Journal of Basic and Applied Engineering Research
Print ISSN: 2350-0077; Online ISSN: 2350-0255; Volume 1, Number 7; October, 2014 pp. 55-60

http://www.krishisanskriti.org/jbaer.html

Assuring Reliability of the Software using Component Based Software Engineering

Rizwan Alam[1], Mohammad Ubaidullah Bokhari[2]

[1,2]*Department of Computer Science, Aligarh Muslim University, Aligarh*

Abstract: **Software reliability is the most measurable aspect of software quality. Unlike hardware, software does not age, wear out or rust, unreliability of software is mainly due to bugs or design faults in the software. Software reliability is dynamic & stochastic. Software reliability improvement is necessary & hard to achieve. It can be improved by sufficient understanding of software reliability, characteristics of software & sound software design.**

Software components as units of independent production, acquisition, and deployment that interact to form a functional system.Both the academic and commercial sectors have devoted considerable effort to defining and describing the terms and concepts involved in component-based software development.The component-based systems approach could potentially overcome difficulties associated with developing and maintaining monolithic software applications. The authors believe that this approach should result in better quality products, rapid development, and an in-creased capability to accommodate change.The authors identify a set of issues within an overall framework that software developers must address for component-based systems to achieve their full potential.

1. INTRODUCTION

Software reliability is defined as the probability of the failure free operation of a software system for a specified period of time in a specified environment.

Software application reliability is defined as follows [1]:

- "The probability of a given system performing its task adequately for a specified period of time under the expected operating conditions".
- "The probability that software will provide failure-free operation in a fixed environment for a fixed interval of time".

Software reliability differs considerably from program "correctness". A program is consistent with its specification, while reliability is related to the dynamic demands that are made upon the system and the ability to produce a satisfactory response to those demands.

The exact value of product reliability is never precisely known at any point in its lifetime. The study of software reliability can be categorized into three parts: Modeling, Measurement and improvement. Many Models exist, but no single model can capture a necessary amount of software characteristics. There is no single model that is universal to all the situations. Simulations can mimic key characteristics of the processes that create, validate & review documents & code. Software reliability measurement is naive. It can't be directly measured, so other related factors are measured to estimate software reliability.

1.1Software reliability and Hardware reliability:

Software reliability is not a direct function of time. Hardware parts may become old and wear out with time, but software will not change over time unless the software is changed or modified intentionally.

In Hardware reliability, in the first phase of the manufacturing, there may be a high number of faults. But after discovering and removing faults this number may decrease and gradually in the second phase (Useful life), there exists only a few number of faults. After this phase, there will be wear out phase in which, the physical component wear out due to the time and usage and the number of faults will again increase.

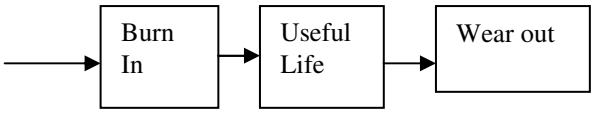

Fig1.Phases of hardware when considering reliability

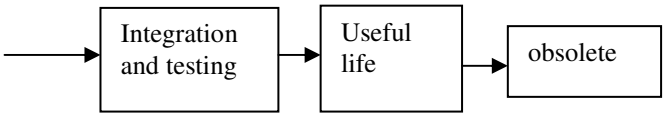

Fig2.Phases of software when considering reliability

But in software reliability, at the first phase, i.e while testing and integration there will be high number of faults, but after removing the faults, there exists only a few number of faults

and this process of removing the faults continues at a slower rate . Software products will not wear out with time and usage, but may become outmoded at a later stage.

1.2Software Metrics for Reliability:

The Metrics are used to improve the reliability of the system by identifying the areas of requirements (for specification), Coding (for errors), Testing (for verifying) phases.The different types of Software Metrics that are used are

a) Requirements Reliability Metrics:-

Requirements indicate what features the software must contain. So for this requirement document, a clear understanding between client and developer should exist. Otherwise it is critical to write these requirements .The requirements must contain valid structure to avoid the loss of valuable information.The requirements should be thorough and in a detailed manner so that it is easy for the design phase. The requirements should not contain inadequate information.

