International Journal Of Advanced Research In Engineering Technology & Sciences
ISSN: 2394-2819
November- 2014 Volume 1, Issue-2
Email: editor@ijarets.org www.ijarets.org

SOFTWARE RELIABILTY METRICES for QUALITATIVE ANALYSIS

Dhowmya Bhatt, Mradula Sharma ,Madan Kumar
Assistant Professor
Department of Information Technology
SRM University NCR Campus
Modinagar

ABSTRACT:
The term reliability in software Engineering is focused on those engineering techniques and technologies that prove helpful in developing and maintaining system software. This is generally called by the term "software Reliability Engineering". This mainly focuses on those software systems in which the reliability can be evaluated quantitatively. But this evaluation shall be possible only if the failure data is measured properly while developing the software and implementing it. Also when reliability analysis is done, a credible software reliability model is needed to trace the failure process. This poses an open challenge to reliability methodologies. Hence this demands the development of new software reliability engineering paradigms taking into consideration the software architectures, testing techniques, and software failure manifestation mechanisms. In this proposed research paper I attempt to review the existing trends and current problems in software reliability. Directions for the future research in software reliability engineering are also discussed.

INDEX TERMS: Reliability, Quantitative Analysis, Failures

INTRODUCTION:

Software is omnipresent in our life today. Infact it has become an inseparable part of many aspects of our society. Hence for making improvements and breakthrough, high-quality software have become essential. It has to be also noted that the complexity of software systems have grown during the last few years and this will continue in the future too. In short, the demand for software in various industries shows a compound growth. Also that the software dependency has increased, it has to be observed that the failures in software systems can lead to fatal consequences, especially when the security factor is taken into account. Software failures in the past have led to heavy and disastrous loss in the business sectors.[8].

Though the software systems have made our life swift, we often are not able to find where and how the software has failed or will fail in the future. It is essential to note that even today software Engineering has not evolved into a full Engineering discipline due to inconsistency in the process of decision making, hence this also makes a very critical situation for Reliability Engineering because it involves measuring quality quantitatively.

Software reliability engineering is centered on a key attribute, software reliability, which is defined as the probability of failure-free software operation for a specified period of time in a specified environment [9]. software reliability is generally accepted as the major factor in software quality since it quantifies software failures, which can make a powerful system inoperative. Software reliability engineering (SRE) is therefore defined as the quantitative study of the operational behavior of software-based systems with respect to user requirements concerning reliability.[1]

The failures in the existing software techniques can be complied into three major categories besides unexpected software breakdowns including those which can or cannot be recovered later.

1. Current SRE techniques collect the failure data during integration testing or system testing phases. Failure data collected during the late testing phase may be too late for fundamental design changes.
2. Failure data collected in the in-house testing may be limited, and they may not represent failures that would be uncovered under actual operational environment.

3. Current SRE techniques or modeling methods are based on some unrealistic assumptions that make the reliability estimation too optimistic relative to real situations.[21]

In this research paper, I shall first survey the current trends in reliability Engineering and towards the end discuss the problems and concerns that remain in this field.

TECHNIQUES IN RELIABILITY ENGINEERING

There have been a number of techniques in use to counter attacks on software reliability. The important of those are,

FAULT LIFECYCLE TECHNIQUE

This summarizes the following four technical areas which are applicable to achieving reliable software systems, and they can also be regarded as four fault lifecycle techniques.

1. **FAULT PREVENTION:** to avoid, by construction, fault occurrences.
2. **FAULT REMOVAL:** to detect, by verification and
3. validation, the existence of faults and eliminate them.
4. **FAULT TOLERANCE:** to provide, by redundancy, service complying with the specification in spite of faults having occurred or occurring.
5. **FAULT OR FAILURE FORECASTING:** to estimate, by evaluation, the presence of faults and the occurrences and consequences of failures. This has been the main focus of software reliability modeling.[17]

FAULT MEASUREMENT IN RELIABILITY ENGINEERING:

While measuring Software reliability models usually make a number of common assumptions. They are

1. The operation environment where thereliability is to be measured is the same as the testing environment in which the reliability model has been parameterized.
2. Once a failure occurs, the fault which causes the failure is immediately removed.
3. The fault removal process will not introduce new faults.
4. The number of faults inherent in the software and the way these faults manifest themselves to cause failures follow, at least in a statistical sense, certain mathematical formulae.[18]

FAULT TOLERANT MODELS

There are two different groups of fault tolerance techniques,

1. **SINGLE VERSION** - includes program modularity, system closure, atomicity of actions, error detection, exception handling, checkpoint and restart, process pairs, and data diversity [6]
2. **MULTI-VERSION SOFTWARE TECHNIQUES** – so called design diversity, is employed where multiple software versions are developed independently by different program teams using different design methods, yet they provide equivalent services according to the same requirement specifications. [7].

