Springer Series in Reliability Engineering

P. K. Kapur · H. Pham · A. Gupta · P. C. Jha

Software Reliability Assessment with OR Applications

Springer

Springer Series in Reliability Engineering

For further volumes:
http://www.springer.com/series/6917

P. K. Kapur · H. Pham · A. Gupta · P. C. Jha

Software Reliability Assessment with OR Applications

Springer

Prof. P. K. Kapur
Faculty of Mathematical Sciences
Department of Operational Research
University of Delhi
Delhi 110007
India
e-mail: pkkapur1@gmail.com

Dr. A. Gupta
Faculty of Mathematical Sciences
Department of Operational Research
University of Delhi
Delhi 110007
India
e-mail: guptaanshu.or@gmail.com

Prof. H. Pham
Department of Industrial and Systems
Engineering
Rutgers University
Frelinghuysen Road 96
Piscataway, NJ 08854-8018
USA
e-mail: hopham@rci.rutgers.edu

Dr. P. C. Jha
Faculty of Mathematical Sciences
Department of Operational Research
University of Delhi
Delhi 110007
India
e-mail: jhapc@yahoo.com

ISSN 1614-7839

ISBN 978-0-85729-203-2 e-ISBN 978-0-85729-204-9

DOI 10.1007/978-0-85729-204-9

Springer London Dordrecht Heidelberg New York

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

Cover design: eStudio Calamar, Berlin/Figueres

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Advances in software technologies have promoted the growth of computer-related applications to a great extent. The proliferation of Internet has gone far beyond even the most optimistic forecasts. Computers and computer-based systems pervade every aspect of our daily lives. This has benefited society and increased our productivity, but it has also made our lives critically dependent on their correct functioning. Successful operation of any computer system depends largely on its software components. In the past three decades abilities to design, test and maintain software have grown fairly, but the size and design complexities of the software have also increased manyfolds, and the trend will certainly continue in future. In addition to this, the critical system operations, in which very high operational precision is required are also becoming more and more dependent on the software. There are numerous instances where failure of computer-controlled systems has led to colossal loss of human lives and money. This is a big challenge to the software developers and engineers. Producing and maintaining the high quality of software products and processes are at the core of software engineering, and only a comprehensive quality improvement and assessment program that have successful outcome can assure it. A lot of research material and book titles are available with focus on tools and methods for monitoring and assuring high quality software. At this stage there is a great need for looking at ways to quantify and predict the reliability of software systems in various complex embedded operating systems. Apart from this, cost and budget limitations, schedule, and due dates are the constraints that encroach on the degree to which software development and maintenance professional can achieve maximum quality. Our title *Software Reliability Assessment with OR Applications* provides in-depth knowledge of quantitative techniques for software quality assessment.

The technology of modern embedded software systems is changing at a very fast rate; such changes are not ever seen in any other areas. On account of these changes, the techniques and models available to measure the system reliability have also increased at the same rate. In contrast to the few available books in this area our book addresses most of the existing research, recent trends, and many more of these techniques and models. Several areas of software reliability

assessment and applications, which have gained interest mainly in the last five years and grown at a very fast pace, have been discussed comprehensively in the book for the first time. Topics such as

- Change point models in software reliability measurement
- Application of neural networks to software reliability assessment
- Optimization problems of optimum component selection in fault tolerant systems
- Unification methodologies in software reliability assessment
- Software reliability growth modeling using stochastic differential equationshave been included for first time, while topics such as
- Literature of reliability analysis for fault tolerant systems
- Study of software release time decision
- Optimum resource allocation problem

have been addressed comprehensively.

The content of this book is useful and provides solution to the problems faced by several groups of people working in the different fields of software industry.

These groups in general are the people

1. Who want to acquire the knowledge of the state-of-the-art of software reliability measurement, prediction and control. These people include the managers of the software development organizations, engineering professional dealing with software, and persons involved in the marketing and use of software.
2. Who are working in different software development groups such as software design team, testing and debugging teams, and maintenance and evolution teams, or practitioners of quality assessment, risk analysis, management, and decision sciences.
3. Who are involved in the research related to software reliability engineering, reliability analysis, operations research, applied statistics and mathematics, and industrial engineering and related disciplines.

