Modern Software Engineering Guidebook

Practical solutions to efficient and economical software management

Dr. Shakti Kundu

www.bpbonline.com

First Edition 2024

ISBN: 978-93-55519-801

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Dedicated to

Every challenging work needs self-effort as well as the guidance of elders, especially those who were very close to our hearts.
My humble effort I dedicate to my sweet and loving father

Shri Shamsher Kundu

and mother

Smt. Anupama Kundu

whose affection, love, encouragement and prays of day and night make me able to get such success and honor.

About the Author

Dr. Shakti Kundu is serving as Associate Professor, School of Engineering and Technology, Computer Science Engineering at BML Munjal University (BMU), Gurugram, Haryana, India. His current research interests are Web Data Mining, Software Engineering, Intelligent Systems, Big Data Analytics, and Digital Marketing. He brings to his classes, over 15+ years of teaching experience. He also worked with various reputed universities and holds administrative experience as well. He has published more than 56 research papers in reputed journals and conferences, which are indexed in various international databases. Called for several invited talks and keynote addresses and chaired sessions in various conferences at the national/international level in the field of Big Data Analytics, Web Data mining, and Software Engineering. Also serving as a member of the reviewer board of various peer-reviewed journals and conferences. His leadership abilities are reflected while successfully hosting various seminars, workshops, short-term training programs, and conferences. He is a reviewer board member of SCI & Scopus indexed journals like International Journal of Healthcare Information Systems and Informatics IGI Global, IEEE Transactions on Computers, International Journal of Engineering & Technology Science Publ Corp UAE, Computer Applications in Engineering Education Wiley USA, Journal of Intelligent and Fuzzy Systems IOS Press Netherlands and, Journal of Intelligent Manufacturing Springer Netherlands. He is a life member of CSI, ISTE, IAENG, AIRCC, and IAEME.

About the Reviewers

- **Chetan Sachdeva** is a tech enthusiast with over 14 years of expertise in C++ and Python software development, with a dynamic career spanning Automotive, AR/VR, IoT, Android native development, Typography, and Printing RIP.

  His true passion lies in solving complex problems with data structures and algorithms. His entrepreneurial spirit has led to the successful design of software and products for startups. Beyond his professional endeavors, Chetan is an avid reader of non-fiction and IT-related books, staying at the forefront of industry trends.

- **Raj Agrawal** is a distinguished Senior Engineering Manager at Salesforce, renowned for his expertise in distributed systems, generative AI, and enterprise mobile applications. He has a proven track record in leading technological strategy and execution, with deep expertise in microservices and cloud computing. Known for fostering high-performing, diverse teams, Raj has significantly contributed to advancing AI technologies while ensuring robust data privacy and security. As an author, he provides insightful perspectives that drive innovation and inspire future engineers. Raj's work continues to impact the broader technological landscape, promoting excellence and progress.

Acknowledgement

There are a few people I want to thank for the continued and ongoing support they have given me during the writing of this book. First and foremost, I would like to thank my parents, wife, and son Shaurya for putting up with me while I was spending many weekends and evenings writing. I could have never completed this book without their support.

This book would not have happened if I had not the support of my colleague(s) and students. My gratitude goes to Prof. Maneek Kumar, Dean, School of Engineering and Technology, for providing valuable insights into some of the new features and allowing me to have access to the impressive Computer Science Engineering lab, BML Munjal University (BMU), Gurugram, Haryana, India.

I am also grateful to BPB Publications for their guidance and expertise in bringing this book to fruition. It was a long journey of revising this book, with valuable participation and collaboration of reviewers, technical experts, and editors.

Finally, I would like to thank all the readers who have taken an interest in my book and for their support in making it a reality. Thank you for your encouragement and kind words.

Preface

Building software applications is a complex task that requires a comprehensive understanding of the latest technologies and trends. The powerful techniques of software engineering have become increasingly popular in the field of software development.

This book is designed to provide a comprehensive guide to building software applications with software engineering. It covers a wide range of topics, including the basics, advanced concepts such as object-oriented design, software implementation, software testing strategies, and the use of software metrics for building robust and scalable software applications.

Throughout the book, you will learn about the key features of software engineering and how to use them to build software applications that are efficient, reliable, and easy to maintain. You will also learn about best practices and design patterns for creating software applications and will be provided with numerous practical examples to help you understand the concepts.

