SOFTWARE RELIABILITY ENGINEERING TECHNIQUES FOR DEVELOPING MAINTAINING SOFTWARE SYSTEMS QUATITATIVELY EVALUTED

[1]Dr.UmeshSehgal, [2]Er.Chetan Sharma

Associate Professor, Department of Computer Application, ARNI University Kathgarh, H.P

Assistant Professor, Department of Computer Science and Engg, ARNI University Kathgarh, H.P

Software reliability engineering is focused on engineering techniques for developing and maintaining software systems whose reliability can be quantitatively evaluated. In order to estimate as well as to predict the reliability of software systems, failure data need to be properly measured by various means during software development and operational phases. Moreover, credible software reliability models are required to track underlying software failure processes for accurate reliability analysis and forecasting. Although software reliability has remained an active research subject over the past 35 years, challenges and open questions still exist. In particular, vital future goals include the development of new software reliability engineering paradigms that take software architectures, testing techniques, and software failure manifestation mechanisms into consideration. In this paper, we review the history of software reliability engineering, the current trends and existing problems, and specific difficulties. Possible future directions and promising research subjects in software reliability engineering are also addressed.

Keywords: *Software Models ,UML and Software Tools*

ABSTRACT

INTRODUCTION

Software permeates our daily life. There is probably no other human-made material which is more omnipresent than software in our modern society. It has become a crucial part of many aspects of society: home appliances, telecommunications, automobiles, airplanes, shopping, auditing, web teaching, personal entertainment, and so on. In particular, science and technology demand high-quality software for making improvements and breakthroughs.

The size and complexity of software systems have grown dramatically during the past few decades, and the trend will certainly continue in the future. The data from industry show that the size of the software for various systems and applications has been growing exponentially for the past 40 years [20]. The trend of such growth in the telecommunication, business, defense, and transportation industries shows a compound growth rate of ten times every five years. Because of this ever-increasing dependency, software failures can lead to serious, even fatal, consequences in safety-

critical systems as well as in normal business. Previous software failures have impaired several high visibility programs and have led to loss of business [28].

The ubiquitous software is also invisible, and its invisible nature makes it both beneficial and harmful. From the positive side, systems around us work seamlessly thanks to the smooth and swift execution of software. From the negative side, we often do not know when, where and how software ever has failed, or will fail. Consequently, while reliability engineering for hardware and physical systems continuously improves, reliability engineering for software does not really live up to our expectation over the years. This situation is frustrating as well as encouraging. It is frustrating because the software crisis identified as early as the 1960s still stubbornly stays with us, and "software engineering" has not fully evolved into a real engineering discipline. Human judgments and subjective favorites, instead of physical laws and rigorous procedures, dominate many decision making processes in software engineering. The situation is particularly critical in software reliability engineering. Reliability is probably the most important factor to claim for any engineering discipline, as it quantitatively measures quality, and the quantity can be properly engineered. Yet software reliability engineering, as elaborated in later sections, is not yet fully delivering its promise. Nevertheless, there is an encouraging aspect to this situation. The demands on, techniques of, and enhancements to software are continually increasing, and so is the need to understand its reliability. The unsettled software crisis poses tremendous opportunities for software engineering researchers as well as ractitioners. The ability to manage quality software production is not only a necessity, but also a key distinguishing factor in maintaining a competitive advantage for modernbusinesses.

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 [2]. Among other attributes of software quality such as functionality, usability, capability, and maintainability, etc., 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. As a proven technique, SRE has been adopted either as standard or as best current practice by more than 50 organizations in their software projects and reports [33], including AT&T, Lucent, IBM, NASA, Microsoft, and many others in Europe, Asia, and North America. However, this number is still relatively small compared to the large amount of software producers in the world. Existing SRE techniques suffer from a number of weaknesses. First of all, 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. Secondly, the 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. This is especially true for high-quality software systems which require extensive and wide-rangingtesting.

