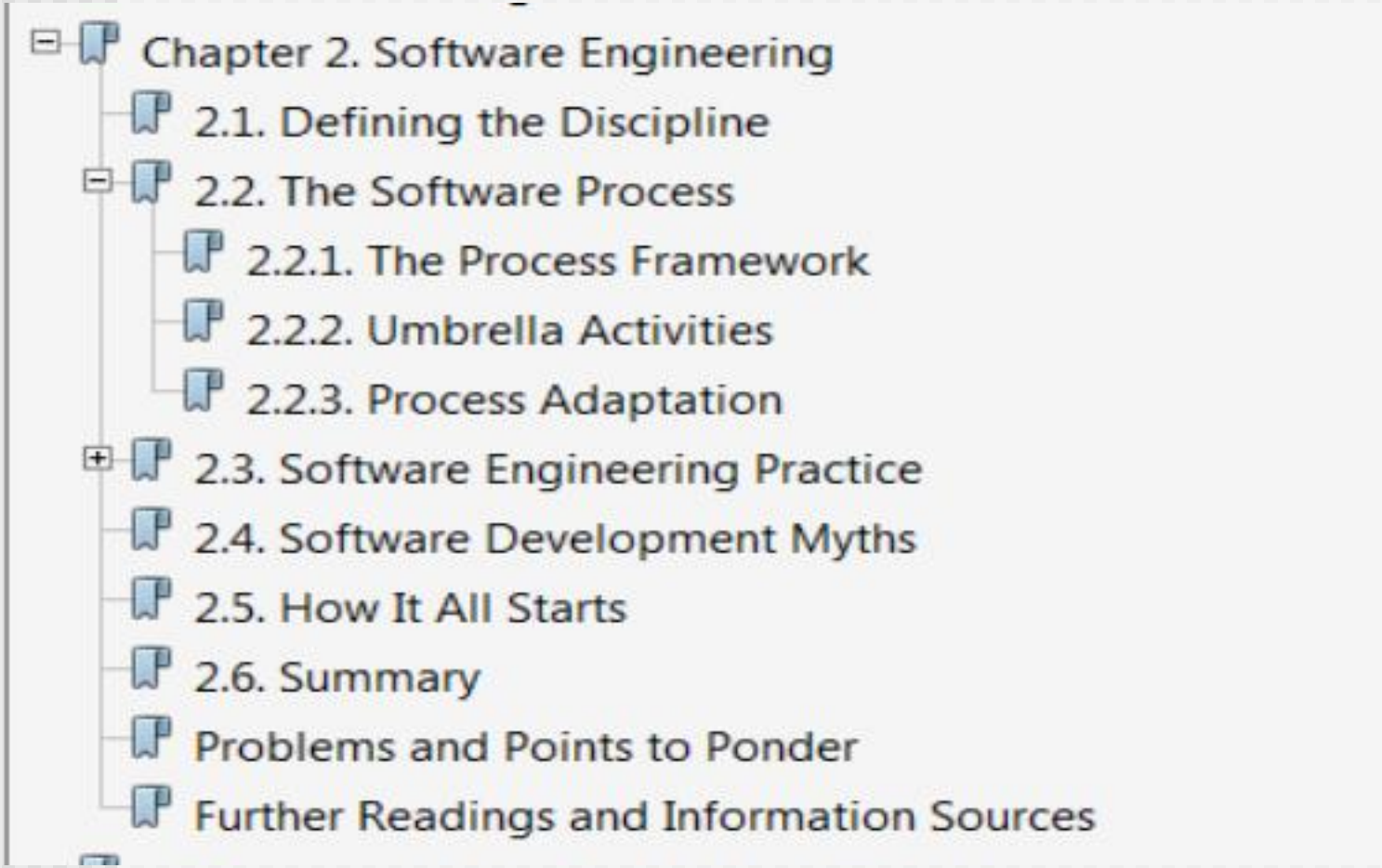
Software Engineering: A Practitioner's Approach, 9/e . Slides copyright 2022 by Du qingfeng,Tongji University

Chapter 2 SOFTWARE ENGINEERING



Chapter 2 SOFTWARE ENGINEERING

2.1 DEFINING THE DISCIPLINE（definition for software engineering）

The IEEE has developed the following definition for software engineering:*Software Engineering : (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).*

And yet, a "systematic, disciplined, and quantifiable" approach applied by one software team may be burden some to another. We need discipline, but we also need adaptability and agility.