Next one is to communicate easily .There should not be any ambiguous data in the requirements. If there exists any ambiguous data, then it is difficult for the developer to implement that specification. Requirement Reliability metrics evaluates the above said quality factors of the requirement document.

b) Design and Code Reliability Metrics

The quality factors that exists in design and coding plan are complexity, size and modularity.If there exists more complex modules, then it is difficult to understand and there is a high probability of occurring errors. So complexity of the modules should be less.Next coming to size, it depends upon the factors such as total lines, comments, executable statements etc. According to SATC, the most effective evaluation is the combination of size and complexity.The reliability will decrease if modules have a combination of high complexity and large size or high complexity and small size. In the later combination also the reliability decreases because, the smaller size results in a short code which is difficult to alter.

c) Testing Reliability Metrics:

Testing Reliability metrics uses two approaches to evaluate the reliability.First, it ensures that the system is fully equipped with the functions that are specified in the requirements. Because of this, the errors due to the lack of functionality decreases.Second approach is nothing but evaluating the code, finding the errors and fixing them.

The current practices of software reliability measurement can be divided into four categories.

1) Product metrics

2) project management busy

3) process metrics

4) Fault and failure metrics

As discussed earlier software size and complexity plays an important role in design and coding phase. One of the product metrics called function point metric is used to estimate the size and complexity of the program.

Project Management metrics increases reliability by evaluating the Management process whereas process metrics can be used to estimate, monitor and improve the reliability and quality of the software.The final one, Fault and Failure Metrics determines, when the software is performing the whole functions that are specified by the requirement documents without any errors. It takes the faults and failures that arises in the coding and analyzes them to achieve this task.

2. RELIABILITY REQUIREMENTS

For any system, one of the first tasks of reliability engineering is to adequately specify the reliability and maintainability requirements derived from the overall availability needs and more importantly, from proper failure analysis or preliminary test results. Setting only availability targets is not appropriate. Reliability requirements address the system itself, including test and assessment requirements, and associated tasks and documentation. Reliability requirements are included in the appropriate system or subsystem requirements specifications, test plans and contract statements. Creation of proper lower level requirements is critical.Provision of only quantitative minimum targets (e.g. MTBF values/ Failure rates) is not sufficient for different reasons. One reason is that a full validation (related to correctness and verifiability in time) of an quantitative reliability allocation (requirement spec) on lower levels for complex systems can (often) not be made as a consequence of

1) The fact that the requirements are probabilistic.
2) The high level of uncertainties involved for showing compliance with all these probabalistic requirements.
3) Reliability is a function of time and accurate estimates of a (probabalistic) reliability number per item are available only very late in the project, sometimes even only many years after in-service use.

Compare this problem with the continues (re-)balancing of for example lower level system mass requirements in the development of an aircraft, which is already often a big undertaking. Notice that in this case masses do only differ in terms of only some %, are not a function of time the data is non-probabalistic and available already in CAD models. In case of reliability, the levels of unreliability (failure rates) may change with factors of decades (1000's of %)as result of very minor deviations in design, process or anything else.

The information is often not available without huge uncertainties within the development phase. This makes this allocation problem almost impossible to do in a useful, practical, valid manner, wich does not result in massive over- or under specification. A pragmatic approach is therefore needed. For example; the use of general levels / classes of quantitative requirements only depending on severity of failure effects. Also the validation of results is a far more subjective task than for any other type of requirement. (Quantitative) Reliability parameters -in terms of MTBF - are by far the most uncertain design parameters in any design.

