Volume 3, Issue 6, June 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Review of Fault Detection Techniques to Detect Faults and Improve the Reliability in Web Applications

Jyoti Tamak
Department of Computer Science and Engineering
University Institute of Engineering & Technology
Kurukshetra University,Kurukshetra,
Haryana, India

Abstract--Software reliability is a big anxiety in industry. Every software shows some minor bugs after being released. Detection and diagnosis of faults in a large-scale distributed system is a difficult task. Interest in monitoring and using traces of user requests for fault detection has been on the rise recently. Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. In the past, many software reliability growth models (SRGM) have been proposed to estimate the reliability of software. In practice, effective debugging is difficult because the fault may not be immediately detected. Software developers need time to read and analyze the collected failure data. The time delayed by the fault detection & correction processes should not be ignored. Research in the field of software reliability has been extensive and various techniques have been developed for locating bugs. Different techniques are used together to provide good results for problems. Software reliability in web based application is more complex rather than other software systems. The customers of web application want more reliability during the use of application. So the reliability of the web based applications is considered as a special case over different kinds of network. This paper discussed the relevant aspect of reliability issues and some best practices of growth model when thinking in terms of web software reliability.

Keywords:- Fault detection, non-homogeneous Poisson process (NHPP), software reliability growth model (SRGM), software testing.

I. Introduction

A software fault refers to a defect in a system. An error is inconsistency between the observed performance of a system and its specified performance. A software failure occurs when the delivered product deviates from correct service and perform unexpected behavior from user requirements. A software fault or error may not necessarily cause a software failure. Fault detection is recognizing that a problem has occurred, even if you don't know the reason. Faults may be detected by a variety of quantitative or qualitative approaches. This includes many of the multivariable, model-based approaches. Fault diagnosis is investigating one or more root causes of problems to the point where corrective action can be taken. This is also referred to as "fault isolation", especially when need to show the distinction from fault detection. A "fault" or "problem does not have to be the result of a complete failure of a software product. In a process plant, root causes of non-optimal operation might be hardware failures but problems might also be caused by poor choice of operating targets, poor feedstock quality or human error. The following are the major classes of software faults:-

- Syntactic faults: interface faults and parameter faults called as syntactic faults.
- Semantic faults: inconsistent behavior and incorrect results called as semantic faults.
- Service faults: QoS faults, SLA(Service Level Agreement) related faults, and real-time violations are called service faults.
- Communication / interaction faults: time out and service unavailability is called communication or interaction faults.
- Exceptions: I/O related exceptions and security-related exceptions are called exception faults.

Approaches of Fault Detection:
There are many different approaches to detect and isolate faults. Because each approach has their benefits and limitations, maximum applications mix multiple approaches. We emphasize some of the key differentiating factors between the different techniques.

- **Model based reasoning:** When models of the observed system are used as a basis for fault detection and diagnosis, this is often referred to as "model based reasoning".One of the major difference in approaches to fault detection & diagnosis is whether or not explicit models are used, and what type of models are used.
- **Fault signatures, pattern recognition and classifiers: Pattern recognition** directly uses the observed symptoms of a problem and compares them to a set of known symptoms for each possible problem. The

"pattern", or "fault signature" can be represent as a vector (1 dimensional array) of symptoms for each defined fault.

