1 Chapter

Introduction

Key Takeaway Points

- Software engineering aims to significantly improve software productivity and software quality while reducing software costs and time to market.
- Software engineering consists of three tracks of interacting life cycle activities—software development, software quality assurance, and software project management activities.
- Object-oriented (OO) software engineering is a specialization of software engineering. It views the world and systems as consisting of objects that interact with each other.

Computers are used in all sectors of our society. It is difficult to find a hospital, school, retail shop, bank, factory, or other organizations in the United States that does not rely on computers. Our cell phones, cars, and televisions are also based on computer-powered platforms. The driving force behind the expanding use of computers is the market economy. However, it is software that makes the computers work in the ways we want. Software or computer programs consist of thousands or millions of instructions that direct the computer to perform complex calculations and control the operations of hardware devices. The demand for computer software has increased rapidly in recent years. In 2009, global software production reached $985.70 billion, growing at 6.80% annually from 1999 to 2009. The Bureau of Labor Statistics shows that from 2010 to 2020 the total number of jobs in application development software engineer and system analyst positions is expected to increase from 520,800 to 664,500 (27.6%) and from 544,400 to 664,800 (22.10%), respectively. To be able to perform the work required of an application development software engineer or system analyst, an education in software engineering is highly recommended.

1.1 WHAT IS SOFTWARE ENGINEERING?

Software systems are complex intellectual products. Software development must ensure that the software system satisfies its requirements, the budget is not overrun, and the system is delivered according to schedule. To accomplish these goals, the

the need for an engineering approach to software production. Since then, software engineering has become a discipline and remarkable progress has been made. The efforts that have taken place in the field have led to the following definition.

Definition 1.1 *Software engineering* as a discipline is focused on the research, education, and application of engineering processes and methods to significantly increase software productivity and software quality while reducing software costs and time to market.

⊗ Def'n

That is, the overall objectives of software engineering are *significantly increasing software productivity (P) and quality (Q) while reducing software production ~~and op~~erating costs (C) and time to market (T).* These objectives are abbreviated as PQCT. Research, education, and application of software engineering processes and ~~methods~~ are the means to accomplish these goals. These processes and methods are classified into three sets of activities: development, quality assurance, and project management activities. The development activities transform an initial system concept into an operational system. The quality assurance activities ensure that the development activities are carried out correctly and that the artifacts produced by the activities are correct. These ensure that the desired software system is produced and delivered. Project management activities plan for the project, schedule and allocate resources to the development and quality assurance activities, and ensure that the system is developed and delivered on time and within budget.

1.2 WHY SOFTWARE ENGINEERING?

First, software is expanding into all sectors of our society. Companies rely on software to run and expand their businesses. Software systems are getting larger and more complex. Today, it is common to develop systems that contain millions of lines of source code. For many embedded systems, software cost has increased to 90%–95% of the total system cost from 5%–10% two decades ago. Some embedded systems use application specific integrated circuits (ASIC) and firmware. These are integrated circuits with the software burned into the hardware. They are costly to replace; and hence, the quality of the software is critical. These call for a software engineering approach to system development.

big projects

Second, software engineering supports teamwork, which is needed for large system development. Large software systems require considerable effort to design, implement, and test. A typical software engineer can produce an average 50–100 lines of source code per day. This includes the time required to perform analysis, design, implementation, integration, and testing. Thus, a small system of 10,000 lines of code would require one software engineer to work between 100 and 200 days or 5 to 10 months. A medium-size system of 500,000 lines of source code would require a software engineer to work 5,000 to 10,000 days or 20 to 40 years. It is not acceptable for most businesses to wait this long for their systems. Therefore, real-world software systems must be designed and implemented by a team, or teams

team-based development

size vs.

problems require discipline

40 software engineers to work for one year. When two or more software engineers work together to develop a software system, serious conceptualization, communication, and coordination problems arise.

