Preface

Theme

Developing efficient software in a timely manner continues to be a struggle for many students and software developers in the real world. As both the size and scope of software systems continues to grow, the complexity of the solutions required to fill the needs of clients, too, increases. Software development has exceeded the mere practice of programming to involve much more planning, analyzing, designing, and delivering; as a result, students must learn how to approach software projects and complete them from start to finish. In light of this trend, a plethora of tools and solutions to any given problem now exists, and developers must know when and how to appropriately use these tools to accomplish the various tasks that come with software engineering projects.

It is not unusual, though, for students to complete many courses only having applied lessons to small problem sets that have limited complexity and freedom. It is also not uncommon for students to learn the science of software engineering without significant hands-on experience. This book aims to ameliorate such gaps in the education of those studying Software Engineering. It recognizes the need to teach key principles, but to also allow the students to gain deeper knowledge that only direct practice can teach. The project-based approach taken by this book reflects the needs observed amongst the students who will one day be a part of teams completing large software projects.

Tools

While many tools for software engineering exist, this book uses some of the most widely used and effective tools available. Most significantly, the object-oriented paradigm and its fundamental principles are used as the primary way to organize and design software. Introduced in phases, the completion of the hands-on project laid out in this book uses various tools for the completion of each phase built on this foundation. UML (Unified Modeling Language), for example, is a choice tool

used to model software and is put into practice by the student for their hands-on project. Also extending the object-oriented paradigm are key object-oriented design patterns and analysis tools that will aid the student during requirements specification, system design, and implementation phases of the project. Lastly, standard documentation for each project phase is presented, exposing students to organizational tools often lacking by those new to software engineering.

Principles

In order to better teach software engineering from a practical approach, this book is written with the following principles as a foundation:

Hands-On Learning: Some skills applicable to software engineering are only learned by being fully engrossed in a project with others. Hands-on learning is very important to bridge the gap between academia and the real world as it gives students the chance to learn from mistakes and successes. Professional skills that are both useful and marketable will be learned by students by working with group members, interacting with clients, and applying the tools and techniques taught throughout the book.

Teamwork: The complex software systems of today do not allow any one developer to create solutions on their own; hence, working in teams is a necessary skill all students must learn and experience. Teamwork requires students to work together to define goals and responsibilities, organize tasks, and provide open communication and feedback. These traits of successful teamwork are just as important as the technical knowledge acquired in this book.

Problem Solving: Every software project begins with a problem that must be defined and solved. The solution to such problems will inherently bring about other subsequent problems along the way. A good developer, while knowing the available approaches and methods, must also know how to solve problems that will likely arise as software engineering projects move from phase to phase. The practice of software engineering is always changing solutions to improve to accommodate the endless number of potential problems and ways to solve them.

The Book

This book covers the process, core concepts, and tools of software engineering, from a primarily object-oriented approach, with a hands-on project as the backbone to support the learning and skill development of the student. Divided into parts, the first set of chapters introduce software engineering principles before the subsequent chapters describe the development of a software project from start to finish. A more detailed description of this book's parts and chapters is given below:

Part I: What Is Software Engineering?

In this first part of the text, a brief history and introduction into the science of software engineering and object-oriented principles used in its process are described.

Chapter 1: In this chapter, *Introduction to Software Engineering*, a summary of the definition and need for software engineering is laid out in a historical context; the complexity of software and lack of success rates are detailed as support. The people, tasks, and activities involved in software engineering are also described to provide evidence that software engineering is not merely limited to programming. Lastly, the life cycles of the development process are depicted.

Chapter 2: In *Object-Oriented Concepts*, object-orientation is introduced and underlying concepts such as classes, modularity, inheritance, and abstraction are discussed. This chapter defines key ideas used by more complex tasks described later in this book.

Chapter 3: Entitled *Modeling with UML*, this chapter describes and depicts the variety of diagrams the Unified Modeling Language provides developers. Class, use case, sequence, component diagrams, and others are discussed as well as the basic parts and association types used by many of them.

Part II: The Software Engineering Project

Part II of this book covers the various phases common to software engineering projects. From defining the problem and requirements to implementing designs and models, these chapters describe the tools and methods used by developers to systematically complete a software project.

Chapter 4: *Starting the Project* provides an overview of the project and non-technical activities such as scheduling, communicating, handling problems, and documenting. The skills covered in this section are practical as they help teams to stay on task, work with each other, and quickly adapt to change.

Chapter 5: The topic of this chapter, *Requirements Elicitation*, is the process of defining the different types of requirements that exist for software projects. First, what exactly a requirement is and the difference between functional versus non-functional, and domain requirements versus constraints is described. Next, the methods of requirements elicitation are discussed as well as the issues and problems concerning requirements. Finally, the production of use case models from requirements is outlined.

