Novel Software Reliability Estimation Model for Altering Paradigms of Software Engineering

Ritika Wason[1], P.Ahmed[2] and M.Qasim Rafiq[3]

**[1] Department of MCA, Institute of Information Technology and Management
Delhi, India**

**[2] Department of Computer Science and Engineering, Sharda University
Greater Noida, Uttar Pradesh, India**

**[3] Department of Computer Engineering, Aligarh Muslim University
Aligarh, India**

Abstract

A number of different software engineering paradigms like Component-Based Software Engineering (CBSE), Autonomic Computing, Service-Oriented Computing (SOC), Fault-Tolerant Computing and many others are being researched currently. These paradigms denote a paradigm shift from the currently mainstream object-oriented paradigm and are altering the way we view, design, develop and exercise software. Though these paradigms indicate a major shift in the way we design and code software. However, we still rely on traditional reliability models for estimating the reliability of any of the above systems. This paper analyzes the underlying characteristics of these paradigms and proposes a novel Finite Automata Based Reliability model as a suitable model for estimating reliability of modern, complex, distributed and critical software applications. We further outline the basic framework for an intelligent, automata-based reliability model that can be used for accurate estimation of system reliability of software systems at any point in the software life cycle.

Keywords: *Software Reliability, Software Reliability Growth Model (SRGM), Automata-Based Software Reliability Model, Software Reliability Paradigm, Finite State Machine (FSM).*

1. Introduction

The current prerogative of the software engineering community is the production of dependable systems. A huge amount of human and financial resources have been devoted for the achievement of the same. The present advancements in computer software can be classified into three major categories, namely: Software Design Paradigms, Software Design Implementation Paradigms and Software Deployment. Software Deployment implies installation of software at customer site, performing its expected functionalities (deliver what the user expects) error free, by what percentage and for how long? This factor forms the basic idea of what is referred to as software reliability, which can be defined as the probability of failure-free software function for a specific period of time in a specific environment.

The ubiquitous approach for software reliability modeling is the posteriori, black box approach, which utilizes post-implementation data regarding the interactions of the software with its environment for estimating software reliability. The inaccuracy of the above needs no proof as despite numerous reliability estimation models available and decades of ongoing research for better practices to produce software with minimal failure the number and degree of software failures continues to multiply. Software engineering continues to confront unreliable or failure prone software. Hence, we can say that software reliability will continue to bug developers as long as we continue to produce unreliable software.

Software Reliability Research has gained momentum in the recent decades due to the increasing penetration of software-based utilities and the increasing demand for reliable software in almost every human endeavor. It has been observed that effective software measurement can play an important role in risk management during software development [17], which being a human-centric task is very much prone to errors. Therefore, the software testing and error handling processes are being increasingly supplemented by different formal verification techniques like model checking [9], [10], finite state machines [18], [21], [23], state based models [5], operational profiles etc [17].

This paper presents the advances and current paradigms in software reliability estimation and suggests a new framework that can help estimate reliability of any software system at any phase of its life accurately. The remainder of this paper is organized as follows. Section 2 discusses the current scenario of software engineering by

highlighting the new paradigms that promise reliable software systems. Section 3 analyzes the various issues and factors impeding software reliability. Section 4 proposes a new intelligent, self-learning paradigm for accurate reliability estimation of state-based software systems. Section 5 discusses the need and scope of the proposed model.

2. Software Engineering: Current Scenario

Software Engineering is an adolescent field. Numerous paradigms for problem solving and software development have evolved over a period of time driven by the increasing complexity of software systems. A solid foundation for all these new paradigms is grounded in the fact that all of them offer the promise of delivering reliable software applications with their own engineering processes and techniques. The major domains of modern research in software reliability estimation in the present times worth mentioning can be classified into those of fault tolerant systems [8], self-healing systems [19], [24], component-based architectural models [22], state-based models [5], [15] and model-based development [10] as depicted in Figure 1 below:

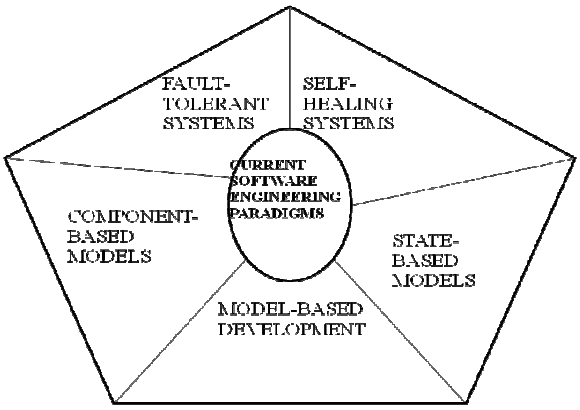

Figure 1: Current Software Engineering Paradigms

The fact that technical systems shall not work correctly or fail due to human or system errors during any phase of the software life cycle is well-known. A major dilemma for these systems is accurate reliability prediction, fault diagnosis and repair. With increasing complexity, heterogeneity and size of real-time software applications the reliability and maintainability (especially recovery from failure) issue of software systems has gained more importance. As a result the current research movement is directed towards achieving Autonomic or Self-Healing systems capable of managing themselves automatically without any human intervention [4], [6], [24]. The IBM autonomic computing initiative envisions a self-managing system which can self-configure, self-optimize, self-heal, and self-protect [4]. Autonomic systems attempt to automate the management of hardware, software and network infrastructures, hence alleviating the need of human intervention, thus reducing operational expenditures on software and increasing dependability, security and adaptability of software resources to varying workloads [6]. A great deal of research is being carried out independently as well as by IT giants like IBM to stimulate autonomic capabilities in their software products like IBMs WebSphere Virtual Enterprise, Oracle 11g etc to name a few.

A relatively naive domain with a far-sighted mission, autonomic systems can be broadly classified into two main categories namely Autonomic Computing and Autonomic Networking [4]. However, the realization of the underlying purpose of this paradigm lies in the merging of these two domains to give rise to a new model of Autonomic Systems. The research community is still uncertain about the future of this technique and many researchers reject it as a futuristic academic vision with no real application. However, this claim can be rejected considering the fact that many large IT giants like IBM, HP and Oracle have and are investing hundreds and millions of dollars to bring autonomic capabilities to many of their products. Though a complete autonomic system does not exist yet, autonomic technologies have already become a vital part of many important systems today.

Another simultaneous research endeavour is the domain of Self-Healing Systems [19], [24]. Self-Healing Systems attempt to heal themselves in order to recover from faults and regain normative performance levels independently without any human intervention [24]. Notable researches have been done for Self-Healing or Survivable systems under the domain of Fault Tolerant or Recovery Oriented Computing [8]. Many different architectural models have also been suggested to realize a self-healing system that can recover from abnormal ("unhealthy") state and return to the normative ("healthy") state it was before disruption. A detailed literature review and a classification based on similarities and relationships of self-healing systems have also been conducted [24]. Though most of the research on self-healing systems is still in infancy, [24] examines some prospective application areas for such systems.

For monitoring the self-CHOP properties of Autonomic or Self-Healing systems utilization of Aspect Oriented Programming (AOP) has been suggested as an alternative [7]. Such systems hold the key to the future as they can help realize our problem domain of realizing a self-learning system that can distinguish between a correct and an incorrect or failure state and once if in failure state can easily learn to retreat to a correct state [25].

Although autonomic computing is being projected as a paradigm that can help achieve reliable systems by eliminating human intervention, the various components for realizing such a system requires the effective interplay of many other design models. Some approaches have already been proposed to address the requirements of specific autonomic components, however Model-Driven Development is one comprehensive approach that deals with all areas of autonomic computing that relate to system dependability improvement: Self-configuring, Self-improving and Self-protecting. Model checking approach has been popularly used for property verification of systems [10]. The approach defines a system using a mathematical model, expressing the properties one wishes to prove for the system in a formal language and then verifies whether the model satisfies the formal property. Invented almost twenty-five years ago, the approach is fully automatic and has gained wide acceptance and is increasingly being used in commercial research and development units. The size of model increases exponentially in the number of variables or sub-models, preventing scalable automation. Model checking formal verification technique also find a number of applications like representation of programs in high-level programming languages, probabilistic or stochastic systems. The technique has already become a part of wide variety of commercial products like that of Microsoft. One of the most significant characteristic of the model-driven approach is that network components (e.g. nodes, channels and traffic) are not monitored randomly [10]. Rather they are monitored based upon the predicted reliability or security provided by the models. For example, the components that are predicted to have lower dependability at any given point in time are monitored more intensively than highly reliable components. Furthermore, the dependability is not static. As time passes a component that may initially be highly reliable could become more prone to failure. Therefore, the monitoring frequency of that component should be adapted accordingly over time. Failure correlations should also be integrated into the monitoring decision [8]. The major advantage of the model-driven design model is the fact that it provides for a reactive, event-driven healing mechanism along with a proactive, predictive technique to prevent failures before they occur. The reason behind the current emphasis on model-based reliability estimation methods is the fact that many researchers have unanimously accepted the fact that formal models aid in constructing software in a dependable manner [9],[10]. Such models help describe desired services precisely and compose them together consistently.