- **Neural networks:-Neural networks** are nonlinear, multivariable models built from a set of input/output data. They can be used as event detectors to detect events and trends. They also used as diagnostic models in model-based reasoning, or directly used as classifiers for recognizing fault signatures.
- **Procedural/workflow approaches-modeling the decision process rather than the observed system:-**Some fault detection and diagnosis is handled by creating procedures for making decisions based on the observed data. This is direct modeling of the decision process rather than modeling the system being diagnosed.
- **Event-oriented fault detection, diagnosis and correlation:-**An event represents a change of state of a monitored object. Alarms are examples of events. Diagnostics involving events can be significantly different than diagnostics involving a fixed set of variables.
- **Passive system monitoring vs. active testing:-**In the case of online monitoring systems, many diagnostic techniques assume routine scanning of every variable of interest. But many times, it is preferable to request non-routine tests. Diagnosis for maintenance purposes is based on testing.
- **Rule-based approaches and implementations:-**Rule-based systems in most cases just implement other approaches discussed above, more as a program control mechanism than a separate diagnostic technique.
- **Hybrid approaches:-**Pattern recognition by itself does not require a model. However, input for construction of the signatures for the known failures may be based on models; for instance, as residuals from models of normal behavior. This general technique applies to static or dynamic models. For dynamic models, the patterns can be based on predicted measurement values vs. observed values in a Kalman filter, for example **Smartsignal** offers products based on an empirical process model of normal operation used for fault detection, combined with other techniques for fault isolation. Pattern recognition can also be combined with models of abnormal behavior. For instance, in the case of the SMARTS In Charge product, the modeling was in terms of a fault propagation model (qualitative cause/effect model of abnormal behavior). But as part of the development process, this model was then used to automatically construct fault signatures - a form of compiled knowledge. At run time, diagnosis was based on matching observed data to the nearest fault signature. So the product at run time had the characteristics of a pattern matching solution. So, the overall methodology is often a combination of pattern recognition with a model-based method. Some tools such as **GDA** are flexible enough to support multiple approaches to fault detection and diagnosis, and also support the upfront filtering and event generation as well. One conclusion was that most applications required a mix of techniques for success [1].

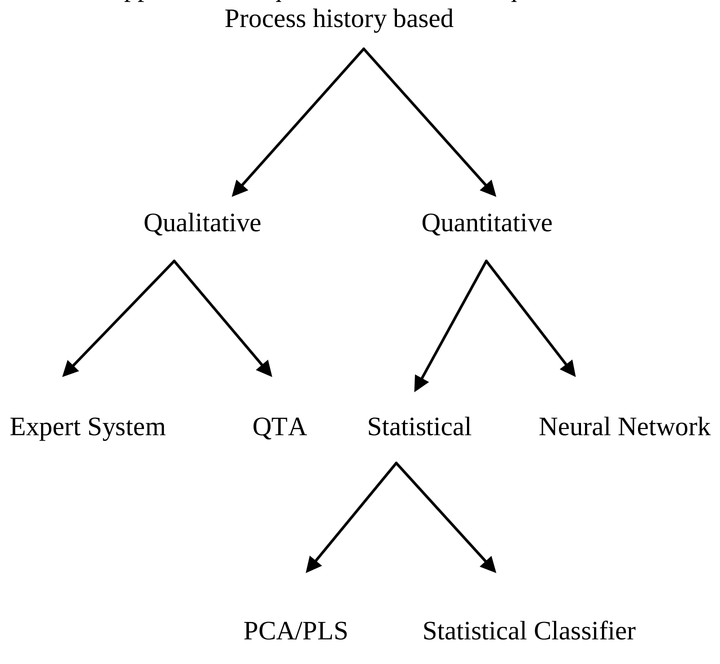

Fig 1 Architecture diagram

Strategy for detection of software faults

To detect the software faults, which have been generated during the development process, two different strategies may be applied:-

1. Static strategies 2. Dynamic strategies

- **Static strategies:-**Techniques guided by a static strategy do not require that the system is executed and may be applied at all stages of the development process. These techniques may be applied as formal reviews like inspections or automatic analyses of the code of a system or associated documents.

- **Dynamic strategies:-**Techniques guided by dynamic strategies ensure that a program is operationally correct which mean that the system is executed with test data. On the other hand, testing is only possible when a prototype or an executable version of a program is available. Both inspections and testing are activities contributing to validation and verification.

II. Software Based Fault Detection Technique

1. Algorithm Based Fault Tolerance (ABFT):- ABFT is used for detecting, locating, and correcting faults with a software procedure. It exploits the structure of numerical operations. This approach is effective but lacks of generality. It is well suited for applications using regular structures, and therefore it is used for a limited set of problems.

