# Software Engineering

HUANG Jie

School of Software Engineering

Tongji University, 2022

# Chapter 3
# Generic Process Model

In chapter 2, we have learned

- ✓ Why Software has Process Framework?
- ✓ Who is involved in a Software Engineering Project?
- ✓ What are Software Engineering Activities & Elements?
- ✓ General Principles of Software Engineering Practice.
- ✓ Examples of software development myths.

# Lesson 3
# Generic Process Model

In this chapter, we will discuss

- ✓ The meaning of Software Process.
- ✓ The generic concept of software engineering process models.
- ✓ The technical issues in development & support of a system.
- ✓ The nontechnical issues of developing and supporting a system.
- ✓ The coordination efforts needed for process.
- ✓ The traditional process models.

# Software & Software Engineering

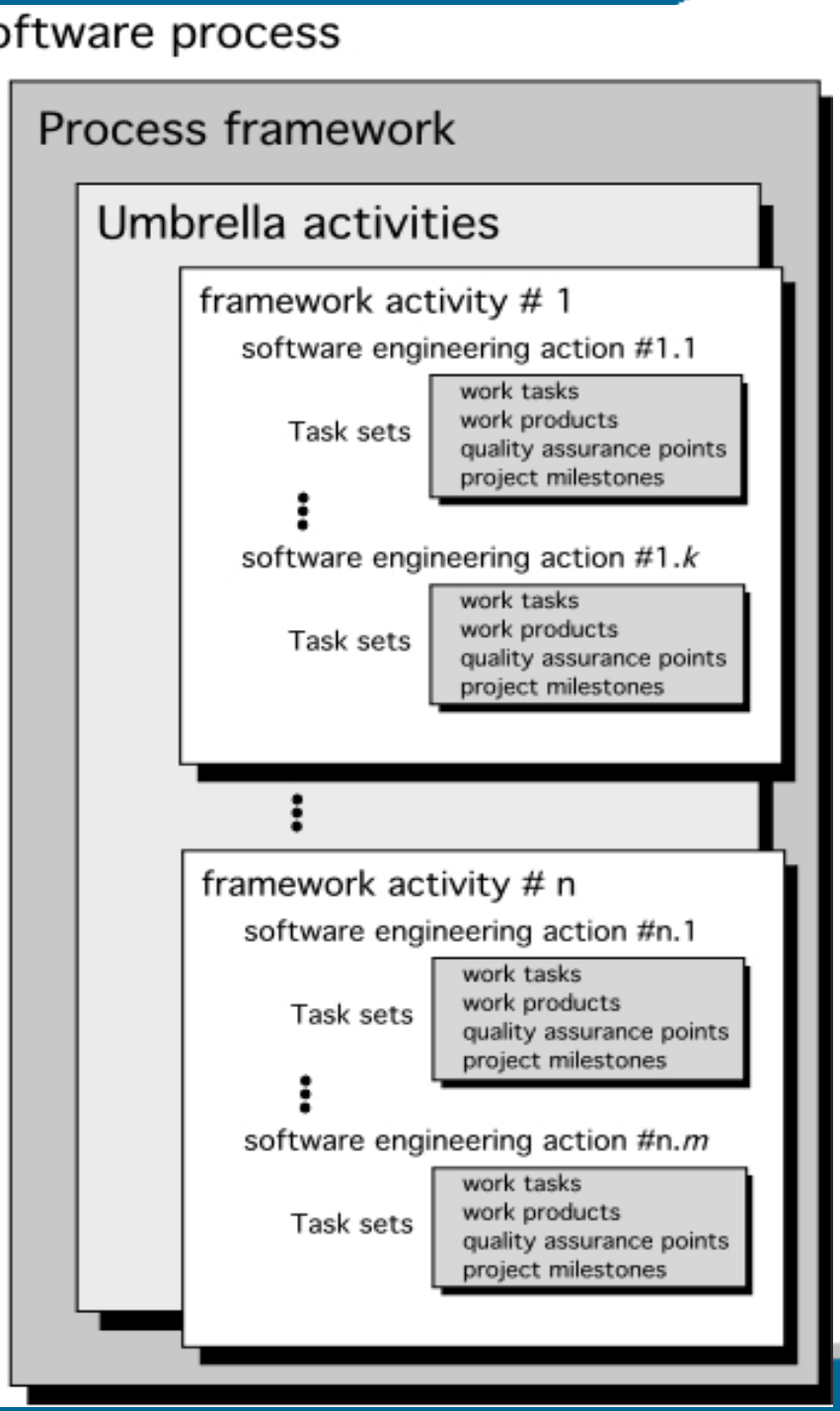
- Howard Baetjer Jr. 1998
  - Because software, like all capital, is embodied knowledge, and because that knowledge is initially dispersed(分散的), tacit(默示的), latent(隐藏的), and incomplete in large measure, software development is a social learning process.
  - The process is a dialogue in which the knowledge that must become the software is brought together and embodied in the software. The process provides interaction between users and designers, between users and evolving tools, and between designers and evolving tools [technology].
  - It is an iterative process in which the evolving tool itself serves as the medium for communication, with each new round of the dialogue eliciting more useful knowledge from the people involved.

# The meaning of Software Process

- Within the context of this book, we define a software process as a framework for the activities, actions, and tasks that are required to build high-quality software.
- Is process synonymous with software engineering"?
  - The answer is Yes and No.
  - A software process defines the approach that is taken as software is engineered.
  - But software engineering also encompasses technologies that populate the process - technical methods and automated tools.
- More important, software engineering is performed by creative, knowledgeable people who should adopt a mature software process so that it is appropriate for the products that they build and demands of their marketplace.

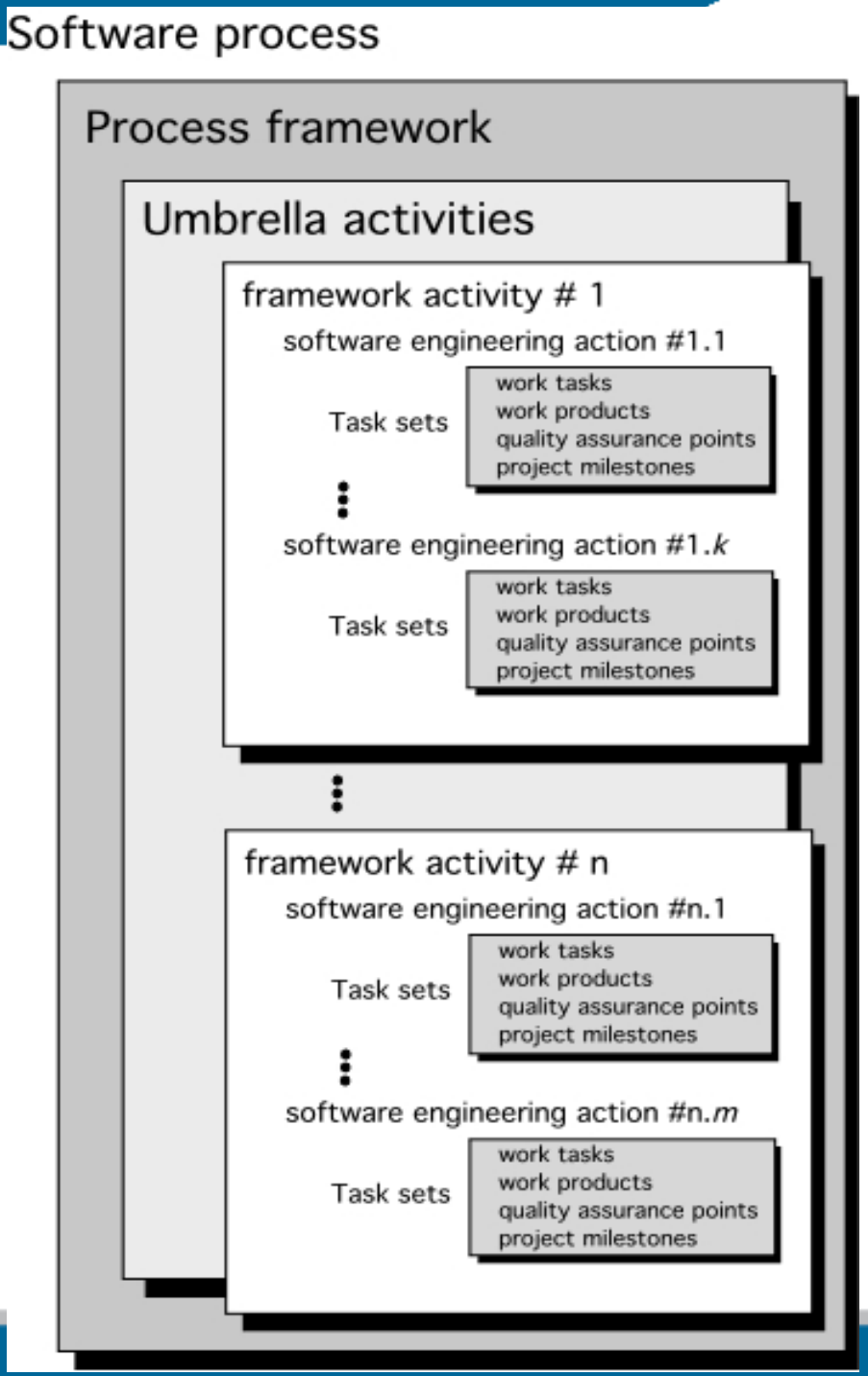
# A Generic Process Model

- Framework activities
  - Communication.
  - Planning.
  - Modeling.
    - Analysis of requirements.
    - Design.
  - Construction.
    - Code generation.
    - Testing.
  - Deployment.



# A Generic Process Model

- Umbrella activities
  - Software project tracking and control.
  - Risk management.
  - Software quality assurance.
  - Technical reviews.
  - Measurement.
  - Software configuration management.
  - Reusability management.
  - Work product preparation and production.