HISTORICAL SOFTWARE RELIABILITY ENGINEERING TECHNIQUES

In the literature a number of techniques have been proposed to attack the software reliability engineering problems based on software fault lifecycle. We discuss these techniques, and focus on two ofthem.

Fault lifecycle techniques

Achieving highly reliable software from the customer's perspective is a demanding job for all software engineers and reliability engineers. [28] summarizes the following four technical areas which are applicable to achieving reliable software systems, and they can also be regarded as four fault lifecycletechniques:

1) Fault prevention: to avoid, by construction, faultoccurrences.

2) Fault removal: to detect, by verification and validation, the existence of faults and eliminatethem.

3) Fault tolerance: to provide, by redundancy, service complying with the specification in spite of faults having occurred oroccurring.

4) Fault/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 reliabilitymodeling.

Software reliability models and measurement

As a major task of fault/failure forecasting, software reliability modeling has attracted much research attention in estimation (measuring the current state) as well as prediction (assessing the future state) of the reliability of a software system. A software reliability model specifies the form of a random process that describes the behavior of software failures with respect to time. A historical review as well as an application perspective of software reliability models can be found in [7, 28]. There are three main reliability modeling approaches: the error seeding and tagging approach, the data domain approach, and the time domain approach, which is considered to be the most popular one. The basic principle of time domain software reliability modeling is to perform curve fitting of observed time-based failure data by a pre-specified model formula, such that the model can be parameterized with statistical techniques (such as the Least Square or Maximum Likelihood methods). The model can then provide estimation of existing reliability or prediction of future reliability by extrapolation techniques.Softwarereliabilitymodelsusuallymakeanumberofcommonassumptions,asfollows.

(1) The operation environment where the reliability 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. Since the number of faults (as well as the failure rate) of the software system reduces when

the testing progresses, resulting in growth of reliability, these models are often called software reliability growth models (SRGMs).

CURRENT TRENDS ANDPROBLEMS

The challenges in software reliability not only stem from the size, complexity, difficulty, and novelty of software applications in various domains, but also relate to the knowledge, training, experience and character of the software engineers involved. We address the current trends and problems from a number of software reliability engineeringaspects.

1. Software reliability and systemreliability
2. Software reliabilitymodeling
3. Metrics andmeasurements
4. Data collection andanalysis
5. Methods andtools
6. Testing and operationalprofiles

POSSIBLE FUTURE DIRECTIONS

SRE activities span the whole software lifecycle. We discuss possible future directions with respect to five areas: software architecture, design, testing, metrics and emerging applications.

Reliability for software architectures and off-the-shelf components

Due to the ever-increasing complexity of software systems, modern software is seldom built from scratch. Instead, reusable components have been developed and employed, formally or informally.On the one hand, revolutionary and evolutionary object-oriented design and programming paradigms have vigorously pushed software reuse. On the other hand, reusable software libraries have been a deciding factor regarding whether a software development environment or methodology would be popular or not. In the light of this shift, reliability engineering for software development is focusing on two major aspects: software architecture, and component-based softwareengineering.
The software architecture of a system consists of software components, their external properties, and their relationships with one another. As software architecture is the foundation of the final software product, the design and management of software architecture is becoming the dominant factor in software reliability engineering research. Well designed software architecture not only provides a strong, reliable basis for the subsequent software development and maintenance phases, but also offers various options for fault avoidance and fault tolerance in achieving high reliability. Due to the cardinal importance of, and complexity involved in, software architecture design and modeling, being agoodsoftwarearchitectisararetalentthatishighlydemanded.Agoodsoftwarearchitectsees

widely and thinks deeply, as the components should eventually fit together in the overall framework, and the anticipation of change has to be considered in the architecture design. A clean, carefully laid out architecture requires up-front investments in various design considerations, including high cohesion, low coupling, separation of modules, proper system closure, concise interfaces, avoidance of complexity, etc. These investments, however, are worthwhile since they eventually help to increase software reliability and reduce operation and maintenance costs.

