Software Architecture and Design for Reliability Predictability

Software Architecture and Design for Reliability Predictability

By

Assefa D. Semegn

CAMBRIDGE SCHOLARS PUBLISHING

Software Architecture and Design for Reliability Predictability,
by Assefa D. Semegn

This book first published 2011

Cambridge Scholars Publishing

12 Back Chapman Street, Newcastle upon Tyne, NE6 2XX, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN (10): 1-4438-3380-0, ISBN (13): 978-1-4438-3380-6

This book is dedicated to my family: My wife, Wubayehu Z. Azmera; My daughter, Soliyana A. Dagne; My son, Abel A. Dagne; as well as my late parents, Mulu Jemberie and Dagne Semegn, for their unlimited love and consistent support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

Reliability prediction of a software product is complex due to interdependence and interactions among components and the difficulty of representing this behavior with tractable models. Models developed by making simplifying assumptions about the software structure may be easy to use, but their result may be far from what happens in reality. Making assumptions closer to the reality, which allows complex interactions and interdependences among components, results in models that are too complex to use. Their results may also be too difficult to interpret.

The reliability prediction problem is worsened by the lack of precise information on the behavior of components and their interactions, information that is relevant for reliability modeling. Usually, the interactions are not known precisely because of subtle undocumented side effects. Without accurate precise information, even mathematically correct models will not yield accurate reliability predictions. Deriving the necessary information from program code is not practical if not impossible. This is because the code contains too much implementation detail to be useful in creating a tractable model. It is also difficult to analyze system reliability completely based on the program code.

This author approached the problem from three tracks:

- Identifying design imperatives that will make the system behavior easier to predict,
- Identifying mathematical documentation techniques to describe the behavior of software systems,
- Adapting structural reliability modeling techniques to predict the reliability of software systems based on their mathematical description.

This book documents the resulting novel approach of designing, specifying, and describing the behavior of software systems in a way that helps to predict their reliability from the reliability of the components and their interactions. The design approach is named *design for reliability predictability* (DRP). It integrates *design for change*, *precise behavioral documentation* and *structure based reliability prediction* to achieve improved reliability prediction of software systems. The specification and documentation approach builds upon precise behavioral specification of interfaces using the trace function method (TFM). It also introduces a number of structure functions or connection documents. These functions capture both the static and dynamic behaviors of component based software systems. They are used as a basis for a novel document driven structure based reliability prediction model. System reliability assessment is studied in at least three levels: *component reliability*, which is assumed to be known, *interaction reliability*, a novel approach to studying software reliability and *service reliability*, whose estimation is the primary objective of reliability assessment. System reliability can be expressed as a function of service reliability. A mobile streaming system, designed and developed by the author as an industrial product, is used as a case study to demonstrate the application of the approach.

ACKNOWLEDGMENTS

First and foremost, I praise Lord, the Almighty, Who gave me all what I need to bring this work to a success and to Whom I owe my very existence.

Numerous people around me: family, colleagues and friends, have contributed directly and indirectly to make the work a success. I am grateful to all who extend their support in different ways..

I would like to extend my acknowledgment to Prof. David Parnas, the father of software engineering, Prof. Eamonn Murphy and Dr. Donal Heffernan for their invaluable feedback, advice, encouragement and support that made the completion of this work a reality. The background image on the cover is a photo captured by Khaironi Yatim. I would also like to thank him for his permission to use it. .

My deepest gratitude and respect goes to my late parents Mulu Jemberie and Dagne Semegn, who raised me with care and love, who instilled in me self-confidence, determination, and who taught me the great lessons of life.

A million thanks and deepest love to my gorgeous children: Soliyana and Abel, for giving me unlimited happiness. You have made my life complete!

Finally, I wish I have special words that can express my gratitude to my darling wife Wubayehu Zewdie, who paid the greatest sacrifice and contributed the most. Wubua, in this journey, your unconditional love, extraordinary strength, and unwavering support have created the energy and your insightful ideas and guidance have given me the light, making your role for the success of this work to have no parallel. You are incredible!