# Characteristics of Building a Software System

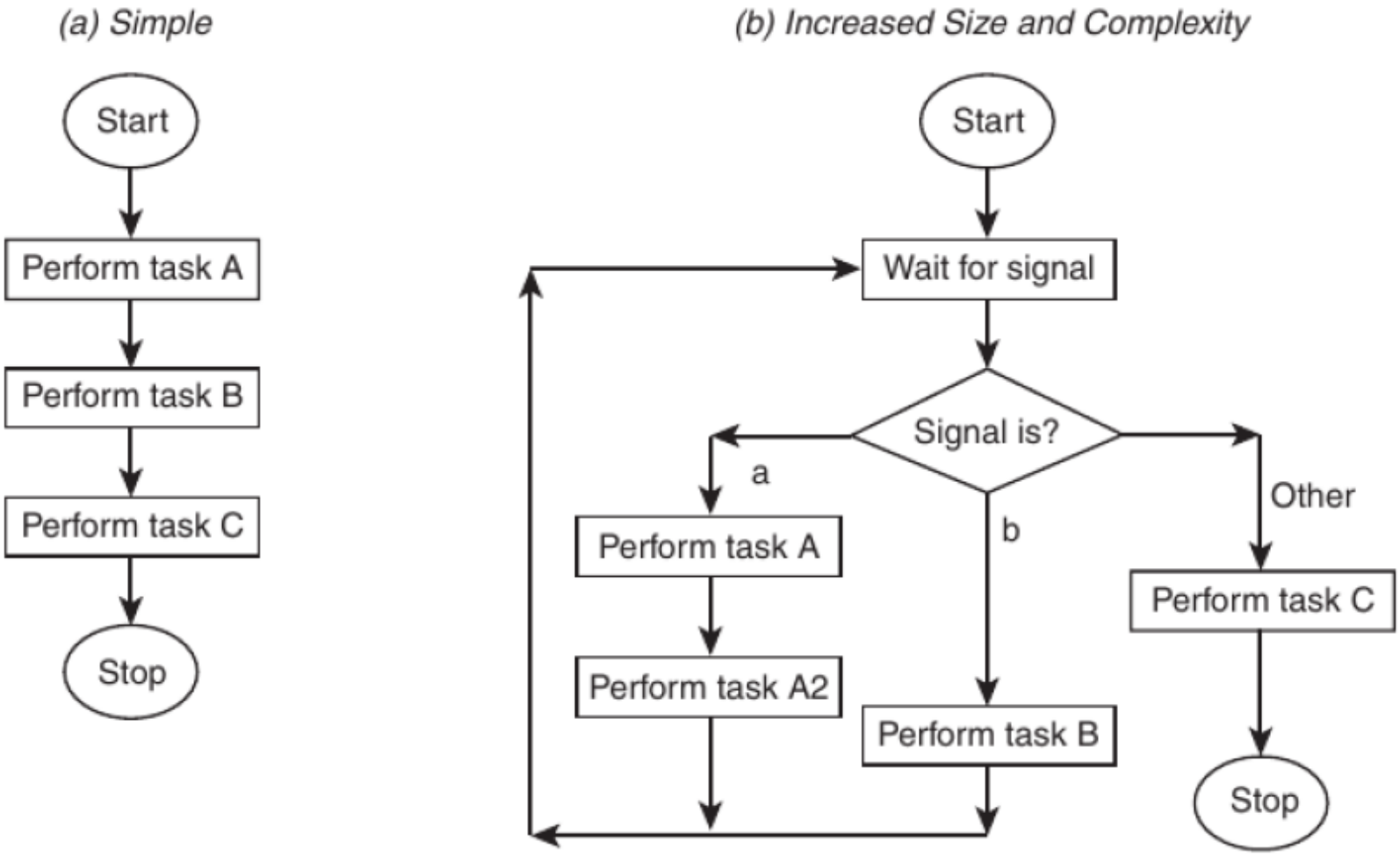
- Size & Complexity
  - As software project becomes large, the development of systems is also becoming more complex. Software engineers are asked to solve both simple and complex problems, deal with the distinct differences between them. The complex problems come in multiple levels of breadth and depth.
  - The breadth issue addresses the sheer numbers involving the following
    - Major Functions.
    - Features within each functional area.
    - Interfaces to other systems.
    - Types of Data and Data Structures.

# Characteristics of Building a Software System

- Size & Complexity
  - The depth issue addresses the linkage and the relationships among items.
    - The linkages may either be through the sharing of data or through the transfer of control, or both.
    - The relationships may be hierarchical, sequential, loop, recursive, or some other form.
  - Example
    - Observe the size in terms of breadth.
    - Observe the complexity in terms of depth and number of interactions.

    See the following chart in next page.

# Characteristics of Building a Software System



# Characteristics of Building a Software System

- Analysis
  - The (a)**simple** case has 3 major segments:
    - Start process.
    - Perform three normal tasks.
    - Stop process.
  - The (b)**right case** has increased tasks, from 3 to 5.
    - Wait for signal task and perform task A and A2 in sequential.
    - A new decision task, represented by the diamond-shaped figure.
    - The decision task has greatly increased the number of paths or choices, and thus it causes the increase of complexity.
    - The complexity is further exacerbated(加剧) by introducing a loop relationship with the decision task. There are many more interactions involved in a loop, more complex than sequential.

# Technical considerations of development and maintenance

- Problem and Design Decomposition
  - When we move from a simple to a complex situation of building software systems, there are some technical issues that we must consider.
  - The basic issue is how to handle all the pieces, parts, and relationships. A common solution is the divide & conquer method. This method has its roots in the modularization concept first presented by Parnas (1972).
  - **Q**uestion

    *How do we divide a large complex problem?*

# Technical considerations of development and maintenance

- Problem and Design Decomposition
  - **Q**uestion: How do we divide a large complex problem?
    - First, simplify the large, complex problem by addressing the problem in smaller segments.
    - Then, decide whether we should design and decompose the solution along the dividing lines of the problem segments.
    - If the problem description or requirement, is segmented by function and feature, should we architect and design the solution along the same function and feature segments?
    - Alternatively, should we choose another decomposition method for the solution, perhaps along the line of "objects"?

# Technical considerations of development and maintenance

- Problem and Design Decomposition
  - **Q**uestion: How do we divide a large complex problem?
  - The key to attacking large and complex problems is to consider some form of simplification through the following types of activities.
    - Decomposition.
    - Modularization.
    - Separation.
    - Incremental iterations.

# Technical considerations of development and maintenance

- Technology and Tool considerations
  - Aside from the important issue of decomposing a problem and its solution, there are problems related to technology and toll considerations that will also need to be addressed.
  - A large complex system requires a Team to develop the software solution. All members in this team may know several languages, each member has different experience. A common development language and development environment needs to be picked.
  - Beyond the programming language and the development tools, there are further considerations of other technical choices related to the following:
    - Database, Network, Middleware, Other technical components such as code version control.

# Technical considerations of development and maintenance

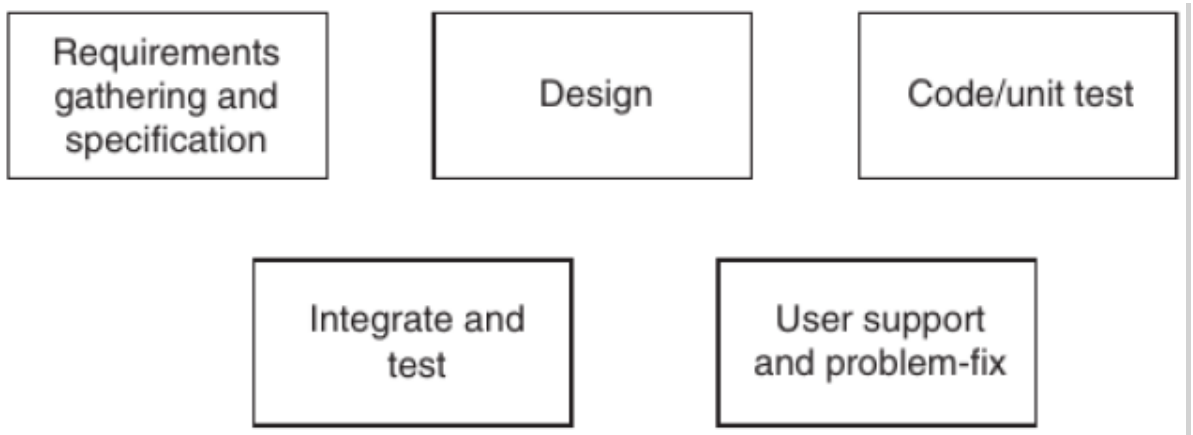
- Process and Methodology
  - Definition: software development and support process
    - The set of tasks, the sequence and flow of these tasks, the inputs to and the outputs from the tasks, and the preconditions and post-conditions for each of the tasks involved in the production of a software.
  - Example
    - The syntax for the expression of a design, need to be agreed upon by all the developers so that they can all review, understand, and produce a consistent and cohesive design.
    - Each method used for a specific task along with the entire development process must be agreed to by the group of people involved in the project.
    - Software development & support processes were invented to coordinate and manage complex projects involving many people.