The book brings out widespread literature of past 40 years of software reliability assessment. It can serve as a first choice and a complete reference guide. The book brings out widespread literature of past 40 years of software reliability assessment. It can serve as a first choice and a complete reference guide

The introduction chapter provides an inclusive material and basic knowledge required to understand the entire content of the book. Various new concept maps and pictures have been designed to facilitate the understanding. The content of rest of the book is organized as follows.

Chapter 2 describes the earlier literature of the software reliability growth models (SRGM). It covers the software reliability modeling with exponential, S-shaped and flexible models. Consideration of testing efforts in reliability growth modeling is also presented. The last section of the chapter concentrates on reliability assessment models for software developed under distributed environment.

Earlier literature of reliability growth modeling assumed a perfect debugging environment. Testing the efficiency of testing and debugging teams makes an important aspect of the reliability growth modeling, and its consideration in the models can give absolutely different results as compared to perfect debugging models. The literature of software reliability modeling under imperfect debugging environment is discussed in Chapter 3.

Testing coverage and testing domain measures are the key factors related to the software reliability growth process. These measures help developers to evaluate the quality of the tested software and determine the additional testing required to achieve the desired reliability. On the other hand, it is a quantitative confidence criterion for the customer in taking the decision to buy the product. A detailed discussion on the testing coverage, domain, and reliability modeling with respect to these measures is done in Chapter 4.

The concept of change point is relatively recent in the software reliability modeling. Developing models using the change point concept provides very accurate results most of the times. A number of reasons are associated for modeling under change point concept such as changes in the testing environment, testing strategy, complexity and size of the functions under testing, defect density, skill, and motivation and constitution of the testing and debugging team. Modeling using the change point concept provides answers to the number of questions related to the changing scenarios during testing phase. Reliability modeling with change point is discussed at length in Chapter 5.

Chapter 6 is addressed to the unification schemes in software reliability growth modeling. Several existing SRGM consider one or the other aspect of software testing but none can describe a general testing scenario. As such, for any particular practical application of reliability analysis one needs to study several models and then decide the most appropriate one. The selected models are compared based on the results obtained and then a model is selected for further use. As an alternative, following a unification approach several SRGM can be obtained from a single approach giving an insightful investigation of these models without making many distinctive assumptions. It can make our task of model selection and application much simpler compared to the other methods. Establishment of unification methodology is one of the very recent topics of research in software reliability modeling and is discussed for the first time in this book.

Like unification schemes, software reliability modeling based on the Artificial Neural Networks has gained interest of software reliability researchers recently. Only limited work has been done in the field by a group of few researchers. In Chapter 7 we introduce and discuss the existing literature in this area.

The topic of software reliability modeling with stochastic differential equations although started in the early nineties but gained much popularity and seen more useful work only in the current years. A comprehensive study of this topic is presented in Chapter 8.

The reliability growth models discussed in the previous chapters are the continuous time models. There is another category of reliability growth models, which use the test cases as a unit of fault detection/removal period. These models are

called discrete time models. A large number of models have been developed in the first group while fewer are there in the second group due to the difficulties in terms of mathematical complexity. The utility of discrete reliability growth models cannot be underestimated. As the software failure data sets are discrete, these models many a time provide better fit than their continuous time counterparts. Chapter 9 addresses to the study of discrete software reliability growth modeling.

The software reliability models find important OR applications. Determination of software release time and allocation of testing resources at unit testing level are among the major applications. Chapters 10 and 11 present an inclusive study of these optimization applications of the reliability growth models.

Maintaining highest possible reliability is most important for the software systems used to automate the critical operations. Fault tolerance is designed in software to achieve the highest level of reliability in these systems as compared to what can be attained with testing. A complete knowledge of fault tolerant schemes, reliability growth modeling, and optimum system composition problem has been described in Chapter 12.