ABBREVIATIONS

ADL	Architecture Description Languages
ADT	Abstract Data Type
CLT	Central Limit Theorem
CN	Communication Networks
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off The Shelf
CTMC	Continuous Time Markov Chain
CTVR	Center for Telecommunication Value Chain Research
DC	Design for Change
DRP	Design for Reliability Predictability
DTMC	Discrete Time Markov Chain
EFG	Event-Flow-Graph
GAO	General Accounting Office
GPRS	General Packet Radio Service
GUI	Graphical User Interface
IC	Input Channel
KLOC	Kilo Lines of Code
LD	Local Device
LRCL	Long Range Communication Link
M&C	Monitoring and Control
MD	Mobile Device
MG	Module Guide
MGF	Moment Generating Function
MID	Module Interface Specification
MIDD	Module Internal Design Document
MIL	Module Interconnection Languages
MSC	Mobile Side Components
MSS	Mobile Streaming System
MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
MVD	Multi-version Design
NATO	North Atlantic Treaty Organization
NHPP	Non-Homogenous Poison Process
NIST	National Institute of Standards and Technology
NVP	N-Version Programming

OC	Output Channel
OES	Other Elementary Systems
OMG	Object Management Group
OODB	Object-oriented database
PA	Parent Application
PBD	Precise Behavioral Documentation
RB	Recovery Blocks
RPC	Remote Procedure Call
RS	Remote Server
S&D	Specification and Documentation
SCR	Software Cost Reduction
SDC	Software Design and Construction
SDI	Strategic Defense Initiative
SFI	Science Foundation of Ireland
SMP	Semi-Markov Process
SQRL	Software Quality Research Laboratory
SRCL	Short Range Communication Link
SRD	Service/System Requirement Document
SRE	Software Reliability Engineering
SSS	Server Side Services
THE.	Technische Hogeschool Eindhoven
TAM	Trace Assertion Method
TFM	Trace Function Method
UA	User Application
UML	Unified Modeling Language

CHAPTER ONE

INTRODUCTION

1.1 Background

Software is one of the most complex engineering products. While it has been among the key components in changing the society from the "industrial age" to the "information age", its complexity has also been a serious bottleneck for the success of various projects. For instance, the key question that could not be answered with respect to United State's strategic defense initiative (SDI) project was dependability of the software (Parnas, 1985).

Recognizing the complexity of software, the interest to consider it as an engineering product that needs to be produced through an engineering process dates back as early as the 1960's. Although, various technological advances have been made in the past four decades, many issues that were discussed in the 1968's NATO sponsored conference (Nato, 1968), are still unresolved.

One such example is related to a reliability, more precisely an availability, requirement that was reported in that conference which reads as: "a design requirement for our Electronic Switching System was that it should not have more than two hours system downtime (both software and hardware) in 40 years." (Nato, 1968, p 323-326). Such requirements are hard to be *guaranteed* even today, after nearly 40 years in light of the inherent complexity of software systems and the daunting task of software reliability prediction.

The main reason for this is the complexity of software systems, which is essential by its nature (Brooks, 1987) that cannot be avoided. As remarked by Dijkstra, "software presents the only discipline and profession where an individual's skull has to bridge from a bit to hundreds of megabytes, a gigantic ratio of 10^9 that baffles our imagination" (Dijkstra, 1988).

As a result, mastering the complexity of software has been the hallmark of software design, besides achieving other qualities. To this end, the principle of *information hiding* (Parnas, 1972a), that laid the foundation of today's technologies (IEEE, 2007), has practically proved itself to be the most fundamental concept. It not only helps to master complexity but also provides such qualities as changeability, maintainability, reusability, and evolvability.

The four decade-old, yet timeless, principle of software design states that "the design begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing. The interface of a module should be chosen to reveal as little as possible about its inner workings."

This principle has become the foundation of object oriented design (IEEE, 2007) and many other current technologies, such as *design for reusability* (Erich et al., 1995) in object-oriented design patterns, and *design for composition* - (Szyperski, 2002) in component-software where interfaces get self standing position in the system and serve as contracts between components. Component level changeability and structural relations are also at the heart of software architecture (Reddy et al., 2006, Guijun and Fung, 2004).