CURRENT TRENDS IN SOFTWARE AND SYSTEM RELIABILITY

The foundation for software reliability is for sure the hardware reliability techniques that had been put into use long time ago. By applying mathematical models, software reliability schemes could be established as same as the hardware frameworks. This had certain advantages.

1. The physical meaning of the failure mechanism can be properly interpreted, so that the effect of failures on reliability,
2. The combination of hardware reliability and software reliability to form system reliability models and measures can be provided in a unified theory. A single formulation can be employed from the reliability modeling and statistical estimation viewpoints.
3. System reliability models inherently engage system structure and modular design in block diagrams. The resulting reliability modelling process is not only intuitive but also informative.

METRICS AND MEASUREMENT:

This is important for the software quality assurance purposes. software complexity and software quality are highly related to software reliability, the measurements of software complexity and quality attributes have been explored for early prediction of software reliability [5]. Static as well as dynamic program complexity measurements have been collected, such as lines of code, number of operators, relative program complexity, functional complexity, operational complexity, and so on. The complexity metrics can be further included in software reliability models for early reliability prediction, for example, to predict the initial software fault density and failure rate.[1] The two measurements related to reliability are

1. the number of failures in a time period
2. time between failures.

DATA COLLECTION AND ANALYSIS:

Data collection, plays a vital role for the success of software reliability measurement. The amounts and types of data to be collected for reliability analysis purposes vary between organizations. The more accuracy is required for analysis, the more effort is required for data collection. Fault-based data are usually easier to collect due to their static nature. [1] While it is much more difficult to collect failure-based data for the following reasons.

1. The dynamic operating condition where the failures occur may be hard to identify or describe.
2. The time when the failures occur must be recorded manually, after the failures are manifested.
3. After collecting the failure data from a real system, the analysis consists of five steps essentially in order [14]
4. Pre-processing of data - the necessary information is extracted from the field data.
5. Analysis of data – Data Interpretation.
6. Model structure identification and parameter estimation – realistic parameters.
7. Model solution – using known techniques to solve the model.
8. Analysis of models –Question addressing.

METHODS AND TOOLS :

a. FAULT TREES –

They provide a graphical and logical framework for a systematic analysis of system failure modes. Fault tree models therefore provide an informative modeling framework that can be engaged to compare different design alternatives or system architectures with respect to reliability.

b. SIMULATION TECHNIQUES –

They can produce observables of interest in reliability engineering, including discrete integer-valued quantities that occur as time progresses. One simulation approach produces “artifacts” in an actual software environment according to factors and influences believed to typify these entities within a given context [4]. This artifact-based simulation allows experiments to be set up to examine the nature of the relationships between software failures and other software metrics.

EFFECTIVENESS TESTING AND CODE COVERAGE:

Effective testing is defined as uncovering of most of the existing detectable faults. As the total number of inherent faults is not known, testing effectiveness is usually represented by a measurable testing index. Code coverage, as an indicator to show how thoroughly software has been stressed, has been proposed and is widely employed to represent the estimated metrics of fault coverage. [1]

	REFERENCE	FINDINGS
POSITVE	Horgan (1994)[17] Frankl (1988) [16] Rapps (1988) [38]	High code coverage brings high software reliability and low fault rate.
POSITIVE	Chen (1992) [13]	A correlation between code coverage and software reliability was observed.
POSITIVE	Wong (1994)	The correlation between test effectiveness and block coverage is higher than that between test effectiveness and the size of test set. Frate (1995)
NEGATIVE	Briand (2000) [6]	The testing result for published data did not support a causal dependency between code coverage and fault coverage.

