Software Quality: Predicting Reliability of a Software Using a Decision Tree

1Ayanloye O. Shakiru, 1Mahmud Ahmad Yusuf, 1Koh Tieng Wei and 2Sukumar Letchmunan
1Software Engineering Research Group, Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, UPM, Selangor, 43400, Selangor, Malaysia
2School of Computer Science, Universiti Sains Malaysia, USM, 11800, Penang, Malaysia

ABSTRACT
System availability can be expressed as an attribute of reliability that determines the total time a system or component is functioning. Most available models try to predict availability of a software during its lifecycle but there are very few or no models that predict a software going days without a failure. Over the years, decision tree model have been used as a reliable technique for prediction. In this study, based on the sample data collected by John Musa of Bell Telephone Laboratories, a decision tree model has been used to predict the availability of a system going days without a failure. This study concluded that a decision tree model is able to decide availability of a software in terms of going days without a failure.

Keywords: Software quality, software measurement, software reliability, software availability, decision tree model

INTRODUCTION

The statement of Tom DeMarco “you can’t control what you can’t measure” has become the motto of expert for software quality trying to develop and apply quality metrics in the software industry (Daniel, 2004). For years, stakeholders of software projects have found it is challenging to apply software technologies, methods and process control to software development (Siok and Tian, 2007). Generally, there is no easy-to-use or standard industrial measurement for software projects (Siok and Tian, 2007). According to previous studies the diversity of technology, running environment and other factors, make it difficult to have specific quality definition for software systems (Schneidewind, 2002). Lyu (2007) identifies software reliability as an activity that spans throughout the entire lifecycle of a software and can be discussed with respect to areas like software architecture, design, testing, metrics and emerging applications.

While software reliability refers to the ability of a software component consistently perform its duty according to its specifications, the ratio of time a system or component is functioning to the total time it is required or expected to function is known as software availability. Availability is an attribute of reliability which is a factor software quality that determines the total time a system or component is expected to be functional. Over the years, different models have been proposed to predict different aspects of software reliability. These models use historical data to solve the problems they addressed. A lot of software reliability models have been developed to meet the needs of managers, software engineers and system engineers to be able to predict software reliability (Aljahdali et al., 2003). When trying to measure software availability, there are two attributes to consider (Buckley and Poston, 1984). The first is the Mean Time to Failure (MTTF). The second is the Mean Time to Repair (MTTR).

On the other hand, many software reliability models have been developed using different classification and prediction techniques. Zhao and Li (2011) define a decision tree as “an analysis skill and a classification algorithm, whose basic principle is the combination of probability theory and an analysis tool of tree shapes”. It derives a hierarchy of partition rules with respect to a target attribute of a large dataset.

This study propose a decision tree model to determine availability of a software in terms of going days without having any failures.
INDUSTRIAL RELATED PROBLEMS

Quality is usually observed from different points of view. The most popular being supplier (developer) and customer (user) points of view. The problem with measurement of software quality arises from the definition of software quality. Many experts have different opinions on what the definition of software quality is. It is a general idea that software quality can be measured subjectively or objectively as supported by Khosravi and Gueheneuc (2004) where quality is said to be perceptual, conditional and to an extent a subjective attribute as understood by different people.

Kotaiah and Khan (2012) stated that there is no particular model applicable to all situations because none of them is capable of capturing sufficient software characteristics. This means most models are specific to performing particular task. Khosravi and Gueheneuc (2004) in their own idea, identified human estimation, software metrics and the quality models being used as tools for assessment of software quality that should be modified and improved.

In Kotaiah and Khan (2012), non-compliance, good architecture and coding practices are the most common causes of poor reliability. Measuring the static quality attributes of software would help in detection of non-compliance. “Assessing the static attributes underlying an application’s reliability provides an estimate of the level of business risk and the likelihood of potential application failures and defects the application will experience when placed in operation”. The three causes identified above can be broken up into application architecture practices, coding practices, complexity of algorithms, complexity of programming practices, compliance with object-oriented and structured programming best practices (when applicable), component or pattern re-use ratio, dirty programming, error and exception handling (for all layers-gui, logic and data), multi-layer design compliance. Most of the problems identified earlier fall into one of these three groups.

Lyu (2007) addressed the trends and problems from the software reliability engineers’ point of view. He expanded the problems by pointing out that the problems of software reliability are not just size, complexity, difficulty and novelty of the application but also included knowledge, training, experience and character of the software engineers.

