Chapter 1

Introduction

Software has become an essential part of industry, medical systems, spacecraft and military systems, and many other commercial systems. The application of software in these safety critical systems has led software reliability to be an important research area [1, 2, 3]. Software reliability denotes the probability of failure free operation of software under pre-defined conditions for a specified duration. The failure of software in safety critical systems may lead to risk significant events. We illustrate only some of the catastrophic accidents that had taken place due to the failure of software to realize the essence of software reliability:

1. Therac -25:

   The Therac-25 was a radiation therapy machine produced by Atomic Energy of Canada Limited (AECL) after the Therac-6 and Therac-20 units (the earlier units had been produced in partnership with CGR of France).

   It was involved in at least six accidents between 1985 and 1987, in which patients were given massive overdoses of radiation, approximately 100 times the intended dose.[4]:425 These accidents highlighted the dangers of software control of safety-critical systems, and they have become a standard case study in health informatics and software engineering.

2. Crash of Air France Flight 447 [5]:

On May 31, 2009, an Airbus A330-200 departed from the Rio de Janeiro-Galeão International Airport due to arrive at Paris 11 hours later. It crashed into the Atlantic Ocean on June 1. The aircraft was carrying 216 passengers, and 12 crew members, all of whom are presumed to be dead. In addition to the loss of the aircraft itself, Air France announced that each victim's family would be paid roughly €17,500 in initial compensation. This accident makes it the deadliest disaster for Air France, surpassing the Air France Flight 4590 in 2000 that killed 109 people.

Regarding software related contributing factors, the onboard automating reporting system transmitted several messages regarding discrepancies in the indicated air speed (IAS) readings before the aircraft disappeared. In total, 24 error messages were generated as systems failed across the aircraft. On June 4, 2009 (three days after the crash of Flight 447), Airbus issued an Accident Information Telex to operators of all Airbus models reminding pilots of the recommended Abnormal and Emergency Procedures to be taken in the case of unreliable airspeed indication. Efforts to retrieve the flight data recorders, critical to determining the exact cause of the crash, will resume in February 2010, but the chance of recovery is low. A final report on the accident is expected to be issued by the end of 2010.

3. Emergency-Shutdown of the Hatch Nuclear Power Plant [6, 7]:

   The Edwin I. Hatch nuclear power plant was recently forced into an emergency shutdown for 48 hours after a software update was installed on a computer. The accident occurred on March 7, 2008 after an engineer installed a software update on a computer operating on the plant's business network. The software update was designed to synchronize data on both the business system computer, and the control system computer. According to a report filed with the Nuclear Regulatory Commission (NRC), when the updated computer rebooted, it reset the data on the control system, causing safety systems to errantly interpret the lack of data as a drop in water reservoirs that cool the plant's radioactive nuclear fuel rods.

   As a result, automated safety systems at the plant triggered a shutdown. Estimating a loss based on electricity prices, the total cost to purchase electricity from external sources during the 48-hour shutdown period would be approximately $5 million. This does not include other operation-related costs.

The main cause of this accident was the installation of a software upgrade on a computer, without the knowledge of its impact on the entire system. System boundaries should be carefully defined, and the business network should be external to what might be safety critical. Although communication can be allowed between the business and control networks, it is better to accomplish this using a query mechanism rather than granting the former unrestricted access to the latter.

4. Loss of Communication between the FAA Air Traffic Control Center, and Airplanes [8,9]:

   On Tuesday, September 14, 2004, the Los Angeles International Airport, and other airports in the region suspended operations due to a failure of the FAA radio system in Palmdale, California. Technicians onsite failed to perform the periodic maintenance check that must occur every 30 days, and the system shut down without warning as a result. The controllers lost contact with the planes when the main voice communications system shut down unexpectedly. Compounding this situation was the almost immediate crash of a backup system that was supposed to take over in such an event. The outage disrupted about 600 flights (including 150 cancellations), impacting over 30,000 passengers. Flights through the airspace controlled by the Palmdale facility were either grounded or rerouted elsewhere. Two airplane accidents almost occurred, and countless lives were at risk.

   A bug in a Microsoft system compounded by human error was ultimately responsible for the three-hour radio breakdown. A Microsoft-based replacement for an older Unix system needed to be reset approximately every 50 days to prevent data overload. A technician failed to perform the reset at the right time, and an internal clock within the system subsequently shut it down. In addition, a backup system also failed. When a system has a known problem, it is never a good idea to continue operation. Instead of relying on an improvised workaround, the bug in the software should be corrected as soon as possible to avoid a potential crisis. Learning from this experience, the FAA deployed a software patch which now addresses this issue. If the backup had been operating correctly, this accident would not have occurred.

For systems where a high degree of safety is of utmost concern, solid redundancies should always be in place.