2. Assertions:- Assertions or the logic statements inserted at different points in the program reflect invariant relationships between the variables of the program and they often lead to various problems as assertions are not transparent to a programmer and their effectiveness depends on the nature of an application and on a programmer's ability.

3. Control Flow Checking (CFC):- The basic task of CFC is to partition an application program in basic blocks or the branch-free parts of code. A deterministic signature (or number) is assigned to each block and faults are detected by comparing the run-time signature with a pre-computed one. In most CFC techniques one of the major problems is to tune the test granularity that should be used.

4. Procedure Duplication (PD):- The programmer decides to duplicate the most critical procedures and to compare the obtained results on executing the procedures on two different processors. This approach requires a programmer to decide which procedures to be duplicated and to introduce proper checking on the results. These code modifications are done manually and might introduce errors. Code duplication for byte-error detection by a single processor.

5. Error Detection by Duplicated Instructions (EDDI):- Computation results from master and shadow instructions are compared before writing to memory. Upon mismatch, the program jumps to an error handler that will cause the program to restart. EDDI has high error coverage at the cost of performance penalty due to time redundancy as introduced into the system. Since we use general purpose registers as shadow registers, more register spilling occurs with EDDI. More spilling causes more performance overhead since it increases the number of memory operations.

6. Software Implemented Error Detection and Correction (EDAC):- Software Implemented EDAC approaches (e.g., Cyclic Redundancy Checks or CRC, Hamming Codes, Bose-Chaudhuri- Hocquenghem or BCH etc,) are effective in error detection but they suffer from very high time overhead. Hamming, BCH and RS codes have nice mathematical structures. However, there is a limitation when it comes to code lengths. These conventional error correcting codes namely, They have limitations and there exists very high time redundancy when they are implemented by software. When the check-bits become erroneous, the stored check bits do not match the computation and as a result, a block code fails. In general, checksum schemes fail when they are corrupted enough to transform to another valid code work (the distance of the code).

7. Periodic Memory Scrubbing:- This approach relies on periodic reloading of code on main memory from an immutable memory. This is effective for protecting the code segment of Operating system and application programs. Performance penalty is due to repetitive memory reading.
8. Masking Redundancy:- This approach means running an application in the presence of faults. Few processors are used to run the same program and vote to identify errors in any single processor. Errors can be masked from application software. No software rollbacks are required to fix errors.

9. Reconfiguration:- This means removing failed modules from the system. When failure occurs in a module, its effects on the remaining portion of the system is isolated. A large number of functional modules are used, which are switched automatically to replace a failing module.

10. Replication:- This ensures reliability but is expensive in terms of hardware or runtime cost. The idea is to take a majority vote on a calculation replicated N times. Its software solution requires each processor to run N copies of surrounding computations and then vote on the result. This slows down the computation by at least a factor of N.

11. Restore Architecture:- Transient errors or soft errors are detected through time redundancy in the ReStore architecture. The novelty of the ReStore architecture is the use of transient error symptoms, such as, memory protection violation and incorrect control flow etc. The tendency for these symptoms to occur quickly after a transient, coupled with a check pointing implementation in hardware to restore clean architectural state, enables a cost effective soft error detection and recovery solution.

12. Dual Modular Redundancy (DMR) & Backward-Error Recovery (BER) & Checkpoint:- Error is detected through differences in execution across a dual modular redundant (DMR) processor pair. DMR is a backward-error recovery (BER) technique where two processors are used to detect errors in execution. BER mechanisms create checkpoints of correct system state and rollback processor execution when an error is detected. A checkpoint of program state consists of a snapshot of architectural registers and memory values. A checkpoint logically represents a single point in time.

13. Triple Modular Redundancy (TMR) & Forward - Error Recovery (FER):- Three processors execute the same program and when one processor fails a majority vote, it determines the erroneous processor. TMR is the classic example of FER where enough redundancy exists in the system to determine the correct operation, should a processor fail.