In the past, a lot of pioneers in the software engineering industry have come up with different ways to measure software reliability based on their understanding of the domain. Regardless of the amount of time and effort put into it, there is no all-purpose model to predicting reliability.

Florac (1992) developed a mechanism in form of a framework which can be used to describe and specify two types of software measures, software problems and defects. The framework will identify measurable attributes and use them against a checklist to identify problem and defect measurements.

After coming across some issues in their research, Khosravi and Gueheneuc (2004) came up with a 9 steps to software quality evaluation which solves some of their issues. The 9 steps, respectively are, choosing category of people, identifying sample programs, building a quality model, human evaluation, computing software metrics over BP, machine learning tools, computing software metrics over EP, adapting metric and finally software evaluation. The method they used was new but still used classical tools of software engineering. Kotaiah and Khan (2012) stating that there is no particular model applicable to all situations because none of them is capable of capturing sufficient software characteristics, proposed a series of machine learning methods for assessing software reliability. These methods are fuzzy approach, neuro-fuzzy approach, artificial neural network approach, genetic algorithm approach, Bayesian classification approach, Support Vector Machine (SVM) approach and self-organizing map approach.

Aljahdali et al. (2003) set an evaluation criterion and used it to carry out experiment on software reliability using parametric and non-parametric methods. At the end of their experiment they concluded that when there is an absence of historical data, non-parametric models performed better than parametric models.

Singh and Kumar (2010) developed a model for prediction that can be used across different circumstance and still be capable of accurate prediction. They used a combination of two models. The neural networks model is used for reliability prediction in real environments and the connectionist model for assessment of reliability. Their results showed an improvement when using artificial neural networks over statistical models based on NHPP.

Aljahdali and Sheta (2011) used a fuzzy logic model which consisted of several linear sub-models put together smoothly by using fuzzy membership functions. In the end they developed a fuzzy model for predicting reliability of software projects.

Musa and Okumoto (1984) developed a simple model capable of predicting failures expected of software. The model was presumed better than most models at the time. The model uses two derived units, execution time and calendar time.

Karunanithi and Whiteley (1992) presented a modeling approach that is adaptive by using connectionist networks and shows how feed forward, recurrent networks and various training regimes are used in prediction of software reliability. An empirical comparison was done between their new approach and five already existing models. The result was that their new connectionist network model adapted better to different dataset and in long term preserved its predictive accuracy over the analytic models.

Brocklehurst et al. (1990) used a recalibration process to improve the accuracy for reliability predictions of already existing techniques. The result was an improvement in prediction reliability in majority of the cases.

The model presented by Malaiya et al. (1990), characterizes the long term predictability of a model by using a predictability measure with two-component. Average predictability being the first component, measures a models
capability to predict well throughout the testing phase. Average bias being the second component measures chances that an underestimation or overestimation of the faults could be possible. In their conclusion they said “Our results seem to support Musa’s observation (Musa et al., 1987) that the logarithmic model appears to have good predictability in most cases. However, at very low fault densities, the exponential model may be slightly better. The delayed S-shaped model which in some cases have been shown to have good fit, generally performed poorly”.

Karunanithi et al. (1992) explored a connectionist method in different network models, data representation methods and training regimes. A comparison between their connectionist method and five other well-known reliability growth models revealed that their connectionist model can adapt across various dataset and still preserve its prediction accuracy. The connectionist method can also be used to model variation in complexity.

**METHODOLOGY**

**Software reliability data source:** The software reliability data used in this study has been collected by John Musa (Bell Telephone Laboratories). He collected failure interval dataset with the purpose of helping software managers monitor test status, predict schedules and help researchers to validate software reliability models. The models are applicable to the area of software reliability engineering. The dataset was collected from failure of 16 projects. Table 1 shows all the 16 projects and the information recorded. The data represents a variety of applications and was recorded in the mid 1970’s. The application types are real time command and control, word processing, commercial and military applications. The attributes recorded for each software are system code, failure number, failure interval and day of failure.

While trying to accomplish the goal, additional data was collected on top of the one provided by John Musa decision tree model construction. The data used is showed in Table 2.

**Methodology of measurement program:** The measurement program done in three phases. The first is to calculate the reliability of each software. Reliability here is measure in terms of the availability of the system. The second phase would be using a reliability scale to determine if the software is reliable or not.

The third and final phase is to classify the reliability and use attribute of each software to model a decision tree.

