Improving Software Reliability using Software Engineering Approach- A Review

Aasia Quyoum
Research Scholar
University of Kashmir (India)

Mehraj – Ud - Din Dar
Director, IT & SS
University of Kashmir (India)

S. M. K. Quadri
Director Computer Sciences
University of Kashmir (India)

ABSTRACT
Software Reliability is an important facet of software quality. Software reliability is the probability of the failure free operation of a computer program for a specified period of time in a specified environment. Software Reliability is dynamic and stochastic. It differs from the hardware reliability in that it reflects design perfection, rather than manufacturing perfection. This article provides an overview of Software Reliability which can be categorized into: modeling, measurement and improvement, and then examines different modeling technique and metrics for software reliability, however, there is no single model that is universal to all the situations. The article will also provide an overview of improving software reliability and then provides various ways to improve software reliability in the life cycle of software development.

Keywords
Reliability, Modeling, Simulation, Software, Engineering.

1. INTRODUCTION
With the advent in the computer era, computes are playing very important role in our daily lives. Dish washers, TV's, Microwave Ovens, AC's are having their analog and mechanical parts replaced by digital devices, CPU's and software's. Increasing competition and high development costs have intensified the pressure to quantify software quality and to measure and control the level of quality delivered. There are various software quality factors as defined by MC Call and ISO 9126 standard, however, Software Reliability is the most important and most measurable aspect of software quality. This paper tries to give general idea for software reliability and the metrics and models used for that. This will also focus on using software engineering principles in the software development and maintenance so that reliability of software will be improved.

2. RELIABILITY
Software Reliability is defined as the probability of the failure free software operation for a specified period of time in a specified environment [ANSI91] [Lyu95].

Unreliability of any product comes due to the failures or presence of faults in the system. As software does not 'wear-out" or "age", as a mechanical or an electronic system does, the unreliability of software is primarily due to bugs or design faults in the software.

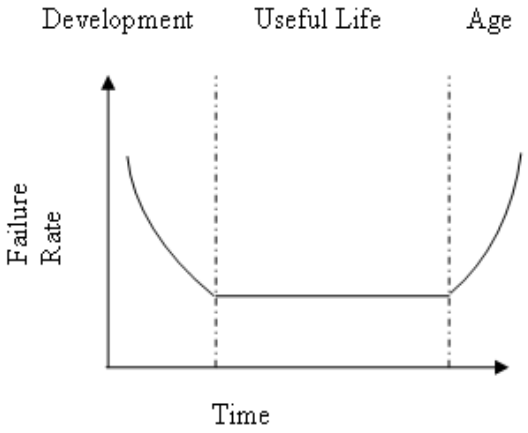
Reliability is a probabilistic measure that assumes that the occurrence of failure of software is a random phenomenon. Randomness means that the failure can't be predicted accurately. The randomness of the failure occurrence is necessary for reliability modeling. In [MIO87], it is suggested that reliability modeling should be applied to systems larger than 5000 LOC.

3. RELIABILITY PROCESS
The reliability process in generic terms is a model of the reliability-oriented aspects of software development, operations and maintenance. The set of life cycle activities and artifacts, together with their attributes and interrelationships that are related to reliability comprise the reliability process. The artifacts of the software life cycle include documents, reports, manuals, plans, code configuration data and test data. Software reliability is dynamic and stochastic. In a new or upgraded product, it begins at a low figure with respect to its new intended usage and ultimately reaches a figure near unity in maturity. The exact value of product reliability however is never precisely known at any point in its lifetime.