One central research issue for software architecture concerning reliability is the design of failure-resilient architecture. This requires an effective software architecture design which can guarantee separation of components when software executes. When component failures occur in the system, they can then be quickly identified and properly contained. Various techniques can be explored in such a design. For example, memory protection prevents interference and failure propagation between different application processes. Guaranteed separation between applications has been a major requirement for the integration of multiple software services in complicated modern systems. It should be noted that the separation methods can support one another, and usually they are combined for achieve better reliability returns. Exploiting this synergy for reliability assessment is a possibility for furtherexploration.

These methods favor reliability engineering in multiple ways. First of all, they directly increase reliability by reducing the frequency and severity of failures. Run-time protections may also detect faults before they cause serious failures. After failures, they make fault diagnosis easier, and thus accelerate reliability improvements. For reliability assessment, these failure prevention methods reduce the uncertainties of application interdependencies or unexpected environments. So, for instance, having sufficient separation between running applications ensures that when we port an application to a new platform, we can trust its failure rate to equal that experienced in a similar use on a previous platform plus that of the new platform, rather than being also affected by the specific combination of other applications present on the new platform.

Achieving design for reliability

1. Faultconfinement
2. Faultdetection
3. Diagnosis.
4. Reconfiguration
5. Recovery
6. Restart
7. Repair

Testing for reliability assessment

Software testing and software reliability have traditionally belonged to two separate communities. Software testers test software without referring to how software will operate in the field, as often the environment cannot be fully represented in the laboratory. Consequently they design test cases for exceptional and boundary conditions, and they spend more time trying to break the software than conducting normal operations. Software reliability measurers, on
the other hand, insist that software should be tested according to its operational profile in order to allow accurate reliability estimation and prediction. In the future, it will be important to bring the two groups together, so that on the one hand, software testing can be effectively conducted, while on the other hand, software reliability can be accurately measured. One approach is to measure the test compression factor, which is defined as the ratio between the mean time between failures during operation and during testing. This factor can be empirically determined so that software reliability in the field can be predicted from that estimated during testing. Another approach is to ascertain how other testing related factors can be incorporated into software reliability modeling, so that accurate measures can be obtained based on the effectiveness of testing effort.

Software engineering targeted for general systems may be too ambitious. It may find more successful applications if it is domain-specific. In this Future of Software Engineering volume, future software engineering techniques for a number of emerging application domains have been thoroughly discussed. Emerging software applications also create abundant opportunities for domain-specific reliability engineering.

One key industry in which software will have a tremendous presence is the service industry. Service oriented design has been employed since the 1990s in the telecommunications industry, and it reached software engineering community as a powerful paradigm for Web service development, in which standardized interfaces and protocols gradually enabled the use of third-party functionality over the Internet, creating seamless vertical integration and enterprise process management for cross-platform, cross-provider, and cross-domainapplications.

Researchers have proposed the publish/subscribe paradigm as a basis for middleware platforms that support software applications composed of highly evolvable and dynamic federations of components. In this approach, components do not interact with each other directly; instead an additional middleware mediates their communications. Publish/subscribe middleware decouples the communication among components and supports implicit bindings among components. The sender does not know the identity of the receivers of its messages, but the middleware identifies them dynamically. Consequently new components can dynamically join the federation, become immediately active, and cooperate with the other components without requiring any reconfiguration of the architecture. Interested readers can refer to [21] for future trends in middleware-based software engineeringtechnologies.

The open system approach is another trend in software applications. Closed-world assumptions do not hold in an increasing number of cases, especially in ubiquitous and pervasive computing settings, where the world is intrinsically open. Applications cover a wide range of areas, from dynamic supply-chain management, dynamic enterprise federations, and virtual endeavors, on the enterprise level, to automotive applications and home automation on the embedded-systems level. In an open world, the environment changes continuously. Software must adapt and react dynamically to changes, even if they are unanticipated. Moreover, the world is open tonew
components that context changes could make dynamically available – for example, due to mobility. Systems can discover and bind such components dynamically to the application while it is executing.