This book is intended for developers who are new to the concepts of software engineering and want to learn how to build software applications. It is also helpful for experienced developers who want to expand their knowledge of these technologies and improve their skills in building robust and reliable software applications.

With this book, you will gain the knowledge and skills to become a proficient engineer in the field of software engineering. I hope you will find this book informative and helpful.

The primary goal of this book is to provide information and skills that are necessary for end users as well as engineers to understand the real scenario of software engineering. This book contains real-life examples that will help you remember, analyze, apply, and create the various trends and techniques of modern software engineering. The book is composed of an introduction with 15 chapters each consisting of conclusions at the end. The conclusions present the original contributions of the author and point out directions for the next chapter. Over the fifteen chapters of this book, you will learn the following:

Chapter 1: Introduction to Software Engineering - explains everything needed for the engineer to develop software applications based on the phases of software engineering, including requirements, analysis, design, development, testing, implementation, and maintenance. Furthermore, the chapter also gives the reader an overview of the software characteristics, software applications, and objectives of software engineering.

Chapter 2: Software Processes - presents a detailed overview of the software engineering process and shows the differences between software process, project, and product. This is essential content for the entire book as this chapter covers fundamental aspects of process assessment that lead to process improvement and capability determination. In addition, the description of the software process capability maturity model and software life cycle models define the task that tasks that are performed at each phase in the development of software.

Chapter 3: Software Life Cycle Models - covers the extended discussion on software life cycle models, including prototyping, object-oriented, agile, rapid application development, iterative enhancement, v-model, and extreme programming. Furthermore, the chapter shows how various software life cycle models work with their benefits and drawbacks.

Chapter 4: Software Requirements - allows the reader to learn fundamental concepts related to functional requirements and non-functional requirements, including product, organizational, and external. Furthermore, the chapter explains user requirements and system requirements with details and practical examples. The users of a requirement document are highlighted in the shape of a software requirements document for better description and understanding.

Chapter 5: Software Requirements Engineering Process - gives special attention to feasibility study and requirements elicitation and analysis, demonstrating how to implement robust and extensible software applications and explaining concepts related to requirement validation and its related techniques. Close-ended and open-ended software prototyping concepts help the user satisfy the requirement and achieve a workable version. For problem analysis, change analysis, and change implementation, the requirement change management process plays a vital role.

Chapter 6: Software Reliability - shows basic concepts of software reliability and its related metrics. It also shows the differences between fault, error, and failure. The chapter includes practical examples of programming for reliability and software reuse. Several levels at which software reuse can be taken into consideration are highlighted for further comprehension.

Chapter 7: Software Design - explains the basics of software design and its design process. This chapter also allows the reader to learn the concepts of data, architectural, component-level, and user interface design. The fundamental design concepts and structured design techniques provide model support for the establishment of software systems.

Chapter 8: Object-Oriented Design - is dedicated to highlighting the role of object and object classes to give the reader more familiarity with the description of objects and their

relationship. It also talks about the various activities in the object-oriented design process with its associated phases. This chapter covers practical examples of design models such as the sequence model and state machine model.

Chapter 9: Software Implementation - presents the structured coding techniques. This chapter covers practical examples of coding styles and coding methodology. It also describes the list of some of the frequently used coding tools. In addition, emphasize the usage of code documentation and further follow code standards with well-stated guidelines.

Chapter 10: Software Maintenance - covers the concept of the software re-engineering process model. This chapter describes the elements, objectives, and issues of change management. For document specification, the essential tasks of configuration management are discussed. Furthermore, several tools and techniques used for software maintenance are described with their respective appellation.

Chapter 11: Software Testing Strategies - allows the reader to understand the importance of software testing and its strategies. This chapter covers various levels of testing such as black box, white box, validation, alpha, beta, system, recovery, security, stress, and performance testing with their benefits and drawbacks. It also highlights the process of debugging and its related approaches.

Chapter 12: Software Metrics - focuses on software quality metrics. This chapter covers practical instances of metrics for analysis, design, source code, testing, and maintenance. The discussion on software metrics is well addressed to quantify object-oriented principles such as polymorphism, inheritance, coupling, and cohesion.

Chapter 13: Quality Management - shows the concept of quality with its quality-determining elements. This chapter covers relevant concepts such as software quality assurance, software reviews, and formal technical reviews. Three standards for ISO-9000 with its related guidelines were discussed for the creation of a quality software system.

