A BRIEF STUDY ON SOFTWARE RELIABILITY

Anil Kumar Mishra Prof. Einstein Academy of Technology and Management, Bhubaneswar, India
Manaswini Pattanayak Asst. Prof. Einstein Academy of Technology and Management, Bhubaneswar, India
Ratiranjan Swain Student, Einstein Academy of Technology and Management, Bhubaneswar, India

Software's increasing role creates both requirements for being able to trust it more than before, and for more people to know how much they can trust their software. A sound engineering approach requires both techniques for producing reliability and sound assessment of the achieved results. Different parts of industry and society face different challenges: the need for education and cultural changes in some areas, the adaptation of known scientific results to practical use in others, and in others still the need to confront inherently hard problems of prediction and decision-making, both to clarify the limits of current understanding and to push them back. We outline the specific difficulties in applying a sound engineering approach to software reliability engineering, some of the current trends and problems and a set of issues that we therefore see as important in an agenda for research in software dependability.

KEY WORDS: Reliability engineering and assessment, COTS reliability, diversity

INTRODUCTION:

We use "dependability" informally to designate those system properties that allows us to rely on a system functioning as required. Dependability encompasses, among other attributes, reliability, safety, security, and availability. These qualities are the ared concern of many sub-disciplines in software engineering, of specialized fields like computer security, and of reliability and safety engineering. In this area, an important factor is the diversity of "the software industry", or, rather, among the many industrial sectors that produce or use software. The demand for software dependability varies widely between industrial sectors, as does the degree of adoption of systematic approaches to it. From many viewpoints, two extremes of the range are found in mass- marketed PC software and in safety-critical software for heavily- regulated industries. A couple of decades ago there was a revolution in dependability of consumer goods such as TV sets, VCRs and automobiles, when companies realized that there was market advantage to be gained by demonstrating higher reliability than their competitors. There has not yet been a similar movement in the corresponding sectors of the software industry.

RELIABILITY:

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

Overview of Software Reliability Prediction Models

These models are derived from actual historical data from real software projects. The user answers a list of questions which calibrate the historical data to yield a software reliability prediction. The accuracy of the prediction depends on how many parameters (questions) and datasets are in the model, how current the data is, and how confident the user is of their inputs. One of the earliest prediction models was the Rome Laboratory TR-92-52. It was developed in 1987 and last updated in 1992 and was geared towards software in avionics systems. Due to the age of the model and data it's no longer recommended but is the basis for several modern models such as the Shortcut model, Full-scale model, and Neufelder assessment model. There are also lookup tables for software defect density based on the capability maturity or the application type. These are very simple models but are generally not as accurate as the assessment based models.

Model	**Number of inputs**	**Industry support ed**	**Effort required to use the model**	**Relative accuracy**	**Year developed/ Last updated**
Industry tables	1	Several	Quick	Varies	1992, 2015
CMMI® tables	1	Any	Quick	Low at low CMMi®	1997, 2012
Shortcut model	23	Any	Moderate	Medium	1993, 2012
Full-scale model	94-299	Any	Detailed	Medium-High	1993, 2012
Metric based models	Varies	Any	Varies	Varies	NA
Historical data	A minimum of 2	Any	Detailed	High	NA
Rayleigh model	3	Any	Moderate	Medium	NA
RADC TR-92-52	43-222	Aircraft	Detailed	Obsolete	1978, 1992
Neufelder model	156	Any	Detailed	Medium to high	2015

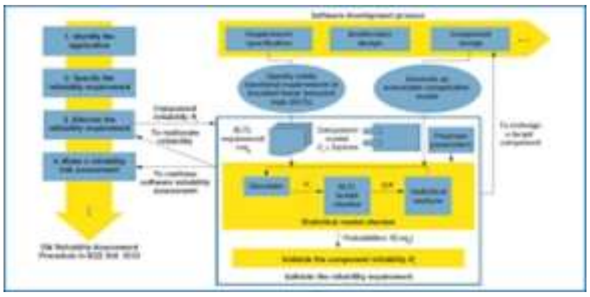
Overview of Software Reliability Growth (Estimation) Models:

