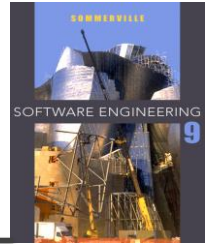


---

# Chapter 2 – Software Processes

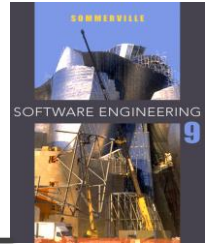


# Topics covered

---

- ✧ Software process models
- ✧ Process activities
- ✧ Coping with change
- ✧ The Rational Unified Process
  - An example of a modern software process.

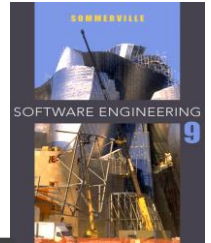
# The software process



- ✧ A structured **set of activities required** to develop a software system.
- ✧ Many different software processes but **all involve**:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- ✧ A **software process model** is an abstract representation of a process. It presents a description of a process from some particular perspective.

# Software process descriptions

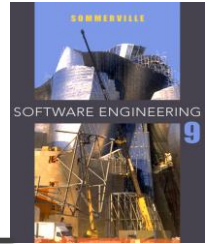
---



- ✧ When we describe and discuss processes, we usually talk about **the activities in these processes** such as specifying a data model, designing a user interface, etc. **and the ordering of these activities.**
- ✧ Process descriptions **may also include:**
  - Products, which are **the outcomes** of a process activity;
  - Roles, which reflect the **responsibilities of the people** involved in the process;
  - Pre- and post-conditions, which are **statements that are true before and after** a process activity has been enacted or a product produced.

# Plan-driven and agile processes

---



- ✧ **Plan-driven processes** are processes where all of the process activities are **planned in advance and progress is measured** against this plan.
- ✧ In agile processes, **planning is incremental** and it is **easier to change the process** to reflect changing customer requirements.
- ✧ Agile methods break tasks into **small increments** with minimal planning and **do not directly involve long-term planning**.
- ✧ Iterations are short time frames ([timeboxes](#)) that typically last from one to four weeks.

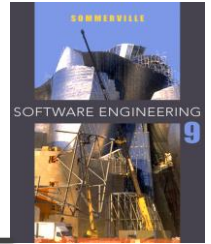


- ✧ Each iteration involves a cross functional team working in all functions: planning, requirements analysis, design, coding, unit testing, and acceptance testing.
- ✧ At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly.
- ✧ An iteration **might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs)** at the end of each iteration.
- ✧ **Multiple iterations might be required** to release a product or new features.

- ✧ In practice, most practical processes **include elements** of both plan-driven and agile approaches.
- ✧ There are **no right or wrong software processes**.
- ✧ Although there is **no ‘ideal’ software process**, there is scope for improving the software process in many organizations.
- ✧ E.g., by **standardization** where **the diversity** in software processes across an organization **is reduced**
- ✧ This leads to **improved communication and a reduction in training time, and makes automated process support more economical.**

# Software process models

---



## ✧ The waterfall model

- Plan-driven model. Separate and distinct phases of specification and development.

## ✧ Incremental development

- Specification, development and validation are interleaved.
- May be plan-driven or agile.

## ✧ Reuse-oriented software engineering

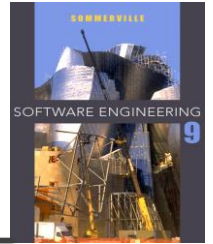
- The system is assembled from existing components. May be plan-driven or agile.

✧ In practice, **most large systems** are developed using a process that **incorporates elements from all of these models.**

- 
- ✧ Parts of the system that **are well understood** can be specified and developed using a **waterfall-based** process.
  - ✧ Parts of the system which are **difficult to specify** in advance, such as the user interface, should always be developed using an **incremental approach**.

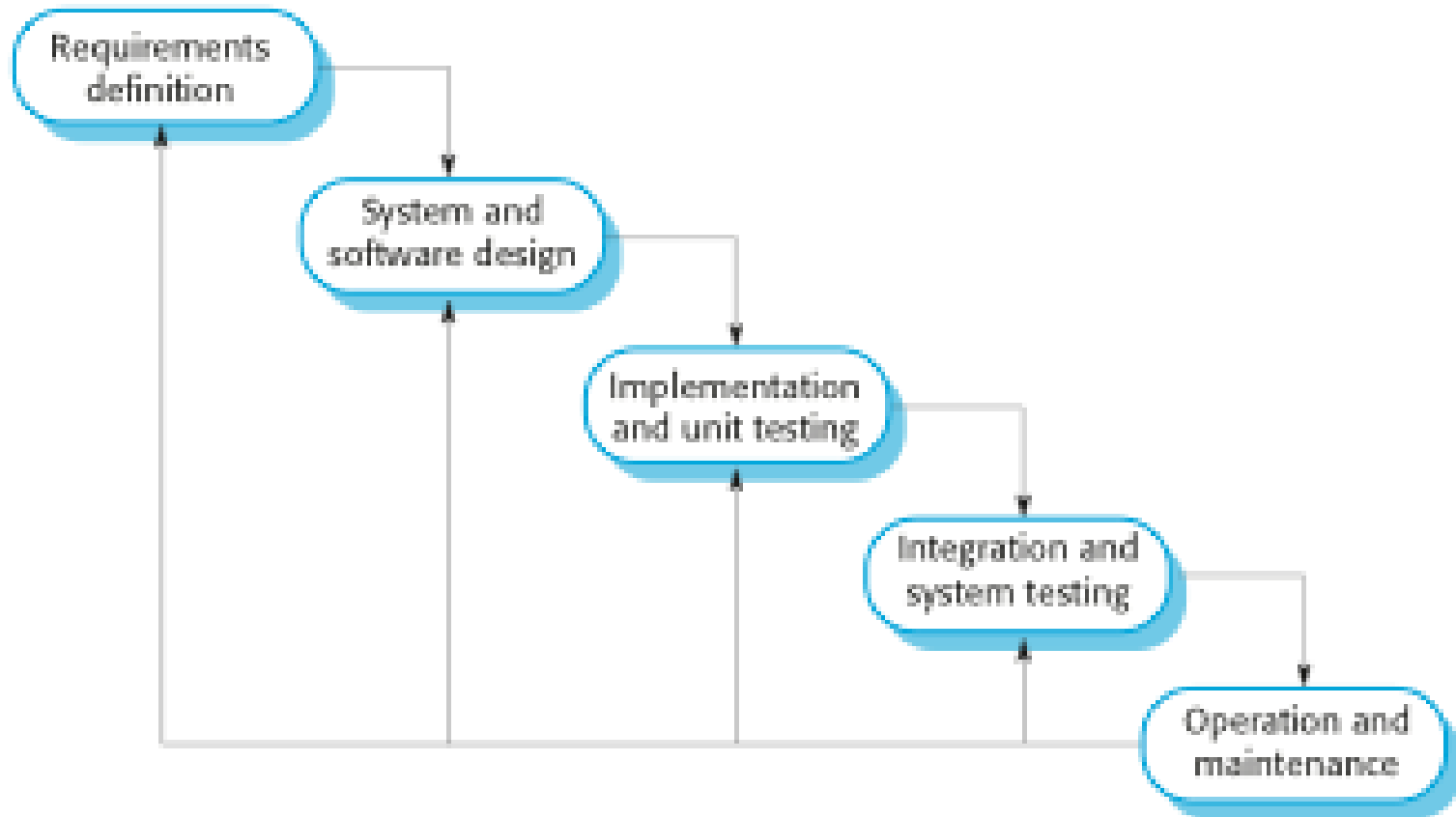
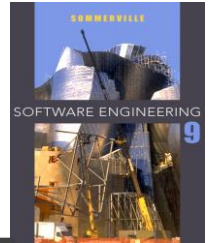
# The waterfall model

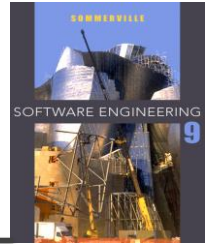
---



- ✧ The waterfall model is an example of a **plan-driven process**—in principle, **you must plan and schedule all of the process activities** before starting work on them.

# The waterfall model





# The principal stages of the waterfall model

---

## 1. Requirements analysis and definition

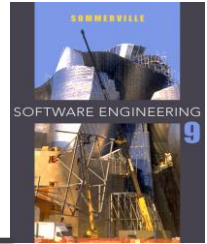
The system's **services, constraints, and goals** are established by **consultation with system users**. They are then defined in detail and **serve as a system specification**.

## 2. System and software design

The systems design process **allocates the requirements to either hardware or software** systems by **establishing an overall system architecture**.

## 3. Implementation and unit testing

During this stage, the software design is **realized as a set of programs** or program units. Unit testing **involves verifying that each unit meets its specification**



## 4. Integration and system testing

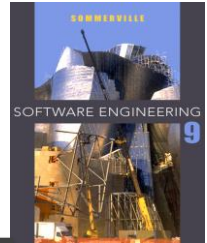
The individual program units or programs are integrated and tested as a complete system **to ensure that the software requirements have been met**. After testing, the software system is delivered to the customer.

## 5. Operation and maintenance

Normally (although not necessarily), this is **the longest life cycle** phase. The system is installed and put into practical use. Maintenance **involves correcting errors** which were not discovered in earlier stages of the life cycle, **improving the implementation** of system units and **enhancing the system's services as new requirements are discovered**.