# Technical considerations of development and maintenance

- Process and Methodology

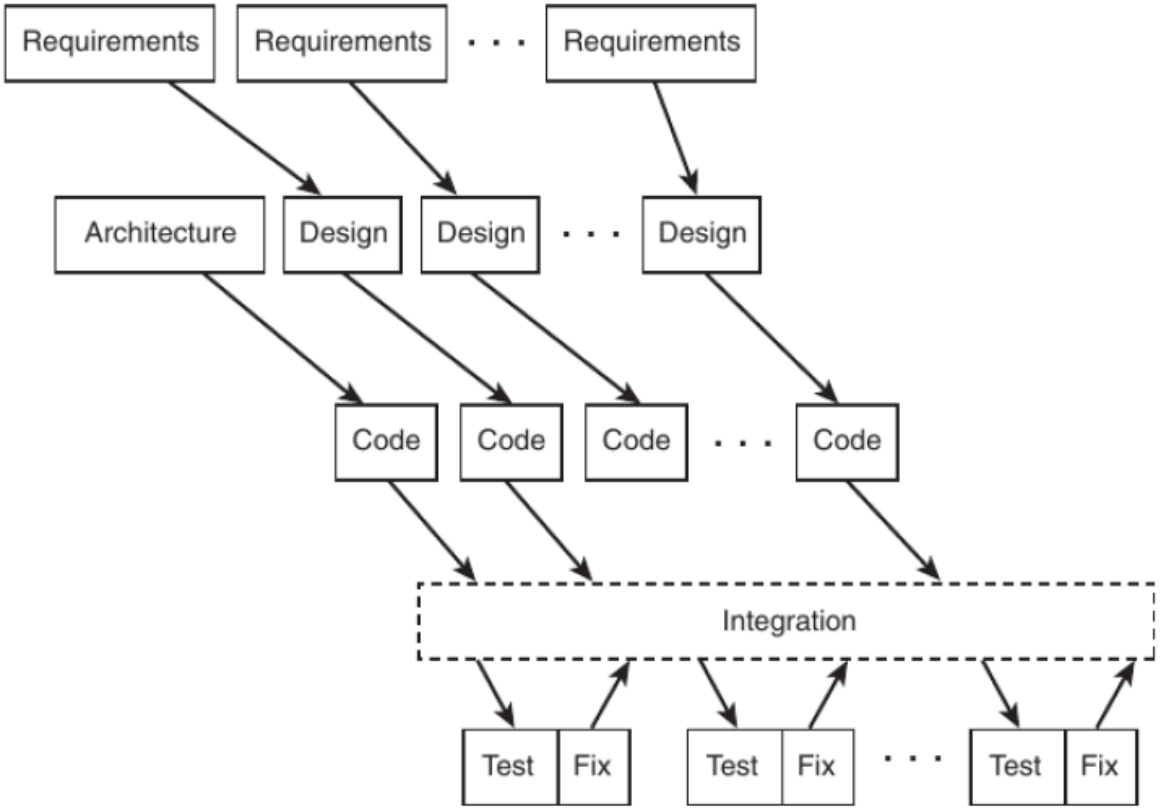
  **E**xample

  - The following tasks are independently. When several individuals are involved in this project, there has to be a clear understanding of the sequence, overlap, and starting conditions.
  - Team members should know the relationships between these tasks.

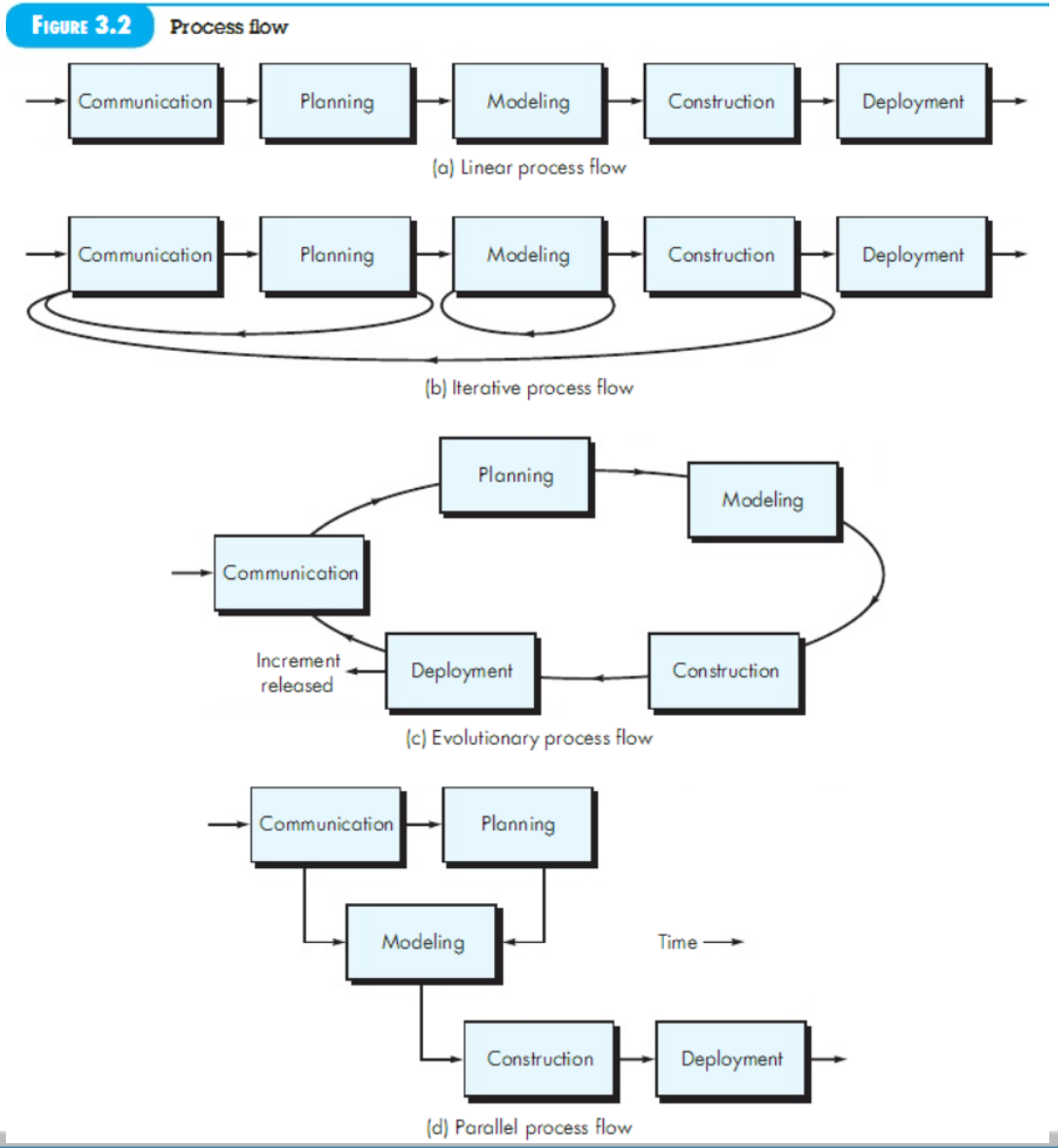


# Technical considerations of development and maintenance

- Example of Process and Methodology



# Types of process flow



# Nontechnical considerations of development and maintenance

- Effort Estimation and Schedule
  - Estimating the total effort and coming up with a reliable project schedule under the difficult condition is one of the main reasons behind so many software project failures.
  - The inaccurate effort estimates and schedules for large, complex systems are often extremely optimistic and aggressive. This places unrealistic expectations on both the customers and the suppliers of these systems.
- Example

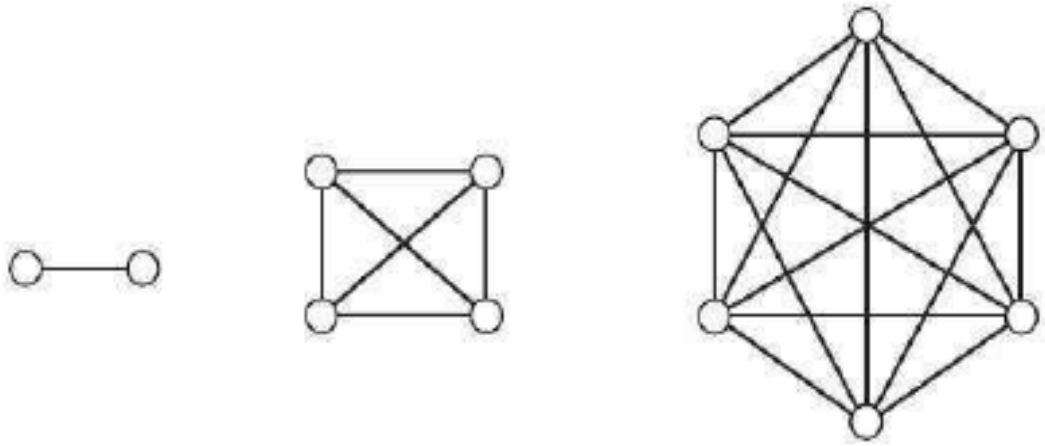
  《The Mythical Man-Month Essay on Software Engineering》,

  Chapter 2 The Mythical Man-Month

# Nontechnical considerations of development and maintenance

- Assignments and Communications

  **E**xample




Maximizing communication paths.

# Coordination Efforts

- A critical concern for large, complex system is the upward scaling of the needed process, of the design structure and contents of the product, and of the required personnel.
- In the opposite direction is the concern of downward scaling of the same parameters for simpler systems.
- Process
  - There is no single process that fits all occasions. Some are more appropriate for large, complex systems that require extensive coordination, and others are much simpler and are appropriate for small and quick software projects.

# Defining a framework activity

- Different projects demand different task sets. The software team chooses the task set based on problem and project characteristics.
- **E**xample
  - Communication activity in small project.
  - What are the contents of this activity?
    - Step 1 ?
    - Step 2 ?
    - Step 3 ?
    - Step 4 ?

    From Textbook《Software Engineering》, P18～P19.