Chapter 14: Software Project Management - is dedicated to showcasing the concept of project planning to give the reader more familiarity with the main principles associated with project progression. This chapter covers practical examples of project scheduling and project staffing. It also describes the goals of the capability maturity model in the five defined levels.

Chapter 15: Latest Trends in Software Engineering – highlights the rise and exponential growth of software engineering. It discusses the trends in software engineering and top IT job profiles in the current environment.

Coloured Images

Please follow the link to download the
Coloured Images of the book:

https://rebrand.ly/iwkhg5s

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

Chapter 1
Introduction to Software Engineering

Introduction

Understanding software engineering is crucial to building high-quality, error-free software on schedule and at a lower cost. The field of software engineering assists in assessing the opportunities and hazards that software brings to our daily lives. Computers are now a necessary component of almost every economic sector. A crucial component of any computer-based system is software. Software development calls for a methodical approach. Even the development of a single element of software is considered a project since it involves numerous tasks. Software developers are responsible for every user-friendly user interface, flawless online transaction, and interactive application. Their work is thorough. They put the user experience first, making sure that software is not only useful but also simple to use and intuitive, which improves our interactions with technology. Software engineering is the result of the evolution of all the engineering facets of software development. You will find the answers to these queries in this and subsequent chapters.

Structure

In this chapter, we will cover the following topics:

- Basics of software engineering
- Principles of software engineering

- Software characteristics
- Software applications
- Objectives of software engineering
- Phases of software engineering

Objectives

In this chapter, we will define software engineering and its related principles. We will also learn about software characteristics, kinds of software applications, objectives of software engineering, and phases of software engineering.

Basics of software engineering

The term *software engineering* was originally mentioned during a meeting that was held in Germany in 1968 and was hosted by the **North Atlantic Treaty Organization** (**NATO**). The creation of high-quality software at a low cost is the primary focus of the technical discipline known as software engineering, which is a subfield of computer engineering. It is a subdiscipline of computer science that seeks to apply engineering concepts to the process of creating, operating, modifying, and maintaining the software components of a variety of different systems. A *generalist* understanding of all the interrelated components of contemporary computers can be obtained through computer science, whereas large-scale, complicated software systems, like personalized recommendation algorithms, are the domain of software engineering or development.

The field of software engineering focuses on the operational aspects of creating and delivering software that is of value to users. The cost of developing software comprises approximately 60% of the total, while the cost of testing accounts for 40%. System models, notations, rules, design guidance, and process guidelines are all components of structured approaches to the software development process. The most significant issues that software engineering must face are accommodating a growing range of requirements, meeting customer expectations for shorter delivery periods, and creating reliable software.

Engineering concepts

Engineering is the development, operation, modification, and maintenance of useful objects and systems via the use of well-understood scientific procedures.

Software

The term *software* refers to not merely the programs themselves but also all of the documentation and configuration data that is associated with the programs and is required for them to function properly. A software system will typically consist of a number of

individual programs, configuration files that are used to set up the programs, system documentation that describes the structure of the system, user documentation that explains how to use the system, and websites that allow users to download the information.

A system is a collection of elements that interact with one another and possibly with the environment outside the system's boundaries. By breaking down systems into their constituent subsystems, we can comprehend them better. Distinguishing a system's software components from its other components is a very challenging task.

Principles of software engineering

Software engineers create software solutions for end users by utilizing engineering principles and their understanding of programming languages. Among the numerous job options open to software engineers include the design and development of operating systems, network control systems, middleware, business applications, and computer games. One of the first scholars to propose a set of software engineering principles was *Alan Davis* in 1994. The following are some software engineering tenets:

- **Delivering goods to clients ahead of time:** Since it is very difficult to fully comprehend and record user needs at the requirement phase, it is more efficient to present a product prototype to consumers, solicit their comments, and then move forward with full-scale development of the product.
- **Identifying the issue before drafting the requirements:** According to this theory, we must make sure the problem is fully understood before the software engineering team rushes to provide a solution. Next, investigate the different options and the possible fix. This principle highlights the necessity of having a thorough understanding of the issue before establishing requirements.
- **Assessing design options:** Examine various design architectures and associated algorithms once the requirements have been comprehended and decided upon. Make sure the design and algorithms you have chosen are the best fit for achieving the requirements.
- **Choosing the right process model:** Every project has to choose a process based on factors like corporate culture, project conditions, user expectations, requirements volatility, and the experience of the project participants. There is not a single process model that works for all projects.
- **Putting technique before tools:** According to this idea, the technique must be thoroughly understood before using the tools. If so, the program only pushes us to complete the incorrect task more quickly. Tools are used to do tasks quickly and error-free. This idea suggests that the tools to be utilized should be determined by the technique.