14. Fingerprinting:- This mechanism detects differences in execution across a dual modular redundant (DMR) processor pair. It summarizes a processor's execution history in a hash-based signature. Differences between two mirrored processors are exposed by comparing their fingerprints.

15. Processor Status Word Tracking (PSWT):- During the execution of a program if at any point of time a PSW is found to be an invalid one then we say that an error has occurred. Invalid PSW means it does not match to any one of our known or valid PSWs in the PSW bank meant for that program.

16. Application Semantic Based Assertions (ASBA):- We apply various assertions that are derived from our understanding about the semantics of an application. Any violation at an assertion indicates an erroneous state.

17. Checksum & Parity:- They are effective for bit error detection but not suitable for error correction. The single parity checks can detect only odd number of single bit errors. Any even number of single bit-errors remains undetected. In a typical Checksum where n bytes are XORed and the result is stored in (n+1)th byte. Now if this byte itself is corrupted due to transients or in the case of even changes, the errors remain undetected by this typical Checksum.

18. Matrix Checksums:- By using typical checksums for each row and column in a matrix, we can detect erroneous element of a matrix. This is useful for detecting errors in application data.

19. Arithmetic Sum & Difference Checks for a pair of elements:- This approach is useful for detecting and correcting multiple bits errors in data words. Even all bits errors in a data word are corrected by this approach.

20. The NOP-PSW Approach:- This is useful for detecting transient errors or soft-errors in microprocessor registers, memory and stack area that might occur during the operational time at various industrial environments. This generalized approach does not need multiple processors and multiple software design. This is a single - version low-cost but an efficient (fast and having low memory-space overhead) approach for tolerating transient faults. Code size grows by 15% and execution time increases by 20.2% (as discussed in section-3). Such redundancy is negligibly small in comparison to other existing techniques. Memory As discussed This is also useful for processor hardening and transient susceptibility testing. This approach cannot detect all control flow errors. This is more efficient than the conventional software implemented EDAC, PD, scrubbing, masking, DMR or TMR, fingerprinting etc. This novel NOP-PSW approach is intended to be an efficient supplement one to be used along with other prevailing software-based fault tolerance approaches. This approach is very useful for designing fault tolerant microprocessor based systems using COTS components as the Electromagnetic Interference (EMI) or transients or radiation hardened components are very costly ones. The approach is also useful for software based fault avoidance [2].

III. Work Done

Claes Wohlin *et al.* [3] discussed three important problems (spreading, detection and costs). The spreading and detection model was used to discover the chances of software faults before testing phase. The cost model was used to estimate the effect of fault detection and fault removal during testing on the total cost. It was easier to develop cost-effective system by knowing the nature and consequence of software faults. These models were help in the planning of process and control of software products. By using these models, it was easier to uncover faults with low cost. So, this model focused on the cost minimization as well as effective fault elimination approaches.

Samaradasa Weerahandi *et al.* [4] proposed an approach to measure the quality of software system in its releasing process. The quality measurement plan was based on the detection of faults during software operation. The analysis of data was described by this approach and reported in a framework. The proposed approach was described with the help of example involving three releases of software product and the fault detection times were exponentially distributed. The procedure was useful for very little data in latest release of software product. A Bayesian approach was used to estimate the quality parameters for each release of software product. The exponential, Weibull or Pareto distributions were used for the detection of faults in specific data. The main focus of this paper was on the use of parametric models which provided the good results to find out the faulty data but the parametric models was not provided adequate results in some applications . So, the concept of nonparametric methods was used but the nonparametric methods were not used widely for the modeling of software reliability.

