Some Observations on Early Software Reliability of Object Oriented System

Anil Kumar Malviya

Abstract

Object-oriented approach is becoming very popular in software development community as an alternative to traditional approach i.e. structured analysis & design methodologies due to obvious reasons. This approach has become de facto standard of software development organization. As we know, software reliability is one of the important dynamic quality factors in overall quality models proposed in the literature. Several guidelines are available in literature that suggest various "do's" and "don'ts" to produce an "easy to maintain" and "reliable" system. Metrics to measure software reliability do exist and can be used starting in requirements phase to software design phase. In this paper we propose models that capture such ideas and prepare background for development of metrics for assessment of early reliability of software.

Keywords: Software metrics, Reliability, Inheritance, Coupling & Cohesion

1. Introduction

Object Oriented software development is a very new way (natural way) of thinking about software based on abstractions of objects that exist in the real world or problem domain. Objected Oriented approaches are getting a lot of attention from software development community. This is due to a variety of claims by many software researchers and practitioners that an object oriented approach to software development leads to better productivity, reliability, maintainability, software reusability and increased extensibility. A number of papers have investigated the relationship between design metrics and detection of faults in Objected Oriented software[6,7,8,9]. Today, there is a compelling need in educating engineers in reliability engineering and developing economic and maintainable solutions in the context of legacy systems, and not only in fast development of software with the latest technology.

Software reliability modeling is the utmost productive research within software reliability engineering and over 100 models have been developed through different approaches ([10], [11], [12], [13],[14]). They provide useful feedback to the management to keep the software process under control. Numerous studies have been performed in the area of early software reliability predictions & importance over the last few years(Tian, Jeff 2000, Nagappan, N. 2004, Cukic 2005, Tripathi, R 2005, Ion, R.A. 2006, Hu, Q.P. et. al 2006 and many others) using different modeling techniques. Early software reliability prediction models are of paramount importance since they provide early identification of cost overruns, software development process issues, optimal development strategies, etc. Smidth, C. et al.[15] proposed an approach to the prediction of software reliability based on a systematic identification of software process failure modes and their likelihoods.

[1] Assistant Professor, Department of Computer Science & Engineering , Amity School School of Engineering & Technology, Amity University, Noida (Uttar Pradesh). Email:anilkmalviya@yahoo.com, akmalviya@amity.edu

A direct consequence of the approach and its supporting data collection efforts is the identification of weak areas in the software development process. Tripathi, R [16] proposed a model for early software reliability and design assessment. This model is based on Reliability Block Diagram (RBD) for representing real-world problems and an algorithm for analysis of these models in early phase of software development. Cukic, B. [17] proposed a model based on UML to find early reliability assessment that spans the life cycle, from requirements modeling to deployment, corrective maintenance, and evalution. Cortellessa, V. et al. [18] proposed a methodology that starts with the analysis of the UML model of software architecture followed by the Bayesian framework for reliability prediction. They utilize three different types of UML diagrams: Use case, Sequence and Deployment Diagrams. They are annotated with reliability related attributes.

Generally, these models are developed through two approaches: analytical and data-driven. Analytical growth models (SRGMs) are stochastic models to describe the software failure process with essential assumptions to provide mathematical traceability. On the other hand, data-driven models are developed from historical software failure data, using regression, time series and artificial neural network approaches[13].

We first, brifly describe the "OOD Methodology", "Error, Fault, and Failure", "The Concept of Reliability", "Software Reliability Model", "Software Metrics", followed by "Known Reliability Metrics".

1.1 OOD Methodology

Object oriented design aims for robust software that can be reused, refined, tested, maintained and extended. There are several object oriented design methods, the Booch method, the Jacobson method, the Rambaugh method and the Wirfs-Brock method. Each of these object-oriented techniques in addition to identifying objects necessary to implement a system, identifying the internal details of these objects too. The OMT methodology uses three kinds of models, the object model, dynamic model and functional model to describe a system. The object model describes the static structure of the object in a system and their relationships. The object model contains object diagrams. The object diagrams represent a structure graphically. This model forms the starting point for object design. The dynamic model of a system describes the state of various objects changes when events occur. The dynamic model is used to specify and implement the control aspects of a system. The functional model is used to describe the functionality of the system i.e. the computation that takes place within a system. The functional model contains data flow diagrams.