The software must therefore exhibit a self-organization capability. In other words, the traditional solution those software designers adopted – carefully elicit change requests, prioritize them, specify them, design changes, implement and test, then redeploy the software – is no longer viable. More flexible and dynamically adjustable reliability engineering paradigms for rapid responses to software evolution are required.

CONCLUSIONS

As the cost of software application failures grows and as these failures increasingly impact business performance, software reliability will become progressively more important. Employing effective software reliability engineering techniques to improve product and process reliability would be the industry's best interests as well as major challenges. In this paper, we have reviewed the history of software reliability engineering, the current trends and existing problems, and specific difficulties. Possible future directions and promising research problems in software reliability engineering have also been addressed. We have laid out the current and possible future trends for software reliability engineering in terms of meeting industry and customer needs. In particular, we have identified new software reliability engineering paradigms by taking software architectures, testing techniques, and software failure manifestation mechanisms into consideration. Some thoughts on emerging software applications have also been provided.

ITEM Software produces reliability analysis tools which are applicable to a wide range of industries. ITEM Toolkit's Fault Tree, Markov, and FMECA modules can be used to model software reliability, physical security, as well as human interaction with systems. Our Event Tree module can be used to perform Decision Tree Analysis, as well as Safety Assessment of any system. Our reliability prediction analysis modules are up-to-date yet flexible enough to suit your evolving needs. The main features :-

- Powerful and user friendly reliability engineering softwaretools
- Extensive printing and report generationcapabilities
- Build and open multiple systems and projectfiles
- Multi-document interface allows easy transfer ofdata
- Import and export to MS Excel, Access Wordetc.
- Take advantage of powerful 'what if' analyticaltools
- More accurate and efficient than manualmethods
- Identify weak areas in a systemdesign
- Select components with regard to reliability and costsavings
- Optimize designs to meet targetedgoals
- Graphically construct system hierarchydiagrams
- Library management facilities to build and customizelibraries
- Powerful chartingfacilities
- Drag and drop components and systems betweenprojects
- Flexible graphical editing capabilities for event sequence diagrams and faulttrees
- Advanced hybrid linking andmodeling
- Combine prediction methods for complexanalysis
- Linked block facility reduces repetitive dataentry

REFERENCES

[1] S. Amasaki, O. Mizuno, T. Kikuno, and Y. Takagi, "A Bayesian Belief Network for Predicting Residual Faults in Software Products," *Proceedings of 14th International Symposium on Software Reliability Engineering (ISSRE2003),* November 2003, pp. 215-226,

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

[3] L. Baresi, E. Nitto, and C. Ghezzi, "Toward Open-World Software: Issues and Challenges," *IEEE Computer*, October 2006, pp.36-43.

[4] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams," *Future of Software Engineering 2007*,L. Briand and A. Wolf (eds.), IEEE-CS Press,2007.

[5] J. Bishop and N. Horspool, "Cross-Platform Development: Software That Lasts," *IEEE Computer*, October 2006, pp.26-35.

[6] L. Briand and D. Pfahl, "Using Simulation for Assessing the Real Impact of Test Coverage on DefectCoverage,".

[7] J. Cheng, D.A. Bell, and W. Liu, "Learning Belief Networks from Data: An Information Theory Based Approach," *Proceedings of the Sixth International Conference on Information and Knowledge Management*, Las Vegas, 1997, pp.325-331.

[8] X. Cai and M.R. Lyu, "The Effect of Code Coverage on Fault Detection Under Different Testing Profiles," *ICSE 2005 Workshop on Advances in Model-Based Software Testing (A-MOST)*, St. Louis, Missouri, May2005.

[9] 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.

