THE DUMMIES' GUIDE TO SOFTWARE ENGINEERING

Rosina S Khan

Dedicated to:

You, the Valued Reader

Copyright Information

Table of Contents

Preface

This book is for Computer Science and Engineering undergraduate students which is simple to comprehend and is especially written in the format these students would enjoy reading and benefit from learning the foundation concepts of Software Engineering. It has been integrated from various resources including but not limited to the following titles:

[1] Roger S Pressman, Software Engineering- A Practitioner's Approach, McGraw Hill, 7th Edition, 2010

[2] Ian Sommerville, Software Engineering, Addison-Wesley, 9th Edition, 2011

[3] Douglas Bell, Software Engineering for Students – A Programming Approach, Addison-Wesley, 4th Edition, 2005

Organization of the Book

The book is organized into nine chapters and two Appendices.

In the first introductory chapter, we become familiar with professional software development that would consist of people developing software products including code, programs and documentation. [2,3]

In chapter 2, we are concerned with Requirements Engineering introducing user and system requirements. The former tells the system users what service the system will provide while the latter defines system's functions, services and constraints in greater detail. In this phase, a requirements specification is also developed for users describing their view of the system, and expressed in natural language. [2,3]

The third chapter on Design is lengthy taking into account all possible types of system designs such as User Interface Design, Modularity, Architectural-Based Design, Pattern-Based Design, WebApp Design, Navigation Design, Component-Level Design, Object-

Oriented Hypermedia Design Method (OOHDM), Object-Oriented Design using UML (Unified Modeling Language) as well as Data Flow Design. [1,2,3]

Chapter 4 deals with software processes such as Waterfall Model, The Spiral Model, Prototyping, Incremental Development, Open Source Software Development, Agile Methods and Extreme Programming and the Unified Process. [1,2,3]

Chapter 5 describes the Project Teams in respect of the principles behind the teams working, how functional and project teams operate and finally how the chief programmer team operates. [3]

Chapter 6 is about Software Metrics and Quality Assurance. Metrics or measures applied on software will help us determine the quality of software, monitor and improve the software as well as they are helpful for assessing different software development approaches. Quality assurance is similar in that monitoring and controlling the process of a software development system helps meet its quality goals. [3].

Chapter 7 portrays about Project Management on the overall. It is an activity trying to verify that the software development is successful. [3]

Chapter 8 summarizes software testing. Testing is done to ensure that the software does what it is intended to do and to discover any defects the program has before putting it to use. [2]

Chapter 9 goes on to explain software evolution. As a matter of fact, software development does not stop after it has been developed but continues to evolve throughout the lifetime of the system. [2]

Last comes the Appendix section. There are two short appendices A and B. Appendix A considers several short sample case studies on software engineering. Appendix B illustrates a UML summary showing the basic concepts and notations. [3]

Acknowledgments

I express my heart-felt gratitude to Roger S Pressmam, Ian Sommerville and Douglas Bell and their publishers for gathering some of their resources and molding them with what I have and therefore, this book becoming alive.

Last but not the least, I am thankful to my family for their support during the write-up of this book.

C H A P T E R 1

Introduction

1.1 Professional Software Development

Teams rather than individuals develop software products such as for integrating in other devices or in the form of information systems or CAD systems etc. The entire system not only consists of software products but also documentation aided by other programs, guides and data to make the whole system in the process of professional software development. An amateur software development would consist of an individual developing software for his needs without documentation and guides but a fully professional software development would consist of people developing software products including code, programs and documentation.

There are two kinds of software products that software engineers develop. They are:

Generic Products: These are stand-alone systems designed to meet customers' needs generally who are able to buy them. They include databases, word processors, drawing packages, and project-management tools. Software products in this category also include library information systems, accounting systems, or systems for maintaining dental records, which are designed for a specific purpose.

Customized Products: These are systems that are tailored according to the needs of a particular customer. Examples of such software products include control systems for electronic devices or systems to support a particular business process.

The specific set of attributes you would like to expect from a software system are summarized below:

Maintainability: There should be scope for software change in a changing business environment.

Dependability and Security: Dependable software should be reliable, secure and safe. They should not damage physically or economically in the case of system failure.

Efficiency: Software efficiency includes responsiveness, processing time and memory utilization.

Acceptability: Software should be acceptable which means they should be understandable, usable and compatible with other systems the users use.

1.2 Software: Problems and Prospects

The problems centered around software development and the goals software developers seek to achieve are:

- meeting users' needs
- low cost of production
- high performance
- portability
- cost of maintenance
- high reliability
- delivery on time

Meeting users' needs:

It is obvious that software engineers should build products according to their clients' needs but all along evidence has shown that 2% is only used as delivered. This shows that requirements engineering or analysis should play an important role in the whole process rather than reliability or cost. However, it should be taken into account that smaller systems can be tailored to serve clients' needs better.

Cost of Software Production

Industrialized nations spend on software development in significant proportions. The cost of software is influenced by the productivity of the software developers and their salaries. The performance of these software developers is dependent not only on their ability to code but also to carry out clarifying the problem specification, software design, coding, testing and documentation. It is sort of difficult to predict how much time a piece of software will take to develop and hence the cost and delivery date of software is also affected.