1.2 Error, Fault and Failure

The term errors, faults and failures are often used interchangeable, but do have different meanings. Error- People make errors. A good synonym is mistake. When people make mistakes while coding, we call these mistakes bugs. Errors tend to propagate: a requirements error may be magnified during design and amplified still more during coding. A fault is the result of an error. It is more precise to say that a fault is the representation of an error, where representation is the mode of expression, such as narrative text, dataflow diagrams, hierarchy charts, source code, and so on. Defect is a good synonym for fault, as is bug. A fault occurs when a human makes a mistake, called an error, in performing some software activity. A failure occurs when a fault executes. Actually, a failure is a departure from the system's required behaviour. In other way , we can say human error can lead to a fault and a fault can lead to failure. Thus we can

say, a fault is an inside view of the system, as seen by the eyes of the developers, whereas a failure is an outside view, a problem that user can sees. Not every fault corresponds to a failure; for example, if faulty code is never executed, then the fault will never cause the code to fail.

1.3 The Concept of Reliability

The reliability of a software system is defined as the probability of failure free operation for a given time duration under specified conditions of operation. The user oriented reliability of a program (in a certain user environment) is defined as the probability that the program will give the correct output with atypical set of input data from the user environment. The sequence of codes executed in a particular run is dependent on the input data and an error in the non executed statements or branches does not have any effect on the output of the program, the system reliability depends on the probability that a bug is activated in the run. The reliability of the system, therefore, depends on the user profile. The user profile summarizes the dynamic characteristics of a typical execution of the program in a particular user environment [1]. The relationship between software quality factor and metrics can be seen in [(2), page 522, fig. 18.2].

The reliability of a component, of an object based software system that is a class can be viewed in terms of initial number of errors of faults (bugs) that can be present when a class is designed or implemented. Certain earlier studied have shown the relationship between the number of initial faults and cyclomatic complexity of a component of structured software systems [3]. Thus the number of initial faults in the methods of the objects and the number of faulty definitions/assumptions/implementation of its attributes can also indicate a measure of reliability of classes that are modules of a software system. Similarly the Control objects responsible for implementation of the dynamic behavior of software system can also be considered.

1.4 Software Reliability Models

Many Software reliability models have been developed over the years. Within these models, one can distinguish two main categories [15] :

i) Software reliability prediction models (SRPMs) typically address the reliability of the software early in the lifecycle at the requirements, preliminary design, detailed design or coding level. Predictive models are used to assess the risk of developing software for a given customer under a given set of requirements within given resources, staff, budget, schedule, and development environment i.e. before the project truly starts.

ii) Sofware reliability estimation models (SREMs) evaluate current and future software reliability from failure data gathered beginning with the integration testing of the software. In the category of estimation models one can count reliability growth models, input domain models and fault seeding models.

1.5 Software Metrics

Software metrics are quantifiable measures that could be used to measure different characteristics of a software system or the software development process. Software metrics is an emerging area. Because the software has no physical attributes, conventional metrics (such as metrics for weight, density, etc) are not much help in designing metrics for software[4].

Recent studies indicate that there are strong relationships between Object oriented metrics and software quality. Building high reliability software depends on the application of quality attributes at each phase of the development life cycle with emphasis on error prevention,

especially in the early life cycle phases. Metrics are needed at each development phase to measure applicable quality attributes[14].

1.6 Known Reliability Metrics

Many models have been proposed for software reliability assessment for structured programs including the Jelinski and Moranda models, the Goel and Okomuto model, Musa's models and Littelwood and Verall model. All these models are based on the estimation of the number of initial faults present in a program component. Hence the estimation of number of initial faults in a program unit of structured programs is also considered as a reliability model. Cheung [1] discusses the definition of user-oriented reliability and its relationship to the user profile. He has developed user-oriented reliability model to measure the quality of service a program provides to a user.

The rest of the paper is organized as follows: Section 2 discusses various points of interest, given as guidelines in the literature regarding design of reliable software systems. We propose some models in section 3, based on the discussion given below in section 2. Section 4 concludes the paper.