Furthermore, reliability design requirements should drive a (system or part) design to incorporate features that prevent failures from occurring or limit consequences from failure in the first place! Not only to make some predictions, this could potentially distract the engineering effort to a kind of accounting work. The predicted failure probabilities of the elementary services are composed using different compositional structures to predict the reliability of the whole software system[2]. A design requirement should be so precise enough so that a designer can "design to" it and can also prove -through analysis or testing- that the requirement has been achieved, and if possible within some a stated confidence.

Any type of reliability requirement should be detailed and could be derived from failure analysis (Finite Element Stress and Fatigue analysis, Reliability Hazard Analysis, FTA, FMEA, Human Factor analysis, etc.) or other lower part or material level reliability tests, e.g. required overload loads (or stresses) and test time needed. To derive these requirements in an effective manner, a systems engineering based risk assessment and mitigation logic should be used. These practical design requirements shall be part of the output from functional or other failure analysis or tests. These requirements (often design constraints) are in this way derived from failure analysis or preliminary tests. Understanding of this difference with only pure quantitative requirement specification (e.g. Failure Rate / MTBF)is paramount in the development of successfull (complex) systems. For the reliability prediction, we can use the PCM Markov translator [3], which predicts a probability of failure on demand for the system of 0.0605 percent.

The maintainability requirements address the costs of repairs as well as repair time. Testability requirements provide the link between reliability and maintainability and should address detectability of failure modes (on a particular system level), isolation levels and the creation of diagnostics (procedures).

As indicated above, reliability engineers should also address requirements for various reliability tasks and documentation during system development, test, production, and operation. These requirements are generally specified in the contract statement of work and depend on how much leeway the customer wishes to provide to the contractor. Reliability tasks include various analyses, planning, and failure reporting.

Task selection depends on the criticality of the system as well as cost. A safety critical system may require a formal failure reporting and review process throughout development, whereas a non-critical system may rely on final test reports. The most common reliability program tasks are documented in reliability program standards, such as MIL-STD-785 and IEEE 1332. Failure reporting analysis and corrective action systems are a common approach for product/process reliability monitoring.

Several reliability issues and metrics proposed by researchers for CBS. Sharma et.al. [4] propose a link list based dependency representation and implements it by using Hash Map in Java.

There are three main models on which the reliability analysis approaches are based

- State based Models.
- Path based Models
- Additive Models.

3. COMPONENT BASED SOFTWARE ENGINEERING

Component-based software engineering (CBSE) or**Component-Based Development (CBD)**) is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems. This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software.

Software engineering practitioners regard components as part of the starting platform for service-orientation. Components play this role, for example, in web services, and more recently, in service-oriented architectures (SOA), whereby a component is converted by the web service into a service and subsequently inherits further characteristics beyond that of an ordinary component. Components can produce or consume events and can be used for event-driven architectures (EDA).A component model is a definition of standards for component implementation, documentation and deployment. Examples of component models are: Enterprise JavaBeans (EJB) model, Component Object Model (COM) model, .NET model and Common Object Request Broker Architecture (CORBA) component Model. The component model specifies how interfaces should be defined and the elements included in an interface definition.

CBS reliability greatly depends upon the interaction among components. Interaction promotes dependencies[5]. Higher dependency leads to a complex system, Hence reliability estimation will be difficult. Usually dependency is represented by an adjacency matrix. However, this representation can check only for the presence of dependencies between components and does not consider the type of interactions. Interaction types have a significant contribution to the complexity of system, hence the reliability. CBSE improves productivity, quality and reusability and reduce maintenance overheads and time to market. CBSE comprises of two separate but related processes namely component engineering and application engineering. The former is concerned with the analysis of domains and development of generic and domain-specific reusable components while the latter involves application development using commercial off-the-shelf components (COTS) or components that have been developed in-house.

3.1 Advantages of CBSE

(a)Functionality: Component-based systems are at a functional levelmuch more adaptable and extendable than traditional systems, because most of the new functionality can be reused some way or another or derived from already existing components.