Traditional system-level reliability estimation techniques where the reliability of the application is estimated for the complete system as a whole are not suitable for most component-based applications as it does not consider compositional properties of systems and do not accommodate the reliability growth of individual components [22]. Hence, several reliability estimation techniques have been proposed to assess the reliability of component-based software. Approaches for random generation of faults in components using a programmatic procedure that returns the inter-failure arrival time of a given component have also been proposed [16]. The total number of failures is calculated for the application under simulation and its reliability estimation using a control flow graph for a program. Program Dependency Graphs and Fault Propagation Analysis for analytical reliability estimation of component-based applications have also been suggested [22]. However the approach is code-based as it generates dependency graphs from source code, which may not be available for off-the-shelf components. Most of the reliability estimation techniques for component-based software are path-based hence assuming independent component failure and thus providing a pessimistic estimate of system reliability.

In today's world, a complex system consists of many components any of which can fail or degrade. A performance analysis, assuming that all components are in perfect working conditions, may not be a good indicator of system's performance in practice. However, if probabilities are assigned to various operating or non-operating modes of the components, a more general analysis taking into account both performance and reliability aspects may be carried out [5]. This problem can be solved using different formalisms like Markov, semi-Markov reward models like (stochastic) Petri net models, process algebra models or systems of independent components. Independent component approach has also been suggested to represent the system by a set of independent components to model large communication networks [5]. By treating a system as a collection $C= \{c_1, \ldots, c_n\}$ of n independent, non-interacting components each of which can occupy a certain number of states (modes) with a given probability. Global state space is the Cartesian product of the component state sets and the probability of a global state is the product of the probabilities of the corresponding component states. The technique provides an improved estimate for bounding the expectations of a measure over the state space of a system composed of a large number of independent components with an arbitrary number of modes, when the values of the measure are known only on a subset of the state space. The technique has also been verified on two example applications as well as proved mathematically.

In times of increasing software complexity, popularity of component-based systems in handling diverse and critical

applications needs to be thoroughly evaluated. Sharma and Trivedi, [4] propose an architecture-based unified hierarchical model for software performance, reliability, security and cache behavior prediction using Discrete time Markov Chains (DTMC) to model various subsystems and components.
A novel layered formal model to realize a strategy for service realization typically for large scale software development was proposed in [2]. The formal model better known as Service Oriented Computing (SOC) has its basics in Service Oriented Architecture (SOA) which originated from component-based design and Hoare's logic [20]. The model is implemented as a two-layered approach, the top one for service deployment and the bottom one for service realization. However, the main advantage of scenario oriented design is the fact that its design is logically decoupled from any service caller (loose coupling) and hence can be easily implemented by different clients.

Many real-time applications are safety-critical; hence they are often constructed as fault tolerant systems [8], [9]. There are several works considering fault tolerant behavior of real-time applications [8].

In current scenario, software development does not follow the software engineering paradigm [9]. While there are reasonably well-defined software components, the developer has no way to analyze the quality, dependability and reliability of a composite design in terms of its individual parts. The ability to calculate the reliability of a software system in the early hours of its development, e.g., during architectural design, can help to improve the system's quality in a cost-effective manner. Recent work in assessing system reliability is motivated by the fact that building reliable software systems requires understanding reliability at the architectural level. All these approaches acknowledge that individual component reliabilities have a significant impact on system reliability. However, almost all of them invariably assume that the reliabilities of the individual components in a system are known. It however remains unclear how the individual component reliabilities are assumed? The real challenge in realizing a truly dependable autonomic system lies in weaving together the best ideas from each of the above discussed models into a cohesive and complete system and building new ideas into the system.