5. Shutdown of the Hartsfield-Jackson Atlanta International Airport [10]:

   Hartsfield-Jackson Atlanta International Airport is one of the world's busiest airports, both in terms of passengers, and number of flights. The alertness of the security screeners is tested by the random appearance of artificial bombs or other suspicious hard-to-detect devices on the X-ray machine displays, followed by a brief delay, then a message indicating that it was a test. On April 19, 2006, an employee of the Transportation Security Administration (TSA), U.S. Department of Homeland Security, identified the image of a suspicious device, but did not realize it was part of the routine testing for security screeners because the software failed to indicate such a test was underway. As a result, the airport authorities evacuated the security area for two hours while searching for the suspicious device, causing more than 120 flight delays, and forcing many travelers to wait outside the airport.

   The false alarm was clearly due to a software malfunction which failed to alert the screeners by showing a warning message to indicate that the image of the suspicious device was just a test. Whenever a critical software system is using sample data for testing purposes, it should not create unnecessary suspicion among the end users. This sample data should be carefully chosen, and a thorough controlled testing should be conducted to ensure that an appropriate test alert message is displayed.

Therefore, practitioners always want an assessment of the software reliability (or quality) and wish to know when to reach to desired target. In order to assess software reliability and assure software quality, one of methods is to apply Software Reliability Growth Models (SRGM). SRGMs can describe failures as a random process, which is characterized in either times of failures or the number of failures at fixed times [11-13]. In general, SRGMs can provide very useful information about how to improve the reliability of software products. It is now well-recognized that reliability growth models are better described by a NHPP. Numerous SRGMs have been proposed, and some appear to be overall better than others. Unfortunately, models that are overall good are not always the best choice for a particular data set, and it is not possible to know which model to use a priori. Even when an appropriate model is used, the predictions made by a model may still be less accurate than desired due to the following reasons:

1. These approaches are black-box based i.e., the software system is considered as a whole and only its interactions with the outside world are modeled, without looking into its internal structure.

2. There is no single SGRM that can be fit to model the failure process of all the types of software.

3. Software systems are being developed in a heterogeneous fashion using components developed in-house, contractually, or picked off-the-shelf, and hence it may be inappropriate to model the overall failure process of such systems using the existing software reliability growth models [14].

4. SGRMs are based on unrealistic assumptions. From our previous studies [15-16], several conventional SRGMs can be unified under a general formulation.

1.1 Need for early prediction of reliability for software systems

The assessment of software reliability based on SGRM will often be too late. These are mostly employed at the later stage of the software development life cycle, before which many critical design decisions are made. Identification of significant problems during implementation or operation can lead to re-engineering of large parts of the system, which has been shown to be prohibitively costly. Therefore, quality attributes must be "built into" the software system throughout design and development, and particularly during architectural design.

The above suggests that building reliable software systems requires understanding reliability at the architectural level. Accessing software reliability during the early stages of SDLC is termed as Software Reliability Early Prediction. There are following advantages of early prediction of software reliability:

1. The impact of proposed design changes on reliability is determined by comparing the reliability predictions of the existing and proposed designs.
2. The ability of the design to maintain an acceptable reliability level under environmental extremes can be accessed through reliability predictions.
3. Assess the reliability earlier to take corrective action if the reliability does not meet the desired expectations.
4. Assess the reliability of an operational application to identify components that provide the highest potential for reliability improvement.

1.2 Motivation of Research

After realizing the substantial benefits of early prediction of software reliability, several recent approaches have begun to quantify software reliability at the level of architectural models, or at least in terms of high-level system structure [17-22]. Some of them are based on Markov Model (explained in chapter 2). A model should include all the functional requirements. Generally, software fails because of ambiguous or incomplete requirements or due to the defect in software design. Ambiguous requirements also penetrates defect in the design. Therefore, a reliability model must contain precise requirements. There is a requirement that the constructed reliability model should take care of all the software requirements and hence should be explainable to all the stakeholders. The existing models are not easily explainable to all the stakeholders, especially to the clients, who may not have knowledge about modeling. Hence, clients whole are the source of requirements cannot know whether the constructed software reliability model have taken care of all the requirements. These issues have been addressed in the present research work.

The approaches that are based on Markov model, for quantification of reliability during architectural design phase, are known as Markov reliability models. Markov models fail to model the concurrency and are limited to model to the probability of changes in the system with the exponential distribution. Markov reliability models are in the classification of probabilistic models due to the uncertainties involved in their input and hence output parameters. Improper treatment of these uncertainties can lead to the inaccurate results, which can make the wrong decisions. Consideration of uncertainty in the analysis gives insights into decision making by giving optimistic and pessimistic sets of solutions. Acceptable degree of confidence in the results can only be achieved by proper management of uncertainty. In case of Markov reliability model, the input parameter is the transition probabilities in between the states of Markov chain. The reliability of the software is the function of the transition probabilities in between the states of the Markov Model. There can be two types of states in Markov Chain, behavioral and failure. Many researchers, academicians and practicing engineers in various fields worked extensively to develop methods for carrying out the computation of transition probabilities. In these approaches, authors have computed them analytically based on some assumptions and hence do not give accurate prediction, which results inaccurate reliability prediction. Therefore there is an essential requirement to address this issue, which has been addressed in the present research work.