- **Getting it right before you make it faster:** According to this theory, software must function correctly before any improvements can be made. This means that after ensuring that the software functions properly, developers should concentrate on enhancing its performance.
- **Inspecting code:** According to this theory, finding problems through inspection—first put out by IBM's *Mike Fagan*—is far more effective. An inspection's initial results revealed a 50% to 90% reduction in test time.
- **Effective management outperforms advanced technology:** According to this theory, an accomplished manager can achieve remarkable outcomes with little in the way of resources.
- **People are the key to success:** According to this theory, success in the labor-intensive field of software development depends on having individuals with the right combination of talent, experience, and motivation. Many of the limitations in process, approach, or tools can be solved by the appropriate people.
- **Accepting accountability:** According to this theory, you should assume full responsibility for doing the system correctly if you designed it.

The following is another set of widely accepted software engineering principles that are adhered to during the software creation process:

- **Formality and rigor:** Exactness or precision are the definitions of rigor. It is important to specify the level of rigor when developing software. For software to be dependable and verifiable, this idea is crucial. Different rigor degrees exist. When outputs are defined by mathematical laws, formality is the maximum level of rigor. Formality and rigor can be described not just in terms of the software being developed but also in terms of the procedures utilized in its creation.
- **Separations of concerns:** There are a number of factors to consider when developing software. There could be additional features in addition to the basic functions, such as security and the capacity to operate with various databases. Each of these facets must be addressed independently while developing and designing the program. Generally, the software should be designed so that distinct modules handle the various functions. A web application must possess distinct modules for managing incoming web traffic and another for gaining access to the database. To carry out that role, the core ought to call upon these modules. It is crucial that the module addresses multiple facets.
- **Modularity:** The ideas of *separation of concerns* and modularity are closely related. Modules for a complex system should be broken down into simpler ones. Only explicitly defined interfaces should be used for communication between the modules. The ability to reuse code across many systems is facilitated by modularity. It also contributes to the system's maintainability. The relationship between the methods and functions inside a module is implied by cohesiveness. The

interdependence between modules is referred to as coupling. The ideal properties of a module are low coupling and high cohesion.

- **Abstractions:** The capacity to extract the functionality's key features without requiring knowledge of its specifics is known as abstraction. Programming languages are a common illustration of the abstraction notion. The language gives us a strong syntax to design software that can address issues on several platforms while abstracting hardware-specific information.
- **Be prepared for changes:** Frequently, the earliest iterations of software products lack clarity regarding their precise requirements. Therefore, it is important to design the program such that it can easily adapt to changes.
- **Generality:** Software might be general or specially designed. Generic software is more advantageous in the long run, even though it costs more to produce. This idea is the foundation of several tools, including spreadsheets and document writers. These programs can be used for a variety of purposes.
- **Incrementality:** This idea dictates how the software must be developed. It is usually advised to develop a behemoth instead of adding functionality to everything at once. We can offer preliminary prototypes for early input. This will facilitate quicker consumer deployment of the product.

Software characteristics

Software is not a physical component of a system; it is logical. As a result, software and hardware have different properties. Software is developed and engineered rather than produced in the conventional sense. There is no *wear down* of the program. Even while component-based assembly is becoming more common in the business, most software is still created on demand. Any physical component of a computer is referred to as hardware. This covers both external components like keyboards and monitors as well as internal components like hard drives and microchips. Software, which includes computer programs and phone apps, is anything that instructs hardware on what to do and how to accomplish it.

Functionality

It speaks to the extent to which the program performs in opposition to its intended use.

The features and capabilities that a software program or system offers to its users are referred to as its functionality. It is among the most crucial features of software since it establishes whether or not the program will be beneficial for the intended usage.

Software can be more powerful and diverse, but it can also be more complex, as its capability increases. It is critical to strike a balance between functionality and requirements for usability, maintainability, and scalability.