Though all the above models promise reliable, intelligent, fault-tolerant systems, complete realization of any software with all such characteristics requires a close interplay of all these models in a well-defined manner using a strong theoretical automata-based reliability model that can be mathematically verified and has its roots in state-based approach.

The main reason for suggesting an automata-based reliability model as the model for accurate reliability estimations and self-learning failure conditions is motivated by the fact that a Finite State Machine is a mathematically defined object that can provide structured and precise understanding of what is going on in systems represented as complex state machines. The major advantage of this formal model for software system representation is the fact that any system can be easily represented as a control flow graph consisting of a number of states and transitions which may further result in some particular states. Petri nets and Finite Automata have also been few of the earliest models used for program analysis and verification techniques including seminal work by Robert Floyd, Tony Hoare and Edsger Dijkstra [1], [20].

Many different formal design models have also been adopted to ensure good quality software (which can heal themselves from system faults) which is more reliable as compared to counterpart software systems designed and developed using traditional formal design techniques. From the ever increasing set of new formal design techniques, finite state machine (FSM) based techniques that treat the software as a finite state machine and then try to handle all states software can acquire during its life as valid and invalid states, have succeeded in achieving sufficiently reliable systems in many varied domains. FSM-based models and its numerous extensions have found extensive use in designing and testing different kinds of systems.

The advantage of the ongoing work at the international platform is the fact that it is not carrying reliability research in one particular domain but instead has well-acknowledged the scenario that extremely wide variability of allegedly identical components under supposedly identical environmental and operating conditions. Keeping in view the above fact the researchers in the international community have identified different promising domains that can help in giving accurate reliability estimates of different software systems designed and built under differing operating conditions. Most of these models like fault-tolerant systems, self-healing systems, model-based software have succeeded in providing accurate reliability estimates for different software systems. However a generalised, self-learning, fault-tolerant model that can be uniformly applied to varied real-time software systems is still a far-fetched dream that shall require combination of different models discussed above.

3. Critical Factors and Issues Challenging Software Reliability Modeling Science

Faults in a system are inevitable. Software developers spend about 80% of system development costs on identifying and correcting system faults. These faults may be the result of wrong practices during any phase of software design and development, but may eventually propagate into failure and may have severe consequences. In traditional system design methodologies, the developer generally works on a specification of the expected behavior, hence ending up generating erroneous code which shall eventually fail at some instance.

Ever-increasing complexity of modern engineering products and systems further ensures that system failure may not always be a result of component part failure [26]. Many other factors may also influence system failure rate, like:

i) Failure of individual software elements.
ii) Failure due to human factors/ operating documentation.
iii) Failure due to environmental factors
iv) Common mode failure, where redundancy is defeated by factors common to replicated units.

A combination of one or more of the above factors combined with factors like failure rate, probability of occurrence, time etc work on almost all software systems under test and operation, hence leading to inaccuracy of software reliability predictions and estimates done using the traditional models and techniques [25].

The common underlying assumption of the popular reliability estimation models is that software failures occur randomly in time. However the assumption has never been verified quantitatively and the defective, inaccurate estimates of these models need no proof. Further another important issue that affects reliability of current software systems is the fact that the overall system reliability is a composite of individual component reliabilities which may be hardware as well as software components and whose reliabilities may be affected by their operational profiles as well as user interactions. As a result, estimation of system reliability at any point in software life cycle is interplay of many factors as well as components of the software. Hence traditional reliability estimation techniques fail to accurately estimate system reliability or even component reliability as they overlook many of the above factors.