## the result of each phase is one or more documents

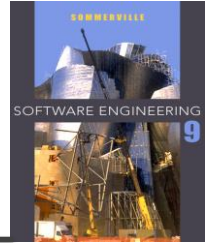
---



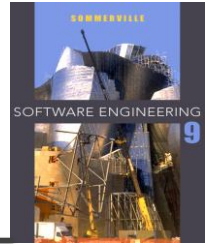
- ✧ In principle, the result of each phase is one or more documents that **are approved ('signed off')**. The **following phase should not start until the previous phase has finished**.
- ✧ In practice, these stages overlap and feed information to each other.
- ✧ **During design, problems with requirements are identified.**
- ✧ **During coding, design problems are found** and so on. The software process is not a simple linear model but involves feedback from one phase to another.
- ✧ Documents produced in each phase may then **have to be modified** to reflect the changes made

# Because of the costs of producing and approving documents

---



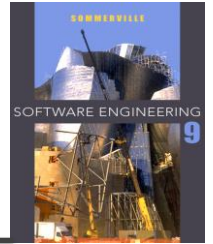
- ✧ iterations can be **costly and involve significant** rework.
- ✧ Therefore, after a small number of iterations, **it is normal to freeze parts of the development**, such as **the specification**, and to continue with the later development stages.
- ✧ **Problems are left for later resolution, ignored, or programmed around.**
- ✧ This premature freezing of requirements **may mean that the system won't do what the user wants.**
- ✧ It may also **lead to badly structured systems** as design problems are circumvented by implementation tricks.



# Deliverables (Documents) in Waterfall Model

---

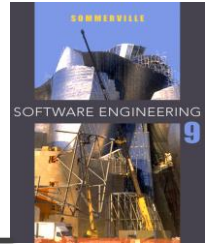
- ✧ Project plan and feasibility report
- ✧ Requirements documents (SRS: Software Requirement Specifications)
- ✧ System Design documents
- ✧ Test plans and test reports
- ✧ Source code
- ✧ Software manuals
- ✧ Each step ends with a review process and report



# Advantages of producing documents

---

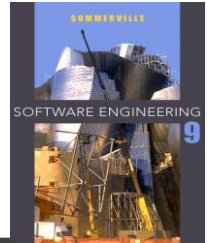
- ✧ Different teams can work on different steps of the projects
- ✧ Important for quality assurance and testing
- ✧ Important for evolution



# Waterfall model drawback

---

- ✧ The **main drawback** of the waterfall model is the **difficulty of accommodating change** after the process is underway.
- ✧ In principle, a **phase has to be complete** before moving onto the next phase.



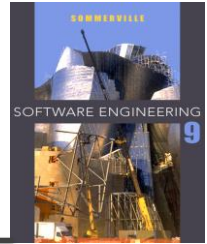
# Waterfall model problems

---

- ✧ Inflexible partitioning of the project into distinct stages makes it **difficult to respond to changing customer requirements.**
  - Therefore, this model is only **appropriate when the requirements are well-understood (e.g. there is a manual counterpart) and changes will be fairly limited** during the design process.
  - **Few business systems have stable requirements.**
  - **Documentation heavy**
- ✧ The waterfall model is mostly **used for large systems engineering projects** where a system is **developed at several sites.**
  - In those circumstances, the **plan-driven nature of the waterfall model helps coordinate the work.**

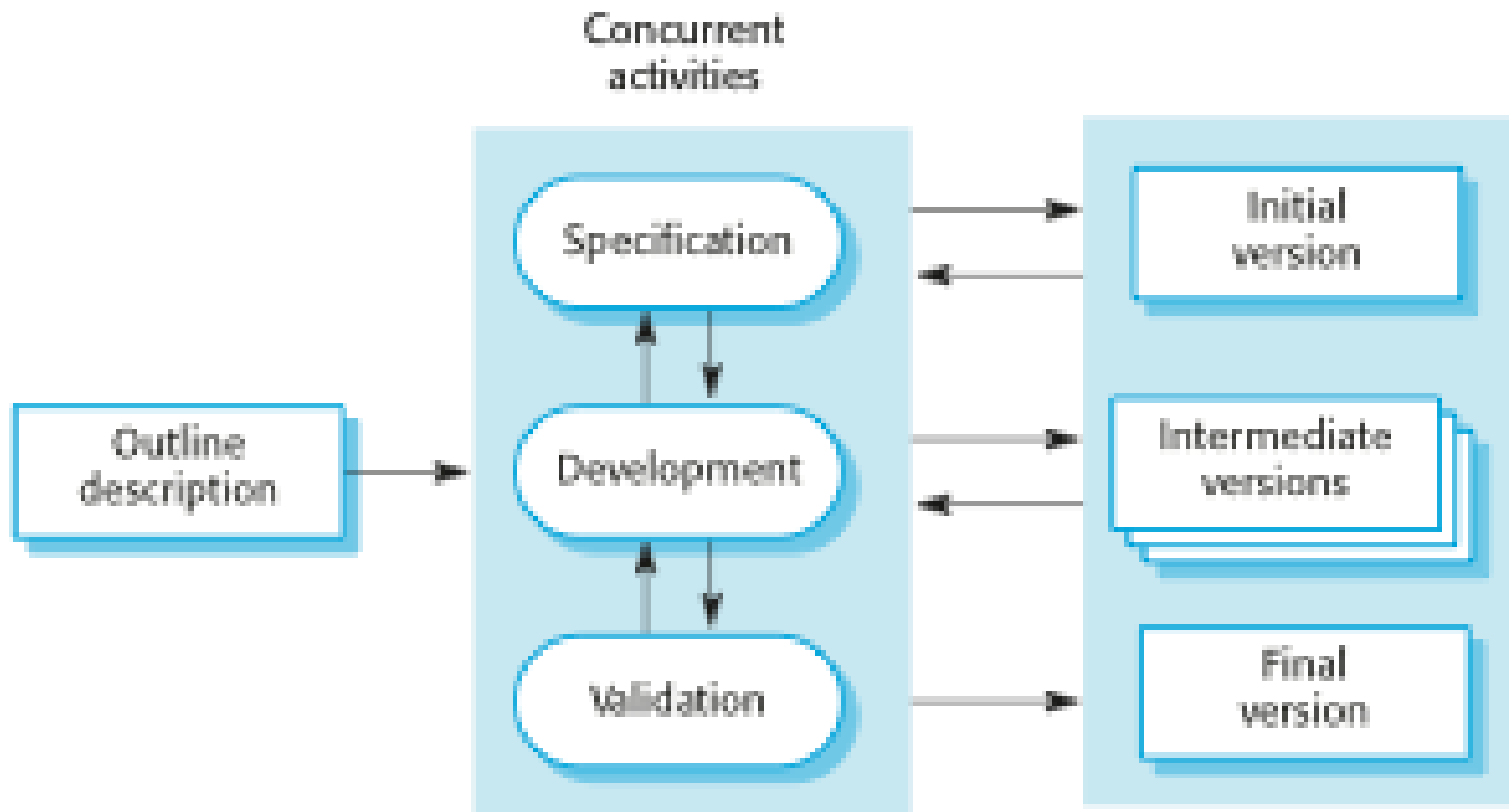
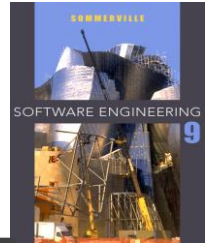
# Incremental development

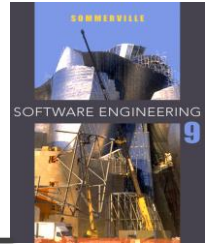
---



- ✧ Incremental development is based on the idea of developing **an initial implementation**, exposing this to **user comment** and **evolving it** through **several versions until** an adequate system has been developed
- ✧ Specification, development, and validation **activities are interleaved** rather than separate, **with rapid feedback across activities.**
- ✧ A **fundamental part of agile** approaches, is **better than a waterfall approach for most business**, e-commerce, and personal systems
- ✧ By developing the software incrementally, **it is cheaper and easier to make changes** in the software as it is being developed.

# Incremental development

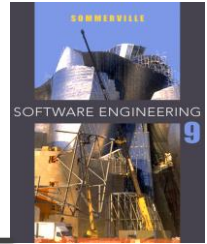




# Incremental development benefits

---

- ✧ The **cost of accommodating changing customer requirements is reduced.**
  - The **amount of analysis and documentation that has to be redone is much less** than is required with the waterfall model.
- ✧ It is **easier to get customer feedback** on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- ✧ More **rapid delivery and deployment** of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

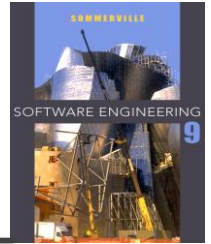


# Incremental development problems

---

- ✧ The process is not visible.
  - Managers need regular deliverables to measure progress. If **systems are developed quickly**, it is **not cost-effective to produce documents** that reflect every version of the system.
- ✧ **System structure tends to degrade as new increments are added.**
  - **Unless time and money is spent on refactoring** to improve the software, **regular change tends to corrupt its structure**. Incorporating further software **changes becomes increasingly difficult and costly**.

