http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 13 (18): 2755-2759, 2014
ISSN 1812-5638

ansinet
Asian Network for Scientific Information

RESEARCH ARTICLE OPEN ACCESS

DOI: 10.3923/itj.2014.2755.2759

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

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

Corresponding Author:
Koh Tieng Wei,
Department of Software Engineering and Information System,
Faculty of Computer Science and Information Technology,
Universiti Putra Malaysia, UPM,
Serdang, 43400, Selangor, Malaysia
Tel: +603-89471799
Fax: +603-89466576

A B S T R A C T

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 life cycle 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.

Key words: 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) point 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 Whitley (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 recaliberation 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 1: Software reliability data project information

System code	Application	Size	Failures	Phases
1	Real time command and control	21,700	136	System test operations
2	Real time command and control	27,700	54	System test operations
3	Real time command and control	23,400	38	System test operations
4	Real time command and control	33,500	53	System test operations
5	Real time commercial	2,445,000	831	System test*
6	Commercial subsystem	5,700	73	Subsystem test
14C	Real time	(Hundreds of thousands)	36	Operations
17	Military	61,900	38	System test
27	Military	126,100	41	System test
40	Military	180,000	101	System test
SS1A	Operating system	(Hundreds of thousands)	112	Operations*
SS1B	Operating system	(Hundreds of thousands)	375	Operations*
SS1C	Operating system	(Hundreds of thousands)	277	Operations*
SS2	Time sharing system	(Hundreds of thousands)	192	Operations*
SS3	Word processing system	(Hundreds of thousands)	278	Operations*
SS4	Operating system	(Hundreds of thousands)	196	Operations*

Table 2: Prediction data

System code	Application	Phases	Software reliability
1	Real time command and control	System test operations	Unreliable
2	Real time command and control	System test operations	Unreliable
3	Real time command and control	System test operations	Unreliable
4	Real time command and control	System test operations	Unreliable
5	Real time commercial	System test*	Unreliable
6	Commercial subsystem	Subsystem test	Unreliable
7	Real time	Operations	Reliable
8	Military	System test	Unreliable
9	Military	System test	Unreliable
10	Military	System test	Unreliable
11	Operating system	Operations*	Data is unreliable
12	Operating system	Operations*	Reliable
13	Operating system	Operations*	Reliable
14	Time sharing system	Operations*	Reliable
15	Word processing system	Operations*	Reliable
16	Operating system	Operations*	Reliable

to measure. The common and most widely acceptable calculation for availability is expressed as in Eq. 1:

$$A = \frac{MTBF}{MTBF + MTTR} \quad (1)$$

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 use 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,..., m}
- The expected information needed to classify:

$$I(S_1, S_2, ..., S_m) = -\sum_{i=1}^{m} \frac{S_i}{S} \log 2 \frac{S_i}{S} \quad (2)$$

- Entropy of attribute A with distinct values $\{a_1, a_2, a_3 \ldots a_v\}$:

$$E(A) = \sum_{j=1}^{v} \frac{S_{1j} + ... + S_{mj}}{S} I(S_{1j}, ..., S_{mj}) \quad (3)$$

- Information gained by branching on attribute A

$$Gain(A) = I(S_1, S_2, ..., S_m) - E(A) \quad (4)$$

The gain 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 generate 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.

```
Begin
Repeat until all data file (i)
        Read File (i)
        Calculate MTBF
        Calculate MTTR
        Calculate Availability
        IF Availability ≥0.7
        THEN Reliability = "Reliable"
        IF Availability < 0.7
        THEN Reliability = "Unreliable"
      Print to new file
   End repeat
End
```

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 Malaiya, 1992; Castner and Ferguson, 1998; Karunanithi *et al*., 1991). 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

Aljahdali, S., A.F. Sheta and M. Habib, 2003. Software reliability analysis using parametric and non-parametric methods. Proceedings of the ISCA 18th International Conference on Computers and their Applications, March 26-28, 2003, Honolulu, Hawaii, USA., pp: 1-4.