# Identifying a Task Set

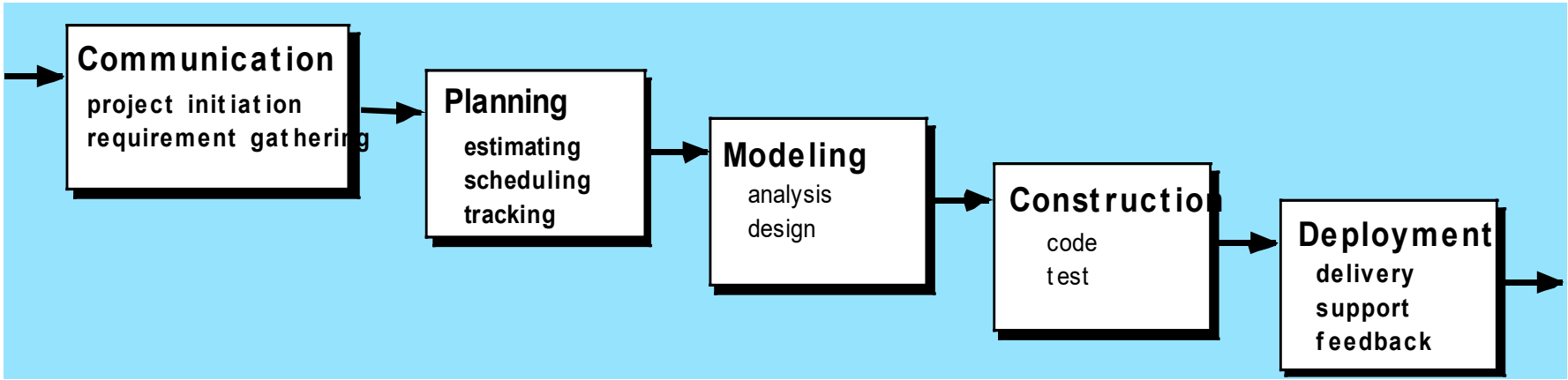
- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
  - A list of the task to be accomplished.
  - A list of the work products to be produced.
  - A list of the quality assurance filters to be applied.
- Example
  - Identifying a Task Set
  - Compare the difference between small and large software projects.

From Textbook《Software Engineering》, P19.

# Prescriptive Models (规范模型)

- Prescriptive process models advocate an orderly & structural approach to software engineering.
- That leads to a few Questions
  - If prescriptive process models strive for structure and order, are they inappropriate for a software world that thrives on change?
  - Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, do we make it impossible to achieve coordination and coherence in software work?
- The Reason, why we use these traditional process models
  - A set of process elements being prescribed.
  - Each model defines a process flow (work flow), the manner in which the process elements are interrelated to one another.

# The Waterfall Model



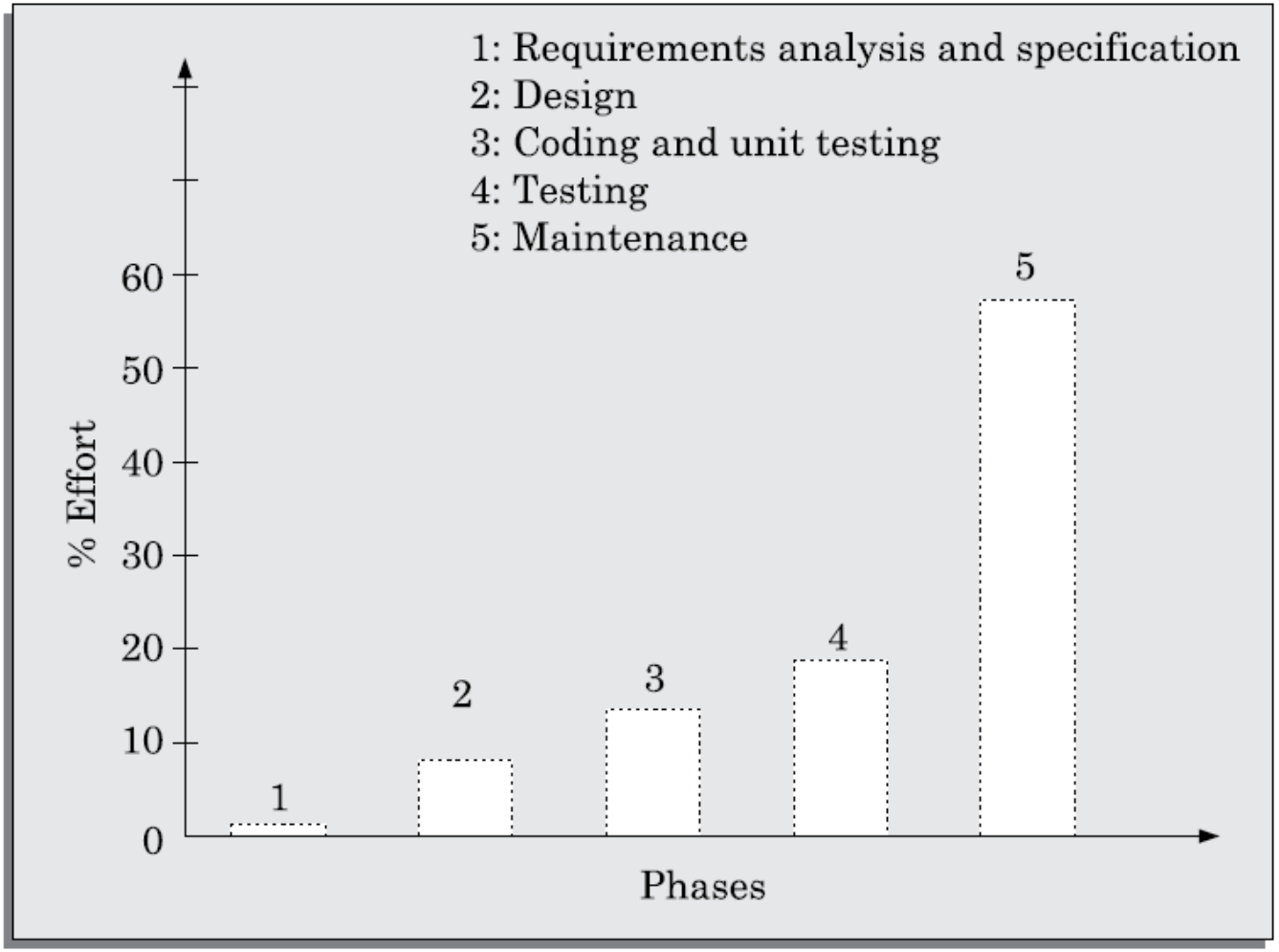
# The Waterfall Model

- The waterfall model, the oldest commonly used software life cycle, is a linear approach to software development that was first suggested by Royce in 1970.
- This development and maintenance paradigm, also known as the classic life cycle, lays out a set of development phases which are to be completed in order.
- A new phase begins only when the previous phase has been fully completed, which includes the finalization of all documentation from that phase, as well the approval of completion by the software quality assurance (SQA) group.

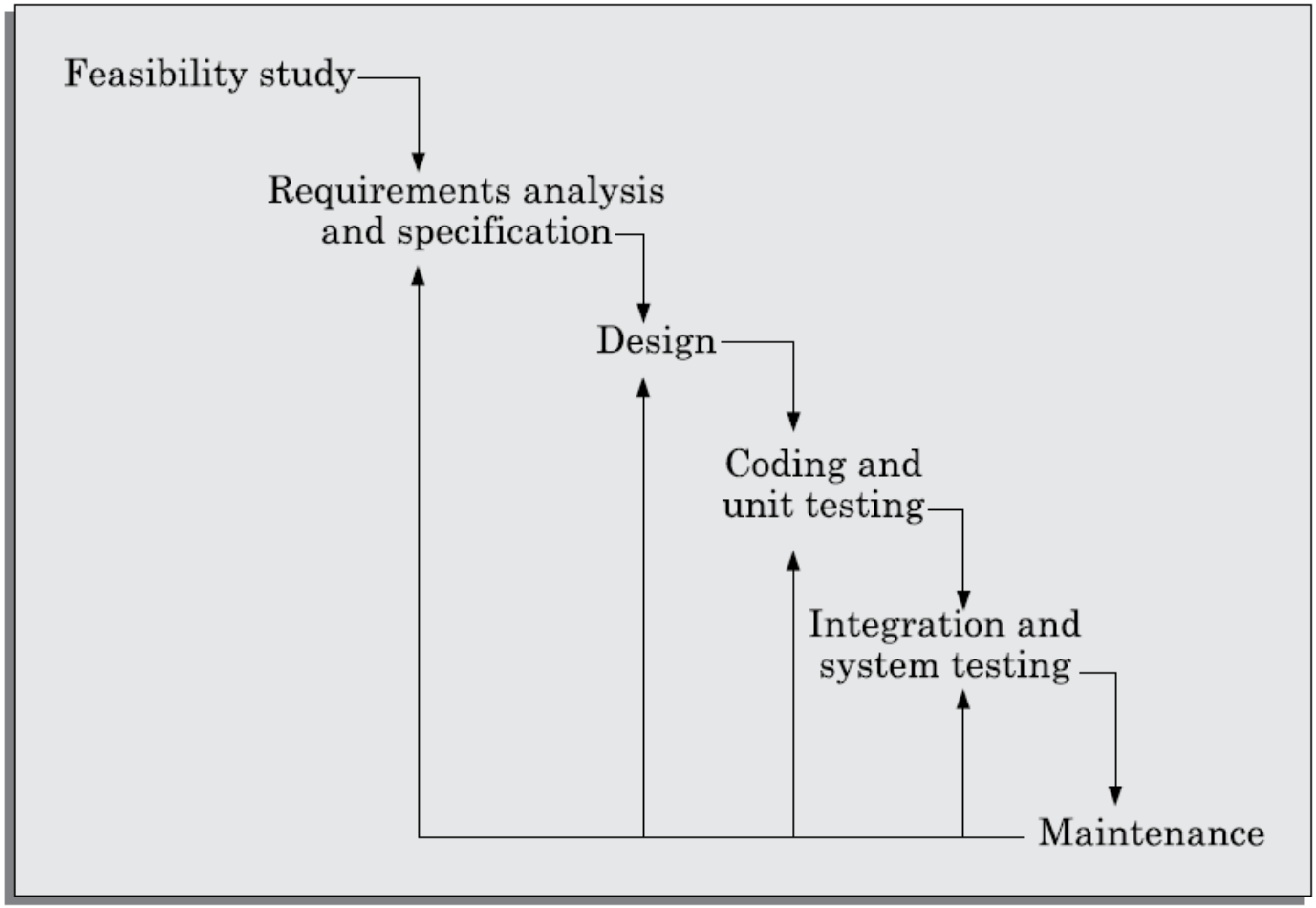
# The Waterfall Model

- The waterfall model represents an idealized version of the software life cycle.
- If a problem that was created in an early development phase is not discovered until a later stage in the development process, the problem must either be ignored or the entire development process must be altered.
- A step-by-step development means that no tangible product is available for assessment by the client until late in the development process.
- The waterfall approach is generally only taken when the requirements of a project are well defined and not likely to change.