Conceptualization is the process of observing and classifying real-world phenomena to form a mental model to help understand the application for which the system is built. Conceptualization is a challenge for teamwork because the software engineers may perceive the world differently due to differences in their education, cultural backgrounds, career experiences, assumptions, and other factors. The ancient story about four blind men and an elephant illustrates this problem. The four blind men wanted to know what an elephant looked like. They obtained permission to touch the elephant. One blind man touched one leg of the elephant and said that an elephant was like a tree trunk. The other three touched the elephant's stomach, tail, and ear, respectively. They said that an elephant was like a wall, a rope, and a fan. We as software developers are like the four blind men trying to perceive or understand an application. If the developers perceive the application differently, then how can they design and implement software components to work with each other? Software engineering provides a solution. That is, the modeling techniques presented in this book help the software engineers establish a common understanding of the application domain and the business processes of the application.

ambiguous

When a team of software engineers work together, they need to exchange their understanding and design ideas. However, the natural language is too informal and often leads to misunderstanding. Software engineering provides the Unified Modeling Language (UML) for software engineers to communicate their ideas. Finally, when teams of software engineers work together, how can they collaborate and coordinate their efforts? For example, how do they divide the work and assign the pieces to the teams and team members? How do they integrate the components designed and implemented by different teams and team members? Again, software engineering provides a solution. That is, the agile unified methodology presented in this book lets the software engineers collaborate in a way that everybody understands and follows.

1.3 SOFTWARE LIFE-CYCLE ACTIVITIES

Software engineering focuses on three tracks of activities as Figure 1.1 exhibits. These activities take place simultaneously throughout the software life cycle.

1. *Software development process.* A software development process transforms the initial system concept into the operational system running in the target environment. It identifies the business needs, conducts a feasibility study, and formulates the requirements or capabilities that the system must deliver. It also designs, implements, tests, and deploys the system to the target environment.
2. *Software quality assurance.* Software quality assurance (SQA) ensures that the development activities are performed properly, and the software artifacts produced by the development activities meet the software requirements and desired quality standards.

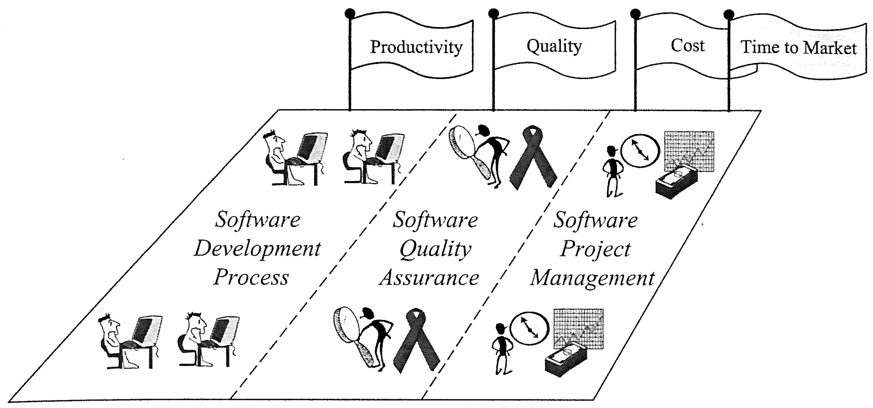

Productivity
Quality
Cost
Time to Market
Software Development Process
Software Quality Assurance
Software Project Management

FIGURE 1.1 Three tracks of life-cycle activities

3. *Software project management.* Software project management oversees the control and administration of the development and SQA activities. Project management activities include effort estimation, project planning and scheduling, risk management, and project administration, among others. These activities ensure that the software system is delivered on time and within budget.

1.3.1 Software Development Process

A software development process is often called a *software process*. The need for a process is similar to custom home construction and many other major undertakings. The activities of custom home construction include acquisition of home buyer requirements, custom home design, build, inspection, and delivery. A software process consists of a series of phases of activities performed to produce the software system. In some cases, the software system is a part of a larger system. For example, telecommunication systems, mail processing systems, industry process control systems, and medical devices use software to process events and control the hardware. These systems are called *embedded systems*. In these cases, the software process is a part of a bigger process called the *system development process* or *system engineering process*. System engineering considers the total system rather than the software system alone.

During the history of software engineering, many software process models are proposed. Among them are the waterfall, prototyping, evolutionary, spiral, unified, and agile processes. Figure 1.2 shows the well-known conventional waterfall process, adapted from a traditional engineering discipline. The process takes into account system engineering activities, that is, system requirements definition, system design, and system requirements allocation. The phases of the waterfall process are described below.