Aljahdali, S. and A.F. Sheta, 2011. Predicting the reliability of software systems using fuzzy logic. Proceedings of the 8th International Conference on Information Technology: New Generations, April 11-13, 2011, Las Vegas, NV., pp: 36-40.

Brocklehurst, S., B. Littlewood and J. Snell, 1990. Recalibrating software reliability models. Software Eng. Trans., 16: 458-470.

Buckley, F.J. and R. Poston, 1984. Software quality assurance. IEEE Trans. Software Eng., 10: 36-41.

Castner, G.J. and C.B. Ferguson, 1998. Development of a scale for measuring software diffusion. Proceedings of the 31st Hawaii International Conference on System Sciences, Volume 6, January 6-9, 1998, Kohala Coast, HI., pp: 518-528.

Daniel, G., 2004. Software Quality Assurance: From Theory to Implementation. Pearson Education India, India, ISBN-13: 9788131723951, Pages: 616.

Florac, W.A., 1992. Software quality measurement: A framework for counting problems and defects. A Technical Report, Software Engineering Institute, September 1992.

Karunanithi, N., Y.K. Malaiya and D. Whitley, 1991. Prediction of software reliability using neural networks. Proceedings of the International Symposium on Software Reliability Engineering, May 17-18, 1991, Austin, TX., pp: 124-130.

Karunanithi, N. and D. Whitley, 1992. Prediction of software reliability using feedforward and recurrent neural nets. Proceedings of the International Joint Conference on Neural Networks, Volume 1, January 7-11, 1992, Baltimore, MD., pp: 800-805.

Karunanithi, N. and Y.K. Malaiya, 1992. The scaling problem in neural networks for software reliability prediction. Proceedings of 3rd International Symposium on Software Reliability Engineering, October 7-10, 1992, Research Triangle Park, NC., pp: 76-82.

Karunanithi, N., D. Whitley and Y.K. Malaiya, 1992. Prediction of software reliability using connectionist models. IEEE Software Eng. Trans., 18: 563-574.

Khosravi, K. and Y. Gueheneuc, 2004. On issues with software quality models. Proceedings of the 11th Working Conference on Reverse Engineering, November 8-12, 2004, Delft, Netherlands, pp: 172-181.

Kotaiah, B. and R.A. Khan, 2012. A survey on software reliability assessment by using different machine learning techniques. Int. J. Sci. Eng. Res., 3: 1-7.

Lyu, M.R., 2007. Software reliability engineering: A roadmap. Proceedings of the Workshop on the Future of Software Engineering, May 23-25, 2007, Minneapolis, MN., USA., pp: 153-170.

Malaiya, Y.K., N. Karunanithi and P. Verma 1990. Predictability measures for software reliability models. Proceedings of the 14th Annual International Computer Software and Applications Conference, October 31-November 2, 1990, Chicago, IL., pp: 7-12.

Musa, J.D. and K. Okumoto, 1984. A logarithmic poisson execution time model for software reliability measurement. Proceedings of the 7th International Conference on Software Engineering, March 26-29, 1984, Orlando, FL., USA., pp: 230-238.

Musa, J.D., A. Iannino and K. Okumoto, 1987. Software Reliability: Measurement, Prediction and Application. McGraw-Hill, New York, ISBN-13: 9780070440937, Pages: 621.

Schneidewind, N.E., 2002. Body of knowledge for software quality measurement. Computer , 35: 77-83.

Singh, Y. and P. Kumar, 2010. Prediction of software reliability using feed forward neural networks. Proceedings of the International Conference on Computational Intelligence and Software Engineering, December 10-12, 2010, Wuhan, pp: 1-5.

Siok, M.F. and J. Tian, 2007. Empirical study of embedded software quality and productivity. Proceedings of the 10th High Assurance Systems Engineering Symposium, November 14-16, 2007, Plano, TX., pp: 313-320.

Zhao, M. and X. Li, 2011. An application of spatial decision tree for classification of air pollution index. Proceedings of the 19th International Conference on Geoinformatics, June 24-26, 2011, Shanghai, pp: 1-6.