The pace of the introduction of new paradigms in Software Engineering and an examination of the underlying aim of paradigms discussed in Section 2 indicates that reliable software systems are no longer a flexible alternative. The Software Industry is now striving for a completely reliable software system that can function as expected irrespective of the operating conditions. To achieve such systems, the software industry is mending its way of designing and implementing software by adopting new paradigms for software development. However, with improved practices of software development what is also required is a generic model for estimating reliability of a software system during any phase of the system's life. The traditional reliability models with their unrealistic assumptions fail to fulfill the above requirement. Hence, we propose a new automata-based framework for reliability estimation in the next Section. This framework with its mathematical foundations in automata- theory shall provide a sound framework for the development of an intelligent, generic software reliability estimation model that will estimate system reliability taking into account the system structure.

4. Proposed Framework

Many different Software Reliability Growth Models (SRGMs) and practices have been proposed time and again by different researchers [6], [7], [8], [9], [15], [16], [18], [22]. All these available reliability estimation models have attempted to predict reliability of software without any regard to its internal structure. The reliability estimation measures are all based on the assumption that reliability is the absence of failures from a system. Contrastingly, they quantify reliability using some kind of failure data (brute force). Further, we can classify the current reliability estimation techniques as either a priori technique (build the software right) or a posteriori techniques (right the wrongs). Much of the current practice today is in a posteriori techniques. We build software that's not very good and through brute force, debug it into correctness [4]. By shifting some of the balance towards a priori efforts; we can go a long way towards correcting some of the most serious problems.

None of the paradigms discussed in Section 2 can individually suffice to provide the breakthrough that the field of software engineering requires today. But if all of them are taken seriously and we succeed in combining them, we may be able to realize major advances in terms of achieving a generalized model for accurate reliability estimation of software systems.

We now propose an automata-based formal reliability model which besides being used for reliability predictions can act as a self-learning model that can handle failure conditions in real time. The proposed model has its basis in network reliability estimation studies [5] and software reliability estimation models for component-based software systems [22].

We hypothesize that a state-based approach for software representation can be used to represent all software along with its component systems. This state-based software

system representation shall be based on an examination of the internal software structure to identify the states of software system during its operation.

To validate the above hypothesis, we further propose the development of a formal model that can help in guiding and monitoring the design and implementation of software to control system reliability at any point in its life. This formal model can achieve the above objective in different stages as depicted in Figure 2 below:

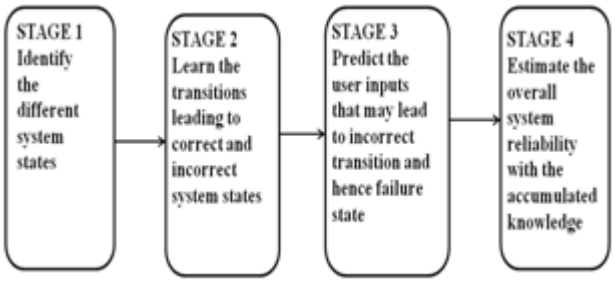

Figure 2: Proposed stages for software reliability modelling

The proposed model in Figure 2 shall first analyze a software system with respect to its internal structure to identify the different states software can acquire during its life cycle (Stage 1). Further in Stage 2 the model shall learn the different system transitions that can lead to each of the identified state. Once this knowledge about a software system is acquired the model in Stage 3 shall identify which user inputs or actions may result in an incorrect system state or the failure state. All this information will then be utilized by this model to estimate system reliability to estimate the reliability of its component parts (Stage 4). The proposed model shall be an intelligent, self-learning software reliability model that can easily detect its error state, register the particular state and the transition that led to such a state in its memory and never repeat the transition that leads to the particular error state.

The above stages for the proposed framework can be realized as a software reliability estimation model depicted in Figure 3. The model initiates monitoring a software system by parsing it into its component sub-systems which can further be represented as a cluster of inter-connected nodes (State-based Software Representation Phase). After representation the framework should be able to compute all possible independent paths through the system and also accumulate knowledge regarding which transitions could lead to undesirable failure states (Knowledge-Acquisition Phase). In its knowledge implementation phase the framework should be able to utilize its accumulated knowledge to ensure operationally reliable software at any point of the software life cycle.