A number of useful references, appendices, and index terms are provided to help further readings.

We expect that our book will meet the expectations of the readers and provide the best of the state-of-the-art on the subject.

Acknowledgments

Prof. Kapur remembers with special fondness, the first in person meeting with Prof. Pham at the International Conference on *Present Practices and Future Trends in Quality and Reliability*, Indian Statistical Institute, Kolkata, in 2008. They shared thoughts together and their thoughts conceptualized the idea of this book. Later Prof. Kapur associated Dr. Jha and Dr. Gupta, who happen to be his former students with his ideas and the concept came into the form of a proposal. It took almost two years to complete the book and it is the time to have the opportunity to acknowledge the people who provided their support directly or indirectly in completing this venture.

This book contains a lot of research material of various researchers across the globe of over nearly four decades and more. The list of authors whose contributions have been incorporated in this book is very big. It was not possible to specifically list them all individually, the authors of the book like to greatly acknowledge their outstanding contributions, appreciate their work, and thank all of them.

Prof. Kapur and Prof. Pham are also grateful to their numerous Ph.D\M.Tech\M.Phil\fellow students for the research work, done jointly with them. Their contribution is immeasurable to the growth of this book.

The authors wish to express deep sense of gratitude to their parents, spouses, children, and other family members, who have provided them unconditional and unfaltering support. Their supportive attitude was always motivating.

We are thankful to almighty God for giving us the strength to complete our work.

Lastly, we apologize for any omissions.

Contents

Acronym

ANN	Artificial Neural Network
ART	Adaptive Resonance Theory
CDF	Cumulative Density Function
CER	Community Error Recovery
CF	Common Faults
CFM	Common Failure Mode
CIF	Concurrent Independent Failures
CIFM	Concurrent Independent Failure Mode
CLNLR	Conditional Logic Nonlinear Regression
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRB	Consensus Recovery Block
DDE	Distributed Development Environment
DIM	Dynamic Integrated Model
DNA	Deoxyribonucleic Acid
DWCM	Dynamic Weighted Combinational Model
EDRB	Extended Distributed Recovery Block
ENHPP	Enhanced Non-Homogeneous Poisson Process
EW	Error Derivative of the Weights
FCP	Fault Correction Process
FDP	Fault Detection Process
FDR	Fault Detection Rate
FIR	Fault Isolation Rate
FRR	Fault Removal Rate
GDIM	Generalized Dynamic Integrated SRGM
GINHPP	Generalized Imperfect Non-Homogeneous Poisson Process
GO Model	Goel and Okumoto Model
GOS	Generalized Order Statistic
GPA	Goal Programming Approach
HPP	Homogeneous Poisson Process

IBM	International Business Machines
IEEE	Institute for Electrical and Electronic Engineers
IF	Independent Faults
ISO/IEC	International Organization for Standardization/International Electro-Technical Commission
IT	Information Technology
KG Model	Kapur and Garg Model
KLOC	Kilo Lines of Code
KT	Kuhn Tucker
LMS	Least Mean Squares
LN	Levenberg-Marquardt
METEF	Modified Exponential Testing Effort Function
MLE	Maximum Likelihood Estimate
MOV	Modified Optimal Values
MRTEF	Modified Rayleigh Testing Effort Function
MSE	Mean Square Error
MTBF	Mean Time between Failures
MTTF	Mean Time to Failure
MWTEF	Modified Weibull Testing Effort Function
NDP	Normalized Detectability Profile
NHPP	Non-Homogeneous Poisson Process
NLLS	Non-Linear Least Square
NLR	Nonlinear Regression
NMR	N-Modular Programming
NN	Neural Network
NVP	N Version Programming
OOV	Original Optimal Values
OS	Operating System
PDF	Probability Density Function
PE	Prediction Error
PGF	Probability Generating Function
R&D	Research and Development
RB	Recovery Blocks
RMSPE	Root Mean Square Prediction Error
RPE	Relative Predictive Error
SDE	Stochastic Differential Equations
SDLC	Software Development Life Cycle Models
SPSS	Statistical Package for Social Sciences
SQP	Sequential Quadratic Programming
SRE	Software Reliability Engineering
SRGM	Software Reliability Growth Model
SRTD	Software Release Time Decision
TEF	Testing Effort Function