Model name	**Inherent defect count**	**Effort required**	**Requires exact time between failures**
Increasing fault rate			
Weibull	Finite/not fixed	High	Yes
Peak			
Shooman Constant Defect Removal Rate Model	Finite/fixed	Low	Yes
Decreasing fault rate			
Shooman Constant Defect Removal Rate Model	Finite/fixed	Low	Yes
Linearly Decreasing			
eneral exponential models including: · Goel-Okumoto (exponential) · Musa Basic Model · Jelinski-Moranda	Finite/fixed	Medium	Yes
Shooman Linearly Decreasing Model	Finite/fixed	Low	Yes
Duane	Infinite	Medium	No

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

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 occurs
- 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.

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. It is tempting to measure something related to reliability to reflect the characteristics, if we cannot measure reliability directly. The current practices of software reliability measurement can be divided into four categories: Product metrics: Software size is thought to be reflective of complexity, development effort and reliability. Lines of Code, or LOC in thousands, is an intuitive initial approach to measuring software size. But there is not a standard way of counting. Typically, source code is used and comments and other non-executable statements are not counted. This method cannot faithfully compare software not written in the same language. 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.

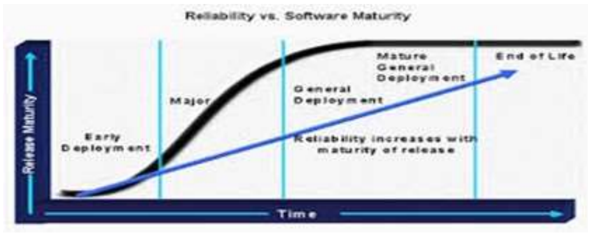
Non-Linearly Decreasing			
Musa-Okumoto (logarithmic)	Infinite	Low	Yes
Shooman Exponentially Decreasing Model	Finite/fixed	High	Yes
Log-logistic	Finite/fixed	High	Yes
Geometric	Infinite	High	No

Increasing and then decreasing			
Yamada (Delayed) S-shaped	Infinite	High	Yes
Weibull	Finite/not fixed	High	

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.

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 ISO.

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 and the failures 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 or other parameters to measure or predict software reliability.



Besides the above metrics, other possible metrics are: 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.

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.

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.

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, it means it needs less maintainability

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.

REFERENCES

1. S. Brocklehurst, B. Littlewood. New ways to get accurate reliability measures. IEEE Software 9, 4 (July 1992), 34-42.
2. R. C. Cheung. A User-Oriented Software Reliability Model. IEEE Transactions on Software Engineering SE-6, 2 (March 1980), 118-125.
3. G. F. Clement, P. K. Giloth. Evolution of Fault Tolerant Switching Systems in AT&T. In A. Avizienis, H. Kopetz and J.-C. Laprie (Eds.) The Evolution of FaultTolerant Computing, Springer-Verlag, 1987, 37-54.
4. FAA. Federal Aviation Administration, Advisory Circular AC 25.1309-1A, 1985.
5. J.-C. Fabre et al., Assessment of COTS microkernels by fault injection, in Proc. DCCA-7 (San Jose, California, USA, January 1999), 25-44.
6. N. Fenton, M. Neil. Software Metrics: a roadmap", in this volume.
7. N. Fenton, S. L. Pfleeger, R. Glass. Science and Substance: A Challenge to Software Engineers. 1EEE Software 11, 4 (July 1994), 86-95.
8. N. E. Fenton, M. Neil. A Critique of Software Defect Prediction Models. IEEE Transactions on Software Engineering 25, 5 (September/October 1999), 675-689.
9. N. Fota et al., Safety analysis and evaluation of an air traffic control computing system, in Proc. SAFECOMP '96 (Vienna, Austria, October 1996), 219-229.
10. M. J. Harrold. Testing: a roadmap", in this volume.
11. L. Hat-ton, Programming Languages and safety-Related Systems, in Proc. Safety- Critical Systems Symposium (Brighton, U.K., 1995), 49-64.