Chapter 6: This chapter, *Object-Oriented Analysis*, uses object-oriented concepts to cover the analysis phase of the software development process. In this intermediary phase before design, system requirements are evaluated and refined by identifying objects and their relations, identifying use cases, developing

scenarios, diagramming, and more. The process and issues in analysis are discussed and demonstrated in this chapter.

Chapter 7: In *System Design*, a chapter covering the transition from analysis to design, different design approaches are detailed for the reader. After learning the categories of system design, approaches detailed include function-oriented, structure-oriented, and finally object-oriented. The key design concepts for each approach are covered and differences between them are analyzed. This phase of the process is important to learn as it allows developers to define how various components will provide functionality to the system.

Chapter 8: The *Object-Oriented Design* chapter is concerned with filling the gaps left by analysis and system design. It describes how object design precisely defines objects, subsystems, and constraints to be used by the software to provide a detailed object model. This stage is the important precursor to implementation in the object-oriented paradigm.

Chapter 9: This chapter, *Implementation*, is about the last phase of the system development life cycle. Described in its contents is the process of planning and executing system design plans using the various tasks of implementation. These tasks are divided amongst the different roles and responsibilities of the stakeholders involved. Furthermore, this chapter uncovers some of the standards for integration, code reuse, language choice, and issues arising from implementation.

Chapter 10: The *Testing* chapter tells students about the ins and outs of software testing. First, the importance of testing is highlighted by describing its important role in the creation of correct software. Then, this chapter details how to plan and manage tests, develop tests, and utilize tests. The appropriate tests to use for certain cases are laid out in this chapter as well. The importance of testing is also highlighted as a means for determining the readiness of a software to be released.

Chapter 11: *Project Wrap-up*, *Delivery*, *and Maintenance* is a chapter on the improvement of project management and definition of success criteria. Students will learn about the purpose and importance of project termination and release, wrap-up and presentation to clients, and post-release maintenance as well as how these phases fit into the software development life cycle.

Chapter 12: The chapter *Software Metrics and Measurements* is a description of the theory and practice of establishing and using metrics of many varieties. Metrics are defined and their importance in measuring software quality are detailed in the sections of this chapter. Many broads types of metrics, including design metrics, object-oriented metrics, and project metrics are discussed in relation to the appropriate phases of development they are used in. Similarly, the benefits and drawbacks that inherently exist are detailed.

Chapter 13: Perhaps the most important chapter, *Hands-On Software Engineering Project* provides a complete guideline for completing an academic software project. Presented in order of phase, this chapter clearly introduces the complete software development life cycle and describes important tools to be used to complete the project.

Courses

Geared towards students with a background in an object-oriented language such as Java or C++, this book is intended for courses in introductory software engineering with a semester-long project. This book can be used for other courses, however, with limited or altered use of the book's parts and chapters.

Project-based course: For a project-based course in software engineering, we recommend using the book nearly as-is, following the chapters in the order provided. This provides a smooth transition between concepts and practical use as well as between phases of the development life-cycle. A semester-long project should accompany course instruction and students should practice models, documentation, and other tools detailed by the book, especially those in Chap. 13.

Introductory course: For a course introducing the concepts of object-oriented programming, we suggest primary attention being paid to the first three chapters of this book. Also, various sections of other chapters can be used as necessary to highlight practical uses of material covered in these first three chapters. For example, parts of Chap. 7 cover object-oriented design concepts that rest on principles of the object-oriented paradigm covered in Chap. 2.

Management course: For a course geared towards project management, focus should be placed primarily on Chaps. 4, 5, 9, 10, 11 and 13 as these covers aspects of overall management such as team work, communication, working with clients, project delivery, etc.

Technical course: For students learning object-orientation and software development beyond and introductory level, this books provides a core set of chapters teaching concepts such as modeling with UML, object-oriented design, implementation, and testing. In this case, Chaps. 2, 3, 6–9 and 12 are worth emphasizing.

Support Materials

To better the learning and teaching experience of those using this software engineering textbook, support materials have been provided to enhance courses in the subject. These include:

- PowerPoint presentations for each chapter of the text.
- Multiple choice tests and answers covering the topics discussed in the book.

For more information and support you may contact the author, Dr. Roger Lee, at lee1ry@cmich.edu.

http://www.springer.com/978-94-6239-005-8

Software Engineering: A Hands-On Approach

Lee, R.Y.

2013, XXIV, 288 p. 70 illus., 3 illus. in color., Hardcover
ISBN: 978-94-6239-005-8
A product of Atlantis Press