TFN	Triangular Fuzzy Number
V&V	Verify and Validate
WRC	Water Reservoir Control
YDSM	Yamada Delayed S-Shaped Model

Chapter 1
Introduction

A popular theory and explanation of the contemporary changes occurring around us is that we are in the midst of a third major revolution in human civilization, i.e., a *Third Wave*. First there was the Agricultural Revolution, then the *Industrial Revolution*, and now we are in the *Information Revolution*. Yet we are, in fact, in the middle of a revolutionary jump. Information and communication technology and a worldwide system of information exchange have been growing for over a 100 years. Information technology (IT) is playing a crucial role in contemporary society. It has transformed the whole world into a global village with a global economy. IT has now become the most important technology in the human world and it is an excellent example of the *law of unintended consequences* as it paves the way for creation of the new technologies (e.g., genetic engineering), extension of the existing technologies (e.g., telecommunications), and the demise of the older technologies (e.g., the printing industry). Today almost every business, industry, services, government agencies, and even our day-to-day activities are directly or indirectly affected by computing systems. The Computer revolution has benefited society and increased the global productivity, but a major threat of this revolution is that the world has become critically dependent on the computing systems for the proper functioning and timing of all its activities. For example, air traffic control, nuclear reactors, patient monitoring system in hospitals, automotive mechanical and safety control, online railways and air ticketing, industrial process control, global networking of various business, and services which include information storing (databases), information sharing and internet marketing, etc. are some diverse applications of IT. If the computer system shows a failure in such systems the impact of failures may range from inconvenience in social life to economic damage to loss of life in the most critical case. A total breakdown of the system functioning is observed in most of the cases until the fault is repaired, and even after restoring the system to a normal state, sometime it takes up huge time, efforts, and resources to make up the losses.

P. K. Kapur et al., *Software Reliability Assessment with OR Applications*,
Springer Series in Reliability Engineering, DOI: 10.1007/978-0-85729-204-9_1,

In the broadest sense, IT refers to both the hardware and software that are used to store, retrieve, and manipulate information. In the past two decades the hardware has attained high productivity and quality due to advances in technology and progress of design and test mechanisms. Large-scale improvement in hardware performance, profound changes in computing architectures, vast increase in memory and storage capacity, a wide variety of exotic input and output options, has further increased the demand of software in automation of complex systems, its use as a problem-solving tool for many complex problems of exponential size, and to control critical applications. With this, size and design complexities of the software has also increased many folds and the trend will certainly continue in future. For instance the NASA Space Shuttle flies with approximately 0.5 million lines of software code on board and 3.5 million lines of code in ground control and processing [1–3].

With the escalation in size, complexity, demand, and depends on the computer systems the risk of crises due to software failures has also increased. There are numerous reported and unreported instances when software failures have caused severe losses [3, 4]. Few examples are the crash of Boeing 727 of Mexicana airlines because the software system did not correctly negotiate the mountain position (1986), overdose given to the cancer patients by the massive Therac-25 radiation machine in Marietta due to flaws in the computer program controlling the device (1985 and 1986), Explosion of the European Space Agency's Ariane 5 rocket, in less than 40 s after lift-off on 4 June 1996 due to software design errors and insufficient testing, blackouts in the North-East US during the month August, 2003 due to an error in the AEPR (Alarm and Event Processing Routine) software, etc.

The abilities to design, test, and maintain software has grown fairly, lot of further improvements are desired in the field. The software development process has become really a challenging task for the developers. Accordingly, the main concern about productivity and quality of computer systems has been changing from the hardware to the software systems. Now the question arises, what makes *productive and quality software*? The answer is, the software that enables a seamless technology experience for people wherever they are—in the home, in the office or on the go. Arguably the most important software development problem is building software to customer demands so that it will be *more reliable, built faster*, and *built cheaper* (in general order of importance) [5]. Success in meeting these demands affects the market share and profitability of a product for the developer. These demands conflict, causing risk and overwhelming pressure, and hence strong need for a practice that can help them to have a tight control over the software development process and develop software to the need of the software market.