Ghani A. Kanawati *et al*. [5] proposed techniques to inject fault and errors in the software, for that purpose a flexible and automated tool FERRARI was developed. A number of experiments were used to present the capabilities of FERRARI. This tool was used to evaluate a prototype system under faults and its tolerance capability. It supported the injection of transient as well as permanent errors in the software. The guidelines and methodologies were used to improve the software performance and the analysis would help for better error or fault detection mechanism. The analysis would help in the design of better error detection mechanisms. The important finding of these experiments was that it would help in the fault detection of input/output routines and system libraries. The results have shown that the coverage of errors or faults was highly dependent on the choice of error detection models. The built in system detected more than 60% errors but this technique had some limitations also. It incurred time overhead and failure to inject faults in logic circuitry, unable to validate the complete embedded system. These disadvantages could be removed by the association of software and hardware techniques.

Sandro Morasca [6] proposed an approach based on reliability techniques to measure the fault detection process. The proposed approach found out the number of faults. There were two types of (static and dynamic) techniques used on the two important factors (ability of the process and quality of the artifact) to uncover all defects but these factors should be measured separately in a process. FDP was used in all software development phases to find out the errors in the software. Static technique like as inspection was used to detect fault directly while the dynamic technique like as testing was used to show the occurrence of faults in the first step and to find out the location of faults in the second phase. For the whole process, a fault-detection process assessment model was prepared. The model was valid if the FDP was unable to discover any type of error and there were no error in the artifacts. The Poisson distribution was used to count the number of faults in the software. The fault detection process assessment model could be applied to all artifacts during the software development.

Gregg Rothermel *et al.* [7] proposed different types of techniques to prioritize test suits for regression testing and checked which technique was best for fault detection during regression testing. Once the techniques prioritized, it becomes easier to detect faults by using the best technique and easily corrected earlier than the other techniques. The numbers of techniques were used for different purposes like as the techniques to cover complete code, the techniques to cover that code which is not previously covered etc. The results were analyzed to estimate the fault detection rate and cost of each technique. A large amount of effort was required to run all the test cases in a test suit. For this reason, the different types of techniques (Regression test selection and test suite minimization Techniques) were considered in a test suit to reduce the cost of regression testing. The different goals were accomplished by prioritized the techniques. The results have shown that the techniques could improve the rate of fault detection of test suits and help in the choice of least expensive techniques.

Nelly Delgado *et al.* [8] proposed a catalog which was used to monitor the software at run time. The wide range of runtime monitoring tools was classified by the use of taxonomy and used to observe the software behavior whether it performed its intended behavior or not. The monitoring would help in analyzing the software and recovery of faulty components because the properties were classified to discover faults before they become failure. The fault detection, diagnosis, fault recovery and performance analysis was performed by runtime software. The main aim of fault detection was to ensure that the all function performed their functionality in desired manner or not. The run-time fault monitoring was done by testing and theorem proving to examine whether the program executed correctly or not. The properties were defined by the specification languages and the result was stored by event handler. The complexity, cost and insufficiency of testing was interesting to minimize in the future.

James H. Andrews *et al.* [9] proposed mutation analysis to estimate and compare testing coverage approach. For checking the effectiveness of test suits, a statistical analysis was performed to check the empirical results. This was difficult to find out the real faults in software and validate the criterion that is why a large number of faults were injected in the software to compare the test suits on the software. The mutants were generated to show the fault detection effectively. The results were very consistent across the investigated criteria as they have shown that the use of mutation operators was trustworthy results. The benefit of the mutant generation was that the mutant operators described in accurate manner and the fault injection process implemented easily to detect faults. A large number of mutants decreased the impact of random variation in the analysis and allowed the use of different analysis approach. This approach focused on the choice of low cost as well as effectiveness of test suits.

Zheng. J. *et al.* [10] proposed the automated static analysis to detect faults in software. The number of techniques like as software review and testing were used to detect faults but every technique was not effective to detect errors. The static analysis tools were used to eliminate errors before the delivery of software. The static analysis was an inexpensive approach. The faults were detected manually and automatically in Nortel software products and there was no need of execution. Automated static analysis used orthogonal Defect Classification proposal to identify assignment and checking faults. The programming error which caused security vulnerabilities were also detected by automatic static analysis techniques. The results indicate that the automated static analysis was efficient technique to improve the quality of software in economic way. But there were some limitations also like as the results only provided by using three larger Nortel networks for c/c++.