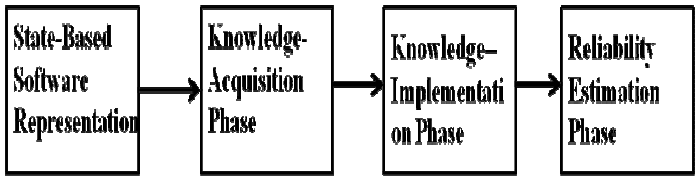

Figure 3: Phases of Automata-Based Software Reliability Model

The working of the proposed model can be understood with the depiction in Figure 4. If each individually compilable software component can be represented as a cluster of nodes. Further, if each node in the cluster marked as the Learning Cluster is achievable from the initial node. Then each of these nodes may finally result in either a correct node or error node. If each such correct and failure node is assigned a probability, then the reliability of the whole cluster can be defined as the sum of probabilities of the correct nodes.

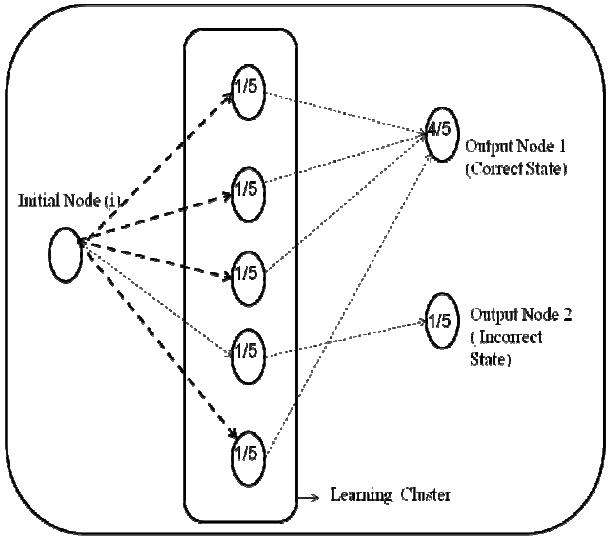

Figure 4: Representation of a Software Block/ Component / Module as a set of distinct states

Further if the whole system is represented as a combination of such integrating and interacting clusters then the reliability of the whole system can also be calculated based on the individual component reliabilities of each of these clusters. The above model treats software as a finite state machine and hence the proposed design is simple and easy to understand. The actual software implementation of this model is being worked out at the time of this writing.

5. Conclusion

An effective observation from this study is the fact that software engineering in the present times is no longer a static science that accommodates current real-time software complexities and demands using traditional software life cycle models. The study establishes the fact that only a formal automata-based reliability model can be successful in providing accurate reliability estimates for next-generation autonomic, component-based, fault-tolerant, self-healing, service-oriented systems. However, to realize such a model we require an effective monitoring as well as self-learning model that can learn different states acquired by system components during its life and then apply this knowledge for estimating system reliability at any point on its life or for recovering from a fault.

References

[1] K.M.S. Faqih, "What is Hampering the Performance of Software Reliability Models? A Literature Review",In International MultiConference of Engineers and Computer Scientists, 2009.

[2] W.T. Tsai, "Service-Oriented System Engineering: A New Paradigm", In IEEE International Workshop on Service-Oriented System Engineering, 2005.

[3] V.S. Sharma, and K.S Trivedi, "Quantifying software performance, reliability and security: An Architecture-Based Approach", The Journal of Systems and Software, 2007, vol 80, pp. 493-509.

[4] Y.S. Dai, T. Marshall, and X. Guan," Autonomic and Dependable Systems: Moving Towards a Model-Driven Approach", Journal of Computer Science, 2006.

[5] J. Bowles, "A Model for Assessing Computer Network Reliability", IEEE Proceedings, 1989.

[6] H. Chan, and T. Chieu, "An approach to monitor application states for self-managing (autonomic) systems", in 18th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages and Applications, 2003, pp. 312-313.

[7] Y.S. Dai, M. Xie and K.L. Poh, "Markov renewal models for correlated software failures of multiple types", IEEE Transactions on Reliability, 2005, Vol. 54, pp. 100-106.

[8] L. Waszniowski, J. Krakora, and Z. Hanzalek, "Case Study on Distributed and Fault Tolerant System Modeling based on Timed Automata", The Journal of Systems and Software, 2009, Vol. 82 pp. 1678-1694.

[9] B. Yang, X. Li, M. Xie and F. Tan, "A generic data-driven software reliability model with model mining technique", Reliability Engineering and System Safety, 2010, Vol. 95, pp. 671-678.