[10] X. Cai, M.R. Lyu, and M.A. Vouk, "An Experimental Evaluation on Reliability Features of N-Version Programming," in *Proceedings 16th International Symposium on Software Reliability Engineering (ISSRE'2005)*, Chicago, Illinois, Nov. 8-11,2005.

[11] X. Cai and M.R. Lyu, "An Empirical Study on Reliability and Fault Correlation Models for Diverse Software Systems," in *Proceedings 15th International Symposium on Software Reliability Engineering (ISSRE'2004)*, Saint-Malo, France, Nov. 2004,pp.125-136.

[12] M. Chen, M.R. Lyu, and E. Wong, "Effect of Code Coverage on Software Reliability Measurement," *IEEE Transactions on Reliability*, vol. 50, no. 2, June 2001,pp.165-170.

[13] M.H. Chen, A.P. Mathur, and V.J. Rego, "Effect of Testing Techniques on Software Reliability Estimates Obtained Using Time Domain Models," In *Proceedings of the 10th Annual Software Reliability Symposium*, Denver, Colorado, June 1992, pp.116-123.

[14] J.B. Dugan and M.R. Lyu, "Dependability Modeling for Fault-Tolerant Software and Systems," in *Software Fault Tolerance*, M. R. Lyu (ed.), New York: Wiley, 1995, pp.109–138.

[15] D.E. Eckhardt and L.D. Lee, "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors," *IEEE Transactions on Software Engineering*, vol. 11, no. 12, December 1985, pp.1511–1517.

[16] P.G. Frankl and E.J. Weyuker, "An Applicable Family of Data Flow Testing Criteria," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, October 1988, pp.1483-1498.

[17] 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.

[18] C.Y. Huang and M.R. Lyu, "Optimal Release Time for Software Systems Considering Cost, Testing-Effort, and Test Efficiency," *IEEE Transactions on Reliability*, vol. 54, no. 4, December 2005, pp.583-591.

[19] C.Y. Huang, M.R. Lyu, and S.Y. Kuo, "A Unified Scheme of Some Non-Homogeneous Poisson Process Models for Software Reliability Estimation," *IEEE Transactions on Software Engineering*, vol. 29, no. 3, March 2003, pp.261-269.

[20] W.S. Humphrey, "The Future of Software Engineering: I," Watts New Column, News at SEI, vol. 4, no. 1, March, 2001.

[21] V. Issarny, M. Caporuscio, and N. Georgantas: "A Perspective on the Future of Middleware-Based Software Engineering," *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds.), IEEE-CS Press,2007.

[22] M. Jazayeri, "Web Application Development: The Coming Trends," *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds.), IEEE-CS Press,2007.

[23] Z. Jelinski and P.B. Moranda, "Software Reliability Research," in *Proceedings of the Statistical Methods for the Evaluation of Computer System Performance*, Academic Press, 1972, pp.465-484.

[24] B. Littlewood and L. Strigini, "Software Reliability and Dependability: A Roadmap," in *Proceedings of the 22nd International Conference on Software Engineering (ICSE'2000),* Limerick, June 2000, pp.177-188.

[25] B. Littlewood and D. Miller, "Conceptual Modeling of Coincident Failures in Multiversion Software," *IEEE Transactions on Software Engineering*, vol. 15, no. 12, December 1989, pp. 1596–1614.

[26] C. Liu, L. Fei, X. Yan, J. Han, and S. Midkiff, "Statistical Debugging: A Hypothesis Testing-based Approach," *IEEE Transaction on Software Engineering*, vol.32, no. 10, October, 2006, pp. 831-848.

[27] N. Looker and J. Xu, "Assessing the Dependability of SOAP-RPC-Based Web Services by Fault Injection," in *Proceedings of 9th IEEE International Workshop on Object oriented Real-time Dependable Systems*, 2003, pp.163-170.