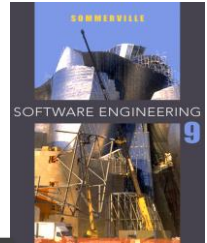
# Reuse-oriented software engineering



- ✧ in the 21st century, software development processes that focus on **the reuse of existing software have become widely used.**
- ✧ Reuse-oriented approaches **rely on a large base of reusable software components** and an integrating framework for the composition of these components.
- ✧ Based on systematic reuse where systems are **integrated from existing components** or COTS (**Commercial-off-the-shelf**) systems.
- ✧ Popular in web software development
- ✧ Reuse is now the **standard approach for building many types of business system**

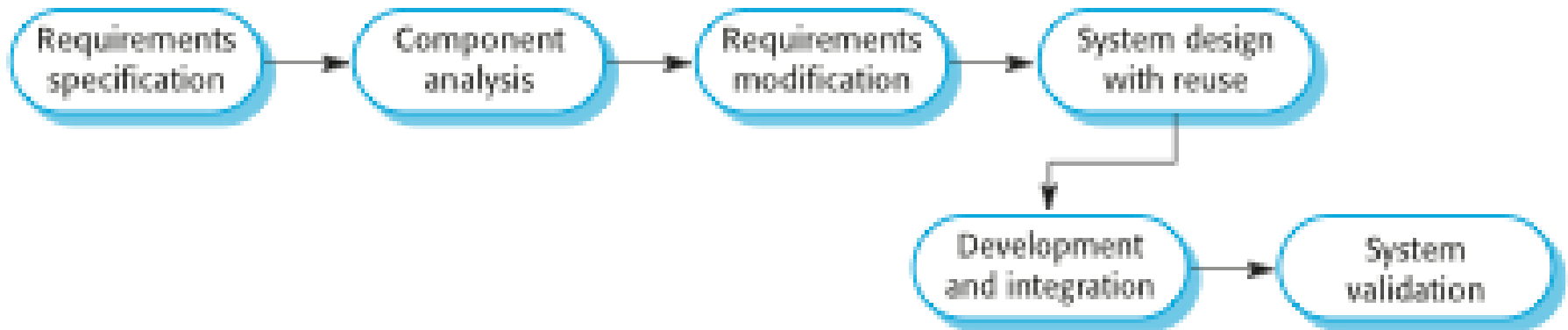
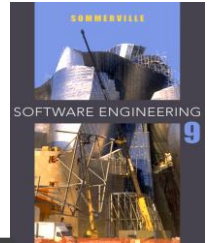
- Reuse covered in more depth in Chapter 16.

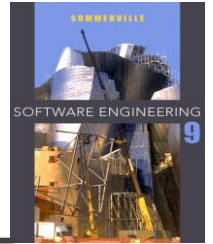
1. **Component analysis** Given the requirements specification, a search is made for components to implement that specification. Usually , **there is no exact match and the components that may be used only provide some of the functionality required.**
2. **Requirements modification** During this stage, the **requirements are analyzed** using information about the components that have been discovered. They are then **modified to reflect the available components.** **Where modifications are impossible,** the component analysis activity may be **re-entered to search for alternative solutions.**



- 
3. **System design with reuse** During this phase, the **framework of the system is designed or an existing framework is reused**. The designers take into account the components that are reused and organize the framework to cater for this. **Some new software may have to be designed if reusable** components are not available.
  4. **Development and integration** Software that **cannot be externally procured is developed**, and the components and COTS systems are integrated to create the new system.

# Reuse-oriented software engineering

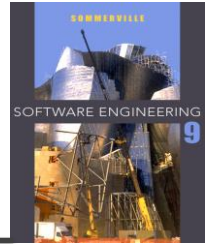




# Types of software component

---

- ✧ Web services that are developed according to service standards and which are **available for remote invocation**.
- ✧ **Collections of objects** that are developed **as a package** to be integrated with a component framework such as .NET or J2EE.
- ✧ **Stand-alone software systems (COTS)** that are configured for use in a particular environment.



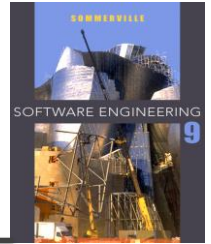
# Advantages and Disadvantages

---

- ✧ Reuse-oriented software engineering has **the obvious advantage of reducing the amount of software to be developed and so reducing cost and risks**. It usually also leads to **faster delivery** of the software.
- ✧ However, **requirements compromises are inevitable** and this may **lead to a system that does not meet the real needs of users**.
- ✧ Furthermore, **some control over the system evolution is lost** as **new versions of the reusable components are not under the control of the organization** using them.

# Process activities

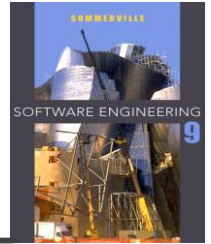
---



- ✧ Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- ✧ The four basic process activities of specification, development, validation and evolution are **organized differently in different development processes**. In the **waterfall** model, they are **organized in sequence**, whereas in **incremental development** they are **inter-leaved**.

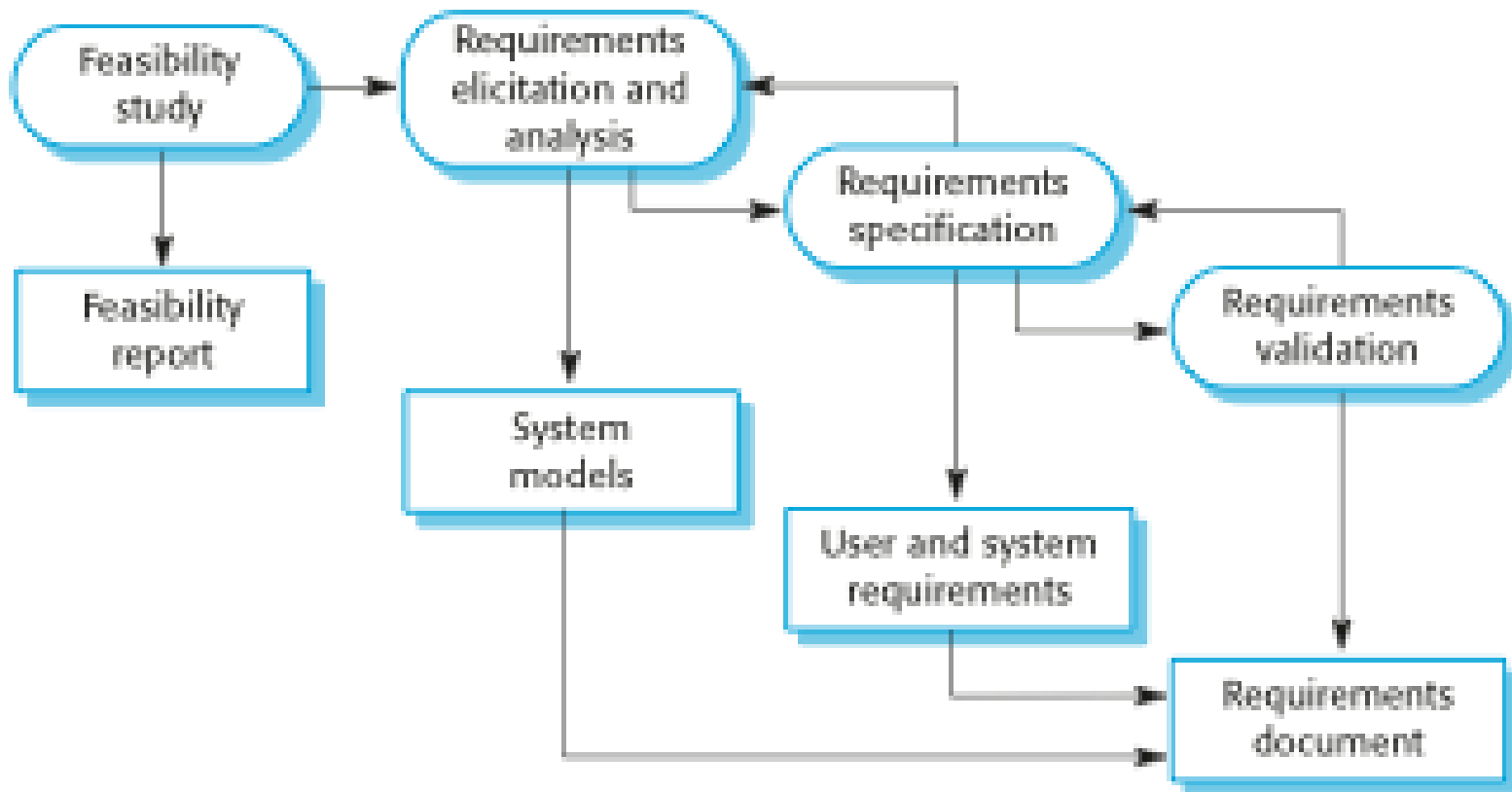
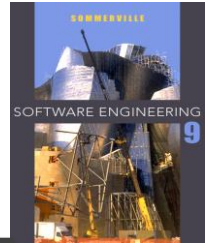
# Software specification