Software engineering is a layered technology. Referring to Figure 2.1:

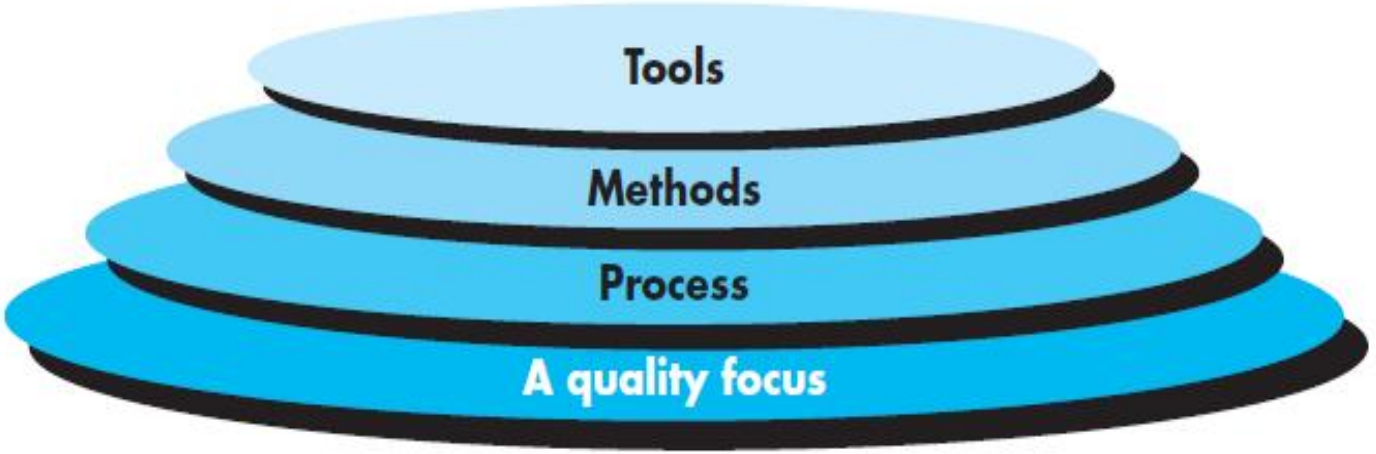
- **The bedrock that supports software engineering is a quality focus.**
- **The foundation for software engineering is the process layer.**

Chapter 2 SOFTWARE ENGINEERING

- **Software engineering methods provide the technical how-to's for building software.**
- **Software engineering tools provide automated or semi-automated support for the process and the methods.When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering(CASE) , is established.**