Table1. Comparison of Investigations on the Relation of Code Coverage to Fault Coverage [1]

INDUSTRY PRACTICES:

There have been only limited success in the industry adoption of software reliability Engineering as it is being seen as cost rather than value.[21] The main reason for the lack of industry enthusiasm in SRE is because its cost-effectiveness is not clear. Current SRE techniques return visible overhead but yield invisible benefits. Reliability is often considered as a cost rather than a value, an investment rather than a return.[12] The reliability attribute of a product takes less priority than its functionality or innovation. Many companies, voluntarily or under compulsion from their quality control policy, collect failure data and make reliability measurements and many software projects claim to set reliability objectives such as five 9's or six 9's that is 0.99999 to 0.999999 availability or 10-5 to 10-6 failures per execution hour. [1]

SOFTWARE ARCHITECTURE:

Software architecture is one of the key factors determining the performance of software systems. Hence a lot of effort has been put in the past to achieve good system architecture so as to support reliability methods. Fault isolation is a major design consideration for software architecture. Good software architecture should enjoy the property that exceptions are raised when faults occur, and module failures are properly confined without causing system failures. In particular, this type of component-based software development approach requires different framework, quality assurance paradigm [2], and reliability modeling [3] from those in traditional software development. Information engineering is becoming the central focus for today's businesses, service-oriented systems and the associated software engineering will be the de facto standards for business development in these sectors for the coming decades. [1].

THE FUTURE DIRECTIONS in SOFTWARE RELIABILITY ENGINEERING

Reliability engineering for software development is focusing on two major aspects,

1. software architecture,
2. Component-based software engineering.

The software architecture of a system consists of software components, their external properties, and their relationships with one another. Effective software architecture not only provides a strong basis for the consequent software development and maintenance phases, but also offers various options for fault avoidance and fault tolerance in achieving high reliability. But it has to be noted while discussing this that Software architecture concerning reliability depends on the design of failure-resilient architecture. This requires an effective software architecture design which guarantees the separation of components when the actual parent software executes. When component failures occur in the system, they can then be quickly identified and fault can be properly dealt with on time everytime.

The other aspect is component-based software engineering that deals with the process view of software engineering. This actually helps us in identifying certain practical factors such as,

1. Generation of reusable components
2. Modification of existing components for reusability
3. Controlled interaction of components
4. Defensive Mechanisms against failures
5. Risk free components design

ADVANTAGES:

1. Reduce risks and failures
2. Fault detection before damaging the system
3. Complete fault diagnosis after failures

DESIGN FOR RELIABLE SYSTEMS:

The basic need to design a reliable system is to install a proper fault tolerant mechanism that will report the failures on time and fight against local failure occurrence. It should have these stage mechanisms,

1. Fault Confinement – fault does not spread and is restricted to one area.
2. Fault detection – detects the unexpected error both offline and online.
3. Diagnosis – if detection fails then diagnosis works when system breaks down.
4. Reconfiguration – when permanent error occurs then the system automatically reconfigures to recover
5. Recovery – it involves few stages,
6. Fault masking
7. Retry
8. Rollback
9. Restart – involves the recovery of undamaged information from the damaged system
10. Repair – the user stands no other choice but to replace the damaged component with a fresh part
11. Reintegration – involves repairing a part and merging it again with the system.

PROBLEMS IN RELIABILITY SYSTEMS:

Taking most of the major common problems in software reliable systems leaving out the smaller ones, I happen to find these difficulties as listed below,

1. Independent hardware failures while in software there is inter-dependency between both and so balancing is tough.
2. Modeling in reliability system is done independently after the entire system is built and hence there is a sense if isolation from the main system.
3. Software metrics and measurements are not consistently defined and interpreted, again due to the lack of physical attributes of software. The achieved reliability measures may differ for different applications, yielding inconclusive results.[1]
4. Insufficient availability of SRE tools to meet out the current requirements and SRE practices.
5. Developing operational profiles for system testing. When a system fails, even under emergency, the operational profiles cannot be duplicated.
6. Cost of implementing reliability solutions in organisations to systems and their increasing need. This today certainly is in the indirect proportion.