---



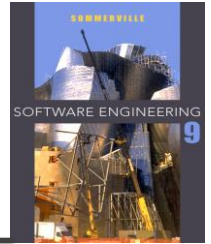
- ✧ The process of establishing **what services are required and the constraints** on the system's operation and development.
- ✧ Requirements engineering process
  - **Feasibility study**
    - Is it technically and financially feasible to build the system?
  - Requirements elicitation and analysis
    - **What do the system stakeholders require** or expect from the system?
  - Requirements specification
    - **Defining the requirements in detail**
  - Requirements validation
    - **Checking the validity of the requirements**

# The requirements engineering process



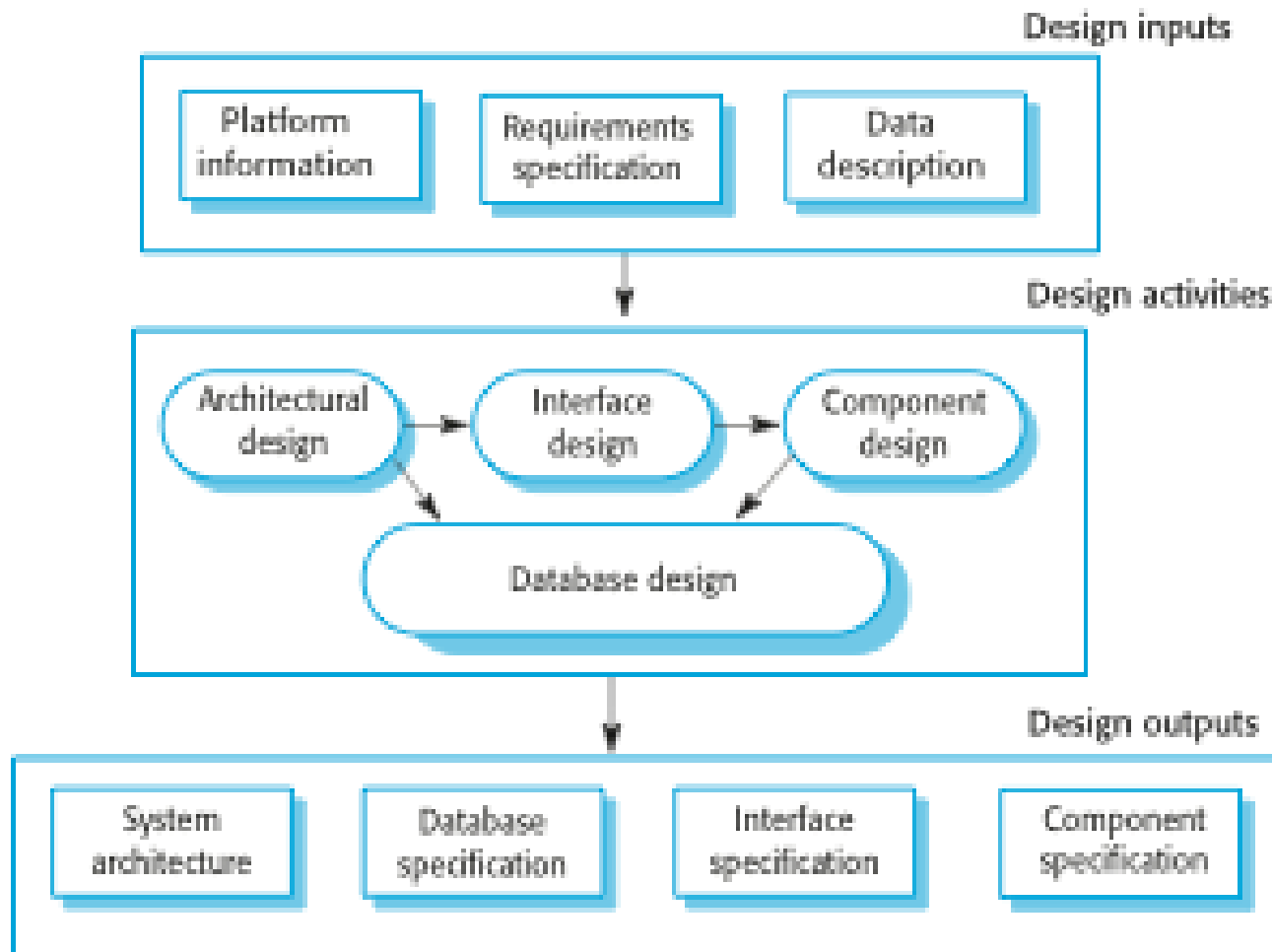
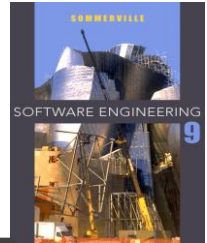
# Software design and implementation

---



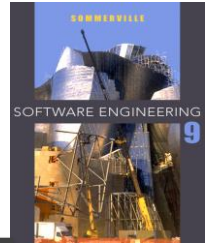
- ✧ The process of **converting the system specification into an executable system.**
- ✧ Software design
  - Design a **software structure** that realises the specification;
- ✧ Implementation
  - Translate this structure into an **executable program**;
- ✧ The activities of **design and implementation are closely related** and may be inter-leaved.

# A general model of the design process



# Design activities

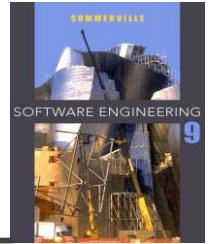
---



- ✧ ***Architectural design***, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- ✧ ***Interface design***, where you define the interfaces between system components.
- ✧ ***Component design***, where you take each system component and design how it will operate.
- ✧ ***Database design***, where you design the system data structures and how these are to be represented in a database.

# Software validation

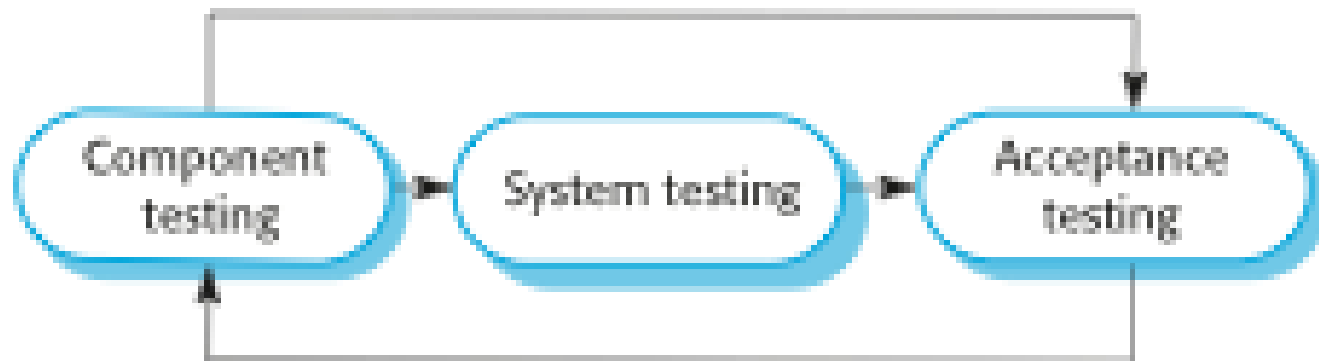
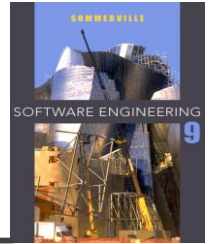
---



- ✧ Verification and validation (V & V) is intended to show that a system **conforms to its specification and meets the requirements of the system customer.**
- ✧ **Involves checking and review processes and system testing.**
- ✧ **System testing** involves executing the system with test cases that are **derived from the specification of the real data** to be processed by the system.
- ✧ Testing is the most commonly used V & V activity.

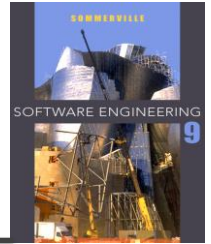
# Stages of testing

---



# Testing stages

---



## ✧ Development or component testing

- Individual components are tested **independently by the people developing the system**;
- Components may be functions or objects or coherent groupings of these entities.
- Test automation tools, such as JUnit (Massol and Husted, 2003), that can re-run component tests when new versions of the component are created, are commonly used