Chin-Yu Huang *et al.* [11] proposed an analysis of software reliability by considering fault dependency and debugging time lag. It was assumed that the faults can be detected and corrected easily by use of mathematical models but it was not realistic. The model covered a range of SRGM (software reliability growth models) to detect and correct the faults in the software. Fault dependency and debugging time lag were considered in SRGM to predict the capability of approach. The goal of the purposed techniques was to minimize the cost of software development. These approaches were also used to decrease the complexity of computer related applications like as credit card and shared ATM systems, air traffic control systems and banking payment systems etc. The evaluation results have shown that the proposed framework for SRGM had a fairly accurate prediction capability.

Guofei Jiang *et al.* [12] proposed new and unique fault detection methods based on abnormal trace detection. It was difficult to detect faults in a large-scale distributed system. The other problem was how to represent the larger training trace data compactly as an oracle. Today, it was important to monitor and trace faults in the system. But it was a serious problem to trace all data efficiently. So, the varied-length n-grams and automata were used to trace all data in efficient and effective manner. An algorithm was developed to extract n-grams and to construct multi-resolution automata from the data. The generalization ability of automata was analyzed and a threshold was introduced to support multiresolution detection. Then the deterministic and multi-hypothesis algorithms were used to detect faults in a system. In the given approach, the faults were injected in real application to be tested and the experiment provided adequate results. The experimental results have shown that the algorithms could work very well in fault detection.

Wu.Y.P.*et al.* [13] considered the time availability to Model and Analyzed the Software Fault Detection and Correction Process. The modeling and estimation of software reliability was very important in software development because it was very difficult to find out the occurrence of faults in software. An extension of SRGM was proposed to improve the reliability. The fault detection and correction processes were very time consuming because it was very difficult to find out the location of fault and correct them in the software. The time dependencies incorporated by an approach between the fault detection and fault correction processes. The combined model of detection and correction processes was proposed to estimate the parameters for fault detection in software. The main focus of the proposed approach was on the time delay issues and based on the hypothesis of perfect debugging, equal testing efforts and no change point. The results have shown that the maximum likelihood (ML) provided the better prediction capability rather than the least square (LS) estimation methods. The analysis of optimal release time has shown that the fault debugging time improves the reliability and overall cost of the software.

Yanjun Shu *et al.* [14] proposed a new model to compare the fault detection and correction processes in analysis of the number of faults. The maximum SRGM used for fault detection but the fault correction was ignored .Because it was assume that the faults were eliminated immediately and efficiently but it was not realistic. The fault correction was difficult part because it was difficult to correct all faults in software. But in this paper, both fault detection and fault correction processes were considered. When the faults were detected, the faults were located and removed by doing some changes in the codes by fault correction process. A logistic function was used to model both fault detection and fault correction processes based on NHHP. The dependency of the two processes was described by the ratio of corrected fault number to detected fault number in software. A data set of software testing was used to evaluate the proposed models. Four paired models were selected for both fault detection and correction processes. The results have shown that the proposed model was useful for both fault detection and fault correction processes.

Chin-Yu Huang *et al.* [15] proposed finite and infinite server queuing models to examine software reliability. It was important to predict the reliability of software system whether all the functions performed in a desired manner or not. It was very difficult to find out the errors in software because the error may not be found out easily and the time was needed to examine and collect the faulty data. But the time should be minimized to detect and correct all faults in a system. The queuing system was used to explain and model the debugging process. The experiments were performed on proposed approach to predict the capability of error detection and correction processes. The problem of imperfect debugging also investigated where fixing one bug created another. The Experimental results have shown that the proposed Models gave a better fit to the observed data and predict future behavior well for the real failure data.