System Requirements Definition, System Design, and Allocation

These are system engineering activities often performed for embedded systems. System requirements definition identifies the capabilities for the total system and

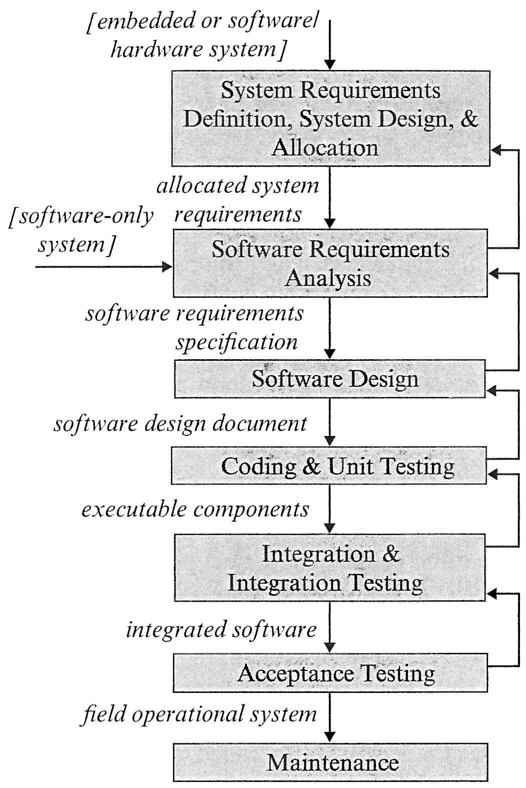

[embedded or software/ hardware system]
System Requirements Definition, System Design, & Allocation
allocated system requirements
[software-only system]
Software Requirements Analysis
software requirements specification
Software Design
software design document
Coding & Unit Testing
executable components
Integration & Integration Testing
integrated software
Acceptance Testing
field operational system
Maintenance

FIGURE 1.2 The waterfall model for software development

(*)

for a radio communication system (RCS). The system is similar to a cellular network except that it has only one high-power base station that serves an area much larger than a cell in a cellular network. The system requirements specify the capabilities for the whole system. The following are four of the many system requirements identified.

(*) Requirements

R1. The RCS shall allow mobile subscribers to initiate calls to other mobile subscribers and land-line telephones.

R2. The RCS shall allow mobile subscribers to answer calls from other subscribers.

R3. The RCS shall provide call accounting to capture and record mobile calls and bill to the subscriber accounts.

R4. The RCS shall allow authorized account administrators to manage subscriber accounts.

System design determines the major subsystems of the total system and specifies the relationships between the subsystems. These are depicted in a system architectural design diagram. Figure 1.3 shows an architectural design for the RCS using a block diagram. It depicts three subsystems: mobile unit, base station, and account manage-

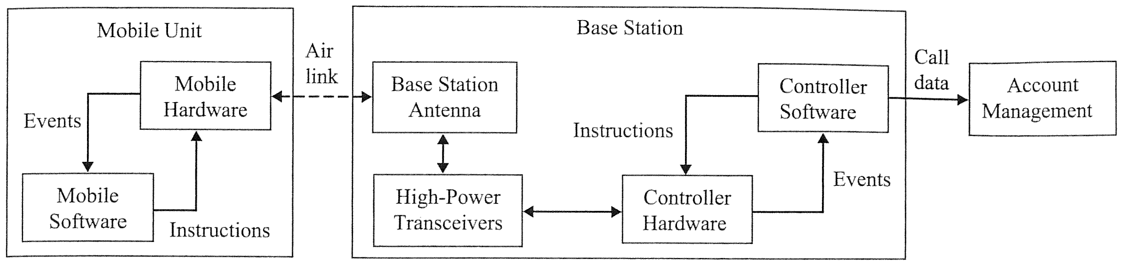


FIGURE 1.3 System design using a block diagram

subsystem. A base station consists of base station antenna, high-power transceiver, controller hardware, and controller software subsystems. The arrow lines in Figure 1.3 show the relationships between the subsystems.