---

## ✧ System testing (Testing of the system as a whole)

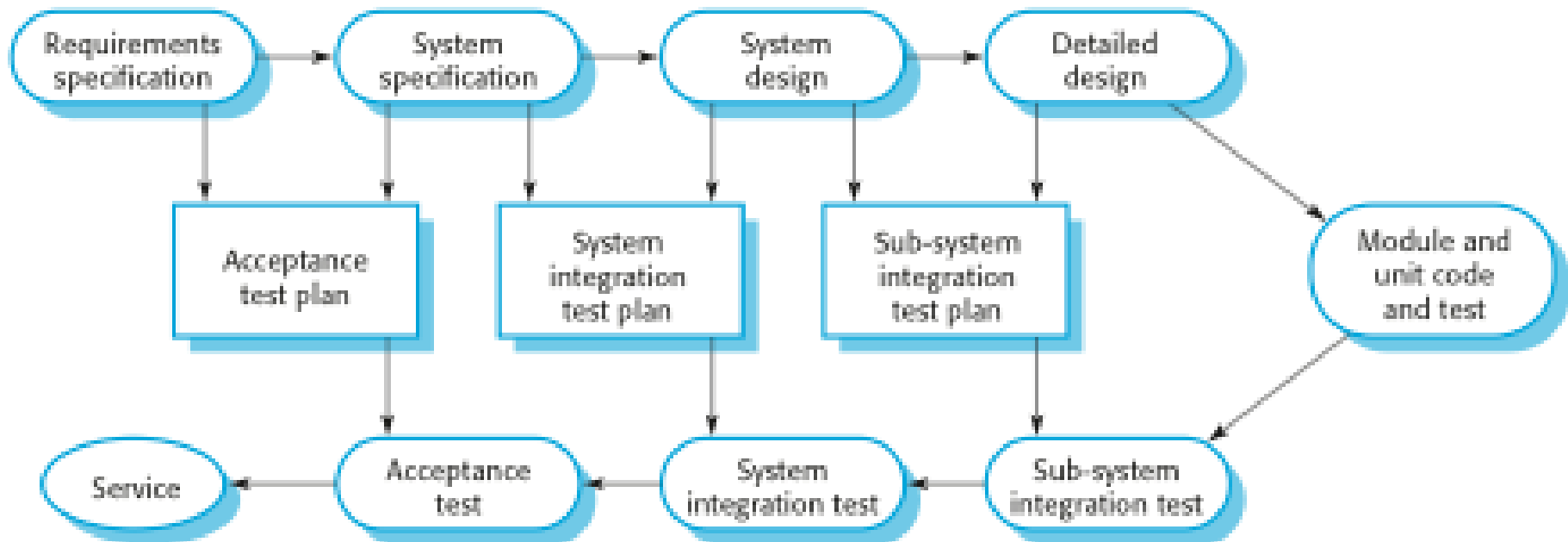
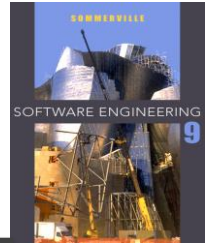
- concerned with finding errors that result from unanticipated interactions between components and component interface problems
- Testing the functional as well as non-functional requirements.
- **Testing of emergent properties (e.g. Reliability, Security, Repair ability )** is particularly important.

---

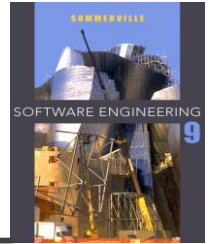
✧ Acceptance testing (sometimes called ‘**alpha testing**’.)

- Testing with **customer data** to check that the system meets the customer’s needs.
- May reveal errors and **omissions in the system requirements** definition
- may also reveal requirements problems where **the system’s facilities do not really meet the user’s needs** or the system performance is unacceptable.

# Testing phases in a plan-driven software process (the V-model of development)



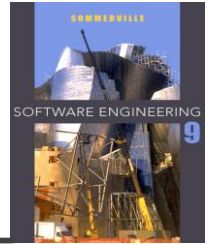
# Beta Testing



- ✧ When a system is to be marketed as a software product, a testing process called 'beta testing' is often used.
- ✧ Beta testing involves **delivering a system to a number of potential customers who agree to use that system.**
- ✧ **They report problems** to the system developers.
- ✧ This **exposes the product to real use** and detects errors that may not have been anticipated by the system builders.
- ✧ After this feedback, **the system is modified and released** either for further beta testing or for general sale.

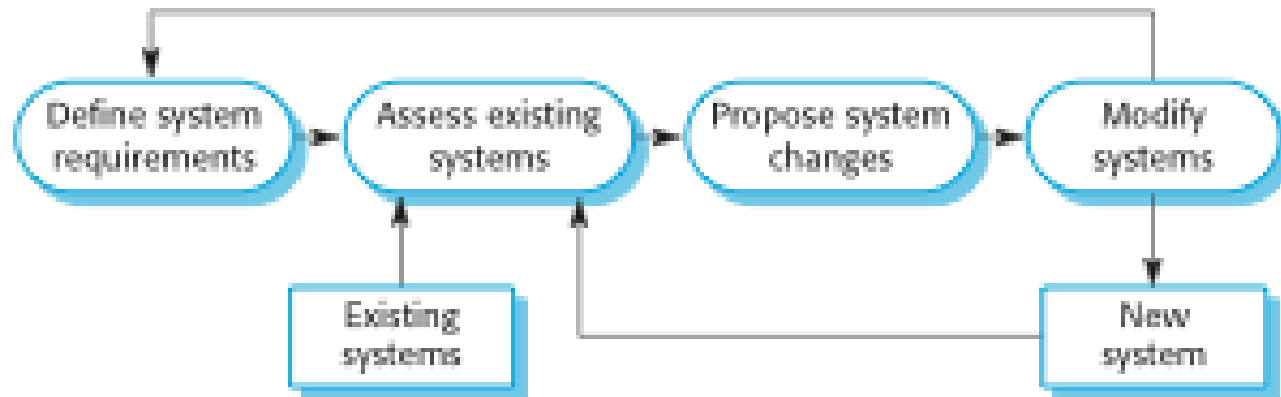
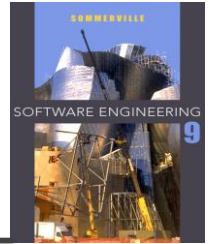
# Software evolution

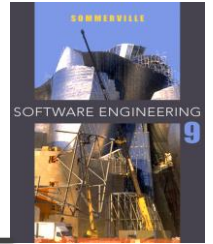
---



- ✧ Software is **inherently flexible and can change**.
- ✧ As requirements change through changing business circumstances, **the software that supports the business must also evolve and change**.
- ✧ Although there has been a **demarcation** between development and evolution (maintenance) this is increasingly irrelevant as **fewer and fewer systems are completely new**.
- ✧ the **costs of maintenance** are often several times the initial development costs,

# System evolution

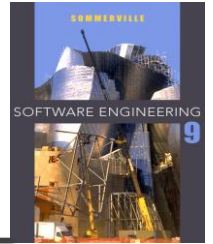




## Key points

---

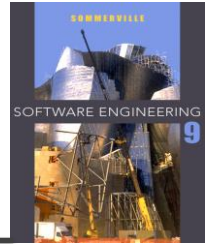
- ✧ **Software processes** are the **activities involved** in producing a software system. Software process models are abstract representations of these processes.
- ✧ General **process models describe the organization of software processes.**
- ✧ **Examples** of these general models include the **‘waterfall’ model, incremental development, and reuse-oriented development.**



## Key points

---

- ✧ **Requirements engineering** is the process of developing a software specification.
- ✧ **Design and implementation** processes are concerned with **transforming a requirements specification into an executable software system.**
- ✧ **Software validation** is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- ✧ **Software evolution** takes place when you change existing software systems to meet new requirements. **The software must evolve to remain useful.**



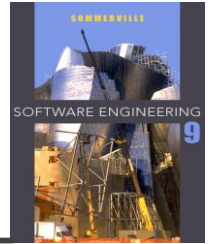
---

# Chapter 2 – Software Processes

## Lecture 2

# Coping with change

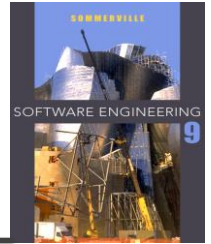
---



- ✧ **Change is inevitable** in all large software projects.
  - **Business changes** lead to new and changed system requirements
  - **New technologies** open up new possibilities for improving implementations
  - **Changing platforms** require application changes
- ✧ Change leads to rework so the **costs of change include both rework (e.g. re-analysing requirements) as well as the costs of implementing new functionality**

# Reducing the costs of rework

---



- 1. Change avoidance**, where the software process includes activities that can **anticipate possible changes** before significant rework is required.
  - For example, a **prototype system** may be developed to show some key features of the system to customers.
  - They can experiment with the prototype and refine their requirements before committing to high software production costs.
- 2. Change Tolerance**

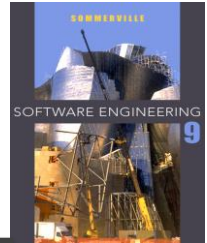
# Reducing the costs of rework

---

- ✧ **Change tolerance**, where the process is designed so that **changes can be accommodated** at relatively low cost.
  - This normally **involves some form of incremental development**.
  - **Proposed changes may be implemented in increments that have not yet been developed**.
  - **If this is impossible, then only a single increment** (a small part of the system) may have be altered to incorporate the change.
  
- ✧ The notion of **refactoring**, namely improving the structure and organization of a program, is also an important mechanism that **supports change tolerance** (see chap3 on Agile development)

# Two ways of coping with change and changing system requirements

---



## ✧ System prototyping

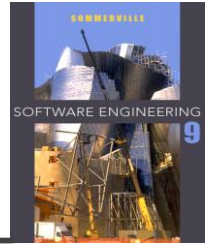
- **supports change avoidance** as it allows users to experiment with the system before delivery and so refine their requirements.
- The number of requirements change proposals made after delivery is therefore likely to be reduced.

## ✧ Incremental delivery

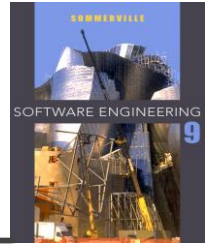
- system increments are delivered to the customer for comment and experimentation
- supports both **change avoidance and change tolerance**
- It **avoids the premature commitment to requirements** for the whole system and **allows changes to be incorporated into later increments at relatively low cost.**

# Software prototyping

---



- ✧ A prototype is an initial version of a system **used to demonstrate concepts and try out design options**, and **find out more about the problem** and its possible solutions.
  