Zhou.C.*et al.* [16] proposed a combined approach to detect faults in embedded control software system. The observed data was monitored with the appropriate specifications at two levels (software level and controlled-system level). The embedded control system contains both software and hardware components. A model-based technique was used to detect errors during run time operation. The proposed approach was used to examine the behavior of software that the functionality was performed in desired manner or not. The software fault was detected when the observed behavior was rejected by a software level monitor. The input-output extended finite automata (I/OEFA) were used to isolate the faults in software. A number of existing techniques (Formal verification, N-version programming and exception handling) were used to detect faults but there were some limitations like as complexity and lack of complete specification etc. So, a two-tier technique was used to detect all faults in software. But the approach guarantied no false-alarm.

Simon Poulding *et al.* [17] proposed statistical testing using automated search to detect faults in software. Statistical and the structural testing were considered to detect faults. The statistical testing was better to detect faults than the structural and random testing because it considered the approach of both techniques to minimize the problems. This paper was based on the automated search methods to detect faults in real world applications. The uniform random testing was more efficient in large test sizes than statistical testing and the distributions have shown the diminished fault detection ability. The probability distribution was used to generate test data in samples. But it was very difficult to derive probability distribution for large and complex software. Search-Based Software Engineering (SBSE) was used to derive probability distribution in statistical testing which was implemented manually. It has shown that the technique was effective and practical for larger input domains.

Chin-Yu Huang *et al.* [18] proposed an effective, adaptable technique to predict and assess the software reliability in operational and testing phases. But it was important to note that the fault detection and elimination process were different during the software development and operation phase. For example, the fault elimination was slower during operation than the development phase. A number of SRGM (software reliability growth models) based on NHPP (non-homogeneous Poisson processes) could be extracted from an integrated theory. The multiple change-points were incorporated into software reliability model within the unified theory. The parameters of the proposed models were estimated by employing real software failure data and compared with other SRGM (Software reliability growth models). The numerical results were used to show that performance of proposed models was good and provided good reliability prediction in the development and operation stage. The approach was flexible to form different environment range from exponential-type to S-shaped NHPP models.

Kaustubh R. Joshi *et al.* [19] proposed a pure model-based approach to manage challenges in the distributed systems. Automatic system monitoring and recovery improved the dependability effectively in distributed software systems but it was very difficult in practice to find out the location of faults. The techniques considered with Bayesian estimation and Markov decision theory to provide recovery in a system. This approach explored more advantages by association of monitoring and recovery methods in comparison of using them separately. The faults were injected in realistic e-commerce systems to validate this approach. The dependability of distributed software system was improved by using of automatic system monitoring and recovery techniques. The recovery was very difficult because it was difficult to choose tools and techniques to find out the location of faults in distributed software system. The results have shown that the approach was of low cost and provided high availability solution for distributed system.

IV Conclusion

There are number of methods and techniques used to detect faults in software system but every technique has some limitations also. We mainly focus on the software reliability growth models. Software reliability growth models are applicable on late stages of testing to detect and correct the faults in software development. It provides useful information in predicting and improving the reliability of software products. SRGM also help to quantify software reliability and to understand how and why software faults was occur. SRGM involves the stochastic process for statistical analysis of software. In this survey, we studied how SRGM is used to calculate the time delay and minimize the cost of software products. We studied applications of SRGM model in some critical applications such as very large-scale commercial software, safety-critical flight software and online banking service. The proposed work is to apply any SRGM model on the web applications to test and calculate its reliability. Firstly, the web application will be tested whether there is any fault or not, then the SRGM model will be used to find out the rate of detect the faults and calculate the reliability of web applications.

Acknowledgment

I would like to express my special thanks and gratitude to my guide Mr. Karambir who gave me the golden opportunity to write this paper on the topic Review of Fault Detection Technique to Detect Faults and Improve the Reliability in Web Applications, which also helped me in carrying out an intensive research through which I came across so many new concepts and terminologies. I am really thankful to him.