More recent programming efforts coin the term *aspect* which is claimed to address *separation of concerns* (Kiczales et al., 1997, Atkinson and Kuhne, 2003, Schult and Polze, 2002). The concept in there is primarly aimed at addressing some of the limitations of object-oriented languages in supporting changeability, more than problems in designs. Otherwise, its main goal stated as *changeability,* is the same goal that started the *design for change* paradigm three decades earlier. A similar work on N-degree *separation of concerns* is also reported in (Tarr et al., 1999).

Unfortunately, these technological developments do not guarantee anyone to meet stated reliability targets nor do they enforce the application of sound principles. Sophisticated tools may come with sophisticated problems resulting in difficult-to-uncover bugs. Programmers can use/abuse tools to construct badly structured programs. Testing alone, the existing predominant mechanism of quality assurance, cannot verify the correctness of these programs. Too much emphasis given to coding, at the expense of disciplined design of systems and careful structuring of its components, results in systems that continue to grow in size and complexity. The increased complexity in turn results in increased uncertainty on the dependability of the products.

As a result, software failure is still commonplace, causing substantial economic problems, as well as, life loss. Some, examples include:

- A study by the National Institute of Standards and Technology (NIST) indicates that software errors cost the U.S. Economy about $59.5 billion (0.6%GDP) annually (RTI, 2002, Newman, 2002). The same study indicates that "software developers spent 80% of development costs on identifying and correcting defects, and yet few products of any type other than software are shipped with such high levels of errors."
- The cumulative effect of a chopping error that missed 0.000000095 sec in every 10^{th} second, accumulated over 100hrs, to create a discrepancy of 0.34 secs. resulted in missing of a Patriotic missile to intercept a Scud missile with a consequence of loss of life of 28 people (GAO, 1992).
- Software problems in radiation therapy equipment, Therac-25, is believed to have caused at least six known accidents of massive radiation overdoes resulting in deaths and serious injuries (Leveson and Turner, 1993).
- The inquiry commissions report on the failure of flight 501 of ARIAN 5 (Lions, 1996), which is one of the most costly system failures due to software bug, shows that the root cause of the disaster is because of unhandled exception resulted from an operand error.

In general, "software crises", a term that was coined many decades ago to indicate the challenges of software construction is still with us as a chronic problem. To this date, while hardware is sold with a guarantee or warranty, software is sold with a disclaimer.

Various approaches have been proposed over the years to address reliability prediction of software systems. Two classes of reliability modeling approaches have evolved: black box approaches (classified as software reliability growth models) and white box approaches (classified as architecture based reliability models).

The first category does not take the internal structure of the software into consideration. As a result, it is not particularly effective for predicting large-scale software systems composed of various components.

The models in the second category have been proposed to take the structure of a software system into consideration. They aim to predict software system reliability from the reliability of its components and their interconnections. Most architecture based reliability models assume components to fail independently and only when executed (Swapna, 2007) (Katerina Goseva-Popstojanova, 2001). Many of them also assume components to be sequentially executed and appear in *series* connection in terms of reliability.

The main difference between these models is how they account for variations on usage profile of components. In most cases, software architecture is modeled by the 'transfer of control' among components, whereby the relative contribution of the components is estimated. Essentially, this is an effort to estimate what classical reliability theory calls *the duty cycle* (Bazovsky, 1961) of components in the system.

Only a few conceptual models (Littlewood et al., 2001) are proposed to address the issue of reliability estimation in the case of redundancy through N-version Programming (NVP) (Avizienis, 1995). These models include those reported in papers (Bev Littlewood Peter et al., 2000), (Wattanapongskorn and Coit, 2007), (Zafiropoulos and Dialynas, 2004), (Ege et al., 2001) and (Ping et al., 2008). However, due to the difficulty of obtaining reliable data to support various models, there are no universally accepted models for estimating the failure rate of redundant software versions.