In the early days of computers, hardware was costly and software relatively cheap. Nowadays due to mass production and miniaturization, hardware is cheap and software costly. For instance in 1955, software cost only 10% of about a project while hardware cost 90%. Nowadays software has risen to about 90% of a project's cost, while the remaining 10% cost comprises hardware. These proportions should be treated carefully. They hold for certain projects only and not in each and every case.

Some software is simple and easy to write but most commercially used software is large and extremely complex. Clearly, the cost of testing is enormous, whereas coding constitutes only a small part of software development.

In summary, what we see today is that software is expensive:

- relative to the gross national product
- because developers exhibit apparently low productivity
- relative to the cost of hardware
- in popular perception.

Software Performance

Examples of software performance include:

- an interactive system responds within a reasonably short time

- a control signal is output to a plant in sufficient time
- a game runs sufficiently so fast that the animation appears smooth
- a batch job is not taking 12 hours when it should take one.

Portability

Software should be portable from one hardware to another and that is what is expected given the advent of high-level languages and international standards. But that's not actually the case because clients are tied to their suppliers for switching allegiance at a considerable cost in converting software.

Maintenance

This factor comes into the picture after a piece of software has been written and put into operation. There are two types:

1) **Remedial Maintenance**: The software is tested for faults or bugs.
2) **Adaptive Maintenance**: The software is modified either because the user's needs have changed or the computer, operating system or the programming language has changed.

Reliability

A software piece is reliable if it keeps working and working without undesirable malfunctions.

Now we need to coin three terms that would make a software piece undesirable:

1) Error : a wrong decision made during software development.
2) Fault: a problem or bug causing software to malfunction.
3) Failure: an event causing software to malfunction.

An error is a mistake causing one or more faults in software. Failure will result while the system is tested. Failures are symptoms that end-users experience while faults are what the developers have to solve.

Delivery on Time

The fewer faults or bugs a software piece has, the higher the chances that it will be delivered to the client on time.

1.3 Software Crisis

A software crisis arises when:

- it fails to do what users want it to do
- it is expensive
- it isn't always fast enough
- it cannot be transferred to another machine easily
- it is expensive to maintain
- it is unreliable
- it is often late
- it is not always easy to use

1.4 Remedy: Software Engineering

There are problems in developing software and so what is the remedy? A number of methods and tools comprising software engineering is the likely answer. Some of them are as follows:

- greater emphasis on carrying out all stages of development systematically.
- computer assistance for software development – software tools.
- an emphasis on finding out exactly what the users of a system really want (requirements engineering and validation)
- demonstrating an early version of a system to its customers (prototyping)
- use of new, innovative programming
- greater emphasis on trying to ensure that software is free of errors (verification).
- incremental development, where a project proceeds in small, manageable steps.

1.5 Software Engineering Ethics

Like other engineering disciplines, software engineering imposes professional responsibilities without pertaining to laws among teams of people working to develop software products. These responsibilities cover more than the appropriate application of technical skills such as honesty and integrity as well as ethical and moral ways on the part of the employees. Some of these responsibilities on a wider range are:

1) Confidentiality: You must respect the confidentiality of your employers or clients although no such treaty may have been signed.
2) Competence: You should accept work only that you know you have the right skill set and are competent. Accepting any other work might end in jeopardy and chaos.
3) Intellectual property rights: You should be aware of local laws governing intellectual property rights such as patent and copyrights and protect those property rights of employers and clients.
4) Computer misuse: You may not mishandle your employer's computers by applying your technical skills. This may range from playing games on their computers, for instance, to infecting them with viruses and malware.

C H A P T E R 2

Requirements Engineering

Requirements Engineering fall under two categories: User Requirements and System Requirements.

User Requirements are natural language statements and diagrams telling the system users what service the system will provide and the constraints under which the system will operate.

System Requirements define the software system's functions, services and constraints in greater detail. It should clearly state what should be implemented. It may even become the part of the contract between the system buyer and software developer.

Software system requirements may be further categorized as functional and non-functional requirements.

For instance, functional requirements define what service the system will provide, how it reacts to particular inputs and how it should behave under certain conditions. It may also mention what the system may not do.

Non-functional requirements consist of constraints and functions imposed by the system such as timing constraints, constraints on the development process and standards-based constraints. These may be applied to the system as a whole.

2.1 How to Elicit Requirements

We can categorize three activities to elicit requirements:

1. Listening
2. Thinking
3. Writing

Listening involves the users' needs or requirements of the system, asking them about goals and constraints of the system and finally recording their viewpoints of the system requirements.

Requirements analysis is the level where the system simply thinks. S/he transforms the users' view of the system as an organized representation of the system as seen by the analyst.

Requirements definition is a clear statement, usually in natural language, of what the system should actually provide for its user. This definition translates into a requirements specification.

2.2 Requirements Specification

Three important factors to be considered are:

- the level of detail
- to whom the document is addressed
- the notation used

The specification, as we have already mentioned, should tell in detail the user's view of the system rather than how the system should be implemented.

The specification is a contract between users and developers. While users will prefer it in natural language, developers would like to use some mathematical notations to be more precise. This problem can be resolved by drawing up two documents. These are:

- A requirements specification written for users, describing their view of the system and expressed in natural language. This is the substance of the contract between the users and the developers.
- A technical specification that is used by developers, expressed in some formal notation and describing only a part of the information in the full requirements specification.