(b) Reusability: In principle, CBD enables the development ofcomponents which completely implement a technical solution or a business aspect. Such components can be used everywhere.Reusability is an important characteristic of a high-quality software component. Programmers should design and implement software components in such a way that many different programs can reuse them. Furthermore, component-based usability testing should be considered when software components directly interact with users.It takes significant effort and awareness to write a software component that is effectively reusable. The component needs to be:

- fully documented
- thoroughly tested
- designed with an awareness that it will be put to unforeseen uses

(c) Maintainability: In a component-based system a piece of functionality ideally is implemented just once. It is self-evident this results in easier maintenance, which leads to lower cost, and a longer life for these systems.New applications will consist for a very large part of already existing components. Building a system will look more like assembly than really building.

3.2 Common Requirements for Component Based Reliability Models

(a) To Identify the Component

Standard software engineering concept of a component is the basic entity in the architecture based approach. Component can be independently designed, implemented, and tested. User can define the component which depends on the factors such that probability of getting required data.

(b) Software Architecture

Software architecture is the way of defining the software behavior with respect to the manner in which different software components interact with each other.

(c) Failure Behavior

Failure behavior is also associated with software architecture. Components failure behaviour can be expressed in terms of their reliabilities or failure rates.

(d) Combining the architecture with the failure behavior

There are three different approaches that are used to combine the software architecture with the failure behavior. These approaches are namely: state based approach, path based approach and additive approach.

4. RELIABILITY MODEL FOR COMPONENT BASED SYSTEMS

CBS reliability greatly depends upon the interaction among components. Interaction promotes dependencies. Higher dependency leads to a complex system, Hence reliability estimation will be difficult. Usually dependency is represented by an adjacency matrix. However, this representation can check only for the presence of dependencies between components and does not consider the type of interactions. Interaction types have a significant contribution to the complexity of system, hence the reliability[6].

CBSE improves productivity, quality and reusability and reduce maintenance overheads and time to market.The growing importance of software dictates that the software reliability is the major stumbling block in highly dependable computer system.

To support these techniques, testing models used to check software developed based on CBSE are needed to:

1. Explain the dependency of failure probability for software on its components.

2. Exploit reused software components of known reliability in estimating overall software system reliability.

There is a need of such type of models which is based upon the system architecture. Many reliability models based upon the system architecture have been proposed. These models are known as Architecture Based Reliability Model.Architecture

based software reliability techniques may be used for the following reasons:

(i) To develop a method that analyzes the application reliability built from the COTS software components.

(ii) To understand system reliability dependency on individual component reliabilities and their interconnection mechanism.

Swapna S. Gokhale [7] proposed some limitations for architecture based analysis technique. She classified the limitations into five categories namely modeling, analysis, parameter estimation, validation and optimization. Modeling limitations include concurrent execution, non markov transfer of control, non exponential sojourn time, and interface failures etc. Analysis limitation includes reliability estimation, sensitivity and interface analysis, uncertainty quantification etc.

There are so many models for reliability analysis that can be incorporated with Component Based Development. A sensitivity analysis is also performed to know the effect of each node on the system reliability[8]. For example, Chao-Jung Hsu's Adaptive Reliability Analysis Using Path Testing is an adaptive approach for testing path into reliability estimation for complex component based systems. For path reliability estimation three methods have been proposed namely sequence, branch and loop structures.A promising estimation of software reliability can be given by this approach when testing information is available.In addition, rules as presented in [9, 10, 11] could be integrated in specialized components.