The allocation activity assigns the system requirements to the subsystems. System design and allocation should result in subsystems that are relatively independent and easy to interface. Moreover, the subsystems can be developed by separate engineering teams such as electrical and electronic engineering teams, mechanical engineering teams, and software engineering teams. System allocation may decompose the system requirements into lower-level requirements and assign them to separate subsystems. For example, requirement R1 involves mobile unit and base station as well as hardware and software functions. Sending a call request to the base station is the function of a mobile unit. Intercepting the request is a hardware function. Checking the call request to ensure that it is a subscriber-to-subscriber call is a software function. Therefore, the requirement is decomposed into the following:

functional decomp.

R1.1. Mobile units shall include automatic number identification (ANI) numbers when sending a call request.

R1.2. The base station shall verify the caller and callee using the ANI numbers before setting up a call.

R1.2.1. The software controller shall verify the caller and callee, and instruct the hardware controller to set up a connection when the verification is successful.

R1.2.2. The hardware controller shall instruct the high-power transceivers to establish an air-link connection under the software control.

After decomposition, requirement R1.1 is assigned to the mobile unit. Requirement R1.2.1 is assigned to the base station software controller, and requirement R1.2.2 is assigned to the base station hardware controller. Similarly, requirement R3 may be decomposed and assigned to the appropriate subsystems. Requirement R4 is assigned to the account management subsystem because it is a software-only requirement.

Software Requirements Analysis

Software requirements analysis refines the system requirements allocated to the software system. It also identifies other capabilities for the software system. These and the refined system requirements are specified in a software requirements specaccount administrator can do to manage the accounts. Thus, the requirement is refined as follows:

R4.1. The RCS shall allow an authorized account administrator to create a subscriber account.

R4.2. The RCS shall allow an authorized account administrator to activate a subscriber account.

R4.3. The RCS shall allow an authorized account administrator to deactivate a subscriber account.

R4.4. The RCS shall allow an authorized account administrator to close a subscriber account.

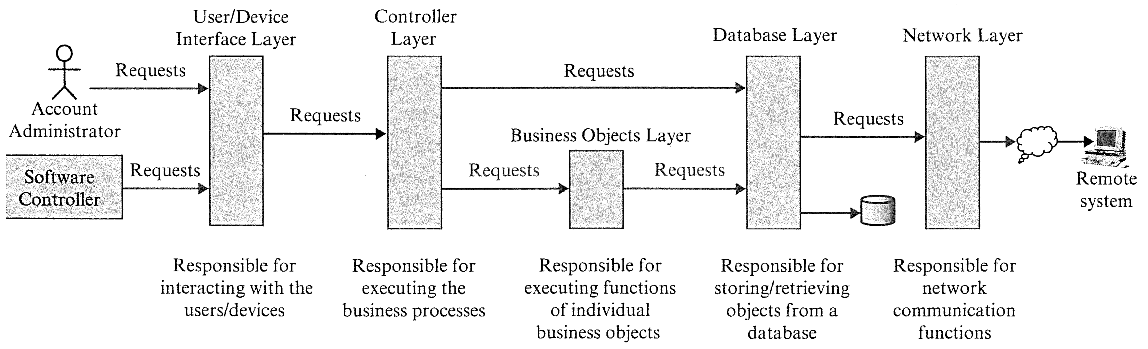
R4.5. The RCS shall allow an authorized account administrator to delete a subscriber account that is already closed.

The account administrator needs to login to perform these operations and logout to prevent others to access the accounts. These are examples of software requirements that are identified during the software requirements analysis phase.

Software Design

Software design determines the software architecture, or the overall structure, of the software system. It specifies the subsystems, their relationships, the subsystems' functions, interfaces, and how the subsystems interact with each other. Design of the user interface is another important activity of software design. That is, it depicts the look and feel of the windows and dialogs, and describes how the system interacts with the users. Software design also specifies the information processing algorithms.

As an example for the architectural design for the RCS, Figure 1.4 shows the N-tier architecture for the account management system. The architecture organizes the classes or objects into separate layers. Each layer has clearly defined responsibilities and requests the services of the next layer. For example, the user/device interface layer provides services to the account administrator and the software controller. It uses the services of the controller layer. The layered architecture simplifies the implementation of the business processes. For instance, when the account administrator wants to create an account, the user interface forwards the request to the create account controller


User/Device Interface Layer
Controller Layer
Database Layer
Network Layer
Requests
Account Administrator
Software Controller
Requests
Requests
Requests
Business Objects Layer
Requests
Requests
Requests
Remote system
Responsible for interacting with the users/devices
Responsible for executing the business processes
Responsible for executing functions of individual business objects
Responsible for storing/retrieving objects from a database
Responsible for network communication functions

in the controller layer. The controller creates the Account object, and saves it to the database through the database layer. This shows that the functions of each layer are relatively easy to implement and use. Another advantage of the N-tier architecture is that change to a layer has little impact on other layers, provided that the layer-to-layer interfaces remain unchanged.