Software reliability engineering (SRE) discipline came forward to create and utilize sound engineering principles in order to economically obtain software systems that are not only reliable but also work proficiently on real machines, in the early 1970s. This made software reliability study recognized as an engineering discipline. The next concern of software engineering was scheduling and systematizing the software development process to monitor the progress of the

various stages of software development using its tools, methods, and process to engineer quality software and maintaining a tight control throughout the development process. Here the most important thing that must be clearly defined is what quality refers to in context to the developers and the end users. More often it is defined in terms of internal quality and external quality with a focus on transforming the user's requirements (external quality characteristics) into the quality characteristics of the software system developers (internal quality characteristics). SRE broadly focuses on quantitatively characterizing the following standardized six quality characteristics defined by ISO/IEC: functionality, usability, reliability, efficiency, maintainability, and portability. *Software reliability* is accepted as the key characteristic of software quality since it quantifies software failures—the most unwanted events, and hence is of major concern to the software developers as well as user. Further it is the multidimensional property including other customer satisfaction factors such as functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. For this reason it is considered to be a "*must be quality*" of the software.

One of the major roles of SRE lies in assuring and measuring the reliability of the software. The tools of SRE known as software reliability growth models (SRGM) are used successfully to develop test cases, schedule status, to count the number of faults remaining in the software, and estimate and predict the reliability of the software during testing and operational environment. Foundation of research in reliability modeling for software seems to be as old as 40 years however, in the past 30 years the field has experienced extensive growth. Many reliability engineering scientists and research scholars have done excellent study of the various aspects of the software quality measurement during the software development and maintenance phases. Many SRE books are available that focus on software reliability modeling. However, similar to the IT advances, the software reliability modeling has also advanced, incorporating the many recent and challenging issues in reliability measurement. In this book we present the state-of-the-art modeling in software reliability over the past 40 years in one place. Various optimization applications of reliability modeling solved using the tools of operational research are also discussed in the later chapters of the book. In the next sections of this chapter we elaborate some important concepts of SRE, software development and testing, SRGM classification, and a brief review of the literature.

1.1 Software Reliability Engineering

Software engineering is relatively a young disciple and was first proposed in 1968 at a conference held to discuss the problem known at that time as software crisis. Software crisis was the result of introduction of the powerful, third-generation computer hardware. Many software projects run over budget and schedule, were unreliable, difficult to maintain, and performed poorly. The software crisis was originally defined in terms of productivity, but evolved to emphasize quality. New

techniques and methods were needed to control the complexities of the large software projects and the techniques developed and adopted lead to the foundation of SRE. The SRE technologies were mainly inherent (such as specification, design, coding, testing, and maintenance techniques) that aid in software development directly and management technologies (such as quality and performance assessment and project management) that support the development process indirectly. Our focus in this book lies on the management technologies.

A number of definitions of SRE are made by several people and it is difficult to say which definition describes it most appropriately. Here we would like to mention that the word engineering is an action word, which aims to find out ways to approach a problem. The problems as well as approaches to resolve them have changed drastically in the past decade and the changes are continued, the definition of SRE is also changing and evolving. The IEEE [6] society has defined SRE as widely applicable, standard, and proven practice that apply systematic, disciplined, quantifiable approach to the software development, test, operation, maintenance, and evolution with emphasis on reliability and the study in these approaches. Further ISO/IEC defined software reliability as "an attribute that a software system will perform its function without failure for a given period of time under specified operational environment".

There are several simultaneous benefits of applying SRE principles on any software development project; broadly they can be listed as—it insures that product reliability conforms to the user requirements, lowers the development cost and time with least maintenance and operation costs, improved customer satisfaction, increased productivity, reduced risk of product failure [5], etc. Conceptually SRE is a layered technology (Fig. 1.1). It rests on the organizational commitment to quality with a continuous process improvement culture and has its foundation in the process layer. Process defines the framework for management control of the software projects, establishes the context in which technical methods are applied, work products are produced, quality is insured, and change is properly managed. SRE methods provide the technical "how to's" for building the software whereas the tools provide automated or semi-automated support for the processes and methods [6].