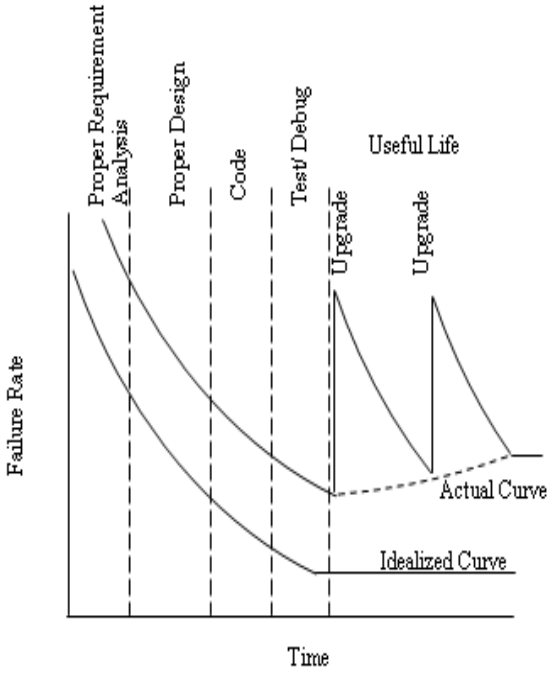
4. SOFTWARE RELIABILITY CURVE
Software errors have caused human fatalities. The cause has ranged from poorly designed user interface to direct programming errors. Software will not change over time unless intentially changed or upgraded. Software does not rust, age, wear-out, or deform. Unlike mechanical parts, software will stay as is unless there are problems in design or in hardware. Software failures may be due to errors, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of software or other unforeseen problems [Keller91].

Over time, hardware exhibits the failure characteristics as shown in Figure 1. Known as bathtub curve.

Software is not susceptible to the environmental maladies that cause hardware to wear out; therefore, the failure rate curve for software should take the form of the "idealized curve" as shown in Figure 2. Undiscovered defects will cause high failure rates early in the life of a program. Once these are corrected (possibly without introducing other errors) the curve flattens. In the useful life phase, software will experience a drastic increase in failure rate each time an upgrade is made. The failure rate levels off gradually, partly because of the defects found and fixed after the upgrade.

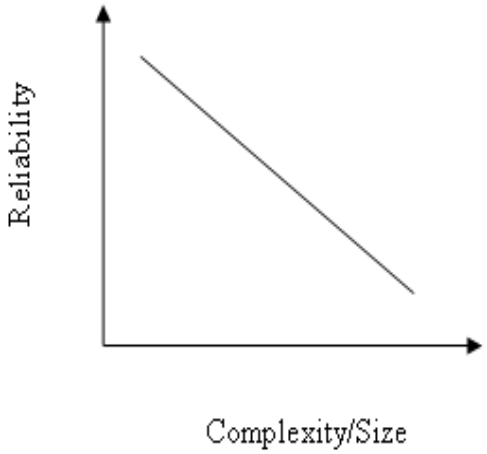


"Figure 1" Bathtub curve for hardware Reliability



"Figure 2" Software Reliability curve

In Figure 2 Considering the "actual curve|", during the software's life, software will undergo feature upgrades. For feature upgrades the complexity of software is likely to be increased, since functionality of software is enhanced, causing the failure rate curve to spike as shown in Figure 2. Before the curve return to the original steady state failure rate, another upgrade is requested causing the curve to spike again. Slowly, the minimum failure rate level begins to rise-the software is deteriorating due to upgrade in feature. Since the reliability of software keep on getting decreased with increase in software complexity, a possible curve is shown in Figure 3.



"Figure 3"

5. SOFTWARE RELIABILITY ACTIVITIES

The reliability process in generic terms is a model of the reliability- oriented aspects of software development, operations, and maintenance. Quantities of interest in a project reliability profile include artifacts, errors, defects, corrections, faults, tests, failures, outages, repairs, validation, and expenditures of resources, such as CPU time, manpower effort and schedule time. The activities relating to reliability are grouped into classes:

Construction

Generates new documentation and code artifacts

Combination

Integrates reusable documentation and code components with new documentation and code components

Correction

Analyzes and removes defects in documentation and code using static analysis of artifacts.

Preparation

Generates test plans and test cases, and readies them for execution.

Testing

Executes test cases, whereupon failure occur

Identification

Makes fault category assignment. Each fault may be new or previously encountered.

Repair

Removes faults and possibly introduces new faults.

Validation

Performs inspections and checks to affirm that repairs are effective

Retest

Executes test cases to verify whether specified repairs are complete if not, the defective repair is marked for repair. New test cases may be needed.

6. SOFTWARE RELIABILITY METRICS

Software Reliability Measurement is not an exact science. Though frustrating, the quest of quantifying software reliability has never ceased. Until now, we still have no good way of measuring software reliability.

Measuring software reliability remains a difficult problem because we don't have a good understanding of the nature of software. There is no clear definition to what aspects are related to software reliability. We cannot find a suitable way to measure software reliability, and most of the aspects related to software reliability.