[10] M. Huth, "Some current topics in model checking", Intl. Journal Software Tools Technology Transfer, 2007, Vol 9, pp. 25-36.

[11] B. Littlewood, "How to measure Software Reliability and How Not To", IEEE Transactions on Reliability, 1879, Vol. 28, No. 2, pp. 103-110.

[12] B. Littlewood, "MTBF is meaningless in software reliability", (letter) IEEE Trans. Reliability, vol R-24, 1975, pp. 82.

[13] B. Littlewood, "Theories of Software Reliability: How Good Are They and How Can They Be Improved?", IEEE Transactions on Software Engineering, SE 6, 1980, pp.489-500.

[14] A.L Goel, "Software Reliability Models: Assumptions, Limitations and Applicability", IEEE Transactions on Software Engineering, vol SE11, No. 12, 1985, pp.1411-1423.

[15] Juncao Li, "An Automata Theoretic Approach to Hardware/Software Co verification", Ph.D. thesis, Portland State University, 2010.

[16] S. Gokhale, "Accurate Reliability Prediction Based on Software Structure", http:// www.engr.uconn. edu/ ~ssg/ cse300/ 397-232.pdf.

[17] N. Fenton, P. Krause, and M. Neil, "Software Measurement: Uncertainty and Causal Modeling", IEEE Software, 2002, vol. 19(4), pp. 116-122.

[18] T.Carmely, "Using Finite State Machines to Design Software", Embedded Systems Design, July 2010, vol.23, Ed.6

[19] D. Ghosh et. al, "Self-Healing Systems- Survey and Synthesis", Decision Support Systems, 2007, vol.42, pp. 2164-2185.

[20] C.A.R Hoare, "An Axiomatic Basis for Computer Programming", Communications of the ACM, 1969, vol 12(10), pp. 576-583.

[21] T.S Chow, "Testing software design modeled by finite state machines", IEEE Transactions on Software Engineering, 1978, vol.**4** (3), pp. 178-187.

[22] R.H Reusnner, H.W Schmidt, and I.H Poernomo, "Reliability prediction for component-based software architectures", The Journal of Systems and Software, 2003, vol. 66, pp. 241-252.

[23] T.Carmely, " Using Finite State Machines to Design Software", Embedded Systems Design, Vol.23, No.6, 2009.

[24] Ghosh et al., "Self-Healing Systems- Survey and Synthesis", Decision Support Systems, vol. 42, 2007, pp. 2164-2185.

[25] A.I. Maniu, "Reliability and its Quantitative Measures", Informatica Economica, vol.14, 2010, pp. 7-12.
[26] D.J Smith, "Reliability, Maintainability and Risk", Butterworth-Heinemann.

Ritika Wason is a Ph.D Computer Science scholar in Sharda University. She received her M.Phil (Computer Science) in 2009 and MCA in 2008. She is an Assistant Professor with Institute of Information Technology and Management since 2008 and a Life-Member of Computer Society of India. Authored three books on Software Testing and another on .NET. She also has many national and international research papers to her credit.

Dr. P.Ahmed is a Professor and Head of the Computer Science and Engineering Department at School of Engineering and Technology, Sharda University, India. He received his Ph.D from Concordia University, Montreal, Quebec, Canada in 1986 and has also been a senior software designer at PHILIPS/MICOM, Montreal, Canada; research fellow (MRI Imaging) at Montreal Neurological Institute, McGill University, Canada and visiting scientist, Centre for Pattern Recognition and Machine Intelligence (CENPARMI), Montreal, Canada. His research interests include Pattern Recognition, Machine Intelligence, Neural Network, Shape Descriptors, Visual Programming and Programming by Demonstration (Software Engineering) and Intelligent Systems. The research has a strong empirical focus resulting into many papers published in international journals and conference proceedings. He is also a lifetime member of Computer Society of India and member IEEE Computer Society (USA). He has also been an associate editor of the Computer Journal of the King Saud University, Riyadh, Saudi Arabia.

Dr. M. Qasim Rafiq is a Professor and Head of Computer Engineering Department at Aligarh Muslim University, India.