SRE management techniques work by applying two fundamental ideas:

- Deliver the desired functionality more efficiently by quantitatively characterizing the expected use, use this information to optimize the resource usage focusing on the most used and/or critical functions, and make testing environment representative of operational environment.

Fig. 1.1 Software reliability engineering layers

- Balances customer needs for reliability, time, and cost-effectiveness. It works by setting quantitative reliability, schedule and cost objectives, and engineers' strategies to meet these objectives.

The activities of SRE include:

- Attributes and metrics of product design, development process, system architecture, software operational environment, and their implications on reliability.
- Software reliability measurement—estimation and prediction.
- The application of this knowledge in specifying and guiding system software architecture, development, testing, acquisition, use, and maintenance.

There exist sound process models of SRE known as software development life cycle (SDLC) models, which describe the various stages of software development in a sequential and planned manner. Most of the models, model the SDLC in the following stages: requirement analysis and definition, system design, program design, coding, testing, system delivery, and maintenance. The tools and techniques of SRE provide means to the software engineer to monitor, control, and improve the software quality throughout the SDLC.

1.2 Software Development Life Cycle

Software development realized using the tools and techniques of SRE enables developers to deliver enough reliability avoiding both excessive costs and development time. Software development involves a set of ordered tasks; each task can be called as a generic process and the process of software development is known as SDLC. The IEEE computer dictionary has defined SDLC as "the period of time in which the software is conceived, developed and used". The software life cycle process model describes software products life from the conceptualization stage to the final implementation and maintenance stage. Many life cycle process models are described in the software engineering literature. The generic process framework applicable to the vast majority of software projects includes the following stages:

- Analysis and specification
- Software development
- Verification and validation
- Maintenance and evolution

Each framework activity is populated by a set of software engineering actions such as software project tracking and control, risk management, quality assurance and measurement, technical reviews, reusability measurement, etc. Following the generic framework activities every software development and engineering organization describes a unique set of activities it adopts with the complemented set of engineering actions in terms of a task set that identifies the work to be accomplished.

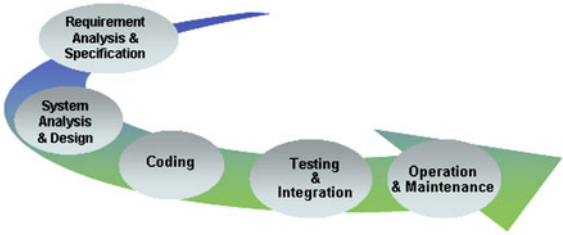

Fig. 1.2 Waterfall model

Almost all known process models bear at least some similarity to the preliminary process model known as waterfall models. The waterfall model was proposed by Royce in 1970. The framework activities of the model are shown in Fig. 1.2 and can be illustrated as follows.

Activity 1: Requirement Analysis and Specification

This phase forms the foundation stage for building successful software. Defining the project scope, software requirements, and providing specifications for the subsequent phases and activities. Project scope definition includes the study of the users' need for the system and their problems. This is accomplished with frequent interaction with the users. Once the scope is defined the requirement collection activity starts. Requirement collection is actually the study of product capabilities and constraints. It includes collection of product functionality, usability, intended use, future expectations, user environment, and operating constraints. Requirement analysis concludes with a feasibility study of user requirements, cost benefit estimation, and documentation of collected information and feasibility report. The document holds the different specific recommendations for the candidate system such as project proposal, environmental specifications and budget, schedule, and method plans. The immediate following activity is system specification. The basic aim of this activity is to transform the user requirement-oriented document to the developer-oriented document (design specifications). This is the first document that goes into the hands of the software engineers and forms the foundation document of the project; hence, it must precisely define essential system functions, performances, design constraints, attributes, and external interfaces. In this phase, the software's overall structure and its nuances are defined. All activities of this phase must be accomplished very crucially. A well-developed specification can reduce the occurrence of faults in the software and minimizes rework.