2. Programming for Reliability

To increase the reliability by preventing software errors, the focus must be on comprehensive requirements and a comprehensive testing plan, ensuring all requirements are tested. Focus also must be on the maintainability of software since there will be a 'useful life' phase sustaining engineering will be needed[14]. Good engineering methods largely improve software reliability. Any Object Oriented System can be characterized mainly by the encapsulation, inheritance and polymorphism. In Object Oriented System, class is the fundamental unit; therefore design of classes has a major impact on the overall quality of the design. So, it is good starting point to introduce reliability into the object oriented system. It is known that system reliability is inversely proportional to the number of unrepaired defects in the system. Here, there are some suggestions available to design reliable Object Oriented systems.

Improved programming techniques, better programming languages and better quality management have lead to very significant improvements in reliability for most software.

Reliability in a program can be achieved by avoiding the introduction of faults and by including fault tolerance facilities which allow the system to remain operational after a fault has caused a system failure. Faults are detected before the software is put into operation. Reliability in a program can also be achieved by defensive programming approach. Defensive programming is an approach to fault tolerance which can be carried out without a fault-tolerant controller. Defensive programming technique which involves incorporating checks for faults recovery code in the program. Faults are detected before they cause a system failure [3]. Reliability in Object Oriented Systems can be achieved by avoiding excess height of inheritance tree, large value of WMC, RFC, CBC and small value of NOC metric.

3. The Observations

The class is the fundamental unit of an Object Oriented System. Therefore, measures and metrics for an individual class, the class hierarchy, and class collaborations will be in invaluable to a software engineer who must assess design quality. The objective is to consider the characteristics of classes and the system for which metrics have been worked out. This section attempts to summarize the relationship between the "numbers of initial faults" in a class of software in terms of known metrics available

for classes of Object based software. The section finally proposes to work out a software reliability model treating this "number of initial faults" as a basic parameter for reliability modeling of object oriented software.

(1) The Depth of Inheritance tree (DIT) of a class C in an inheritance hierarchy is the depth from the root class in the inheritance tree. In other words, it is the length of the shortest path from the root of the tree to the node representing C or the number of ancestors C has. In case of multiple inheritances, the DIT metric is the maximum length from a root to C. The experiments show that DIT is very significant in predicting defect proneness of a class; the higher the DIT the higher is the probability that a class is defect-prone [4, 6]. Thus we can say that

No. of fault $\propto$ DIT .....(i)

(2) Suppose class C has methods $M_1, M_2, \ldots\ldots M_n$ defined on it. Let the complexity of the method M_i be C_i. The WMC (Weighted methods per class) metric is defined as WMC= $\sum_{i=1}^{n} C_i$. The analysis shows that the WMC metric has a reasonable correlation with fault–proneness of a class. The larger the WMC of a class the better the chances that the class is fault prone [4, 6]. Thus we can say that

No. of fault $\propto$ WMC ..... (ii)

(3) The RFC (response for a class) value for a class C is the cardinality of the response set for a class. The response set of class C is the set of all methods that can be invoked, if a message is sent to an object of this class. This includes all the methods of C and of other classes to which any method of C sends a message. The experiment shows that the RFC value is very significant in predicting the fault proneness of a class. The higher the RFC value the larger the probability that the class is defect prone [4, 6]. Thus we can say that

No. of fault $\propto$ RFC .....(iii)

(4) The CBC (Coupling between Classes) value for a class C is the total number of other classes to which the class is coupled. Two classes are considered coupled, if methods of one class use the method of instance variables defined in the other class. The experiment shows that the CBC value is very significant in predicting the fault proneness of a class. The higher the CBC value the larger the probability that the classes defect prone [4, 6]. Thus we can say that

No. of fault CBC .....(iv)

(5) The NOC (Number of Children) metric value of a class C is the number of immediate subclasses of C. The experiment shows that the larger the NOC, the lower the probability of detecting defects in a class. That is the higher NOC, classes are less defect prone [4, 6].
Thus we can say that

No. of fault 1/ NOC .....(v)

(6) The LCOM metric is the number of methods that access one or more of the same attributes. If LCOM is high, methods may be coupled to one another via attributes. This increases the complexity of the class design, thereby increasing the likelihood of errors during development. Thus we can state that

No. of fault $\propto$ LCOM .....(vi)

From equations (i), (ii), (iii), (iv) and (v), we can say that
No. of fault = K*(DIT * WMC *RFC *CBC*LCOM) / NOC

Where, the constant K will have to be worked out for a specific software team of concern organization based on experience of team members and other characteristics related to software processes. The number of faults can be used to determine the total number of faults in software in terms of number of initial faults in all the various class of the object based software. An alternate approach could

be to develop a reliability model of one single class using the number of initial faults in that class as a basic parameter and then a general reliability model for the total software can be expressed in terms of individual classes/objects.