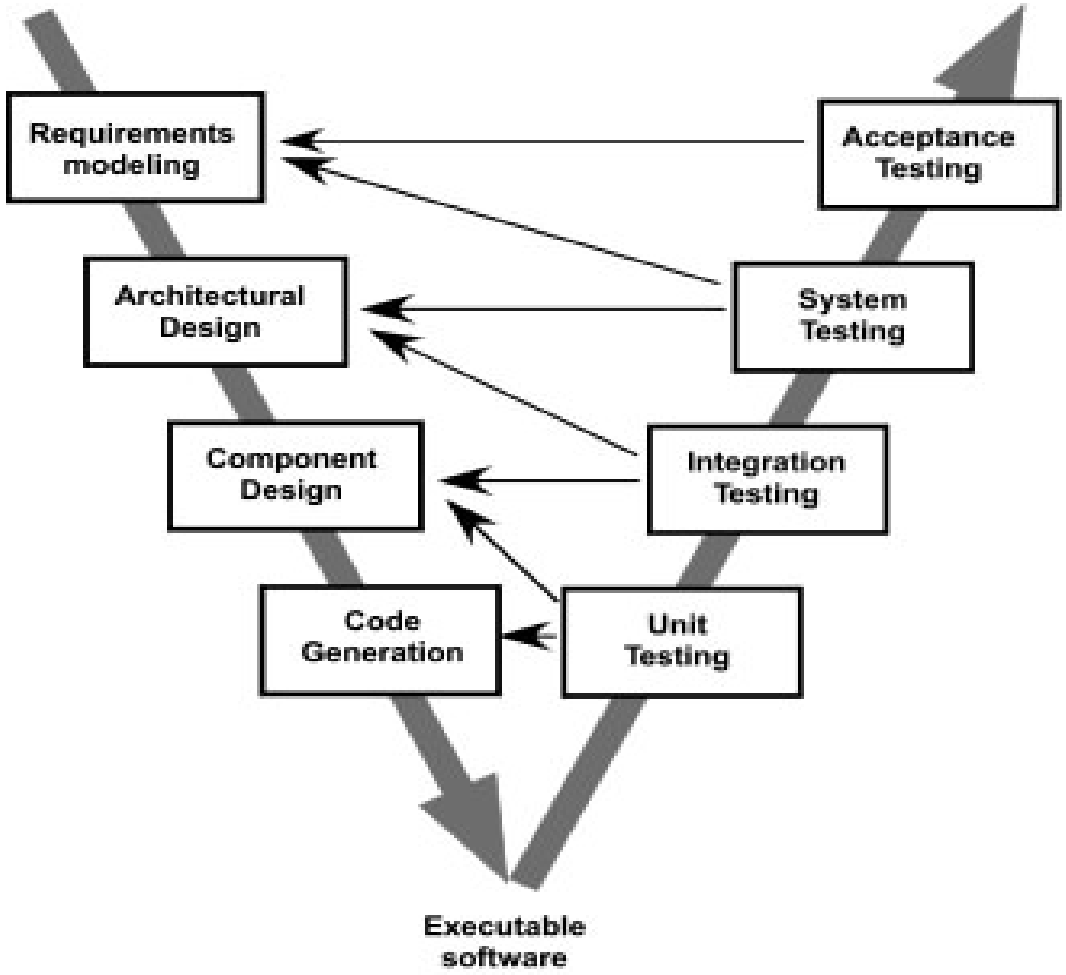
# The Waterfall Model



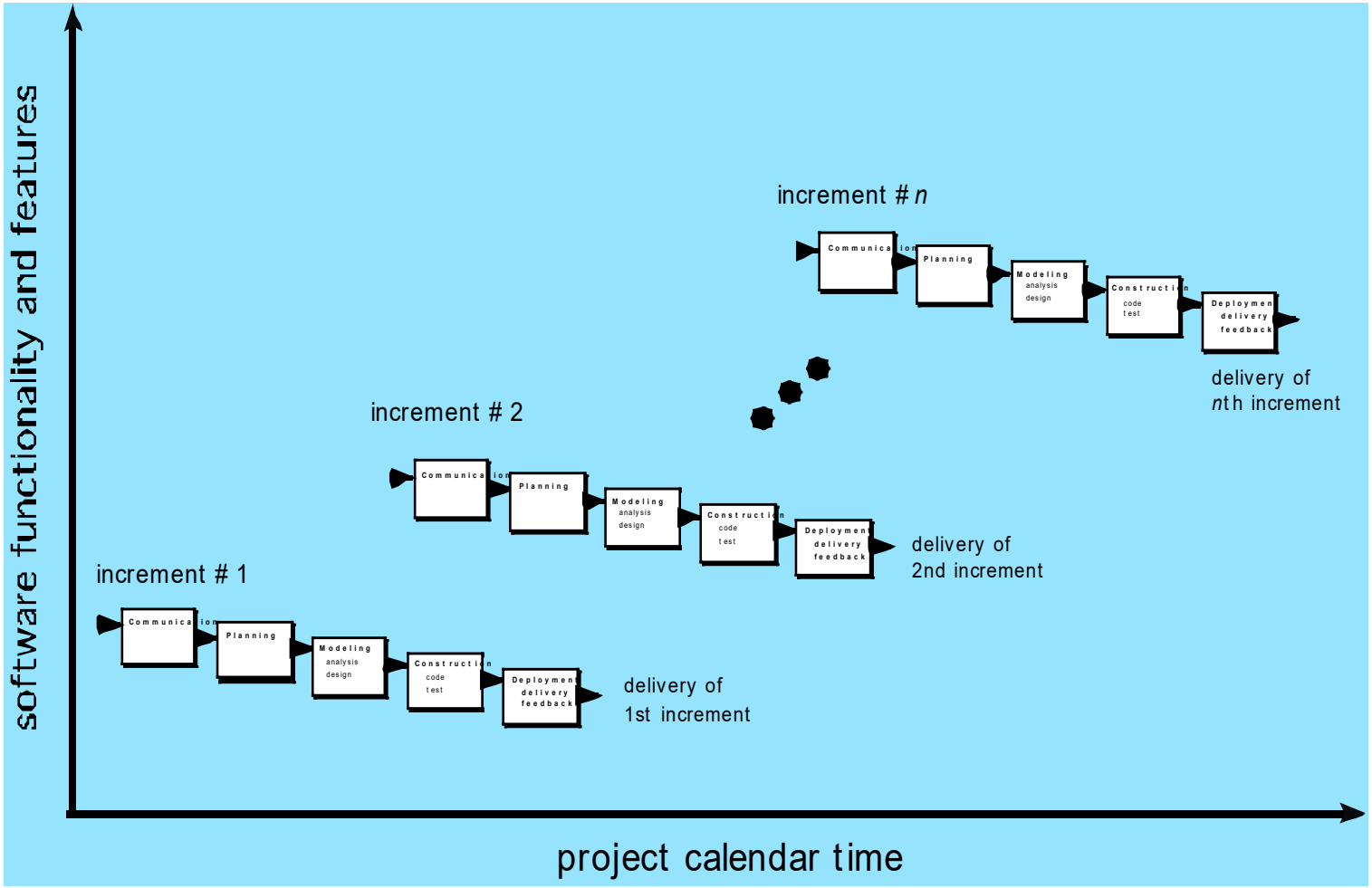
# Iterative waterfall model



# The V-Model


Requirements modeling
Architectural Design
Component Design
Code Generation
Unit Testing
Integration Testing
System Testing
Acceptance Testing
Executable software