FIGURE 2.1

Software engineering layers



Chapter 2 SOFTWARE ENGINEERING

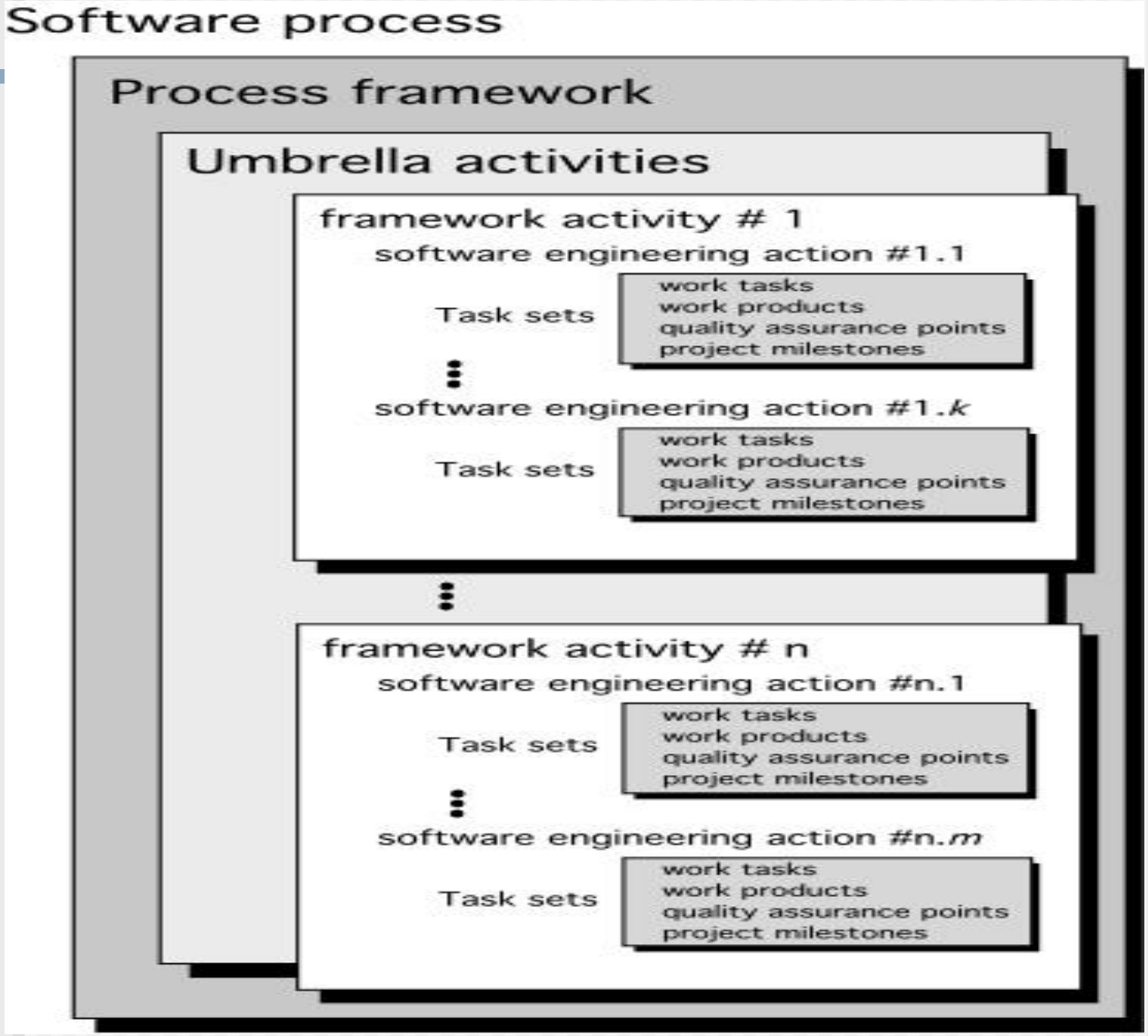
2.2 THE SOFTWARE PROCESS

A process is a collection of *activities, actions*, and *tasks* that are performed when some work product is to be created. An activity strives to achieve a broad objective (e.g., architectural design). An action (e.g., interface design in architectural design) encompasses a set of tasks that produce a major work product (e.g., interface specification). A task focuses on a small, but well-defined objective (e.g., conducting interface design) that produces a tangible outcome.

2.2.1 The Process Framework

A process framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects,

Chapter 2 SOFTWARE ENGINEERING



Chapter 2 SOFTWARE ENGINEERING

regardless of their size or complexity. In addition, the process framework encompasses a set of umbrella activities that are applicable across the entire software process. A generic process framework for software engineering encompasses five activities:

Communication.

Planning（umbrella activities）.

Modeling.

Construction.

Deployment.

These five generic framework activities can be used for different project or system development. The details of the software process will be quite different in each case, but the framework activities remain the same.

Chapter 2 SOFTWARE ENGINEERING

For many software projects, framework activities are applied iteratively as a project progresses. That is, communication, planning, modeling, construction, and deployment are applied repeatedly through a number of project iterations. Each iteration produces a software increment that provides stakeholders with a subset of overall software features and functionality. As each increment is produced, the software becomes more and more complete.

2.2.2 Umbrella Activities

Software engineering process framework activities are complemented by a number of umbrella activities. In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:

Chapter 2 SOFTWARE ENGINEERING

- **Software project planning,tracking and control.**
- **Risk management.**
- **Software quality assurance—defines and conducts the activities required to ensure software quality.**
- **Technical reviews.**
- **Measurement —defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.**
- **Software configuration management.**
- **Reusability management —defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.**

Chapter 2 SOFTWARE ENGINEERING

- **Work product preparation and production —encompass the activities required to create work products such as models, documents, logs, forms, and lists.**

2.2.3 Process Adaptation

Previously in this section, we noted that the software engineering process is not a rigid prescription that must be followed dogmatically by a software team. Rather, it should be agile and adaptable (to the problem, to the project, to the team, and to the organizational culture). Therefore, a process adopted for one project might be significantly different than a process adopted for another project. Among the differences are:

- **Overall flow of activities, actions, and tasks and the interdependencies among them.**

Chapter 2 SOFTWARE ENGINEERING

- **Degree to which actions and tasks are defined within each framework activity.**
- **Degree to which work products are identified and required.**
- **Manner in which quality assurance activities are applied.**
- **Manner in which project tracking and control activities are applied.**
- **Overall degree of detail and rigor with which the process is described.**
- **Degree to which the customer and other stakeholders are involved with the project.**
- **Level of autonomy given to the software team.**
- **Degree to which team organization and roles are prescribed.**

Chapter 2 SOFTWARE ENGINEERING

2.3 SOFTWARE ENGINEERING PRACTICE

In Section 2.2, we introduced a generic software process model composed of a set of activities that establish a framework for software engineering practice. Generic framework activities—communication, planning, modeling, construction, and deployment —and umbrella activities establish a skeleton architecture for software engineering work. But how does the practice of software engineering fit in? In the sections that follow, you'll gain a basic understanding of the generic concepts and principles that apply to framework activities.