5. LIMITATIONS AND FUTURE WORK

Besides inheriting all limitations of the underlying quality prediction techniques for our approach exhibits the following limitations:

a) **No guaranteed optimality**: The approach itself is a best effort approach and does not guarantee to find the real Pareto-front, i.e. the globally optimal solutions, because metaheuristics are used.

b) **Questionable efficiency**: As the evaluation of each candidate solution, mainly due to the performanceevaluation, takes several seconds, the overall approach is considerably time consuming. Here, software architects should run it in parallel to other activities orovernight. A distribution of the analyses on a cluster of workstations could lead to significant improvements. It could also be possible to split the optimization problem into several independent parts that are solved separately and thus quicker. Problem-specific heuristics allowing faster convergence and thus requiring less evaluations are a crucial extension.

c) **No regard for uncertainties**: For the results, uncertainty of estimations, uncertainty of the workload, and the resulting risks are not taken into account. Here, sensitivity metrics could be an additional quality criterion.

This paper estimates the reliability factor issues of data for different metrics of software process model under Component Based Software Engineering..An important aspect of future work is to combine our approach with subordinate heuristics to make use of performance domain knowledge to assure the reliability of each component of the software eventually leading to the reliability of the whole software. For example, heuristics to improve allocation based on the resource demands of components and utilization.

6. CONCLUSION

Reliability is an attribute of quality and software quality can be measured .So reliability depends on high software quality. So at each development phase, some quality attributes are applied and the reliability and quality of the software can be improved by applying software metrics at each of these development phases. This metrics measures software reliability in Requirements, Design and coding, and testing phases.

We considered some criteria on basis of which we examined the available approaches as scope, model, technique, method and critique. Most of the proposed approaches are mathematical and based upon the operational profile.To calculate the overall application reliability existing work take two important considerations one is reliability of individual component and another is operational profiles of the system.

REFERENCES

[1] ANSI/IEEE, 1991 *Standard Glossary of Software Engineering Terminology*, STD-729-1991

[2] Franz Brosch, " *Reliability Prediction for Fault-Tolerant Software Architectures*", QoSA+ISARCS'11, June 20–24, 2011, Boulder, Colorado, USA. ACM 978-1-4503-0724-6/11/06 (pp 75-84)

[3] F. Brosch and B. Zimmerova.*Design-Time Reliability Prediction for Software Systems.In Proceedings of the International Workshop on Software Quality and Maintainability* (SQM'09), pages 70{74, March 2009.

[4] Arun Sharma, P.S.Grover, Rajesh Kumar, 2009, *Dependency Analysis for Component Based Software Systems*, ACMSIGSOFT Software Engineering Notes Vol. 34, No 4 pp 1-6.

[5] S. Becker, H. Koziolek, and R. Reussner.aThe*Palladio component model for model-driven performance prediction.* J. of Systems and Software, 82:3{22, 2009.

[6] H. Koziolek and F. Brosch.*Parameter dependencies for component reliability specifications. In Proc. of Workshop on Formal Engineering approaches to Software Components and Architectures*. Elsevier, 2009.

[7] S. Gokhaleet al.2007, *Architecture Based Software Reliability Analysis: Overview and Limitations.*IEEE Transactions on Dependable and Secure Computing.pp 32-40.

[8] Chao-Jung Hsu and Chin-Yu Huang, 2011, *An Adaptive Reliability Analysis Using Path Testing for Complex Component based Software Systems*, IEEE transaction on reliability Vol. 60, No 1 pp 158-170.

[9] J. Xu. *Rule-based automatic software performance diagnosis and improvement*.In Proc. of WOSP'08, pages 1{12, New York, NY, USA, 2008.ACM.

[10] J. Xu. *Rule-based automatic software performance diagnosis and improvement.*In Proc. of WOSP'08, pages 1{12, New York, NY, USA, 2008.ACM.

[11] V. Cortellessa and L. Frittella.*A framework for automated generation of architectural feedback from software performance analysis.* In K. Wolter, editor, Proc. of Fourth European Performance Engineering Workshop, volume 4748 of Lecture Notes in Computer Science, pages 171{185. Springer, 2007.

[12] J. D. McGregor, F. Bachmann, L. Bass, P. Bianco, and M. Klein. *Using arche in the classroom: One experience. Technical Report* CMU/SEI-2007-TN-001, Software Engineering Institute, Carnegie Mellon University, 2007.