It is tempting to measure something related to reliability to reflect the characteristics, if we can not measure reliability directly. The current practices of software reliability measurement can be divided into four categories: [RAC96].

6.1 Product metrics

Software size is thought to be reflective of complexity, development effort and reliability. Lines of Code (LOC), or LOC in thousands (KLOC), is an intuitive initial approach to measuring software size. But there is not a standard way of counting. Typically, source code is used (SLOC, KSLOC) and comments and other non-executable statements are not counted. This method cannot faithfully compare software not written in the same language. The advent of new technologies of code reuses and code generation technique also cast doubt on this simple method.

Function point metric is a method of measuring the functionality of a proposed software development based upon a count of inputs, outputs, master files, inquires, and interfaces. The method can be used to estimate the size of a software system as soon as these functions can be identified. It is a measure of the functional complexity of the program. It measures the functionality delivered to the user and is independent of the programming language. It is used primarily for business systems; it is not proven in scientific or real-time applications.

Complexity is directly related to software reliability, so representing complexity is important. Complexity-oriented metrics is a method of determining the complexity of a program's control structure, by simplifying the code into a graphical representation. Representative metric is McCabe's Complexity Metric.

Test coverage metrics are a way of estimating fault and reliability by performing tests on software products, based on the assumption that software reliability is a function of the portion of software that has been successfully verified or tested.

6.2 Project management metrics

Researchers have realized that good management can result in better products. Research has demonstrated that a relationship exists between the development process and the ability to complete projects on time and within the desired quality objectives. Costs increase when developers use inadequate processes. Higher reliability can be achieved by using better development process, risk management process, configuration management process, etc.

6.3 Process metrics

Based on the assumption that the quality of the product is a direct function of the process, process metrics can be used to estimate, monitor and improve the reliability and quality of software. ISO-9000 certification, or "quality management standards", is the generic reference for a family of standards developed by the International Standards Organization (ISO).

6.4 Fault and failure metrics

The goal of collecting fault and failure metrics is to be able to determine when the software is approaching failure-free execution. Minimally, both the number of faults found during testing (i.e., before delivery) and the failures (or other problems) reported by users after delivery are collected, summarized and analyzed to achieve this goal. Test strategy is highly relative to the effectiveness of fault metrics, because if the testing scenario does not cover the full functionality of the software, the software may pass all tests and yet be prone to failure once delivered. Usually, failure metrics are based upon customer information regarding failures found after release of the software. The failure data collected is therefore used to calculate failure density, Mean Time between Failures (MTBF) or other parameters to measure or predict software reliability.

Besides the above metrics, other possible metrics are:

6.5 Efficiency

The amount of computing time and resources required by software to perform desired function it is an important factor in differentiating high quality software from a low one.

6.6 Integrity

The extent to which access to software or data by unauthorized persons can be controlled Integrity has become important in the age of hackers.

6.7 Flexibility

The effort required to transfer the program from one hardware to another.

6.8 Interoperability

The effort required to couple one system to another as indicated by the following sub-features: adaptability, insatiability, conformance, replacebility.

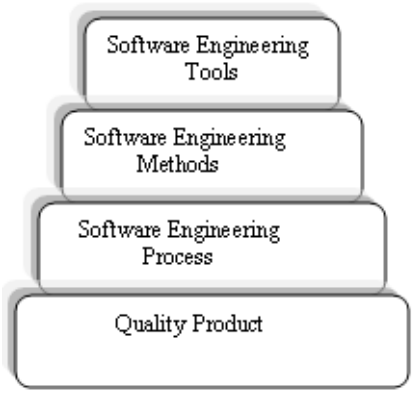
6.9 Maintainability

It is the ease with which repair may be made to the software as indicated by the following sub-feature: analyzability, changeability, stability, testability. If a software needs" less mean time to change (MTTC), it means it needs less maintainability.

7. SOFTWARE RELIABILITY IMPROVEMENT TECHNIQUES

Good engineering methods can largely improve software reliability. In real situations, it is not possible to eliminate all the bugs in the software; however, by applying sound software engineering principles software reliability can be improved to a great extent.

The application of systematic, disciplined, quantifiable approach to the development operation and maintenance of software will produce economically software that is reliable and works efficiently on real machines [IEEE93].Figure 4. shows Software Engineering being the layered technology focuses on the quality and reliability of software.