- ✧ A prototype can be **used in:**
  - The **requirements engineering** process to help with requirements elicitation and validation of system requirements;
  - In **design processes** to **explore options** and **develop a UI** design (to **check the feasibility** of a proposed design or to check that it **supports efficient data access**);
  - In the **testing process** to run back-to-back tests.

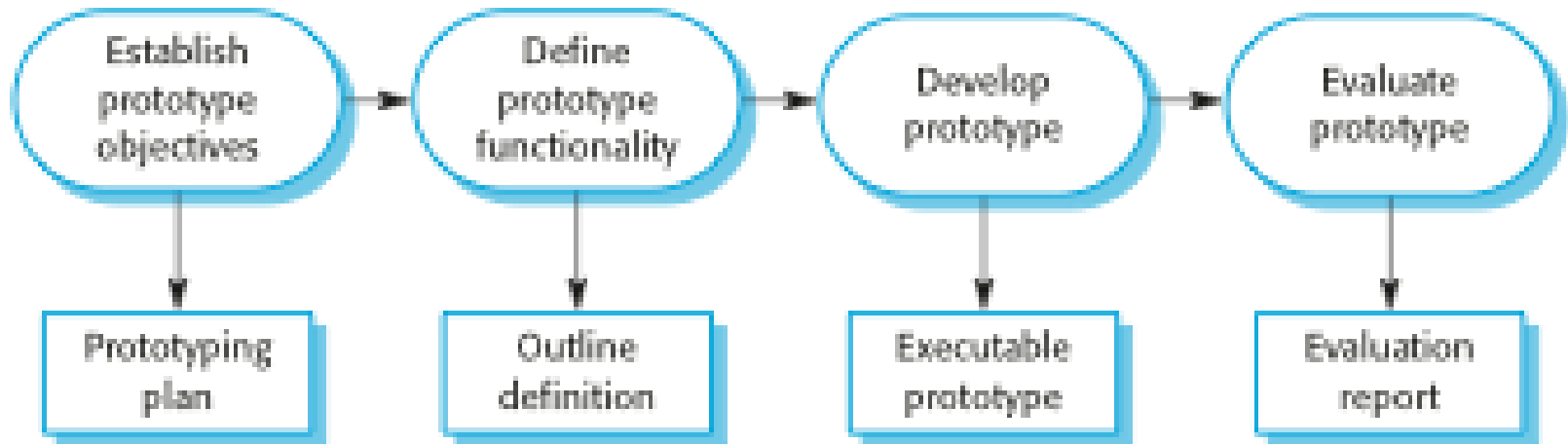
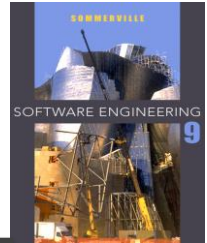


# Benefits of prototyping

---

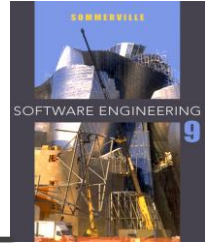
- ✧ Improved system usability.
  - Prototyping is also an essential part of the **user interface** design process.
- ✧ A closer match to users' real needs.
  - A prototype may **reveal errors and omissions** in the requirements that have been proposed.
- ✧ Improved design quality.
  - For example, a **database design** may be prototyped and tested to check that it supports efficient data access for the most common user queries.
- ✧ Improved maintainability.
- ✧ Reduced development effort.

# The process of prototype development

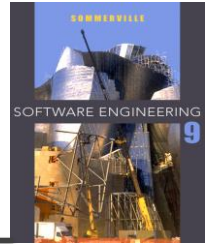


# Prototype development

---



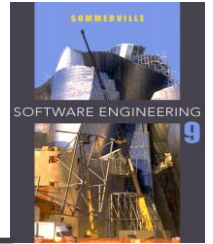
- ✧ May be based on rapid prototyping languages or tools
- ✧ May involve leaving out functionality
  - Prototype should **focus on areas of the product that are not well-understood**;
  - **Error checking and recovery may not be included in the prototype**;
  - **Focus on functional rather than non-functional requirements** such as reliability and security



# Throw-away prototypes

---

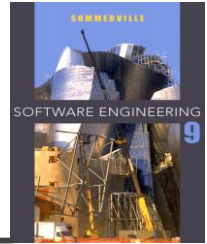
- ✧ Prototypes **should be discarded** after development as they are not a good basis for a production system:
- ✧ Developers are sometimes **pressured by managers to deliver throwaway prototypes**, particularly when there are delays in delivering the final version of the software.
- ✧ However, **this is usually unwise**:
  - It may be **impossible to tune** the system to meet **non-functional requirements**;
  - Prototypes are **normally undocumented**;
  - The prototype **structure is usually degraded** through rapid change;
  - The prototype probably will **not meet normal organizational quality standards**.



# Paper-based mock-ups Prototypes

---

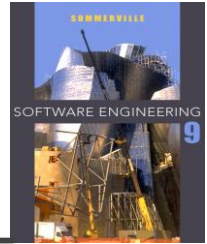
- ✧ Prototypes **do not have to be executable** to be useful. **Paper-based mock-ups of the system user interface** (Rettig, 1994) can be effective in helping users refine an interface design and work through usage scenarios.
- ✧ These are **very cheap to develop** and can be constructed **in a few days**.
- ✧ An extension of this technique is a **Wizard of Oz prototype** where only the user interface is developed.
- ✧ Users interact with this interface but **their requests are passed to a person** who interprets them and outputs the appropriate response.



# Incremental delivery

---

- ✧ Rather than deliver the system as a single delivery, the development and delivery is **broken down** into increments with **each increment delivering part of the required functionality**.
- ✧ **User requirements are prioritised** and the highest priority requirements are included in early increments.
- ✧ Once the development of an increment is started, **the requirements are frozen** though **requirements for later increments can continue to evolve**.
- ✧ Once an increment is completed and delivered, **customers can put it into service**
- ✧ **Unlike prototypes, increments are part of the real system**



# Incremental development and delivery

---

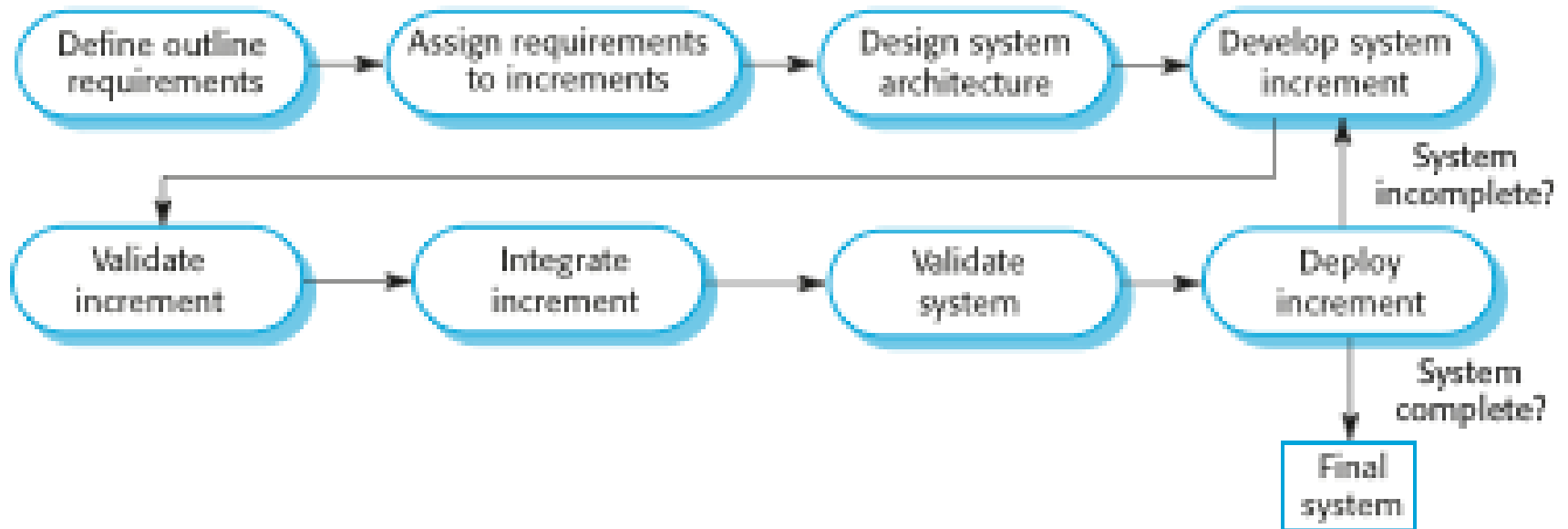
## ✧ Incremental development

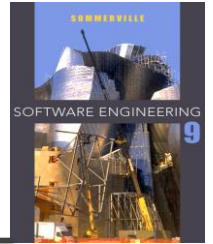
- Develop the system in increments and **evaluate each increment before proceeding** to the development of the next increment;
- Normal approach **used in agile methods**;
- **Evaluation done by user/customer** proxy.

## ✧ Incremental delivery

- Deploy an increment for use by end-users;
- **More realistic evaluation** about practical use of software;
- **Difficult to implement for replacement systems (i.e. to replace an existing system)** as increments have less functionality than the system being replaced.