[28] M.R. Lyu (ed.), *Handbook of Software Reliability Engineering*, IEEE Computer Society Press andMcGraw-

[29] M.R. Lyu and X. Cai, "Fault-Tolerant Software," *Encyclopedia on Computer Scienceand Engineering*, Benjamin Wah (ed.), Wiley, 2007. [30] M.R. Lyu, Z. Huang, S. Sze, and X. Cai, "An Empirical Study on Testing and Fault Tolerance for Software Reliability Engineering," in *Proceedings 14th IEEE International Symposium on Software Reliability Engineering (ISSRE'2003)*, Denver, Colorado, November 2003, pp.119-130. *Transactions on Reliability*, vol. 51, no. 4, December 2002, pp.420-426.

[32] T. Margaria and B. Steffen, "Service Engineering: Linking Business and IT," *IEEE Computer*, October 2006, pp.45-55.

[33] J.D. Musa, *Software Reliability Engineering: More Reliable Software Faster and Cheaper (2nd Edition)*, AuthorHouse,2004.

[34] J.D. Musa, "Operational Profiles in Software Reliability Engineering," *IEEE Software*, Volume 10, Issue 2, March 1993, pp.14-32.

[35] J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw- Hill, Inc., New York, NY,1987.

[36] H. Pham, *Software Reliability*, Springer, Singapore,2000.

[37] P.T. Popov, L. Strigini, J. May, and S. Kuball, "Estimating Bounds on the Reliability of Diverse Systems," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, April 2003, pp.345–359.

[38] S. Rapps and E.J. Weyuker, "Selecting Software Test Data Using Data Flow Information," *IEEE Transactions on Software Engineering*, vol. 11, no. 4, April 1985, pp. 367-375.

[39] Rome Laboratory (RL), *Methodology for Software Reliability Prediction and Assessment*, Technical Report RLTR- 92-52, volumes 1 and 2,1992.

[40] M.L. Shooman, *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis and Design*, Wiley, New York,2002.

[41] R. Taylor and A. van der Hoek, "Software Design and Architecture: The Once and Future Focus of Software Engineering," *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds.), IEEE-CS Press,2007.

[42] X. Teng, H. Pham, and D. Jeske, "Reliability Modeling of Hardware and Software Interactions, and Its Applications," *IEEE Transactions on Reliability*, vol. 55, no 4, Dec. 2006, pp.571-577.

[43] L.A. Tomek and K.S. Trivedi, "Analyses Using Stochastic Reward Nets," in *Software Fault Tolerance*, M.R. Lyu (ed.), New York: Wiley, 1995, pp.139–165.

[44] W. Torres-Pomales, "Software Fault Tolerance: A Tutorial," NASA Langley Research Center, Hampton, Virginia, TM-2000-210616, Oct.2000.

[45] K.S. Trivedi, "SHARPE 2002: Symbolic Hierarchical Automated Reliability and Performance Evaluator," in *Proceedings International Conference on Dependable Systems and Networks*,2002.

[46] K.S. Trivedi, K. Vaidyanathan, and K. Goseva- Postojanova, "Modeling and Analysis of Software Aging and Rejuvenation", in *Proceedings of 33 rd Annual Simulation. Symposium*, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp.270-279.

[47] 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.

[48] M.A. Vouk, "Using Reliability Models During Testing With Nonoperational Profiles," in *Proceedings of 2nd Bellcore/Purdue Workshop on Issues in Software Reliability Estimation*, October 1992, pp.103-111.

[49] W. Wang and M. Tang, "User-Oriented Reliability Modeling for a Web System," in *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE'03)*, Denver, Colorado, November 2003,pp.1-12.

[50] M. Xie, *Software Reliability Modeling*, World Scientific Publishing Company,1991.

[51] 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.

[52] A.X. Zheng, M.I. Jordan, B. Libit, M. Naik, and A. Aiken, "Statistical Debugging: Simultaneous Identification of Multiple Bugs," in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006, pp.1105-1112.