"Figure 4" Engineering approach to high quality software development

7.1 Process

Process defines a framework [PAU93] that must be established for effective delivery of software engineering technology. It forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products (models, documents, data, reports, forms etc) are produces, milestones are established, quality is ensured, and change is properly managed.

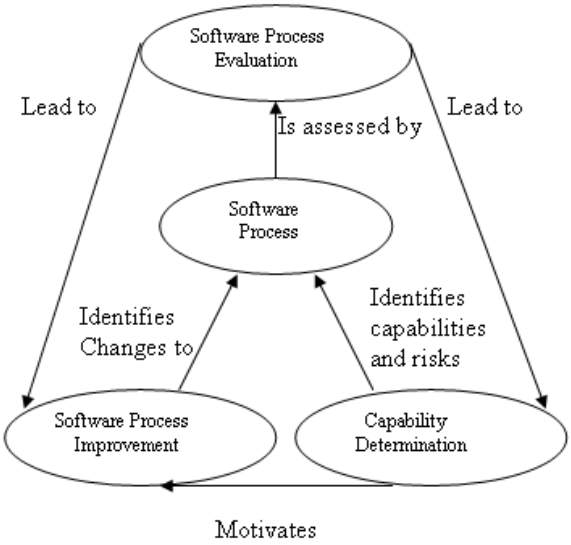
The process itself should be assessed to ensure that it meets the basic process criteria that are necessary for successful software engineering. The possible relationship between the software process and the methods applied for evaluation and improvement is shown in Figure 5.

7.2 Software Engineering Methods

Software engineering methods provide technical "how to's" for building software. These methods consist of a broad array of tasks that include requirement analysis, design modeling, program construction, testing and support.

"Table 1"

Omission	Incorrect Fact	Inconsistency	Ambiguity
26%	10%	38%	26%



"Figure 5"

7.2.1 Requirement Analysis

In the early days of software development, emphasis was on coding and testing but researchers have shown that requirement analysis is the most difficult and intractable activity and is very error prone .In this phase software failure rate and hence the reliability can be increased by:

- **a)** Properly identifying the requirements.
- **b)** Specifying the requirements in the form of software requirement specification (SRS) document. The basic goal of SRS is to describe the complete external behavior of proposed system [Dav93].
- **c)** Requirement reviews (Validating the SRS.)
- **d)** Developing the prototypes.
- **e)** Performing structured analysis for developing conceptual models using data flow diagrams (DFDs).
- **f)** Make estimations of effort, cost and task duration.
- **g)** Performing the Risk management which involves risk management and control.

Some projects have collected data about requirement errors. In [Dav89] the effectiveness of different methods and tools in detecting requirement errors in specifications for a data processing application is reported in Table 1. On an average, a

total of more than 250 errors were detected, and the percentage of different types of errors was:

7.2.2 Modeling Design

Design activity is the first step in moving from problem domain to solution domain. The goal of the design is to produce the model of the system which can be later used to build up the system. In this phase reliability can be improved by:

a) Using "Divide and conquer" principle that is dividing the system into smaller pieces (modules) so that each piece can be conquered separately.
b) Abstraction of components so that maintenance will become easy.
c) Performing different levels of factoring.
d) Controlling and understanding the interdependency among the modules.
e) Design Reviews to ensure that design satisfies the requirements and is of good quality.
f) Reducing the coupling between modules and increasing cohesion within a module.
g) Developing design iteratively.

7.2.3 Program Construction

It includes coding and some testing tasks. In this phase software reliability can be increased by:

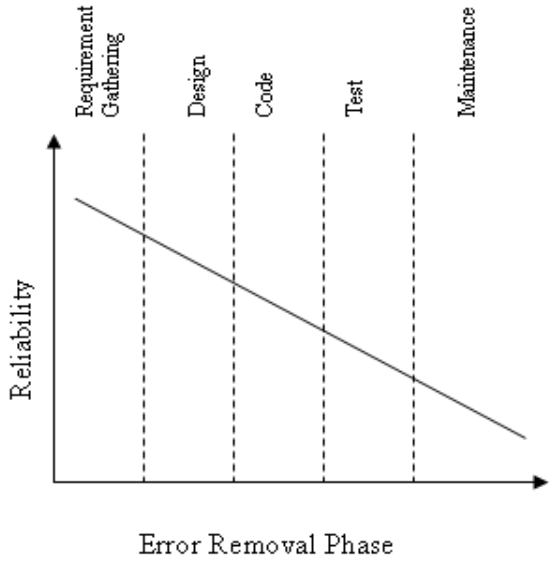
a) Constraining algorithms by following structured programming [BOH00] practice.
b) Write self-documenting code.
c) Creating interfaces that are consistent with architecture,
d) Conducting a code walkthrough.
e) Performing unit tests.
f) Refactoring code.

