy form.
3. Combine the fuzzified inputs according to the fuzzy rules for establishing rule strength.
4. Determine the consequent of the rules for establishing rule strength and the output membership function.
5. Combine all consequents to get an output distribution.
6. Defuzzify the output distribution.

2.1 Inputs and Output of Model

Software maintainability is an important attribute of software quality. It can not be measured directly. It can be measured indirectly using measures of design structures such as class diagram. For object-oriented software, Chidamber and Kemerer proposed a metrics suite consisting of six metrics that can be used to measure the complexity, reusability, coupling and cohesion of a class [11]. These metrics are further correlated with the maintainability of a class. This paper uses the CK metrics suite as input for the adopted model. The CK metrics suite includes the following six metrics.

2.1.1 Weighted Methods Per Class (WMC)

WMC is the sum of the complexities of all the methods of a class. It is one the predictors of class maintainability. A higher value of WMC indicates a lower maintainability of the class.

2.1.2 Depth of Inheritance Tree (DIT)

The DIT of a class is its maximum depth from the root node in the inheritance hierarchy. An empirical study by Daly et al. [12] indicates that a class with DIT 3 is easier to maintain compared to a class with no inheritance.

2.1.3 Number of Children (NOC)

The NOC of a class is the total number of immediate subclasses of that class. A class with higher NOC requires more testing efforts.

2.1.4 Coupling Between Object Classes (CBO)

The CBO of a class is the total number of other classes with which it is coupled. A higher value of CBO indicates lower maintainability of a class due to higher sensitivity to changes in the other classes.

2.1.5 Response For a Class (RFC)

The RFC of a class is the total number of methods that can be executed in response to a message received by an object of that class. It is the sum of the methods defined in the class and the methods that are directly invoked by methods of the class. A higher value of RFC indicates a higher complexity of the class, therefore less maintainability.

2.1.6 Lack of Cohesion in Methods (LCOM)

The LCOM of a class is the difference between the number of pairs of methods that have no common attribute and the number of pairs of methods that have common attributes. It measures the dissimilarity of methods in a class on the basis of attributes used by methods of the class. A positive value of LCOM for the class indicates that the class is less cohesive. For good quality design a positive value of LCOM is not desirable. Low cohesion is an indicator of higher complexity.

2.2 Membership Functions for Inputs and Output

All inputs and outputs are represented using three fuzzy values - Low, Medium and High. The trapezoidal membership function is used for representing fuzzy values. The trapezoidal function depends upon four scalar parameters - a, b, c and d.

\[
f(x;a,b,c,d) = \begin{cases} 
0, & x \leq a \\
\frac{x-a}{b-a}, & a \leq x \leq b \\
1, & b \leq x \leq c \\
\frac{d-x}{d-c}, & c \leq x \leq d \\
0, & d \leq x 
\end{cases}
\]

Different threshold values of CK metrics are given by various researchers [13,14,15,16]. This paper designs three membership functions (LOW, MEDIUM and HIGH) for each input and output. Details of all inputs (Input1 to Input6) and output (Output1) along with their graphics representations are as follows. MF and trapmf stand for membership function and trapezoidal membership function respectively. Each membership function is defined in terms of four parameters. 

[Input1]
Name='WMC'
Range=[0 30]
NumMFs=3
MF1='LOW':'trapmf',[0 0 5 10]
MF2='MEDIUM':'trapmf',[5 10 15 20]
MF3='HIGH':'trapmf',[15 20 30 30]
Figure 2: Membership functions of WMC

[Input2]
Name='DIT'
Range=[0 10]
NumMFs=3
MF1='LOW':'trapmf',[0 0 1 3]
MF2='MEDIUM':'trapmf',[1 3 4 6]
MF3='HIGH':'trapmf',[4 6 10 10]

Figure 3: Membership functions of DIT

[Input3]
Name='NOC'
Range=[0 12]
NumMFs=3
MF1='LOW':'trapmf',[0 0 2 4]
MF2='MEDIUM':'trapmf',[2 4 6 8]
MF3='HIGH':'trapmf',[6 8 12 12]

Figure 4: Membership functions of NOC

[Input4]
Name='RFC'
Range=[0 60]
NumMFs=3
MF1='LOW':'trapmf',[0 0 10 20]
MF2='MEDIUM':'trapmf',[10 20 30 40]
MF3='HIGH':'trapmf',[30 40 60 60]

Figure 5: Membership functions of RFC

[Input5]
Name='CBO'
Range=[0 10]
NumMFs=3
MF1='LOW':'trapmf',[0 0 2 4]
MF2='MEDIUM':'trapmf',[2 4 5 7]
MF3='HIGH':'trapmf',[5 7 10 10]

Figure 6: Membership functions of CBO

[Input6]
Name='LCOM'
Range=[0 10]
NumMFs=3
MF1='LOW':'trapmf',[0 0 1 3]
MF2='MEDIUM':'trapmf',[1 3 4 6]
MF3='HIGH':'trapmf',[4 6 10 10]

Figure 7: Membership functions of LCOM

[Output1]
Name='Maintainability'
Range=[0 1]
NumMFs=3
MF1='LOW':'trapmf',[0 0 0.2 0.4]
2.3 Rule Base

In general, with n inputs and m membership functions, m^n rules can be generated. In this model, six inputs are used and for each input three membership functions are used; as a result, 729 rules are designed. The format of a rule is as follows.

