CHAPTER 1. INTRODUCTION

1.1 Foreword

Software design, development and testing have become very intricate with the advent of modern highly distributed systems, networks, middleware and interdependent applications. The demand for complex software systems has increased more rapidly than the ability to design, implement, test and maintain them and hence, the reliability of software systems has become a major concern. Today software is being deployed in safety applications due to the advancement of technology. In a nuclear reactor, many systems are being used in safety applications, which demand high reliability [1.1]. As software becomes an increasingly important part of many systems that perform complex and critical functions, such as military defense, nuclear reactors, etc., the risk and impacts of software caused failures have increased dramatically. There is now general agreement on the need to increase software reliability by eliminating errors made during software development [1.2].

Software is a collection of instructions or statements in a computer language also known as a program, which is designed to perform a set of specified functions. Upon execution of a program, an input state is translated into an output state. An input state can be defined as a combination of input variables or a typical transaction to the program. When the actual output deviates from the expected output, a failure occurs. The definition of failure, however, differs from application to application and should be clearly defined in specifications. For instance, a response time of 30s is a serious failure for air traffic system, but acceptable for a railway ticket reservation system.
Within the last decade of the 20th century and the first few years of the 21st century, many reported system outages or machine crashes were traced back to computer software failures. For example, in the health industry, the Therac-25 radiation therapy machine was hit by software errors in its sophisticated control systems and claimed several patients’ lives in 1985 and 1986 [1.2]. In the telecommunications industry, known for its high reliability, the nationwide network of a major carrier suffered with embarrassing network outage on 15 January 1990, due to a software problem [1.3]. In 1991, a series of local network outages occurred in a number of US-cities due to software problems in central office switches [1.2]. Software failures have impaired several high visibility programs in space, telecommunications, defense and health industries. The Mars Climate Orbiter crashed in 1999. The Mars Climate Orbiter Mission Failure Investigation Board concluded, the ‘root cause’ of the loss of the spacecraft, was the failed translation of English units into metric units in a segment of ground-based, navigation-related mission software [1.3]. Besides the loss of money, it delayed the space program by more than a year. Current versions of the Osprey aircraft, developed at a cost of billions of dollars are not deployed because of software-induced field failures. The costly “Y2K” problem resulted because of a design failure, a problem that occupied tens of thousands of programmers and costs running to tens of billions of dollars [1.3].

Software faults are most often caused by requirement and design faults that occur when a designer either misunderstands a specification or simply makes a mistake. Software faults are common for the simple reason that the complexity in modern systems is often pushed into the software part of the system. It is estimated that 60-90% of current computer errors is from software faults [1.4]. Software faults may also occur from hardware where these
faults are usually transitory in nature and can be masked using a combination of current software and hardware fault tolerance techniques.

The current assumption is that software cannot be made without bugs. This assumption may be true, but software does not have to be as traditional buggy as it is now. It is well recognized that assessing the reliability of software applications is a major issue in reliability engineering. Predicting software reliability is not easy. Perhaps the major difficulty is that we are concerned primarily with design faults, which is a very different situation from that tackled by conventional hardware theory. A fault (or bug) refers to a manifestation in the code of a mistake made by the programmer or designer with respect to the specification of the software. Activation of a fault by an input value leads to an incorrect output. Detection of such an event corresponds to an occurrence of a software failure. Input values may be considered as arriving to the software randomly. So, although software failure may not be generated stochastically, it may be detected in such a manner. Therefore, this justifies the use of stochastic models of the underlying random process that governs the software failures.

Software reliability is defined as the probability of failure-free software operations in a specified environment. The software reliability field discusses ways of quantifying it, using it for improvement and control of the software development process. Software reliability is operationally measured by the number of field failures, or failures seen in development, along with a variety of ancillary information. The ancillary information includes the time at which the failure was found, the part of the software where it was found, the state of software at that time, the nature of the failure, etc. Most quality
improvement efforts are triggered by lack of software reliability. Thus, software companies recognize the need for systematic approaches to measure and assure software reliability, and devote a major share of project development resources to this. A number of standards have emerged in the area of developing reliable software consistently and efficiently. The Software Engineering Institute (SEI) has proposed an elaborate standard known as the software Capability Maturity Model (CMM) that scores software development organizations on multiple criteria and gives a numeric grade from one to five. The software reliability is being viewed more in terms of software quality, measurements and control. SEI has devised the CMM by adopting two parameters viz, level and key problem areas as shown in Figure 1.1 [1.5]. This methodology provides a step-by-step procedure to improve reliability from level 1 to level 5. SEI-CMM level certification is issued based on the process adopted in the organization.