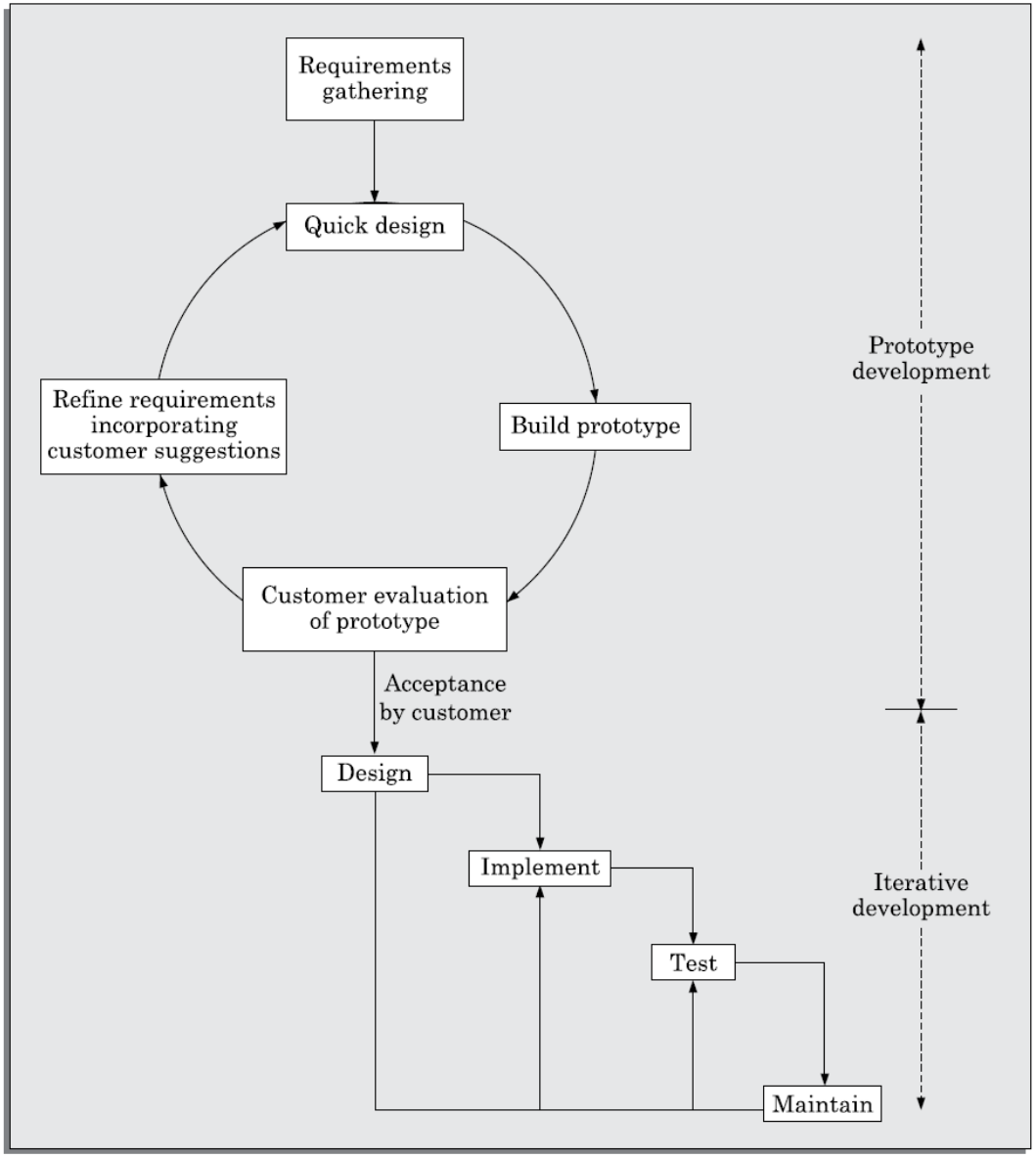
# The Incremental Model



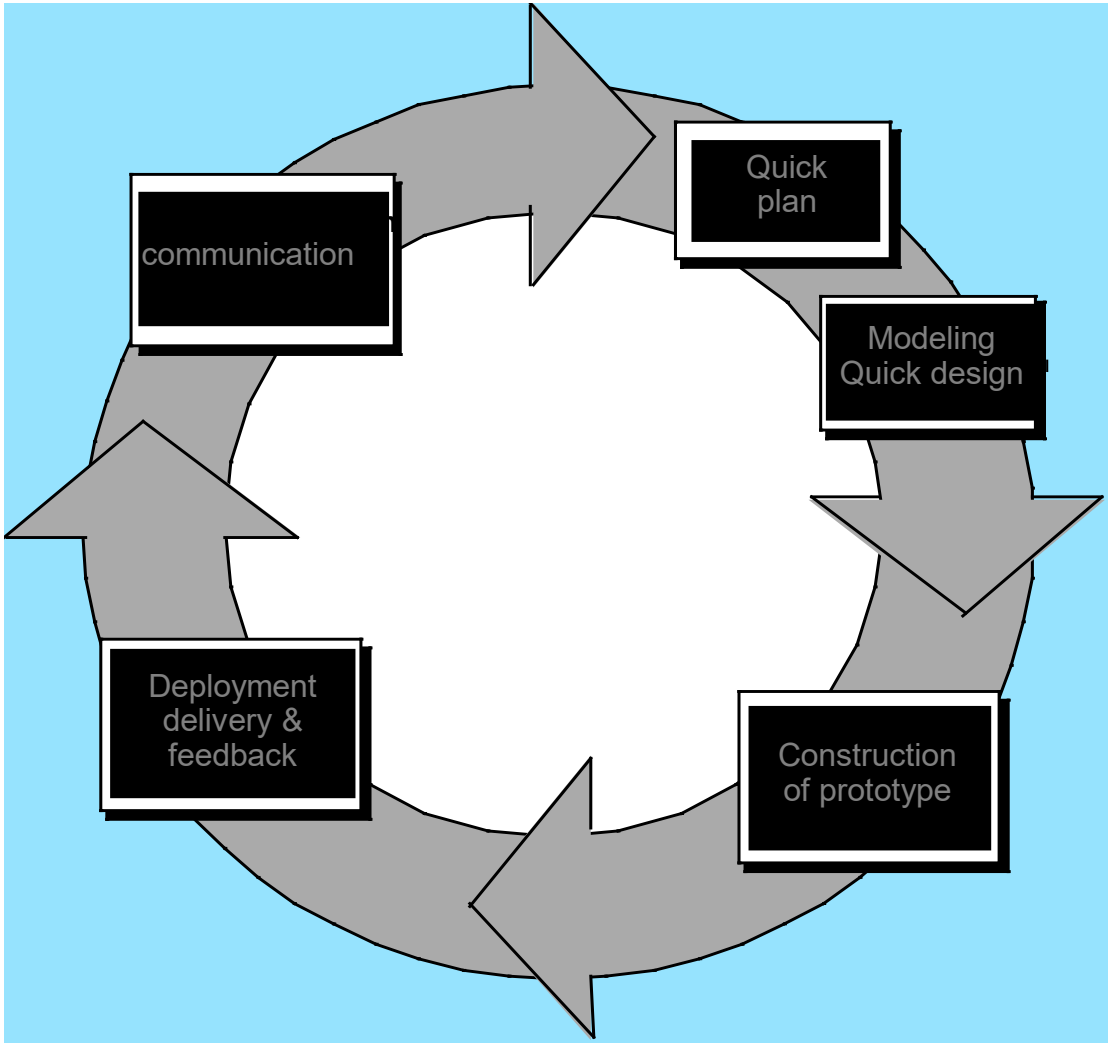
# The Incremental Model

- The incremental model combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping.
- Figure 4.3, the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable “increment” of the software.
- Example: Word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; More sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment… and so on.

# Evolutionary Models: Prototyping



# Evolutionary Models: Prototyping



# Evolutionary Models: Prototyping

- Prototyping can also be problematic for the following reasons:
  - What are questions of Prototying model?

    See Textbook, page 46.
- Although problems can occur, prototyping can be an effective paradigm for software engineering.
- The key is to define the rules of the game at the beginning. that is, the customer and developer must both agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded (at least in part) and the actual software is engineered with an eye toward quality and maintainability.

# Evolutionary Models: Prototyping

- Prototype model can be used for two types of project
  - The prototype model turns out to be especially useful in developing the GUI. For the user, it becomes much easier to form an opinion regarding what would be more suitable by experimenting with a working user interface, rather than trying to imagine the working of a hypothetical user interface.
  - The prototype model is especially useful when the exact technical solutions are unclear to the development team. A prototype can help them to critically examine the technical issues associated with product development. A prototype is often the best way to resolve the technical issues.

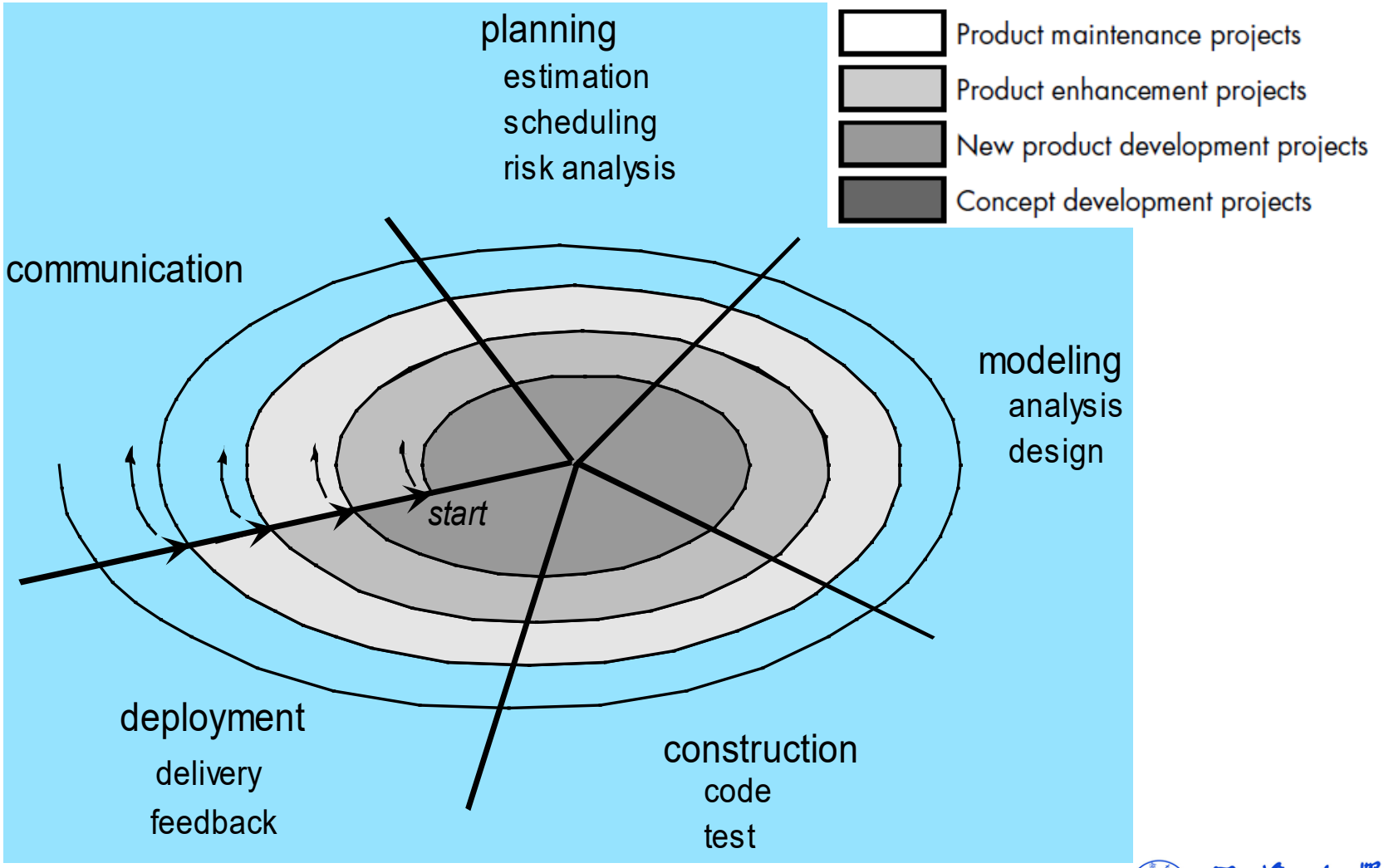
# The Incremental Model and Prototype Model