Activity 2: System Analysis and Design

System design activity is concerned with architectural and detailed project design. A detailed analysis of the specification document is carried to know the

performance, security and quality requirements, system assumptions, and constraints. This study enables partitioning of full system into smaller subsystems and definition of internal and external interface relationships. The needed hardware and software support are also identified. In terms of the client/server technology, the number of tiers needed for the package architecture, the database design, the data structure designs, etc. are all defined in this phase. The architectural design is completed with an architectural document design. This document is followed by a detailed system design activity. Here the program structure, algorithmic details, programming language and tools, test plans are specified. The final outcome of this phase is a detailed design document. The design engineers must take care that the designed system architecture, program structure, and algorithm design conforms to the specification document. Any glitch in the design phase could be very expensive to solve in the later stage of the software development.

Activity 3: Coding

The program structures and algorithms specified in the design document are coded in some programming language—a hardware readable form. This phase consists in identifying existing reusable modules, coding of new modules, modifications in existing modules, code editing, code inspection, and a final test plan preparation. If the program design is performed in a detailed manner, code implementation can be accomplished without much complication. Programming tools like compilers, interpreters, debuggers are used to generate the code. Different high level programming languages like C, C++, Visual basic, and Java are used for coding. With respect to the type of application, the right programming language is chosen. Once the independent programs are implemented they are linked to form the modular structure of the software according to the interface relations defined in the design document.

Activity 4: Testing and Integration

Once the code is generated, the software testing begins. Testing is the key method for dynamic verification and validation of a system. The objectives of the testing phase are to uncover and remove as many faults as possible with a minimum cost, to demonstrate the presence of all specified functionalities, and to predict the operational reliability of the product. Testing is generally focused on two areas: internal efficiency and external effectiveness. The goal of internal testing is to make sure that the computer code is efficient, standardized, and well documented. The goal of external external testing is to verify that the software is functioning according to system design and that it is performing all necessary functions or sub-functions. Initially testing begins with unit testing of independent modules

then the modules are integrated and system testing is performed followed by acceptance testing.

Activity 5: Operation and Maintenance

The system or system modifications are installed and made operational in the operational environment. The phase is initiated after the system has been tested and accepted by the user. Installation also involves user training primarily based on major system functions and support. The users are also provided installation and operation manuals. This phase continues until the system is operating in accordance with the defined user requirements. Inevitably the system will need maintenance during its operational use. During this period the software is maintained by the developer to conquer the faults that remain in it at its release time. Software will definitely undergo change once it is delivered to the customer. There are many reasons for a potential change. Change could happen because of some unexpected input values into the system. Changes in the system could directly affect the software operation. The software should be developed to accommodate changes that could happen during the post-implementation period.

The waterfall model maintains that one should move to a phase only when its proceeding phase is completed and perfected. Phases of development in the waterfall model are discrete, and there is no jumping back and forth or overlap between them. Several modifications of waterfall model are known in the literature to allow the prototyping such as phased, evolutionary, and agile development of the software. The basic difference between waterfall model and its modifications is the flexibility in the sense that the task performed in any stage of the development can be verified and validated with the previous stages so as to reduce the development cost, time, and rework. For example the user and the requirement analyst can review the specifications once they have been defined to insure that the proposed product is what the users want. This allows the user and the software team to visualize the actions performed and to find the aspects of further improvements in the accomplished tasks. Figure 1.3 demonstrates a modified waterfall model that includes reviews and feedbacks in between various development stages.

1.3 Why Software Testing Is Important

Despite using the best engineering methods and tools during each stage of the software development the software is subject to testing in order to *verify* and *validate* it (software V&V). The previous discussion on the importance of computing systems and human dependence on them clarifies the need of software testing. Bugs if appear during software operation in user environment can be fatal to the users in terms of loss of time, money, and even lives depending on criticality