# Incremental delivery

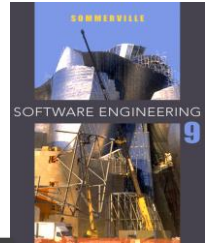




# Incremental delivery advantages

---

- ✧ Customer value can be delivered with each increment so **system functionality is available earlier.**
- ✧ **Early increments act as a prototype** to help elicit requirements for later increments but unlike prototypes, these are part of the real system.
- ✧ **Lower risk** of overall project failure.
- ✧ **The highest priority system** services tend to **receive the most testing.**

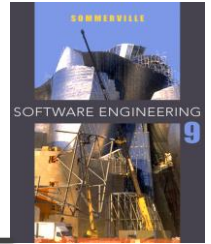


# Incremental delivery problems

---

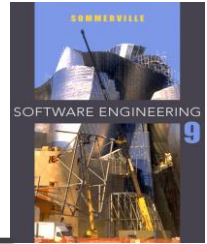
- ✧ Most systems require a **set of basic facilities** that are used by different parts of the system.
  - As requirements are not defined in detail until an increment is to be implemented, **it can be hard to identify common facilities that are needed by all increments.**
- ✧ Users want all of the functionality of **the old system** and are often **unwilling to experiment with an incomplete new system.** Therefore, **getting useful customer feedback is difficult.**
- ✧ The essence of iterative processes is that the **specification is developed in conjunction with the software.**
  - However, **this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.** In the incremental approach, there is **no complete system specification until the final increment is specified.**

# The spiral model



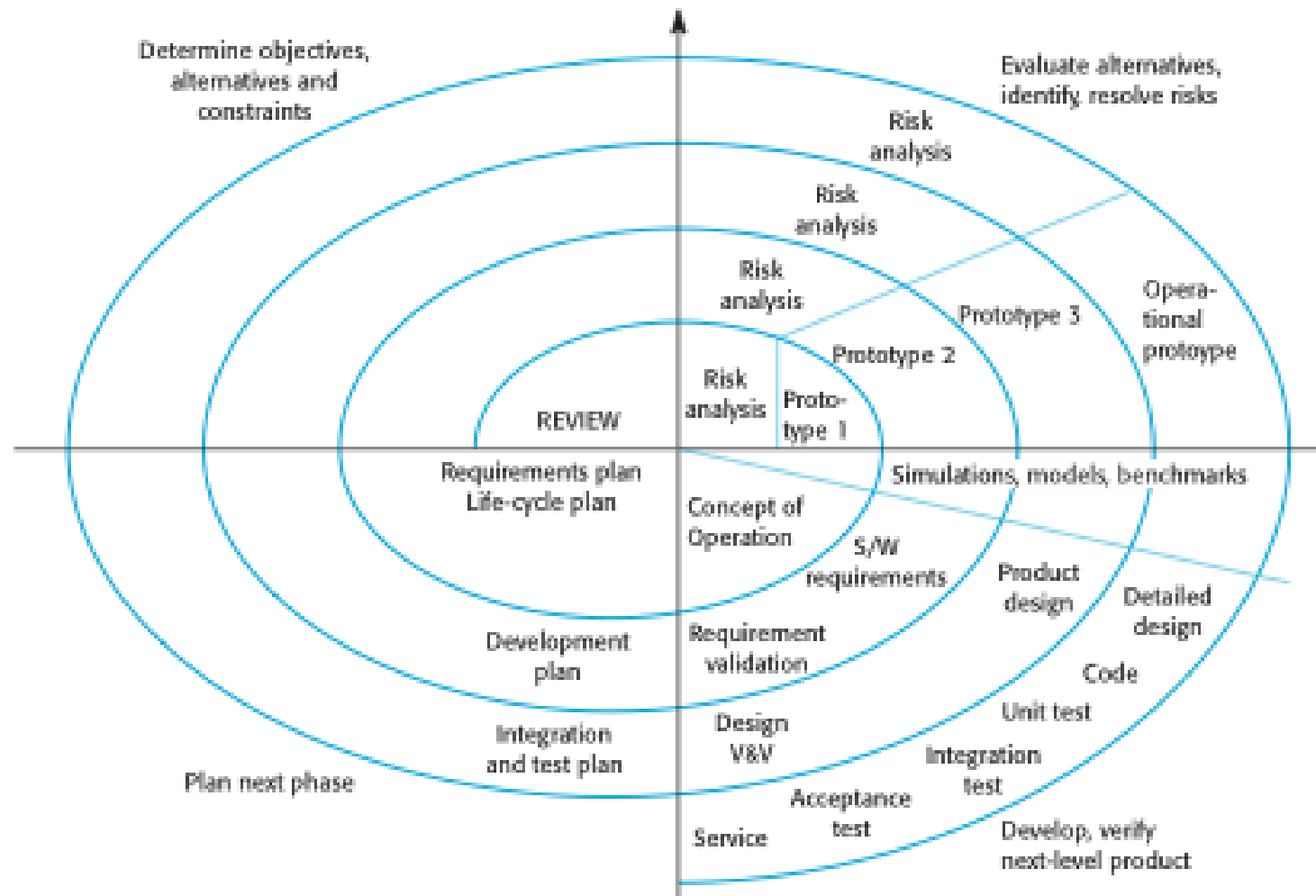
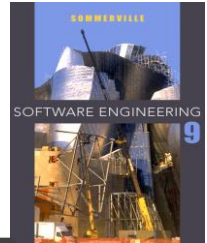
- ✧ Here, the software **process is represented as a spiral**, rather than a sequence of activities with some **backtracking** from one activity to another.
- ✧ Each loop in the spiral **represents a phase of the software process.**
- ✧ Thus, **the innermost loop** might be **concerned with system feasibility**, the **next loop with requirements definition**, the **next loop with system design**, and so on.
- ✧ **No fixed phases such as specification or design** - loops in the spiral are chosen **depending on what is required.**

# The spiral model

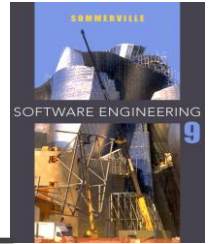


- ✧ The spiral model **combines change avoidance with change tolerance.**
- ✧ The main difference between the spiral model and other software process models is its **explicit recognition of risk.**
- ✧ This model of development **combines the features of the prototyping and the waterfall model.**
- ✧ The spiral model is **intended for large, expensive and complicated projects.**
- ✧ It allows for **incremental releases** of the product, or incremental refinement through each time around the spiral.
- ✧ The spiral model also explicitly includes **risk**

# Boehm's spiral model of the software process



- 
- ✧ Starting at the center, each turn around the spiral goes through several task regions :
- **Determine the objectives**, alternatives, and constraints on the new iteration.
  - **Evaluate alternatives** and identify and **resolve risk issues**.
  - **Develop and verify** the product for this iteration.
  - **Plan the next iteration**.



# Spiral model sectors

---

## ✧ Objective setting

- Specific objectives for the phase are identified.

## ✧ Risk assessment and reduction

- Risks are assessed and activities put in place to reduce the key risks.

## ✧ Development and validation

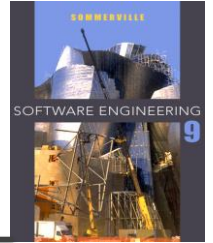
- A development model for the system is chosen which can be any of the generic models.

## ✧ Planning

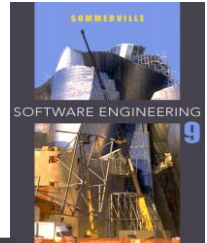
- The project is reviewed and the next phase of the spiral is planned.

# Spiral model usage

---



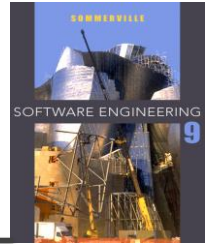
- ✧ Spiral model has been very influential in helping people think about **iteration in software** processes and **introducing the risk-driven approach** to development.
- ✧ In practice, however, the model is **rarely used as published** for practical software development.



# The Rational Unified Process

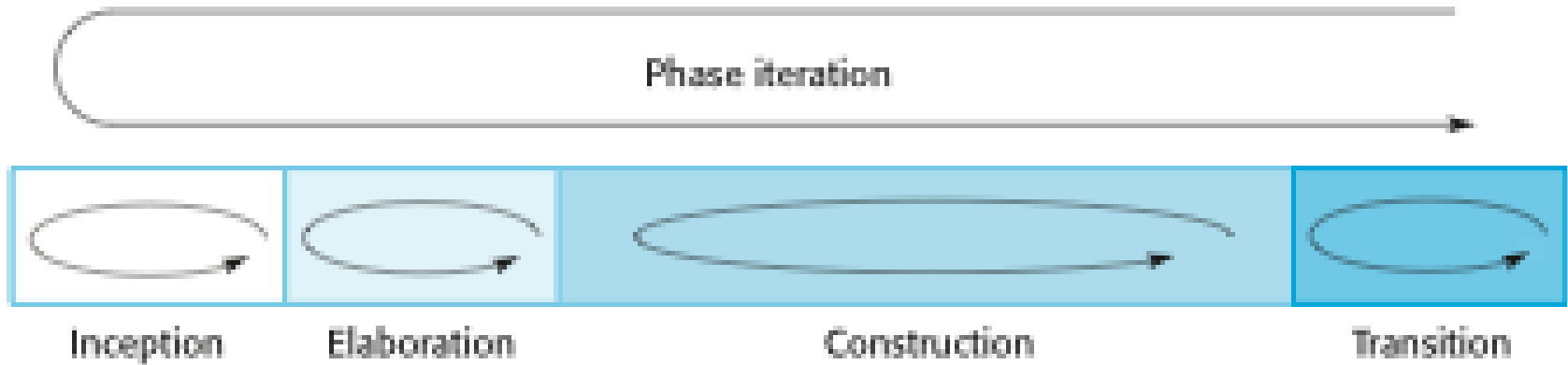
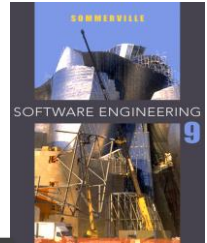
---