Besides N-version programming, there are various ways to achieve software fault tolerance including multilevel restoration systems. One model that considers the possibility of multilevel restorations is provided by Vilkomer et. al. (Vilkomir et al., 2005). Like many other models that model software systems at different levels of performance, this model does not use system structural information.

The author's analysis reveals the following basic gaps between possible software structures and architecture-based reliability models:

- For a network of components that can be executed concurrently, either in a distributed or centralized environment, the concept of 'transfer of control' is not well defined and may not exist at all. Many components can be active or operational at the same time communicating with each other using some communication signals.
- There is a wide range of reliability improvement components, such as protective components, redundant components, failover switches, backup and restorative components where the 'series connection' assumption does not hold.
- Design approaches that allow fault-tolerance and degradation behavior result in components that differ in level of criticality, level of importance, type of output and level of relevance. Therefore,

duty cycle alone, does not fully explain the relative importance of components to the system reliability even for sequentially executing components.

- Components are subjected to both primary failures and dependent failures. Primary failures are failures caused by design faults in each software component. Dependent failures, on the other hand, are failure in which components fail due to failure of other components in their environment, or because of interactions. In case of the later one, interaction failures, no particular component may be singled out as the cause of failure. It can result from the joint interaction of various components, usually through shared resources, leading to an unacceptable system behavior.

The gap between software structure and architecture based reliability prediction approaches may have two causes. The first cause is poor structure where the reliability as well as functional role of components cannot be known precisely. The second cause is lack of precise specification or description of system behavior in well-structured systems where the functional as well as the reliability role of components can clearly be identified.

In both cases, the problem is related to that of a software product, where, by product we mean including its specification and description. It is not possible to predict the reliability of a product whose requirements are not known since it cannot be determined whether an output or an observed behavior is acceptable or not. Similarly, it is not possible to make structural assessment about the product if its components and their relations are not known precisely.

Therefore, our study focuses on software product, its design and its structure. We identify design properties that will facilitate the application of structural reliability assessment techniques. This will enable to estimate the reliability of a software system from the reliability of its components and their interactions.

This is a challenging problem since, even when all information is available, structural reliability assessment of component based software systems is believed to be one of the most difficult problems in software engineering. Kappes et. al. (Kappes et al., 2000) proved the absence of an algorithm (and hence *program*) that can calculate or even approximate the reliability of component oriented software systems by showing the problem to be equivalent to the well known *halting* problem. Butler et. al. (Butler and Finelli, 1993) have discussed the infeasibility of quantifying the reliability of life critical real-time software. Littlewood et. al.

(Littlewood and Strigini, 2000) commented on the difficulty of pre-assessment of reliability of software systems.

1.2 Objective of the Book

The main objective of this book is to identify design properties and documentation techniques that make the reliability of a software product easier to predict based on the analysis of the reliability of its components and information about their interconnections. The book aims to answer the following basic research question.

What design imperatives and documentation techniques can help to make the reliability of a software product more predictable based on analysis of its structure and information about the reliability of its components?

Specifically, the book seeks to find answers for the following questions:

- Can we improve software reliability prediction by applying specific ways of software design?
- What are the design criteria that will help us to better estimate/predict the reliability of a system composed of components whose reliabilities are known?
- What kind of mathematical documentation and model would enable us to understand the reliability structure of software systems?

The author calls the problem of predicting the reliability of software systems, the *reliability predictability problem* and tackles it from three tracks:

1. Identifying design imperatives that will make the system behavior easier to predict
2. Identifying mathematical documentation techniques to describe the behavior of software systems
3. Adapting structural reliability modeling techniques to predict the reliability of software systems based on their mathematical description

1.2.1 Identify Design Imperatives for Improved Reliability Prediction

The first objective of this book is set to identify design imperatives that will make the system reliability easier to predict.

One may develop systems whose behavior is hard to understand and describe using any approach. Such systems may have no uniquely identifiable components. The inter-relationships among their components may not be well known. No realistic tractable model may exist that is capable of predicting the reliability of such poorly structured systems.

Structural reliability assessment can be applied only when the structural relation among the different components in the system can be clearly identified and known. Thus, one part of this book is to identify the necessary decomposition and composition criteria that will make a software product easier to analyze and its reliability simpler to predict.