- The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.
- The prototype used to be identifying software requirements.
- Brooks [BRO75] *In most projects, the first system built is barely usable. It may be too slow, too big, awkward in use or all three. There is no alternative but to start again, smarting but smarter, and build a redesigned version in which these problems are solved . . . The management question, therefore, is not whether to build a pilot system and throw it away. You will do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers . . .*

# The Incremental Model and Prototype Model

- **S**cenario (**E**xample from the textbook p47)
  - A customer often defines a set of general objectives for software but does not identify detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take. In these, and many other situations, a prototyping paradigm may offer the best approach.
- The prototyping paradigm (Figure 4.4) begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory(必须的).
  - A "quick design" leads to the construction of a prototype.

# Evolutionary Models: The Spiral Model



# Evolutionary Models: The Spiral Model

- The Spiral model, originally proposed by Boehm [BOE88], is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.
- It provides the potential for rapid development of incremental versions of the software.
- Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

# Evolutionary Models: The Spiral Model

- The spiral model is a realistic approach to the development of large-scale systems and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level.
- The spiral model uses prototyping as a risk reduction mechanism but, more important, enables the developer to apply the prototyping approach at any stage in the evolution of the product.
- It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world.
- The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic.

# Evolutionary Models: The Spiral Model

- Spiral model as a meta model
  - As compared to the previously discussed models, the spiral model can be viewed as a meta model.
  - Since this model subsumes all the discussed models.
  - Example
    - A single loop spiral actually represents the waterfall model. The spiral model incorporates the systematic step-wise approach of the waterfall model.
    - The spiral model uses the approach of the prototyping model by first building a prototype in each phase before the actual development starts. This prototype is used as a risk reduction mechanism.
    - The spiral model can be considered as supporting the evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built. This enables the developer to understand and resolve the risks at each level (i.e. iteration along the spiral).

# Evolutionary Models: the final word

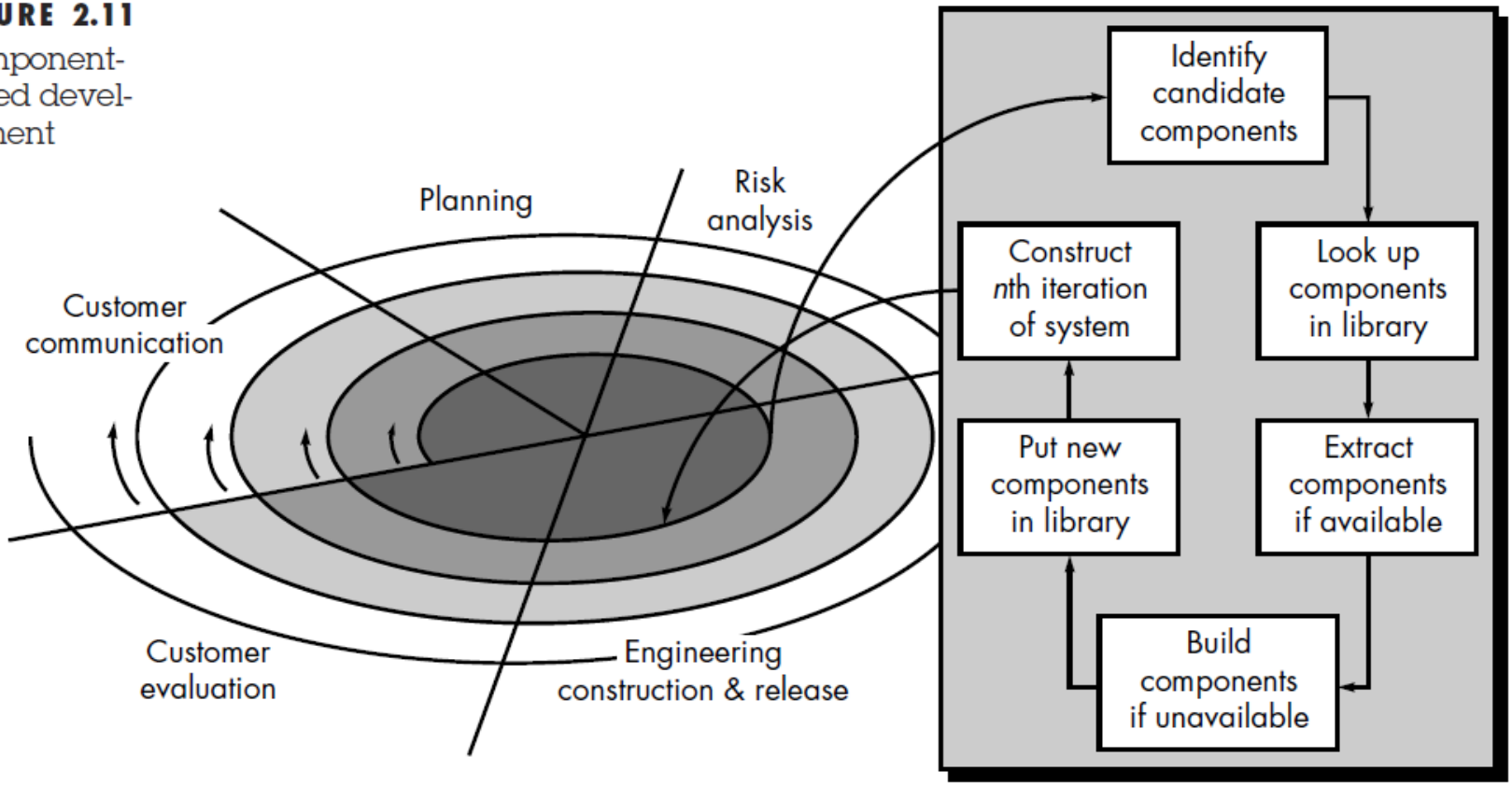
- Evolutionary process models were conceived to address these issues, and yet, as a general class of process models, they too have weaknesses. These are summarized by Nogueira and his colleagues. Please look at textbook p52.
- Indeed, a software process that focuses on flexibility, extensibility, and speed of development over high quality does sound scary. And yet, this idea has been proposed by a number of well-respected software engineering experts (e.g., [You95], [Bac97II).
- The intent of evolutionary models is to develop high-quality software in an iterative or incremental manner. However, it is possible to use an evolutionary process to emphasize flexibility, extensibility, and speed of development. The challenge for software teams and their managers is to establish a proper balance between these critical project and product parameters and customer satisfaction (the ultimate arbiter仲裁人 of software quality).