7.2.4 Testing

After the code construction of software products, testing, verification and validation are necessary steps. Software testing is heavily used to trigger, locate and remove software defects. Software testing is still in its infant stage; testing is crafted to suit specific needs in various software development projects in an ad-hoc manner. Various analysis tools such as trend analysis, fault-tree analysis, Orthogonal Defect classification and formal methods, etc, can also be used to minimize the possibility of defect occurrence after release and therefore improve software reliability.

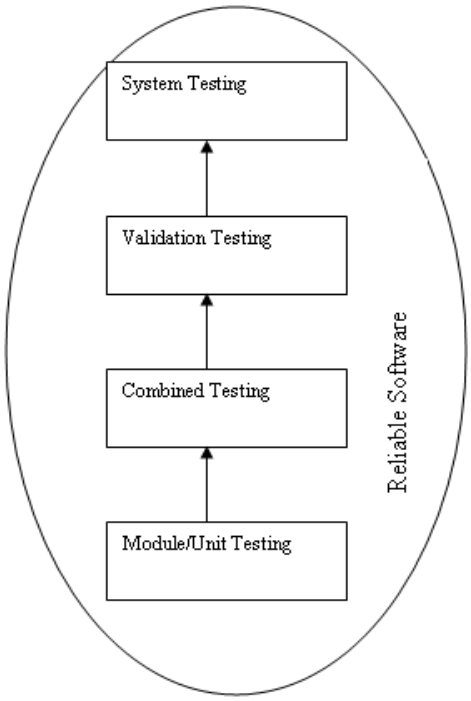
A strategy for testing may be viewed as shown in Figure 6.It starts with testing the individual modules and then progresses by moving upward to integration testing where the modules are collectively tested for errors. In validation testing customer requirements are validated against the software that has been developed. Finally in system testing, the entire software is tested as a single unit. Once the above testing strategy will be followed for software testing, software reliability can be highly improved.

Figure 7 shows the effect of identifying and removing errors in the early phases of software development, on the software reliability.



"Figure 7"

After deployment of the software product, field data can be gathered and analyzed to study the behavior of software defects. Fault tolerance or fault/failure forecasting techniques will be helpful techniques and guide rules to minimize fault occurrence or impact of the fault on the system.



"Figure 6" Testing Strategy

7.3 Software Engineering Tools

Software engineering provides a collection of tools that helps in every step of building a product and is termed as CASE (Computer Aided Software Engineering) tools. Case provides the software engineer with the ability to automate manual activities and assist in analysis, design, coding and test work. This leads to high quality and high reliable software

8. SOFTWARE RELIABILITY MODELING

To study a system, it is possible to experiment with the system itself or with the model of the system, but experimenting with the system itself is very expensive and risky. The objective of many system studies, however is to predict how a system will perform before it is built. Consequently, system studies are generally conducted with a model of a system. A model is not only a substitute of a system; it is also a simplification of the system.

A number of software reliability models have emerged as people try to understand the attributes of how and why software fails, and try to quantify software reliability. Over 200 models have been proposed since 1970s, but how to quantify software reliability still remains unsolved. There is no single model that can be used in all the situations. No model is complete; one model may work well for a set of certain software, but may be completely off track for other kinds of problems.

Most existing analytical methods to obtain reliability measures for software systems are based on the Markovian models and they rely on the assumption on exponential failure time distribution. The Markovian models are subject to the problem of intractably large state space. Methods have been proposed to model reliability growth of components which can not be accounted for by the conventional analytical methods but they are also facing the state space explosion problem. A simulation model, on the other and offers an attractive alternative to analytical models as it describes a system being characterized in terms of its artifacts, events, interrelationships and interactions in such a way that one may perform experiments on the model, rather than on the system itself, ideally with indistinguishable results.