2.3.1 The Essence of Practice

The essence of software engineering practice:

1. Understand the problem (communication,planning,analysis modeling).

Chapter 2 SOFTWARE ENGINEERING

2. Plan a solution (design modeling).

3. Carry out the plan (code generation).

4. Examine the result for accuracy (testing and quality assurance).

Understand the problem. Unfortunately, understanding isn't always that easy. It's worth spending a little time answering a few simple questions:

- **Who has a stake in the solution to the problem? That is, who are the stakeholders?**
- **What are the unknowns? What data, functions, and features are required to properly solve the problem?**
- **Can the problem be compartmentalized? Is it possible to represent smaller problems that may be easier to understand?**
- **Can the problem be represented graphically? Can an analysis model be created?**

Chapter 2 SOFTWARE ENGINEERING

Plan the solution. Now you understand the problem (or so you think), and you can't wait to begin coding. Before you do, slow down just a bit and do a little design:

- **Have you seen similar problems before? Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?**
- **Has a similar problem been solved? If so, are elements of the solution reusable?**
- **Can subproblems be defined? If so, are solutions readily apparent for the subproblems?**
- **Can you represent a solution in a manner that leads to effective implementation?Can a design model be created?**

Chapter 2 SOFTWARE ENGINEERING

Carry out the plan. The design you've created serves as a road map for the system. It's possible that you'll discover an even better route as you go, but the "plan" will allow you to proceed without getting lost.

- **Does the solution conform to the plan? Is source code traceable to the design model?**
- **Is each component part of the solution correct? Has the design and code been reviewed, or better, have correctness proofs been applied to the algorithm?**

Examine the result. You can't be sure that your solution is perfect, but you can be sure that you've designed a sufficient number of tests to uncover as many errors as possible.

- **Is it possible to test each component part of the solution? Has a reasonable testing strategy been implemented?**

Chapter 2 SOFTWARE ENGINEERING

- **Does the solution produce results that conform to the data, functions, and features that are required? Has the software been validated against all stakeholder requirements?**

It shouldn't surprise you that much of this approach is common sense. In fact, it's reasonable to state that a commonsense approach to software engineering will never lead you astray.

2.3.2 General Principles

David Hooker has proposed seven principles that focus on software engineering practice as a whole:

- **The First Principle: The Reason It All Exists**
- **The Second Principle: KISS (Keep It Simple, Stupid!)**
- **The Third Principle: Maintain the Vision**
- **The Fourth Principle: What You Produce, Others Will Consume**

Chapter 2 SOFTWARE ENGINEERING

- **The Fifth Principle: Be Open to the Future**
- **The Sixth Principle: Plan Ahead for Reuse**
- **The Seventh Principle: Think!**

If every software engineer and every software team simply followed Hooker's seven principles, many of the difficulties we experience in building complex computer-based systems would be eliminated.

2.4 SOFTWARE DEVELOPMENT MYTHS

Software development myths can be traced to the earliest days of computing. Today, most knowledgeable software engineering professionals recognize myths for what they are—misleading attitudes that have caused serious problems for managers and practitioners alike. However, old attitudes and habits are difficult to modify, and remnants of software myths remain.

Chapter 2 SOFTWARE ENGINEERING

Management myths. Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality.

Myth: We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable?

Myth: If we get behind schedule, we can add more programmers and catch up.

Chapter 2 SOFTWARE ENGINEERING

Reality: Software development is not a mechanistic process like manufacturing. In the words of Brooks: "adding people to a late software project makes it later."However, as new people are added, people who were working must spend time educating the newcomers.....

Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer myths.

Myth: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.

Chapter 2 SOFTWARE ENGINEERING

Reality: Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster. Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

Myth: Software requirements continually change, but change can be easily accommodated because software is flexible.

Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small. However, as time passes, the cost impact grows rapidly.

Practitioner's myths. Myths that are still believed by software

Chapter 2 SOFTWARE ENGINEERING

practitioners have been fostered by over 60 years of programming culture. During the early days, programming was viewed as an art form. Old ways and attitudes die hard.

Myth: Once we write the program and get it to work, our job is done.

Reality: Someone once said that "the sooner you begin 'writing code,' the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth: Until I get the program "running" I have no way of assessing its quality.

Reality: One of the most effective software quality assurance

Chapter 2 SOFTWARE ENGINEERING

mechanisms can be applied from the inception of a project— the technical review. Software reviews are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

Myth: The only deliverable work product for a successful project is the working program.

Reality: A working program is only one part of a software configuration that includes many elements. A variety of work products (e.g., models, documents, plans) provide a foundation for successful engineering and, more important, guidance for software support.

Myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

Chapter 2 SOFTWARE ENGINEERING

Reality: Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Exercise: Understand Figure 2.1 and find some process tools indcluding opensource tools