If (WMC is LOW/MEDIUM/HIGH) and (DIT is LOW/MEDIUM/HIGH) and (NOC is LOW/MEDIUM/HIGH) and (RFC is LOW/MEDIUM/HIGH) and (CBO is LOW/MEDIUM/HIGH) and (LCOM is LOW/MEDIUM/HIGH) then (Maintainability is LOW/MEDIUM/HIGH). For example, If (WMC is LOW) and (DIT is LOW) and (NOC is LOW) and (RFC is LOW) and (CBO is LOW) and (LCOM is LOW) then (Maintainability is HIGH). The following snapshot displays a graphical representation of the rules.

3. Measuring Class Complexity and Maintainability

The given model was applied to 15 classes and the results are shown in Table 1.

<table>
<thead>
<tr>
<th>Class</th>
<th>WMC</th>
<th>DIT</th>
<th>NOC</th>
<th>RFC</th>
<th>CBO</th>
<th>LCOM</th>
<th>Composite Complexity</th>
<th>Maintainability (Mamdani Model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>5</td>
<td>5.37</td>
<td>0.83</td>
</tr>
<tr>
<td>C2</td>
<td>21</td>
<td>3</td>
<td>0</td>
<td>32</td>
<td>5</td>
<td>5</td>
<td>12.86</td>
<td>0.44</td>
</tr>
<tr>
<td>C3</td>
<td>28</td>
<td>7</td>
<td>0</td>
<td>36</td>
<td>0</td>
<td>0</td>
<td>13.53</td>
<td>0.34</td>
</tr>
<tr>
<td>C4</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>12</td>
<td>3</td>
<td>4</td>
<td>5.48</td>
<td>0.63</td>
</tr>
<tr>
<td>C5</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>2.74</td>
<td>0.84</td>
</tr>
<tr>
<td>C6</td>
<td>25</td>
<td>7</td>
<td>0</td>
<td>46</td>
<td>0</td>
<td>5</td>
<td>16.13</td>
<td>0.17</td>
</tr>
<tr>
<td>C7</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>4.72</td>
<td>0.72</td>
</tr>
<tr>
<td>C8</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>4.47</td>
<td>0.63</td>
</tr>
<tr>
<td>C9</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td>18</td>
<td>1</td>
<td>0</td>
<td>6.25</td>
<td>0.84</td>
</tr>
<tr>
<td>C10</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2.43</td>
<td>0.83</td>
</tr>
<tr>
<td>C11</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>3.18</td>
<td>0.85</td>
</tr>
<tr>
<td>C12</td>
<td>26</td>
<td>4</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>11.4</td>
<td>0.5</td>
</tr>
<tr>
<td>C13</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>3.21</td>
<td>0.83</td>
</tr>
<tr>
<td>C14</td>
<td>27</td>
<td>4</td>
<td>2</td>
<td>50</td>
<td>6</td>
<td>1</td>
<td>17.84</td>
<td>0.38</td>
</tr>
<tr>
<td>C15</td>
<td>15</td>
<td>2</td>
<td>0</td>
<td>25</td>
<td>6</td>
<td>5</td>
<td>10.43</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Jubair et al. [7] suggested an importance factor for each CK metric, which is represented by LOW(1), MEDIUM(2) and HIGH(3). The importance factor reflects the importance of the metric in determining the software quality. WMC, DIT, NOC, CBO, RFC and LCOM are assigned importance factors 2, 2, 1, 3, 3 and 2 respectively [7]. On the basis of the normalized values (0.08 for Low, 0.15 for Medium and 0.23 for High) of the importance factors, a weight is assigned to each CK metric. WMC, DIT, NOC, CBO, RFC and LCOM are assigned the weights 0.15, 0.15, 0.08, 0.23, 0.23 and 0.15 respectively. Using these weights, the composite complexity is computed. The coefficient of correlation between the composite complexity and the maintainability of the class is -0.92, which indicates that a higher value of class complexity lowers the maintainability of class.

4. Conclusion and Future Directions

This paper presents the application of MFIS for computing the maintainability of a class in object-oriented software. The results produced by the MFIS satisfy the relationship between complexity and maintainability, i.e., higher complexity leads to lower maintainability. However, the given model was applied to small academic-level classes and the same study can be replicated with larger industrial projects.
5. References