1.2.2 Identifying Mathematical Documentation Techniques to Describe the Behavior of Software Systems

The second objective of the book is finding ways of mathematically describing the behavior of software systems.

Other engineering disciplines such as Electrical and Mechanical Engineering depend on mathematics to unconditionally characterize and precisely analyze different aspects of their systems. The mathematical models provided in the form of transfer functions or system of equations gives all the necessary information required to study the behavior of their products.

In software engineering, this is not common. The descriptions about the product are usually informal or semiformal which are often known to be ambiguous, inconsistent, and difficult to keep up to date with the actual product. In the absence of precise relevant documentation, it is not uncommon to resort to reading program code for reliability assessment or other purpose. However, deriving the necessary information from program code is not practical, if not impossible, because the code contains too much implementation detail to be useful in creating a tractable model. Besides, program codes tell us what the system does, not what it is supposed to do. Therefore, another part of this book is aimed at identifying precise mathematical documentation technique of software products that would help to derive its reliability structure.

1.2.3 Predicting the Reliability of Software Systems based on their Precise Mathematical Description

Although many reliability models have been proposed over the years, nearly all of them are based on pure theoretical or mathematical basis having very little connection with the actual software structure, its behavior and its failure characteristics. Only few, such as the one discussed in (Katerina Goseva-Popstojanova et al., 2005), attempt to connect reliability models with real software products. Even in such cases, the models usually take simplifying assumptions on software structure, which makes the reliability prediction questionable. These problems arise in attempting to derive structural information that is necessary for reliability prediction based on program code. Program code can generally be considered as a poor source of information for reliability assessment because:

- It contains too much unnecessary details that make the resulting information intractable to model
- The reliability role of components may not be clearly identified from the code
- The textual organization of the program code could be quite different from its runtime organization

Therefore, the third objective of this book is to use precise mathematical descriptions of software for making reliability assessment.

1.3 Achievement of the Book

This book documents the result of these investigations. It introduces a novel approach of designing, specifying, and describing the behavior of software systems in a way that helps to predict their reliability from the reliability of the components and their interactions. It identifies design imperatives and relevant mathematical documentation techniques for improved reliability predictability of software systems.

The design approach, which the author names: *design for reliability predictability* (DRP), integrates *design for change*, *precise behavioral documentation* and *structure based reliability prediction* to achieve improved reliability predictability of software systems. It enhances changeability to the level of supporting modular redundancy (when necessary), fault tolerance and restorability for improved reliability.

However, since the gain in reliability from redundancy can be compromised due to statistical dependencies and interaction failure causes, the design is supplemented with precise documentation and structure based reliability assessment techniques.

The specification and description approach builds upon precise behavioral specification of interfaces using the trace function method (TFM) (Parnas and Dragomiroiu, 2006), (Parnas, 2009b) and introduces a number of structure functions: module-interface connection matrix, input-output connection matrix and event-flow-graphs. These functions capture both the static and dynamic behavior of a software system. They are used to derive reliability structure functions.

The author integrates these structure functions with standard reliability and probabilistic models to develop a novel document based multilevel structural reliability assessment technique.

DRP has been successfully applied in the design and development of a mobile streaming system (MSS) as a case study. MSS is a system that consists of a network of software, hardware, and wireless communication components in a distributed and stochastic environment with real-time characteristics. The time sensitivity and unreliability of the wireless channels, the non-fault tolerant behavior of the peripheral device (where missing of a single bit from tens of megabits causes unacceptable output), and the limited computing and storage resource size of the mobile device make the system nontrivial to design, document and perform reliability assessment.

These features make the system reliability sensitive to its various structures, most notably its event structures. As a result, various design alternatives have been assessed based on their reliability estimates. The design with the best service reliability from the different alternatives is implemented finally. The main lesson learnt from the case study is a successful marriage among the three components of design for reliability predictability, which are design for change, precise behavioral specification/description and structure based reliability assessment that allowed quantification of different reliability concerns among different design alternatives.