- ✧ A modern generic process **derived from the work on the UML and associated process.**
- ✧ A good example of a **hybrid process model.**
- ✧ It brings together **elements from all of the generic process models**, illustrates **good practice in specification and design** and **supports prototyping and incremental delivery.**
- ✧ Normally described from **3 perspectives**
  - A **dynamic perspective** that shows phases over time;
  - A **static perspective** that shows process activities;
  - A **practice perspective** that suggests good practice



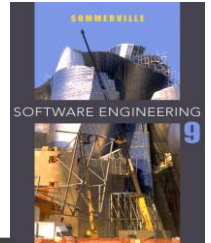
- 
- ✧ the RUP is a **phased model** that identifies four discrete phases in the software process.
  - ✧ However, **unlike the waterfall** model where phases are equated with process activities, the phases in **the RUP are more closely related to business rather than technical concerns.**

# Phases in the Rational Unified Process (Dynamic Perspective)



# RUP phases

---



## ✧ Inception

- **Establish the business case for the system.**
- You should **identify all external entities** (people and systems) that will interact with the system and define these interactions.
- You then use this information **to assess the contribution** that the system makes to the business.
- **If this contribution is minor, then the project may be cancelled** after this phase.

## ✧ Elaboration

- Develop an **understanding of the problem domain** and the **system architecture**.
- develop **the project plan**,
- and identify **key project risks**.
- On completion of this phase you should have
  - a **requirements model** for the system, which may be a set of **UML use-cases**,
  - **an architectural description**,
  - And a **development plan for the software**.

---

## ✧ Construction

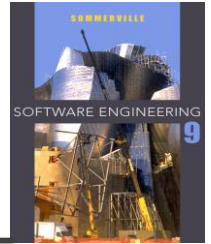
- System design, programming and testing.
- **On completion of this phase, you should have a working software system and associated documentation that is ready for delivery to users.**

## ✧ Transition

- Deploy the system (move it to the user community)
- This is something that is **ignored in most software process models** but is, in fact, an **expensive and sometimes problematic activity.**
- **On completion of this phase, you should have a documented software system that is working correctly in its operational environment.**

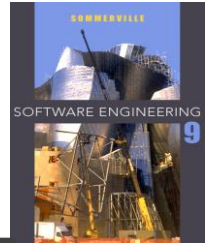
# Inception

---



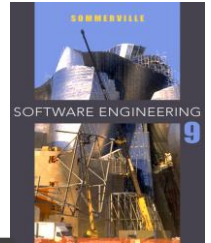
- ✧ In this phase the business case which includes business context, success factors (expected revenue, market recognition, etc.), and financial forecast is established.
- ✧ To complement the business case, a basic use case model, project plan, initial risk assessment and project description (the core project requirements, constraints and key features) are generated.

# Elaboration Phase



- ✧ The elaboration phase is where the project starts to take shape.
- ✧ In this phase the problem domain analysis is made and the architecture of the project gets its basic form.
- ✧ **Outcomes**
  - ✧ **A use-case model** in which the use-cases and the actors have been identified and most of the use-case descriptions are developed. The use-case model should be 80% complete.
  - ✧ A description of the software architecture in a software system development process.
  - ✧ Business case and risk list which are revised.
  - ✧ A **development plan** for the overall project.
  - ✧ **Prototypes** that demonstrably mitigate each identified technical risk.
  - ✧ A preliminary user manual (optional)

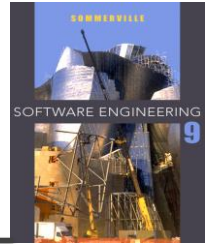
# Construction Phase



- ✧ In this phase, the main focus is on the **development of components** and other features of the system.
- ✧ This is the phase when the **bulk of the coding** takes place.
- ✧ In larger projects, **several construction iterations** may be developed in an effort to divide the use cases into manageable segments that produce demonstrable prototypes.
- ✧ This phase produces the **first external release** of the software.

# Transition Phase

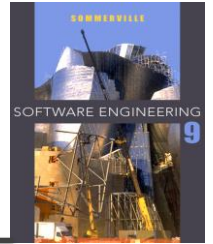
---



- ✧ The primary objective is to **'transit' the system** from development into production, **making it available to and understood by the end user**
- ✧ The activities of this phase include **training the end users** and **maintainers and beta testing** the system to validate it against the end users' expectations.
- ✧ The product is also **checked against the quality level set in the Inception phase.**

# RUP iteration

---



✧ Iteration within the RUP is supported in two ways.

- In-phase iteration

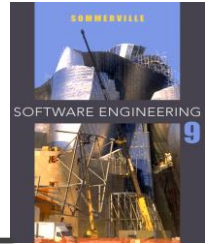
- Each phase **is iterative** with results developed incrementally.

- Cross-phase iteration

- As shown by the loop in the RUP model, the whole set of phases may be enacted **incrementally**.

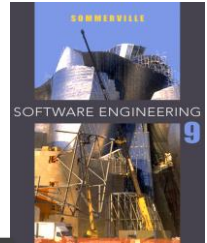
# The static view of the RUP

---



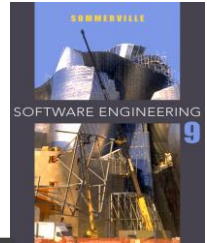
- ✧ focuses on the activities that take place during the development process.
- ✧ These are called **workflows** in the RUP description.
- ✧ There are **six core process** workflows identified in the process and **three core supporting work-flows**.
- ✧ The RUP has **been designed in conjunction with the UML**, so the workflow description is **oriented around associated UML models** such as **sequence models, object models**

# Static workflows in the Rational Unified Process



Workflow	Description
Business modelling	The business processes are modelled using business <b>use cases</b> .
Requirements	<b>Actors</b> who interact with the system <b>are identified</b> and <b>use cases are developed</b> to model the system requirements.
Analysis and design	A design model is created and documented using <b>architectural models, component models, object models and sequence models</b> .
Implementation	The components in the system are implemented and structured into implementation sub-systems. <b>Automatic code generation</b> from design models helps accelerate this process.

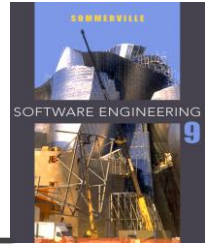
# Static workflows in the Rational Unified Process



Workflow	Description
Testing	Testing is <b>an iterative process</b> that is carried <b>out in conjunction with implementation</b> . System testing follows the completion of the implementation.
Deployment	<b>A product release is created, distributed to users and installed in their workplace.</b>
Configuration and change management	This supporting workflow <b>managed changes to the system</b> (see Chapter 25).
Project management	This supporting workflow <b>manages the system development</b> (see Chapters 22 and 23).
Environment	This workflow is concerned with <b>making appropriate software tools available</b> to the software development team.

# The advantage in presenting dynamic and static views

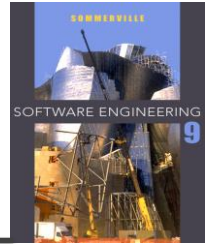
---



- ✧ is that phases of the development process are not associated with specific workflows.
- ✧ In principle at least, all of the RUP workflows may be **active at all stages of the process.**
- ✧ **In the early phases** of the process, most effort will probably be spent on workflows such as business modeling and requirements and,
- ✧ **in the later phases**, in testing and deployment

# RUP good practice

---



## ✧ Develop software iteratively

- Plan increments based on customer priorities and deliver highest priority increments first.

## ✧ Manage requirements

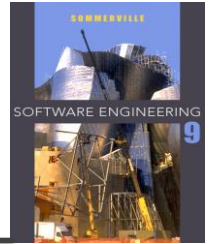
- Explicitly document customer requirements and keep track of changes to these requirements.
- Analyze the impact of changes on the system before accepting them.

## ✧ Use component-based architectures

- Organize the system architecture as a set of reusable components.

# RUP good practice

---



## ✧ Visually model software

- Use graphical UML models to present static and dynamic views of the software.

## ✧ Verify software quality

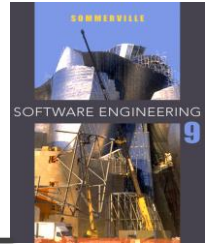
- Ensure that the software meet's organizational quality standards.

## ✧ Control changes to software

- Manage software **changes using a change management system and configuration management tools.**

# Key points

---



- ✧ Processes should include activities to cope with change. This may involve a prototyping phase that helps avoid poor decisions on requirements and design.
- ✧ Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.
- ✧ The Rational Unified Process is a modern generic process model that is organized into phases (inception, elaboration, construction and transition) but separates activities (requirements, analysis and design, etc.) from these phases.