Apart from it, the reliability requirements of every component of the software may not necessarily be same due to the variance in the criticality of every component. The reliability of overall software system is a function of the reliabilities of each component of the system. The early assessment of the impact of change in reliability of any component on the reliabilities on its associated components is useful to take preventive actions (during design), like incorporating redundancy, diversity, etc. Also, if the change in reliabilities, on the basis of test data, of every component can be informed, the same impact analysis mechanism can be used to assess the overall system reliability to take corrective action, in case the target reliability is deviated. These issues have been addressed in the present research work.

1.3 Objectives of the present work

The issues related to early prediction of reliability of software systems are addressed in this work. It is the requirement phase, where the fault can get embed at the first point. Faults are propagating in nature. The faults of the requirement phase propagate in the design phase, thereafter in the implementation phase. It is impossible to discover all the faults in the testing phase and hence that will lead to a faulty system, which, when gets uncovered in operational phase, can lead to catastrophic failures. Similarly faults in design, implementation and testing phase will lead to a faulty system.

Therefore researchers are continuously putting effort in proposing methods for assessment of software reliability in the early phase of software development life cycle (SDLC). These methods include the approaches that can be applied in requirements analysis and design phase. The objective of the present work is to address the reliability related issues during the architectural design phase and project its benefit for taking preventive and corrective actions.

1.3.1 Uncertainty in Markov reliability models

There are several methods which can be used for early prediction of software reliability, that are based on Markov models. A software reliability model must consider all the functional and non-functional requirements. Missing requirements will result into an unreliable system. There should be a method to verify the incorporation of all the requirements in Markov reliability model. In the existing literature, there is no method through which the incorporation of all the requirements can be verified. In order to account for such uncertainty, sophisticated reliability modeling techniques are necessary. This issue is addressed in this research work.

1.3.2 Uncertainty in input parameters in Markov reliability models and their limitations

The reliability based on Markov models is a function of the transition probabilities in between the states of Markov chain. In the existing approaches, these probabilities are either assumed or computed based on unrealistic assumptions. Since inaccurate transition probabilities will give inaccurate reliability figure, there is a need to address these issues, if the reliability analysis is to serve as a tool in the decision making process. Moreover Markov reliability models are limited to model the probability of changes in the system with exponential distributions and incapable to model many facets like concurrency, multithreading. These issues are addressed in this research work.

1.3.3 Impact analysis of component reliability on the reliability of a software system

A software system consists of many software components. The reliability of a software system is the function of the reliabilities of each component. In case of early prediction, sensitivity analysis of component's reliability on the reliability of its associated components, and hence on the overall system, is useful to take preventive action for the assurance of the reliability of the overall software system. The reliability depends on the failure rate and hence, the reliability of the components of a software system gets changed in the operational phase. The technique for sensitivity analysis will be proved beneficial to take corrective action as well, if there is any mechanism through which the change in the reliability of the software components can be notified to the maintainer to take corrective action like repair or replacement of the faulty component. Further, it will be beneficial to a large extent, in case a software system consist one or many COTS components. All these issues are addressed in this work.

1.4 Scope of the Research

1. The proposed methodologies are applicable to any kind of software systems and the case studies have been focused only on safety systems of NPP.

2. All the proposed models are available in the form of analytical expression for the quantification of associated parameters.

3. In case of embedded software systems, the wear and tear out of the chip in which software resides has not been considered.

1.5 Thesis Outline

This thesis is structured as follows:

In Chapter 2, literature overview for the work that is proposed in this thesis is given. The classification of CBS is described. It covers the concepts and software reliability and its early prediction.

Based on the literature survey carried out, certain unavoidable limitations related to the existing software reliability models, for early prediction, are identified. The impractical issues and their planned strategies have been brought out in Chapter 3.

In Chapter 4, by describing the power of UML modeling in context of software requirements capturing and analysis, an approach to predict system reliability, by extending the UML is proposed. The validation of the proposed approach through a control logic component of safety critical system of Indian Nuclear Power Plant is shown. This gives the Markov Model, which can be used to predict the reliability of the system using Cheung's method.

In Chapter 5, the issues in the Markov reliability model are discussed and solutions to address them are proposed. A framework to predict the transition probabilities in between the states of Markov reliability model more accurately, on which the reliability depends has been proposed. A tool, TimeNET, for steady state distribution of the transition rates in between the states of Markov reliability model is used. Experimental

Validation of the proposed approach is shown using a safety critical system of Nuclear Power Plant. Sensitivity analysis of the transition probabilities and parameter assignment in TimeNET on the software reliability is also shown.

In Chapter 6, the impact analysis methodology on the change in reliability of any component on the reliabilities of other components and hence on the reliability of the software system, has been show. Bayesian network has been used. The experimental validation of this approach on the same case study is also shown.

Chapter 7 presents conclusions regarding this study and suggestions for future work.