4. Conclusion

The above model explains the relationship between the number of initial faults present in an object and some metrics defined for an object oriented software system. The "number of initial faults" serves as an important parameter of reliability model for software. These observations will be helpful in working out a software reliability model for object based software. The model can be used to measure of how good is one class design from the other class design, especially in terms of internal structure. Further research in the area of reliability of object oriented systems should focus on the areas: (i) Empirical validation of the above models and (ii) providing the reliability models earlier in the life cycle at the requirements and preliminary design phase.

References

[1] Roger C. Cheung, *"User Oriented Software Reliability Models"*, IEEE Transactions on Software Engineering, Vol. SE-6, No.2, PP. 118-125, March 1980.

[2] Roger S. Pressman, *"Software Engineering-A Practitioner's Approach"*, Fourth edition, The McGraw-Hill Companies, Inc.

[3] Ian Sommerville , *" Software Engineering"*, Fifth edition, Addison-Wesley Publishing Company.

[4] Pankaj Jalote, *"An Integrated Approach to Software Engineering"*, Second edition, Narosa Publishing Hose.

[5] Linda Rosenberg, Ted Hammer & Jack Shaw, *"Software Metrics & Reliability"*, Software Assurance Technology Centre, NASA.

[6] S.R. Chidamber & C.H. Kemerer, *"A Metrics Suite for Object Oriented Design"*,IEEE Transactions on Software Engineering, Vol.20, No. 6, June 1994.

[7] Briand. L.C, Wust.J, Daly. J.W and porter. D.V, *" Exploring the Relationships between Desgn Measures ans Software Quality in Object Oriented Systems"*, The Journal of Systems and Software, Vol. 51,PP. 245-273, 2000.

[8] Briand. L.C , Melo. W.L and Wust. J, *"Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects"*,ISERN Report No. ISERN-00-06, Version 2.

[9] EI-Emam, K.EI and Melo. W, *"The Prediction of Faulty Classes Using Object-Oriented Design Metrics"*, National Research Council Canada, Nov. 1999.

[10] Musa. J.D, Iannino. A & Okumoto. K, *"Software Reliability: Measurement, Prediction, Application"*, Mc-Graw-Hill, New York, 1987.

[11] Xie. M, *"Software Reliability Modeling"*, World Scientific, Singapore 1991.

[12] Lyu. M.R, *"Hand Book of Software Reliability Engineering"*, IEEE Computer Society press ,1996.

[13] Hu. Q.P, Dai.Y.S, Xie. M and Ng S.H, *"Early Software Reliability Prediction with Extended ANN Model"*, Proceeding of the 30th Annual International Computer Conference COMPSAC, 2006.

[14] Rosenberg. L, Hammer. T and Shaw. J, *"Software Metrics and Reliability"*, NASA, Software Assurance Technology Centre.

[15] Smidths. C, Stoddard. R.W, Stutzke. M, *"Software Reliability Models: An Approach to Early Reliability Prediction"*, IEEE Software, 1996.

[16] Tripathi. R and Mall. R, *"Early Stage Software Reliability and Design Assessment"*, Proceeding of the 12th Asia-Pacific Software Engineering Conference (APSEC, 2005).

[17] Cukic. B, *"The Virtues of Assessing Software Reliability Early"*, IEEE Software, May/June 2005.

[18] Cortellessa. V, Singh. H, Cukic. B, *"Early reliability assessment of UML based software models."*, IEEE Workshop on Software and Performance (WOSP, 2002).

Author's Biography

Anil Kumar Malviya is an Assistant Professor in Computer Science & Engineering Department at Amity University, Noida (U.P.). He received his B.Sc. and M.Sc. both in Computer Science from Banaras Hindu University, Varanasi respectively in 1991 and 1993 and Ph.D. degree in Computer Science from Dr. B.R. Ambedkar University, Agra in 2006. His research interests are Data Mining and Software Engineering including Software reliability, maintainability, Usability and Software Testing.