Implementation, Testing, and Maintenance

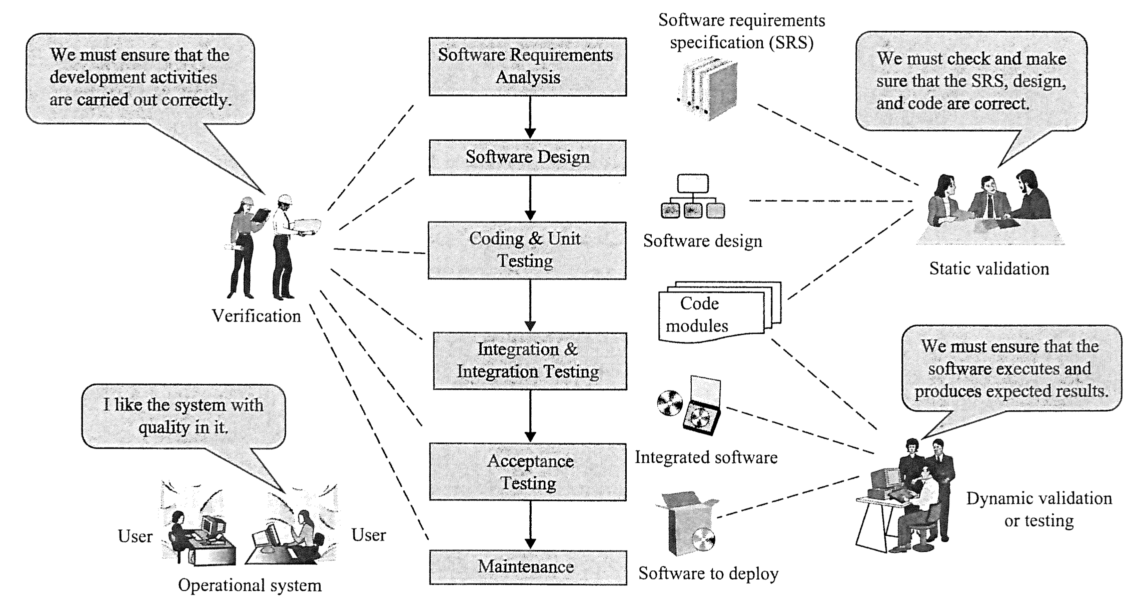
During the implementation and unit testing phase, programs are written to implement the design. The programs are tested, and reviewed by peers to ensure correctness and compliance to coding standards. During the integration phase, the program modules are integrated, and tested to ensure that they work with each other. During acceptance testing, test cases are designed and run to check that the software indeed satisfies the software requirements. The software system is then installed in the target environment and tested by users. The software enters the maintenance phase. During the maintenance phase, corrections, improvements, and enhancements are made continually until the system is replaced.

ver.: right thing.

val.: thing right

1.3.2 Software Quality Assurance

SQA ensures that the software system under development will satisfy the software requirements and desired quality standards. Verification, validation, and testing are the means to accomplish these goals. As shown in Figure 1.5, these activities are life-cycle activities. In particular, verification ensures that the development activities are carried out according to the software processes and methods selected for the software project. It also ensures that the required software artifacts are produced and


We must ensure that the development activities are carried out correctly.
Verification
I like the system with quality in it.
User
User
Operational system
Software Requirements Analysis
Software Design
Coding & Unit Testing
Integration & Integration Testing
Acceptance Testing
Maintenance
Software requirements specification (SRS)
Software design
Code modules
Integrated software
Software to deploy
We must check and make sure that the SRS, design, and code are correct.
Static validation
We must ensure that the software executes and produces expected results.
Dynamic validation or testing

conform to the quality standards. For example, if the RCS project uses the waterfall process, then verification ensures that the phases of the waterfall are carried out correctly. It also checks that the RCS system requirements specification, system design, allocation, software requirements specification, software design, and the other artifacts are produced and meet the required quality standards. Validation checks the correctness of the software artifacts produced by the development activities. For example, validation for the RCS checks the system and software requirements to ensure that they specify the real business needs. It also checks that the system as well as the software design satisfies the requirements and constraints, and the implemented software system solves the intended business problems.