9. RELIABILITY SIMULATION

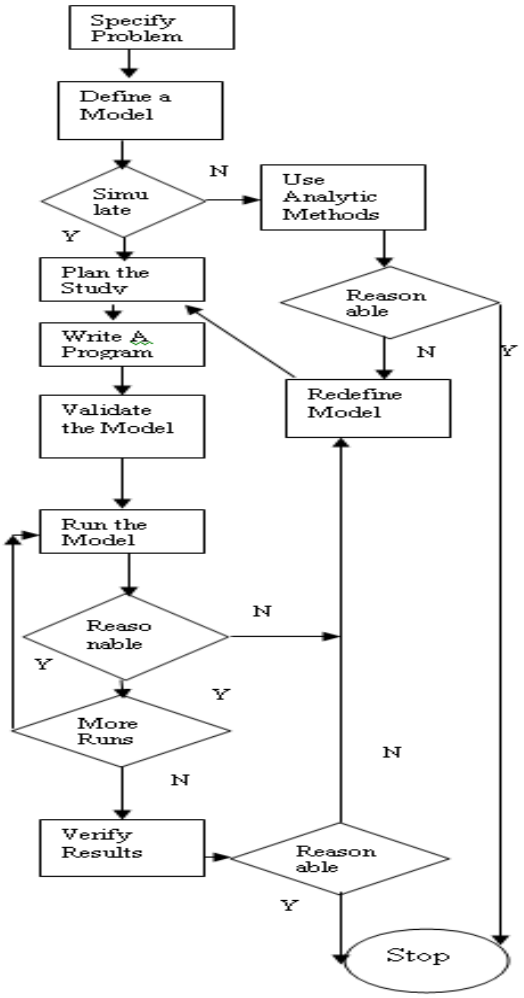
Simulation refers to the technique of imitating the character of an object or process in a way that permit us to make quantified inferences about the real object or process. In the area of software reliability, simulation can mimic key characteristics of the processes that create, validate, and revise documents and code.

Reliability modeling ultimately requires good data. But software projects do not always collect data sets that are comprehensive, complete, or consistent enough for effective modeling research or model application. Further, data required for software reliability modeling in general seem to be even more difficult to collect than other software engineering data.

10. SIMULATION PROCESS

Since good data sets are so scarce, one purpose of simulation is to supply carefully controlled, homogeneous data or software artifacts having known characteristics for use in evaluating the various assumptions upon which existing reliability models have been built. Since actual software artifacts(such as faults in computer programs) and processes(such as failure and fault removal) often violate the assumptions of analytic software reliability models, simulation can perhaps provide a better understanding of such assumptions and may even lead to a better explanation of why some analytic models work well in spite of such violations.

Some of the steps involved in the process of simulation study are illustrated by the flowchart of Figure 6.



"Figure 6" The process of simulating

An initial step is to describe the problem to be solved in a concise manner. Based on this problem definition, a model must be defined. It is at this point that it becomes apparent whether the model can be kept in a form that allows analytical techniques to be used. When it is decided to simulate, the experimental nature of the simulation technique makes it essential to plan the study by deciding upon the major parameters to be varied, the number of cases to be conducted and the order in which runs are to be made. Given that the simulation is to be on the digital computer, a program must be written.

Once the model is decided, we need to verify the model and then executing a series of runs according to the study plan. As results are obtained, it is likely that there will be many changes in the

model and the study plan. The early runs may make parameter significance clear and so lead to the reassessment of the model. Verification of results is important after each run. Sometimes it is useful to repeat runs so that parts of model have different random numbers on each run.

11. CONCLUSION

Computers are playing very important role in our day-to-day life and there is always a need of high quality software. 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. 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 & 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. Software reliability improvement is necessary & hard to achieve. It can be improved by sufficient understanding of software reliability, characteristics of software & sound software design. Complete testing of the software is not possible; however sufficient testing & proper maintenance will improve software reliability to great extent.

12. REFERENCES

[1] C. T. Lin, C. Y. Huang, and C. C. Sue, "Measuring and Assessing Software Reliability Growth Through Simulation-Based Approaches," Proceedings of the 31st IEEE Annual International Computer Software and Applications Conference (COMPSAC 2007), pp. 439-446, Beijing, China, July 2007.