**Calculating software reliability:** Software reliability can be expressed in different ways depending on what you intend

<table>
<thead>
<tr>
<th>System code</th>
<th>Application</th>
<th>Size</th>
<th>Failures</th>
<th>Phases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Real time command and control</td>
<td>21,700</td>
<td>136</td>
<td>System test operations</td>
</tr>
<tr>
<td>2</td>
<td>Real time command and control</td>
<td>27,700</td>
<td>54</td>
<td>System test operations</td>
</tr>
<tr>
<td>3</td>
<td>Real time command and control</td>
<td>33,500</td>
<td>55</td>
<td>System test operations</td>
</tr>
<tr>
<td>4</td>
<td>Real time command and control</td>
<td>2,445,000</td>
<td>831</td>
<td>System test*</td>
</tr>
<tr>
<td>5</td>
<td>Commercial subsystem</td>
<td>5,700</td>
<td>73</td>
<td>Subsystem test</td>
</tr>
<tr>
<td>14C</td>
<td>Real time (Hundreds of thousands)</td>
<td>36</td>
<td>Operations</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Military</td>
<td>61,900</td>
<td>38</td>
<td>System test</td>
</tr>
<tr>
<td>27</td>
<td>Military</td>
<td>126,100</td>
<td>41</td>
<td>System test</td>
</tr>
<tr>
<td>40</td>
<td>Military</td>
<td>180,000</td>
<td>101</td>
<td>System test</td>
</tr>
<tr>
<td>SS1A</td>
<td>Operating system (Hundreds of thousands)</td>
<td>112</td>
<td>Operations*</td>
<td></td>
</tr>
<tr>
<td>SS1B</td>
<td>Operating system (Hundreds of thousands)</td>
<td>375</td>
<td>Operations*</td>
<td></td>
</tr>
<tr>
<td>SS1C</td>
<td>Operating system (Hundreds of thousands)</td>
<td>277</td>
<td>Operations*</td>
<td></td>
</tr>
<tr>
<td>SS2</td>
<td>Time sharing system (Hundreds of thousands)</td>
<td>192</td>
<td>Operations*</td>
<td></td>
</tr>
<tr>
<td>SS3</td>
<td>Word processing system (Hundreds of thousands)</td>
<td>278</td>
<td>Operations*</td>
<td></td>
</tr>
<tr>
<td>SS4</td>
<td>Operating system (Hundreds of thousands)</td>
<td>196</td>
<td>Operations*</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System code</th>
<th>Application</th>
<th>Phases</th>
<th>Software reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Real time command and control</td>
<td>System test operations</td>
<td>Unreliable</td>
</tr>
<tr>
<td>2</td>
<td>Real time command and control</td>
<td>System test operations</td>
<td>Unreliable</td>
</tr>
<tr>
<td>3</td>
<td>Real time command and control</td>
<td>System test operations</td>
<td>Unreliable</td>
</tr>
<tr>
<td>4</td>
<td>Real time command and control</td>
<td>System test operations</td>
<td>Unreliable</td>
</tr>
<tr>
<td>5</td>
<td>Real time commercial</td>
<td>System test*</td>
<td>Unreliable</td>
</tr>
<tr>
<td>6</td>
<td>Commercial subsystem</td>
<td>Subsystem test</td>
<td>Unreliable</td>
</tr>
<tr>
<td>7</td>
<td>Real time</td>
<td>Operations</td>
<td>Reliable</td>
</tr>
<tr>
<td>8</td>
<td>Military</td>
<td>System test</td>
<td>Unreliable</td>
</tr>
<tr>
<td>9</td>
<td>Military</td>
<td>System test</td>
<td>Unreliable</td>
</tr>
<tr>
<td>10</td>
<td>Military</td>
<td>System test</td>
<td>Unreliable</td>
</tr>
<tr>
<td>11</td>
<td>Operating system</td>
<td>Operations*</td>
<td>Data is unreliable</td>
</tr>
<tr>
<td>12</td>
<td>Operating system</td>
<td>Operations*</td>
<td>Reliable</td>
</tr>
<tr>
<td>13</td>
<td>Operating system</td>
<td>Operations*</td>
<td>Reliable</td>
</tr>
<tr>
<td>14</td>
<td>Time sharing system</td>
<td>Operations*</td>
<td>Reliable</td>
</tr>
<tr>
<td>15</td>
<td>Word processing system</td>
<td>Operations*</td>
<td>Reliable</td>
</tr>
<tr>
<td>16</td>
<td>Operating system</td>
<td>Operations*</td>
<td>Reliable</td>
</tr>
</tbody>
</table>
to measure. The common and most widely acceptable calculation for availability is expressed as in Eq. 1:

\[
A = \frac{MTBF}{MTBF + MTTR}
\]  

(1)

where, MTBF is the mean time before failure and MTTR is the mean time to repair.