Validation activities are classified into static validation and dynamic validation activities. Static validation checks the correctness of the software artifacts without executing the software. It is applicable to artifacts that are not executable, such as requirements specification and software design documents. This is similar to examining the specifications of a car to determine whether it satisfies the needs of the car buyer. Dynamic validation executes the software to ensure that it works and produces the correct result. Software testing is a dynamic validation technique. It executes the software using test cases and checks that the test result matches the expected result. It is similar to test-driving a car to ensure that it indeed meets the functional and performance requirements.

Unit testing for the account management system generates test cases to check that each of the classes in the interface, controller, business objects, database, and network layers correctly implements the functionality of the class. Integration testing generates test cases to test and ensure that these classes work with each other. Acceptance testing derives test cases from the requirements of the account management system to ensure that the system satisfies the requirements.

1.3.3 Software Project Management

Software project management activities ensure that the software system under development will be delivered on schedule and within the budget constraint. To accomplish these goals, project management performs the following activities, among others:

- *Effort estimation*. Effort estimation derives the human resources and durations required to perform the development and SQA activities. For example, how many developers are needed to perform the analysis, design, implementation, and testing activities, respectively? What is the duration of each of these activities? Answers to these questions are used to plan the project and schedule the activities. Effort estimation derives these estimates from various factors such as the estimated software size and complexity as well as the required delivery date.
- *Project planning and scheduling*. Project planning and scheduling are aimed at producing an overall plan for the project. The project plan will guide the project teams throughout the life-cycle process. However, due to changes in the real world, the plan needs to be updated to reflect changes. The project plan specifies the project goals, the process to reach the goals, the project milestones, the schedule of project activities and deliverables, the project teams and how the

plan also includes a quality assurance plan. The quality assurance plan specifies the quality standards, the SQA activities, and a schedule of such activities. Also included in the project plan is a risk management plan, described below.

- *Risk management.* Many events could jeopardize a project. For example, a management person or a key technical staff leaves the project, or the project is far behind schedule. These are called risk items. Risk management attempts to reduce the impact of such events through risk management planning. The risk management plan identifies the risk items, prioritizes them according to their likelihood and damage to the project, and specifies risk resolution measures to counter the risks when any of them occur.
- *Project administration.* Project administration is an ongoing function of project management. It performs the management activities as specified in the project plan. It is concerned with the continuous monitoring of project progress and executing the actions necessary to adapt the project to the new situation. It also includes daily management of project activities such as coordinating the project teams or team members, scheduling and conducting meetings, and solving day-to-day problems.
- *Software configuration management.* During the development process, numerous software artifacts are produced. These include requirements specification, software design, code, test cases, user's manual, and the like. These compose the software, or part of it, under different stages of the development process. For example, the requirements specification is the software in its nonexecutable form. The design specification is a refinement of the requirements specification. The code is a refinement of the design. These documents depend on each other. For example, software design depends on the requirements. If the requirements are changed, the design has to change. This in turn may require change to the code that implements the design. Therefore, software engineering needs a mechanism to coordinate so that changes are made consistently. This is software configuration management (SCM).

1.4 OBJECT-ORIENTED SOFTWARE ENGINEERING

Object-oriented software engineering (OOSE) is a specialization of software engineering. OOSE views the world and ~~systems~~ as consisting of objects that relate to and interact with each other. OOSE provides the following to support OO software development:

1. *OO modeling and design languages* for the team members to communicate their analysis and design ideas. A modeling and design language defines the notions and notations as well as rules for using the notations. The Unified Modeling Language (UML) [36] is the most widely used OO modeling and design language.
2. *OO software development processes* to guide the development effort. The unified process (UP) is a well-known development process while agile processes have

3. *OO software development methodologies* that detail the steps or how to carry out the activities of a software process.
4. *OO development tools and environments* to support the development processes and methodologies. There are commercial products as well as public domain software. For example, the NetBeans integrated development environment (IDE) is a free, open source software. It comes with a bundle of plugins that support activities of the entire software development life cycle.