References

[1] Greg Stanely and associates "A guide to fault detection and diagnosis", 2010-2013.
[2] Goutam Kumar Saha "Software-Implemented Fault Detection approach" , may 2008.
[3] Claes Wohlin and Ulf Korner "Software faults : spreading, detection and costs " Software Engineering Journal, 1990.
[4] Samaradasa Weerahandi and Robert E. Hausman "Software Quality Measurement Based on Fault-Detection Data" lEEE Transactions on Software Engineering,1994.
[5] Ghani A. Kanawati, Nasser A. Kanawati, and Jacob A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System", IEEE Transactions on Computers, Vol. 44, No. 2, February 1995
[6] Sandro Morasca "Assessment of Fault-Detection Processes: An Approach Based on Reliability Techniques" IEEE Transactions on Reliability, Vol. 45, NO. 4,1996 December.
[7] Gregg Rothermel , Roland H. Untch and Chengyun Chu, and Mary Jean Harrold, "Prioritizing Test Cases For Regression Testing", IEEE Transactions on Software Engineering, Vol. 27, No. 10, October 2001
[8] Nelly Delgado, Ann Quiroz Gates and Steve Roach "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools", IEEE Transactions on Software Engineering, Vol. 30, No. 12, December 2004

[9] James H. Andrews , Lionel C. Briand , Yvan Labiche and Akbar Siami Namin "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria", IEEE Transactions on Software Engineering, Vol. 32, No. 8, August 2006

[10] Zheng.J.,Williams.L.,Nagappan.N.Snipes.W.,Hudepohl.J.P and Vouk.M.A. "On the Value of Static Analysis for Fault Detection in Software" IEEE Transactions on Software Engineering , 2006.

[11] Chin-Yu Huang and Chu-Ti Lin "Software Reliability Analysis by Considering Fault Dependency and Debugging Time Lag", IEEE Transactions on Reliability, Vol. 55, No. 3, September 2006.

[12] Guofei Jiang, Haifeng Chen, Cristian Ungureanu, and Kenji Yoshihira "Multiresolution Abnormal Trace Detection Using Varied-Length *n*-Grams and Automata", IEEE Transactions on Systems, Man and Cybernetics—Part c: Applications and Reviews, Vol. 37, No. 1, January 2007.

[13] Wu.Y.P. , Hu.Q.P. ,Xie.M. And Ng.S.H "Modeling and Analysis of Software Fault Detection and Correction Process by Considering Time Dependency**"** IEEE Transaction on reliability, 2007.

[14] Yanjun Shu, Zhibo Wu, Hongwei Liu, and Xiaozong Yang "Considering the Dependency of Fault Detection and Correction in Software Reliability Modeling" International Conference on Computer Science and Software Engineering,2008.

[15]. Chin-Yu Huang and Wei-Chih Huang "Software Reliability Analysis and Measurement Using Finite and Infinite Server Queueing Models" IEEE Transactions on Reliability, Vol. 57, No. 1, March 2008.

[16] Zhou.C.,Kumar.R. and Jiang.S. " Hierarchical Fault Detection in Embedded Control Software"in Annual IEEE International Computer Software and Applications Conference,2008.

[17] Simon Poulding and John A. Clark "Efficient Software Verification:Statistical Testing Using Automated Search" , IEEE Transactions on Software Engineering, Vol. 36, No. 6, November/December 2010.

[18] Chin-Yu Huang and Michael R. Lyu "Estimation and Analysis of Some Generalized Multiple Change-Point Software Reliability Models", IEEE Transactions on Reliability, Vol. 60, No. 2, June2011.

[19] Kaustubh R. Joshi **,** Matti A. Hiltunen **,** William H. Sanders and Richard D. Schlichting "Probabilistic Model-Driven Recovery in Distributed Systems" IEEE Transactions on Dependable and Secure Computing, Vol. 8, No. 6, November/December 2011.