CONCLUSIONS:

Software Reliability Systems greatly influence business performance and hence their failure will also inturn lead to great loss and damage to the economy. Hence this demands employing super effective reliable systems for process and product reliability. Only this can help meet major challenges in the field of reliability engineering. In this paper I have discussed the trends and problems in the field of software reliability Engineering and also possible future directions and further research in this filed have also been addressed. In specific few new software reliability engineering paradigms by taking software architectures, testing techniques, and software failure manifestation mechanisms into consideration have been put forth. Lime light also upon emerging software applications is thrown.

ACKNOWLEDGEMENTS:

I wish to extend my sincere thanks to Prof. Michael R. Lyu, Department of CSE, Chinese University, Hongkong for sharing his research on SRE and for inspiring me write this research paper.

REFERENCES:

1. Michael R. Lyu "*Software Reliability Engineering: A Roadmap*"
2. X. Cai, M.R. Lyu, and K.F. Wong, "*A Generic Environment for COTS Testing and Quality Prediction*," Testing Commercial-off-the-shelf Components and Systems, S. Beydeda and V. Gruhn (eds.), Springer-Verlag, Berlin, 2005, pp. 315-347.
3. S. Yacoub, B. Cukic, and H Ammar, "*A Scenario-Based Reliability Analysis Approach for Component-Based Software,"* IEEE Transactions on Reliability, vol. 53, no. 4, 2004, pp. 465-480.
4. A. von Mayrhauser and D. Chen, "*Effect of Fault Distribution and Execution Patterns on Fault Exposure in Software: A Simulation Study,"* Software Testing, Verification & Reliability, vol. 10, no.1, March 2000, pp. 47-64.
5. Rome Laboratory (RL), *Methodology for Software Reliability Prediction and Assessment, Technical Report RLTR*- 92-52, volumes 1 and 2, 1992.
6. W. Torres-Pomales, "*Software Fault Tolerance: A Tutorial,"* NASA Langley Research Center, Hampton, Virginia, TM-2000-210616, Oct. 2000.
7. M.R. Lyu and X. Cai, "*Fault-Tolerant Software," Encyclopedia on Computer Science and Engineering*, Benjamin Wah (ed.), Wiley, 2007.
8. M.R. Lyu (ed.), *Handbook of Software Reliability Engineering*, IEEE Computer Society Press and McGraw- Hill, 1996
9. ANSI/IEEE, *Standard Glossary of Software Engineering Terminology*, STD-729-1991, ANSI/IEEE, 1991.
10. J. Bishop and N. Horspool, "*Cross-Platform Development: Software That Lasts,*" IEEE Computer, October 2006, pp. 26-35.
11. L. Briand and D. Pfahl, "*Using Simulation for Assessing the Real Impact of Test Coverage on Defect Coverage*," IEEE Transactions on Reliability, vol. 49, no. 1, March 2000, pp. 60-70.
12. J.R. Horgan, S. London, and M.R. Lyu, "*Achieving Software Quality with Testing Coverage Measures*," IEEE Computer, vol. 27, no.9, September 1994, pp. 60-69.
13. T. Margaria and B. Steffen, "*Service Engineering: Linking Business and IT,"* IEEE Computer, October 2006, pp. 45-55.
14. J.D. Musa, *Software Reliability Engineering: More Reliable Software Faster and Cheaper*, 2nd Edition , AuthorHouse, 2004.

15. J.D. Musa, *"Operational Profiles in Software Reliability Engineering,"* IEEE Software, Volume 10, Issue 2, March 1993, pp. 14-32.
16. J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw- Hill, Inc., New York, NY, 1987.
17. W.S. Humphrey, *"The Future of Software Engineering: I,"* Watts New Column, News at SEI, vol. 4, no. 1, March, 2001
18. M. Jazayeri, *"Web Application Development: The Coming Trends,"* Future of Software Engineering 2007, L. Briand and A. Wolf (eds.), IEEE-CS Press, 2007.
19. M. Xie, *Software Reliability Modeling, World Scientific Publishing Company*, 1991.
20. H. Pham, *Software Reliability, Springer*, Singapore, 2000.
21. L. Baresi, E. Nitto, and C. Ghezzi, "*Toward Open-World Software: Issues and Challenges,"* IEEE Computer, October 2006, pp. 36-43.
22. A. Bertolino, *"Software Testing Research: Achievements, Challenges, Dreams,"* Future of Software Engineering 2007, L. Briand and A. Wolf (eds.), IEEE-CS Press, 2007.