1.4.1 Object-Oriented Modeling and Design Languages

The rapid spread of C++ in the 1980s motivated the need for a development methodology to guide OO software development efforts. Three influential OO development methodologies, among many others were proposed and widely used in the software industry. These are Booch Diagram, Object Modeling Technique (OMT), and Use Case Engineering. The industry soon discovered that it was a monumental challenge to integrate systems designed and implemented using different methodologies. The reason is that different methodologies use different modeling concepts and notations. To solve this problem, the Object Management Group (OMG) adopted the Unified Modeling Language (UML) [36] as an OMG standard. UML is a family of diagrams for modeling and designing different aspects of an OO system. UML diagrams are used in the requirements analysis phase to help the development team understand the business of the existing application. They are used in the design phase as part of the design specification. UML diagrams will be presented throughout the rest of this book.

1.4.2 Object-Oriented Development Processes

The sequential nature of the waterfall process implies that changes to the requirements are difficult and costly. This is because any change to the requirements affects the design and implementation; these must be changed as well. Unfortunately, requirements change is a common occurence for many real-world projects due to change in market conditions, advances in technology, and other factors. The long development duration of the waterfall process implies that the system is dated as soon as it is released. This is because the capabilities or requirements of the system were identified long ago. To overcome these problems, several software process models have been proposed. All of these adopt an iterative, rather than a strictly sequential, process of development activities. Examples are the spiral process, the unified process, and agile processes.

dumb customers?

1.4.3 Object-Oriented Development Methodologies

(*) A software process specifies "when to do what," but not "how to do them." That is, it defines the development activities but not how to perform the activities. UML is a modeling language. It lets the software engineers describe their analysis and design ideas using the diagrams. It does not help the software engineers to produce the analysis and design ideas. A software development methodology fills the gap. It specifies the steps and how to perform the steps to carry out the activities of a software pro-

Modeling Techniques (OMT), Use Case Engineering, and other methods. Agile methods include Scrum, Dynamic System Development Method (DSDM), Feature Driven Development (FDD), Crystal Clear, Extreme Programming (XP), Lean Development Method, and others.

1.4.4 Will OO Replace the Conventional Approaches?

The answer is no, for a number of reasons. First, maintaining numerous conventional systems is required. Second, numerous organizations still use the conventional approaches. Third, a conventional methodology may be more appropriate for some projects such as scientific computing. Finally, a system may consist of components developed by conventional and OO approaches. Therefore, OO and conventional approaches will coexist for many years. ⊛

have to

1.5 SOFTWARE ENGINEERING AND COMPUTER SCIENCE

Software engineering and computer science are different. Computer science disciplines such as algorithms and data structures, database systems, artificial intelligence, operating systems, and others emphasize computational efficiency, resource sharing, accuracy, optimization, and performance. These attributes can be measured quantitatively and immediately. Indeed in the last several decades (1950–present), all efforts and resources spent in computer science research are aimed at improving these aspects. Most chapters of a computer science textbook are written about methods, algorithms, and techniques to improve these aspects.

Unlike computer science, software engineering emphasizes software PQCT. For example, obtaining an optimal solution is often the goal of computer science. Software engineering would use a good-enough solution to reduce development time and effort. Efforts and resources spent in software engineering R&D are aimed at improving software PQCT. Most chapters of a software engineering textbook are written about methods and techniques to improve these four aspects. Unfortunately, the impact of a software engineering process or methodology cannot be measured immediately. ⊛ To be meaningful, the impact must be assessed during a long period. For example, researchers take more than one decade to realize that the uncontrolled goto statement is harmful. That is, the uncontrolled use of the goto statement results in poorly structured programs, which are difficult to understand and test.

Computer science focuses only on technical aspects. Software engineering has to consider nontechnical issues. For example, the early stages of the development process focus on identifying business needs and formulating requirements and constraints. These activities require knowledge, experience, and skill in communication, customer relations, and business analysis. Software engineering also requires knowledge and experience in project management. User interface design has to consider human factors such as user preference and how users would use the system. In addition, software development must consider political issues because the system may affect many people in one way or another.