# Component based Process Model

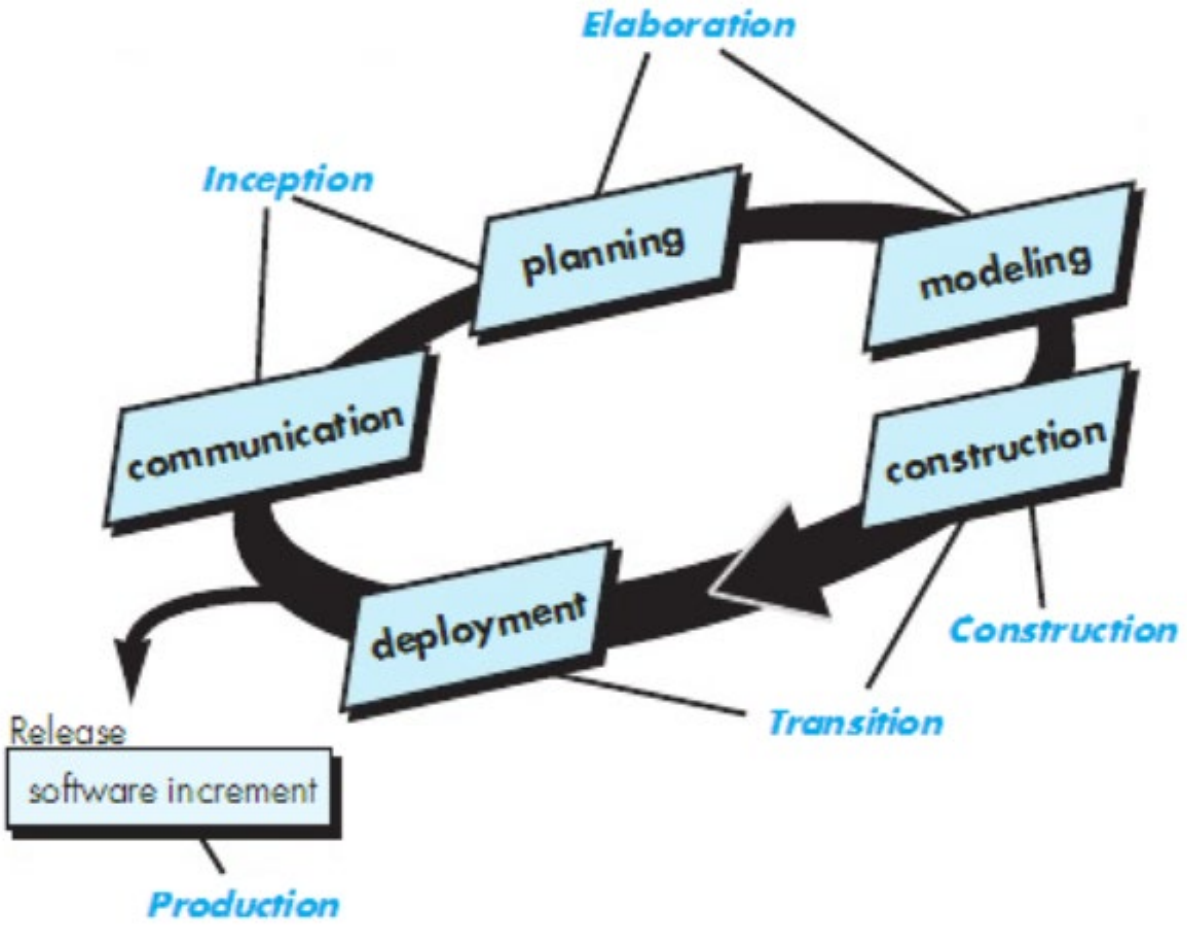
- Component based development

The process to apply when reuse is a development objective.

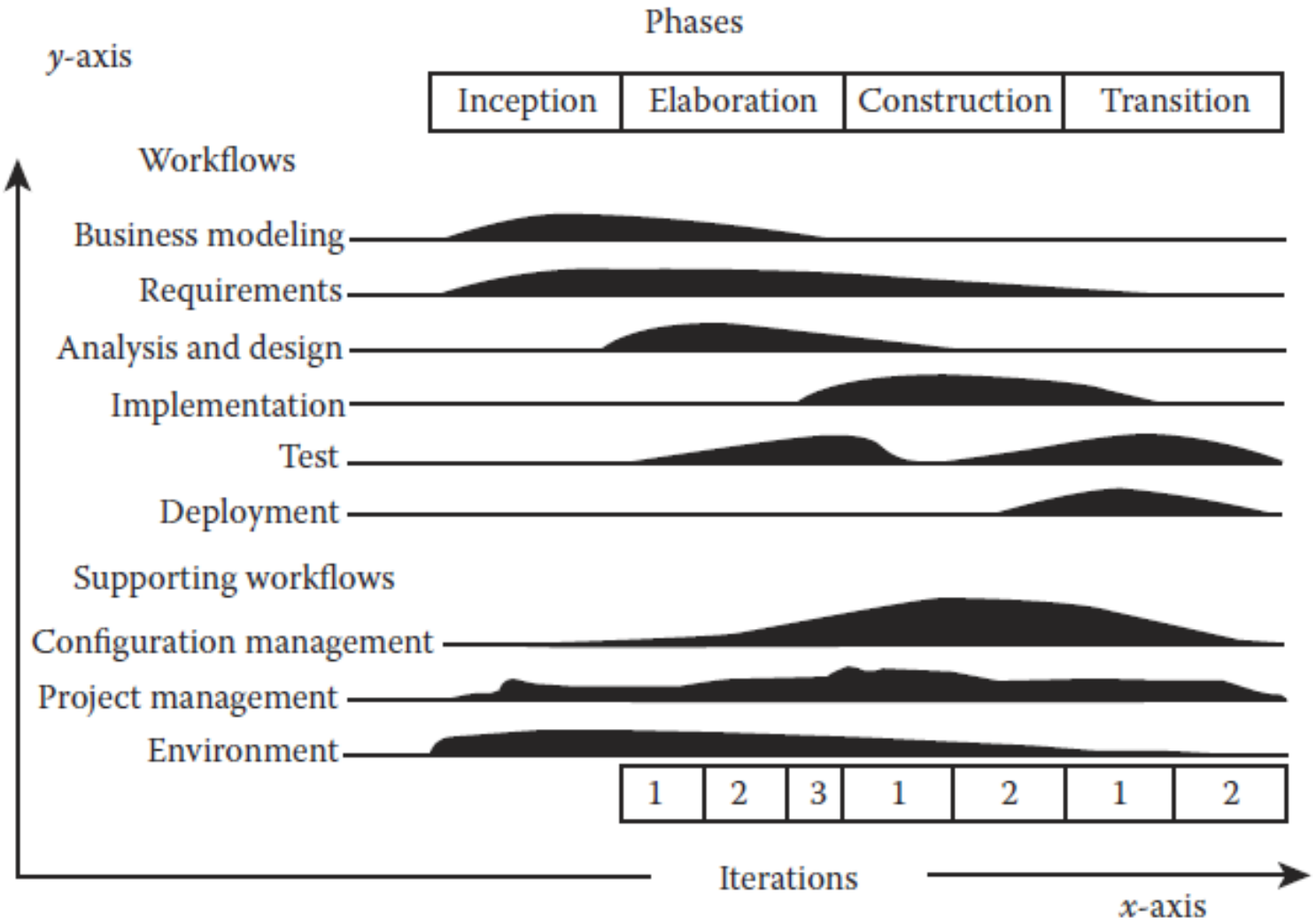
**FIGURE 2.11**
Component-based development



# The Unified Process (UP)



# UP Phases


Phases
y-axis
Inception
Elaboration
Construction
Transition
Workflows
Business modeling
Requirements
Analysis and design
Implementation
Test
Deployment
Supporting workflows
Configuration management
Project management
Environment
1
2
3
1
2
1
2
Iterations
x-axis

# UP Work Products

**Inception phase**

- Vision document
- Initial use-case model
- Initial project glossary
- Initial business case
- Initial risk assessment.
- Project plan,
  - phases and iterations.
- Business model,
  - if necessary.
- One or more prototypes

**Elaboration phase**

- Use-case model
- Supplementary requirements
  - including non-functional
- Analysis model
- Software architecture
  - Description.
- Executable architectural
  - prototype.
- Preliminary design model
- Revised risk list
- Project plan including
  - iteration plan
  - adapted workflows
  - milestones
  - technical work products
- Preliminary user manual

**Construction phase**

- Design model
- Software components
- Integrated software
  - increment
- Test plan and procedure
- Test cases
- Support documentation
  - user manuals
  - installation manuals
  - description of current
    - increment

**Transition phase**

- Delivered software increment
- Beta test reports
- General user feedback

# How to Select an Appropriate Model

- Characteristics of the software to be developed.
- Characteristics of the development team.
- Characteristics of the needs from the customer.
- Characteristics of the market.
- Characteristics of the trends of technology development.

# Software product & process

- Table: Comparison of process models

  From Textbook《Software Engineering》, P19.

# Q&A in classroom

- What are the shortcomings of the waterfall model?
  - Difficult to accommodate change requests.
  - Incremental delivery not supported.
  - Phase overlap not supported.
  - Error correction unduly expensive.
  - Limited customer interactions.
  - Heavy weight.
  - No support for risk handling and reuse.
- What is the difference between V-model and waterfall model?
  - Development and validation(testing) activities proceed hand in hand.
  - Being a derivative of the waterfall model, V-model inherits most of the weaknesses of the waterfall model.

# Summary

- A generic process model for software engineering encompasses a set of Framework and Umbrella activities, actions, and work tasks.
- Each of a variety of process Models can be described by a different process flow – a description of how the framework activities, actions, and tasks are organized sequentially and chronologically(按时间顺序).
- Software engineering is a discipline that integrates process, methods, and tools for the development of computer software.

# Summary

- A number of different process models for software engineering have been proposed, each exhibiting strengths and weaknesses, but all having a series of generic phases in common.
- The principles, concepts, and methods that enable us to perform the process that we call software engineering are considered throughout the remainder of this course.

# Assignment

- Questions for Classroom
  - We have said that software engineering is more than just programming and computer science. What do you understand this to mean?
  - The potential list of professional skills involved in a software engineering project is nearly limitless. List three people involved in a software engineering project who are not computer programmers.
  - What are the risks and benefits of the waterfall model development paradigm?
  - We have described software engineering as a practice in problem solving. Pick any software application that you are familiar with, and describe its purpose in terms of an existing challenge, and the solution that it represents.

# Assignment

- Chapter 2: Problems to ponder
  - 2.4 A common problem during Communication occurs when you encounter two stakeholders who have conflicting ideas about what the software should be. That is, you have mutually conflicting requirements.

    Develop a process pattern (this would be a stage pattern) using the template presented in Section 3.4 that addresses this problem and suggest an effective approach to it.
  - 2.8 Can we use several process models? If possible, please give an example.
  - 2.9 What are the advantages and disadvantages of developing software in which quality is “good enough”? That is, what happens when we emphasize development speed over product quality?

huangjie@tongji.edu.cn

# Assignments

- Preview

  《Software Engineering》, written by R.S.Pressman

  Chapter 3 Agile and Agile Process


huangjie@tongji.edu.cn