1.4 Contribution of the Book

The main contribution of this work is the integration of design for change, precise specification and description, and structural reliability prediction into one design approach, which is called design for reliability predictability (DRP).

The *design for change* paradigm that has been the main focus of research in the past couple of decades is enhanced with the objective of reliability to support modular redundancy, whenever needed, restorability, and fault-tolerance. The contribution in here is the application of changeability property of designs for creating a reliability structure that is possibly different from mere series connection of components. Having assessed various design approaches proposed over the years, this author is convinced that properties, such as changeability, are design features rather than specific tool or programming language features. Even when one applies identical design techniques and tools, various designs may differ from each other in their qualities in general and reliability in particular.

The *specification and description* is built upon a newly developed black-box specification technique, called the trace-function-method (TFM) and introduces a number of novel structure functions, namely ‘module-interface connection matrices’, ‘input-output connection matrices’, and ‘event-flow-graphs’. Of course, connection matrices are well-known mathematical models used in different disciplines such as in the functional description of switching networks (Deo, 1974). The contributions in here relates to their application in specification and/or description of a software structure.

The document based multilevel structural reliability assessment approach reported in this book differs from those covered in the literature in the following ways:

- It is based on precise component and system behavioral documentation as opposed to program code, or informal or semi-formal documentation
- It makes use of various structures about software systems: module-interface structures, composite structures, event-structures, type structures, etc. instead of just ‘control-transfer’ concept
- It does not consider all components to be in mere series connection but rather uses the reliability role played by each component, its failure mode and effect. To this end, components may play a protective role, redundant role, restorative role, supportive, etc.
- It addresses reliability assessment at least from three levels – component reliability, interaction reliability and service reliability
- It takes into account failure causes that may arise due to interactions among a network of components and scarcity of resources that have barely been explored in the literature. Various interaction reliability estimation formulas are also derived based on relevant reliability or probability theory.

- Reliability assessment is not limited to the possible presence or absence of incorrect program statements or bugs. It is rather an assessment of the possible occurrence of undesired events that affect required system outputs.

1.5 Organization of the Book

The rest of the book is structured as follows: Chapter 2 explores related work to this book. Three areas are covered: software design and construction, specification and description and structure based reliability prediction. Chapter 3 provides analysis of the problem where software failure characteristics and software fault mitigating approaches are discussed. Chapter 4: provides the definitions and basic concepts used in design for reliability predictability. Chapter 5 identifies the design imperatives based on reliability theory. These imperatives are further elaborated and mathematical models are developed in chapter 6. Chapter 7 gives the document based structural reliability assessment method developed by the author. Novel formulas for component reliability, interaction reliability and service reliability are provided. Chapter 8 presents DRP as applied to a case study, a mobile streaming system (MSS). Chapter 9 presents reliability assessment of the case study. Chapter 10 provides concluding remarks.

Chapter Two

The Three Tributaries of DRP

Design for reliability (DRP) is derived from a combination of three related fields. These are:

1 Software Design and Construction (SDC)
2 Specification and Documentation (S&D)
3 Software Reliability Prediction (SRP)

DRP can be considered to exist at the intersection point of these three areas and may be depicted as shown in Figure 2-1.

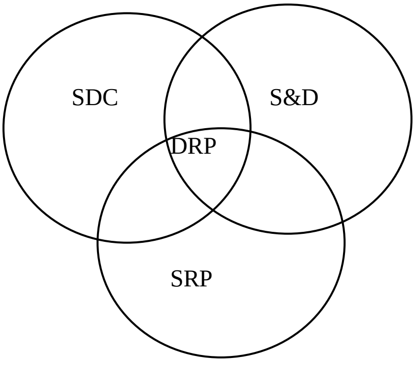

Figure 2-1: Areas Related to Design for Reliability Predictability (DRP)

Many decades of research effort have been spent in each of these areas independently. However, the integration of these three areas as a unified research work has not been addressed thoroughly or adequately enough. Additionally, the software reliability problem is a critical problem that continues to affect many projects negatively.

Thus, this book attempts a new beginning in integrating these three components into one unified design approach. This is aimed at improving