Recognizing the differences between software engineering and computer science and principles. Consider, for example, the design of a software system that needs to access a database. Computer science disciplines might emphasize efficient data storage and retrieval and favor a design in which the business objects access the database directly. Such a design is said to create a tight coupling between the business objects and the database. Software engineering would not consider this as a good design unless performance is a serious concern. If the database, or the database management system (DBMS) is changed, then the business objects have to change. This may be difficult and costly. If the database is at a remote location, then the business objects must know how to communicate with the remote database. These result in complex business objects, which are difficult to test and maintain.

Despite the differences, software engineering and computer science are closely related. Computer science to software engineering is like physics to electrical and electronics engineering, or chemistry to chemical engineering. That is, computer science is a theoretical and technological foundation of software engineering. Software engineering is the application of computer science. However, software engineering has its own research topics. These include research in software processes and methodologies, software verification, validation, and testing techniques, among many others. Software engineering research is aimed at significantly improving software PQCT and the trade-off between them.

Software engineering is a broad area. A software engineer should know the different areas of computer science including database systems, operating systems, data structures and algorithms, programming languages, and compiler construction, to mention a few. Embedded systems development requires the software engineer to have a basic understanding of electronic circuits and how to interface with hardware devices. Finally, it takes time for a software engineer to gain domain knowledge and design experience to become a good software architect. These challenges and the ability to design and implement large complex systems to meet practical needs make software engineering an exciting area. The ever-expanding computer application creates great opportunities for the software engineer and software engineering researcher.

SUMMARY

Software engineering is defined as a discipline that investigates and applies engineering processes and methodologies to improve software PQCT. The need for software engineering is discussed and software life-cycle activities are described. OO software engineering is a specialization of software engineering. It views the world and systems as consisting of objects relating to and interacting with each other. The chapter ends with a discussion of the differences and relationships between computer science and software engineering. That is, computer science is a foundation of software engineering. While computer science is mainly concerned with optimization and efficiency, software engineering is concerned with software PQCT. Knowing these helps a developer understand software engineering and the rationale behind the processes, methodologies, modeling languages, design patterns, and many others. All these are designed to improve software PQCT.

FURTHER READING

Many books provide comprehensive coverages of the discipline of software engineering. Some examples are [123, 125, 139]. OOSE books include [40, 133], among many others. An excellent introduction to OO analysis and design is presented in [104]. Numerous books that describe UML and UP have been published, notably [36, 63, 91]. The Object Management Group (OMG) website (http://www.omg.org) publishes UML specifications and related articles. Many books on agile development are available. These include [21, 50, 52, 119, 124, 140]. The agile alliance website (http://www.agilealliance.org/) is an excellent resource. It publishes numerous good articles and discussions on the topic.

CHAPTER REVIEW QUESTIONS

1. What is software engineering. Why is it needed?
2. What is a software development process?
3. What is software quality assurance?
4. What is software project management?
5. What is software configuration management?
6. What are the differences between object-oriented software engineering and conventional software engineering?
7. What are the differences and relationships between software engineering and computer science? Can we have one without the other?

EXERCISES

1.1 Search the literature and find four other definitions of software engineering in addition to the one given in this chapter. Discuss the similarities and differences between these definitions.

1.2 Describe in a brief article the functions of software development process, software quality assurance, software project management, and software configuration management. Discuss how these work together during the software development life cycle. Discuss how they improve software PQCT.

1.3 Should optimization be a focus of software engineering? Briefly explain, and justify your answer with a practical example.

1.4 Identify three computer science courses of your choice. Show the usefulness of these courses in the software life-cycle activities.

1.5 There are interdependencies between software productivity, quality, cost, and time to market. For example, more time and effort spent in coding could increase productivity. This may result in less time and effort in software quality assurance because the total time and effort are constants. Poor quality could cost productivity due to rework. Identify three pairs of such interdependencies of your choice. Discuss their short-term and long-term impacts on the software development organization. How should software engineering solve the "dilemmas" induced by the interdependencies?

1.6 What are the differences and relationships between OO software engineering and conventional software engineering? Discuss whether object-oriented software engineering will replace conventional software engineering.