[2] J. Lo, S. Kuo, M.R. Lyu, and C. Huang, "Optimal Resource Allocation and Reliability Analysis for Component-Based Software Applications," *Proc. 26th Ann. Int'l Computer Software and Applications Conf. (COMPSAC)*, pp. 7-12, Aug. 2002.

[3] John D. Musa, "Operational Profiles in Software-Reliability Engineering," IEEE Software, v.10 n.2, p.14-32, March 1993

[4] Kishor S. Trivedi, "Probability and statistics with reliability, queuing and computer science applications," John Wiley and Sons Ltd., Chichester, UK, 2001

[5] K. Kanoun M. Kaaniche C. Beounes J.C. Laprie and J. Arlat, "Reliability Growth of Fault-Tolerant Software," IEEE Trans. Reliability, vol. 42, no. 2, pp. 205-219, June 1993.

[6] Kumar, M., Ahmad, N., Quadri, S.M.K. (2005), "Software reliability growth models and data analysis with a Pareto test-effort", RAU Journal of Research, Vol.15, No. 1-2, pp 124-8

[7] Norman F. Schneidewind, "Fault Correction Profiles, "Proceedings of the 14th International Symposium on Software Reliability Engineering, p.257, November 17-21, 2003

[8] Quadri, S.M.K., Ahmad, N., Peer, M.A. (2008), "Software optimal release policy and reliability growth modeling", Proceedings of 2nd National Conference on Computing for Nation Development, INDIACom-2008, New Delhi, India, pp 423-31

[9] R.C.Tausworthe, "A General Software Relibility Process Simulation Technique,"NASA JPL Publication, 91-7, April 1991.

[10] R.C.Tausworthe and M.R, Lyu, "A generalized technique for simulating software relibility,"IEEE Software,"Vol.13, No.2, pp.77-88, March 1996.

[11] Robert C. Tausworthe , Michael R. Lyu, "A Generalized Technique for Simulating Software Reliability, "IEEE Software, v.13 n.2, p.77-88, March 1996

[12] S.Gokhale, M. R.Lyu, and K. S. Trividi," Reliability Simulation of Component-Based Software Systems,"Proceedings of the 19th International Symposium on Software Reliability Engineering, pp. 192-201, Paderborn, Germany, November 1998.

[13] S.Gokhale, Michael R. Lyu, and K.S. Trivedi." Reliability Simulation of Fault-Tolerant Software and Systems". In Proc. of Pacific Rim International Symposium on Fault-Tolerant Systems (PRFTS '97), pp. 197-173, Taipei, Taiwan, December 1997.

[14] S. Krishnamurthy , A. P. Mathur, On The" Estimation Of Reliability Of A Software System Using Reliabilities Of Its Components," Proceedings of the Eighth International Symposium on Software Reliability Engineering (ISSRE '97), p.146, November 02-05, 1997

[15] Swapna S. Gokhale , Kishor S. Trivedi, "A time/structure based software reliability model," Annals of Software Engineering, v.8 n.1-4, p.85-121, 1999

[16] Swapna S. Gokhale , Kishor S. Trivedi , Michael R. Lyu, "Reliability Simulation of Fault-Tolerant Software and Systems," Proceedings of the 1997 Pacific Rim International Symposium on Fault-Tolerant Systems, p.167, December 15-16, 1997

[17] S. Y. Kuo, C. Y. Huang, and M. R. Lyu, "Framework for Modeling Software Reliability, Using Various Testing-Efforts and Fault-Detection Rates," IEEE Transactions on Reliability," Vol. 50, No. 3, pp. 310-320, Sept. 2001.

[18] Tausworthe, Robert C., "A General Software Reliability Process Simulation Technique," Technical Report 91-7, Jet Propulsion Laboratory,Psaadena, CA, March 1991.

[19] Wood, A. (1996), "Predicting software reliability", IEEE Computers, Vol.11, pp 69-77 Von Mayrhauser, A.,Malaiya, Y.K.,Keables, J.,and Srimani, P. K., "On the Need for Simulation for better Characterization of Software Reliability, "International Symposium on Software Reliability Engineering," Denver, CO, 1993.