A Phase-Based Approach to Creating Highly Reliable Software

Michael R. Lyu
Computer Science & Engineering Department
The Chinese University of Hong Kong
Shatin, Hong Kong
lyu@cse.cuhk.edu.hk

1. Introduction

Our demand for complex hardware/software systems has increased more rapidly than our ability to design, implement, test, and maintain them. When the requirements for and dependencies on computers increase, the crises of computer failures also increases. The impact of these failures ranges from inconvenience (e.g., malfunctions of home appliances), economic damage (e.g., interruptions of banking systems), to loss of life (e.g., failures of flight systems or medical software). The reliability of computer systems has become a major concern for our society.

Within the computer revolution progress has been uneven: software assumes a larger burden while based on a less firm foundation than hardware. In stark contrast with the rapid advancement of hardware technology, proper development of software technology has failed to keep pace in all measures, including quality, productivity, cost, and performance. Software has become the bottleneck of system development, and its delay and cost overrun have often put modern complex projects in jeopardy.

To this end, many software companies see a major share of project development costs identified with the design, implementation, and assurance of reliable software, and they recognize a tremendous need for systematic approaches to assure software reliability within a system. Clearly, developing the required techniques for software reliability engineering is a major challenge to computer engineers, software engineers, and engineers of various disciplines for now and the decades to come.

2. Phase-based approach: an overview

Software reliability engineering is centered around a very important software attribute: reliability. Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. It is one of the attributes of software quality, a multi-dimensional property including other factors like functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software reliability engineering therefore includes:

(1) software reliability measurement, which includes estimation and prediction, with the help of software reliability models established in the literature;

(2) the attributes and metrics of product design, development process, system architecture, software operational environment, and their implications on reliability; and

(3) the application of this knowledge in specifying and guiding system software architecture, development, testing, acquisition, use, and maintenance.

My position is that we should attack the problem of software reliability engineering in three phases: (1) Modeling and Analysis Phase, (2) Design and Implementation Phase, and (3) Testing and Measurement Phase. All these phases deal with the management of software faults and failures. In the Modeling and Analysis Phase, reliability of the software system is being modeled according to the structure of the system and possible fault scenarios. The key topic of this phase is to provide fault modeling of the system, and ask the ”what if” questions. The available modeling approaches include system reliability modeling block diagrams, reliability models by Markov chains, fault tree analysis, and stochastic Petri-nets. In the Design and Implementation Phase, reliability of the software system is being achieved by reliable components built into the system. The key topic of this phase is to provide fault avoidance and fault tolerance. The available techniques we emphasize include reusable software fault tolerance routines, and software fault tolerance by design diversity. In the Testing and Measurement Phase, reliability of the software system is being evaluated and verified by measurement and evaluation techniques. The key topic of this phase is to provide fault removal and fault prediction. The available techniques include data flow testing, reliability measurement tasks, and software reliability tools. We discuss the details of these techniques in the following three sections.

3. Phase 1: modeling and analysis phase

To provide reliability modeling and analysis of a software system during the pre-design phase, the overall system architecture based on requirement can be modeled by several techniques. The available modeling approaches include system reliability modeling block diagrams, Makov-chains reliability modeling, fault tree analysis, and stochastic Petri-nets. These approaches can be used to establish system reliability and performance model for the study of system behavior under various scenarios. The reliability of the system, for example, can be predicted in a coarse basis for the overall system given its architectural options are defined. Sensitivity analysis can then be performed to locate important parameters of the system, and critical components of the system can be identified for enforcement of each component's individual reliability. Note that the reliability model established in this phase can be refined and revised for evaluation purpose in a post-design phase for the purpose of a fine prediction and estimation.

We first perform reliability modeling and analysis using block diagrams for an actual project. It is shown that software reliability has become a major critical factor in system reliability. We further show how fault tree models can be used for the qualitative and quantitative analysis of the failure modes of critical systems. A fault tree provides a mathematical and graphical representation of the combinations of events which can lead to system failure. The construction of a fault tree model can provide insight into the system by illuminating potential weaknesses with respect to reliability of the system. The usage of software tools is a must in the modeling and analysis phase. We consider SHARPE and UltraSAN as two leading tools in this arena.

4. Phase 2: design and implementation phase

In the Design and Implementation Phase, reliability of the software system is being achieved within the system. The key topic of this phase is to provide fault avoidance and fault tolerance. Fault avoidance is the subject of many software engineering techniques and is beyond the scope of this paper. Fault tolerance, on the other hand, is the focus of our discussion. We examine fault tolerance techniques used in single-version as well as multiple-version environments.

Software fault tolerance in single-version software environment is achieved by introducing special fault detection and recovery features, including modularity, system closure, atomicity of actions, decision verification, and exception handling. One successful approach is accomplished by reusable software fault tolerance routines.

A middleware platform containing a set of reusable software components `watchd, libft, REPL, libckp`, and `addrejuv`) to perform these reactive and pro-active software fault tolerance tasks will be described. The hardware platform for using these reusable software components is a network of standard computers where each computer provides a back-up facility for another one on the network. The components provide mechanisms to checkpoint, log messages, watch, detect, rollback, restart, and recover from failures and rejuvenate to avoid failures.

In addition, we will show the evolution of techniques for building fault-tolerant software out of simplex units, and a design paradigm in achieving multi-version software fault tolerance systems.

5. Phase 3: testing and measurement phase

In this phase the reliability of the software system should be evaluated and verified, and testing and measurement techniques are available to achieve this goal. Testing techniques are for fault removal purpose, and reliability assessment techniques are for fault prediction purpose. This includes schemes and tools for software testing and software reliability measurement.

There are many ways of testing software. The terms functional, regression, integration, product, unit, coverage, user-oriented, are only a few of the characterizations we encounter. White-box, or coverage, testing uses the structure of the software to measure the quality of testing. This structural coverage and its measurement is believed to be connected with reliability estimation. These testing schemes include statement coverage testing, decision coverage testing, data-flow coverage testing, and fault-injection testing. We will address these testing schemes in detail. In particular, we demonstrate the usage of ATAC for data-flow coverage testing schemes, and the procedure applied for fault-injection testing.

Software reliability measurement, on the other hand, is the application of statistical inference procedures to failure data taken from software testing and operation to determine software reliability. We have established a framework for software reliability measurement purpose, including four major components in this software reliability measurement process, namely,

(1) reliability objective, (2) operational profile, (3) reliability modeling and measurement, and (4) reliability validation.

We designed and implemented a software reliability modeling tool, called Computer-Aided Software Reliability Estimation (CASRE) system, for an automatic and systematic approach in estimating software reliability. This tool will also be illustrated and demonstrated.