The main limitation of software maturity model is that it does not relate to risk or reliability. The levels are arranged to a large variety of software developments and it is not tuned for specific software. It has the provision to accommodate all software organization in to the certification. On the other hand, for nuclear power plants, the safety is of main concern and systems use mainly the “Structured Programming” for safety systems instead of Object oriented or Commercially Off The Shelf (COTS) components [1.6]. So, this research work is aimed at developing a reliability model for the safety systems with software components deployed in Nuclear Power Plants.
<table>
<thead>
<tr>
<th>Level</th>
<th>Key Problem Areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Project Management</td>
</tr>
<tr>
<td></td>
<td>Project Planning</td>
</tr>
<tr>
<td></td>
<td>Configuration Management</td>
</tr>
<tr>
<td></td>
<td>Quality Assurance</td>
</tr>
<tr>
<td>Repeatable</td>
<td>Training</td>
</tr>
<tr>
<td></td>
<td>Technical Practices</td>
</tr>
<tr>
<td></td>
<td>Process focus</td>
</tr>
<tr>
<td>Defined</td>
<td>Process Management</td>
</tr>
<tr>
<td></td>
<td>Process Analysis</td>
</tr>
<tr>
<td></td>
<td>Quantitative quality plans</td>
</tr>
<tr>
<td>Managed</td>
<td>Change in Technology</td>
</tr>
<tr>
<td></td>
<td>Problem Analysis</td>
</tr>
<tr>
<td></td>
<td>Problem Prevention</td>
</tr>
<tr>
<td>Optimizing</td>
<td>Automation</td>
</tr>
</tbody>
</table>

**Figure 1.1 SEI - Capability Maturity Model levels**

Software reliability engineering quantifies the operational behavior of software-based systems with respect to user requirements with bearing on reliability. It includes data collection on reliability, statistical estimation, metrics and attributes of product architecture, design, software development, and the operational environment. Besides its use for operational decisions like deployment, it includes guiding software architecture, design, development and testing. Testing process is driven by software reliability concerns and software reliability models to improve the effectiveness.
1.2. Motivation

From the software disasters, it is clear that software errors have a strong potential to cause serious damage to economy in terms of rework and productivity. It is well recognized that assessing the reliability of software applications is a major issue in reliability engineering [1.7]. Prediction of software reliability is highly involved. Perhaps the major difficulty is that we are concerned primarily with design faults, which is a very different situation from that tackled by conventional hardware theory. Input values may be considered as arriving to the software randomly. So although software failure may not be generated stochastically, it may be detected in such a manner. Therefore, we can use stochastic models of the underlying random process that governs the software failure [1.7].

Hence, for safety / highly dependable software systems the estimation of software reliability becomes prime important to evaluate the overall system reliability with the data available on hardware reliability. The hardware and software reliability together becomes more meaningful and useful for predicting the system performance and availability [1.8].

The University of Maryland (UMD) based its study on previous research carried out by Lawrence Livermore National Laboratory (LLNL) identified a pool of 78 software engineering measures. These measures related to software reliability and established a set of software engineering ranking criteria which is used to assess the metric’s potential as software reliability indicator. This set was then reduced to 30 using importance considerations and these 30 software-engineering measures constitute the basis of the UMD study [1.9].
Four categories of models have been considered as “potential candidates” for modeling the reliability of software. The four categories include reliability growth models, input domain models, architectural models and early prediction models. The first category captures failure behavior during testing and extrapolates it to behavior during operation. Hence, this category of models uses failure data and trends observed in the failure data to derive reliability predictions. The second category of models uses properties of the input domain of the software to derive a correctness probability estimate from test cases that executed properly. The third category of model emphasizes on the architecture of the software and derives reliability estimates by combining estimates obtained for the different modules of the software. Finally, the fourth category of models uses characteristics of the software development process from requirements to test and extrapolates this information to behavior during operation.

1.3. Objectives and Scope of the Present Research Work

Further to the literature review and the motivation behind this research, the main objective of the research is the estimation of software reliability for safety systems of Nuclear Power Plants (NPP). In the process of assessment of reliability, the following sub-objectives are envisaged since software reliability is more concerned with design, methodologies, practices and the tools used in the process of software development.

The sub-objectives are

(a) Development of software life cycle model for safety systems.

(b) Determination of software metrics and development of software metric tool.

(c) Development of Software Verification and Validation methodology.

(d) Development of Software Reliability Model.
These form the basis for the Estimation of software reliability for safety systems of NPP. The present work is focussed towards software development on Instrumentation and Control systems employed in NPP. Around fifty software systems of I&C of fast reactor are taken for analysis. All these systems have been developed using “C” Programming language as part of the embedded systems to perform a specific task. The unreliability attached with Human Error Probability is assumed to be minimal which is basically the boundary within which the exercise has been carried out.

1.4. Organisation of the thesis

The thesis is arranged in eight chapters and they are

Chapter 1. Introduction

Chapter 2. Software life cycle model for safety systems

Chapter 3. Issues and Importance of Software Testing With Respect To Reliability

Chapter 4. Evaluation of Software Metrics and Tool Development

Chapter 5. Verification and Validation Methodology

Chapter 6. Software Reliability Modeling

Chapter 7. Estimation of Software Reliability

Chapter 8. Conclusions and Scope for Future Work
REFERENCES