The availability of each software was calculated using the equation above and the data provided. The output should be from a range of 0.0-1.0. The reliability should lie somewhere between. Reliability scale: For the purpose of classification a threshold is needed for the acceptable level of availability before software can be accepted as reliable. The suggested threshold before reliability can be achieved is set to 0.7. This means after the calculation for individual software is done, those with availability lower than 0.7 will be considered as unreliable while those with availability of 0.7 and above will be considered reliable. For what so ever reason if the availability falls outside the range of 0.0-0.1, the data for that software will be considered unreliable.

**Decision tree model:** Based on the attributes of the software and newly derived attribute (availability), a decision tree is used to predict the reliability of each software. Table 2 shows the data used to model the decision tree.

**Rules generation:**
- Each attribute-value pair along a given path forms a conjunction in the rule antecedent and the leaf node is the consequent

**Basic algorithm (a greedy algorithm):**
- Tree is constructed in a top-down recursive divide-and-conquer
- At start, all the training examples are at the root
- Attributes are categorical (if continuous-valued, they are discretized in advance)
- Examples are partitioned recursively based on selected attributes
- Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)

**Conditions for stopping partitioning:**
- All samples for a given node belong to the same class
- There are no remaining attributes for further partitioning- majority voting is employed for classifying the leaf
- There are no samples left

**Attribute selection measure: Information gain:**
- Select the attribute with the highest information gain
- \( S \) be a set consisting of \( s \) data samples
- Let \( s_i \) no of tuples in class \( C_i \) for \( i = \{1, \ldots, m\} \)

\[
I(S, S_1, \ldots, S_m) = -\sum_{i=1}^{m} \frac{S_i}{S} \log_2 \frac{S_i}{S}
\]  

(2)

- Entropy of attribute \( A \) with distinct values \( \{a_1, a_2, a_3, \ldots, a_l\} \):

\[
E(A) = \sum_{i=1}^{l} \frac{S_{a_i}}{S} I(S_{a_i}, \ldots, S_m)
\]  

(3)

- Information gained by branching on attribute \( A \)

\[
\text{Gain}(A) = I(S, S_2, \ldots, S_m) - E(A)
\]  

(4)

The gain \( \text{Gain}(A) \) for each attribute was calculated by subtracting the entropy of the attribute \( E(A) \) from the information gain \( I \). After calculating the gain for each attribute, selects the largest as the most eligible to become the root node.

**Implementation:** For implementation of the measurement program, a program written in C++ to read the data being used and compute the necessary attribute require to model the decision tree. The program calculates the reliability of each software in the data set and based on the set threshold determines if the software is reliable or not and generates a new table with an extra attribute, reliability. The program uses the attribute of the new table to determine the information gain for the table. It then calculates the entropy of each attribute and subtracts the entropy of each attribute from the information gain. This would repeat itself until the information is sufficient to model the decision tree.

---

**RESULTS AND DISCUSSION**

Although this project has not compared the model to current and existing models, it is certain that the model is capable to predict software reliability in terms of availability to a modest contempt.

The justification for scaling reliability between 0 and 1 is because this scaling system has been used before (Karunanithi and Malaia, 1992; Castner and Ferguson, 1998; Karunanithi et al., 20191). Setting the threshold at 0.7 comes from the perception that 0.7 is the accepted minimum
threshold for reliability to be achieved (Castner and Ferguson, 1998). Size of the project was removed from the original data (Table 1). This is because size of the project on its own has little or no impact on the result of the table.

The measurement of reliability is subjective to different people but here the model proposed is specific to the prediction of availability of software. The decision tree model has not been compared with other existing model, so it cannot say at the moment to what extent is it better than others. The model has been applied to only the dataset discussed here. Although, it is expected that the result if applied to a different dataset would be similar.

CONCLUSION

This study developed a decision tree model which is able to predict the reliability of a software. It described using a threshold of 0.7 to determine the reliability and unreliability of software. The future work would check the effectiveness of this model against other common and well known models.

REFERENCES


