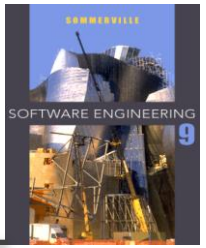
CS 3323



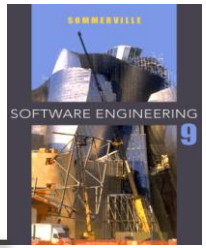
Chapter 2 – Software Processes

Ian Sommerville,

Software Engineering, 9th Edition

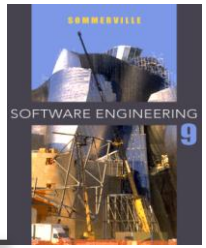
Pearson Education, Addison-Wesley

Topics covered



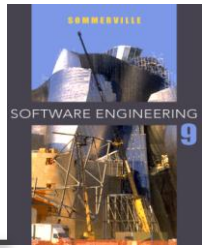
- Basic software process models (1, 2, 3)
- Process activities
- Coping with change & additional software process models (4, 5)
- The Rational Unified Process (6)
  - An example of a modern software process.

The software process



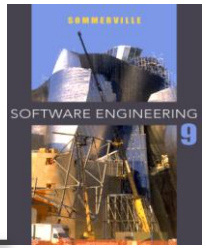
- ***Software process***:
  - a structured set of activities required to develop a software system. (SommerVille)
  - a framework for the activities, actions, and tasks that are required to build high-quality software. (Pressman)
  - Software Process defines the approach that is taken as software is engineered.
  - Is not equal to software engineering, which also encompasses **technologies** that populate the process– technical methods and automated tools.

The software process



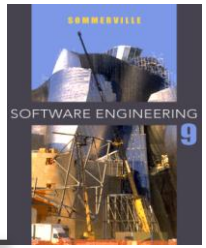
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.

Software process descriptions



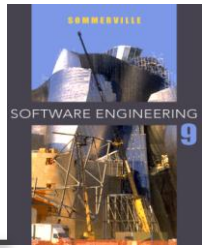
- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
- Process descriptions may also include:
  - Products, which are the outcomes of a process activity;
  - 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



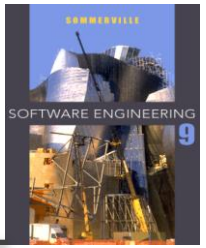
- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

Process model



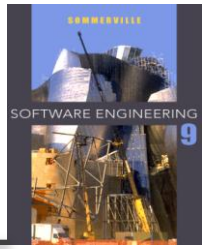
- A *software process model* is an abstract representation of a process. It presents a description of a process from some particular perspective.

What / who / why is Process Models?



- **What**: Go through a series of predictable steps--- a road map that helps you create a timely, high-quality results.
- **Who**: Software engineers and their managers, clients also. People adapt the process to their needs and follow it.
- **Why**: Provides stability, control, and organization to an activity that can if left uncontrolled, become quite chaotic. However, modern software engineering approaches must be agile and demand ONLY those activities, controls and work products that are appropriate.
- **What Work products**: Programs, documents, and data
- **What are the steps:** The process you adopt depends on the software that you are building. One process might be good for aircraft avionic system, while an entirely different process would be used for website creation.
- **How to ensure right**: A number of software process assessment mechanisms that enable us to determine the maturity of the software process. However, the quality, timeliness and long-term viability of the software are the best indicators of the efficacy of the process you use.

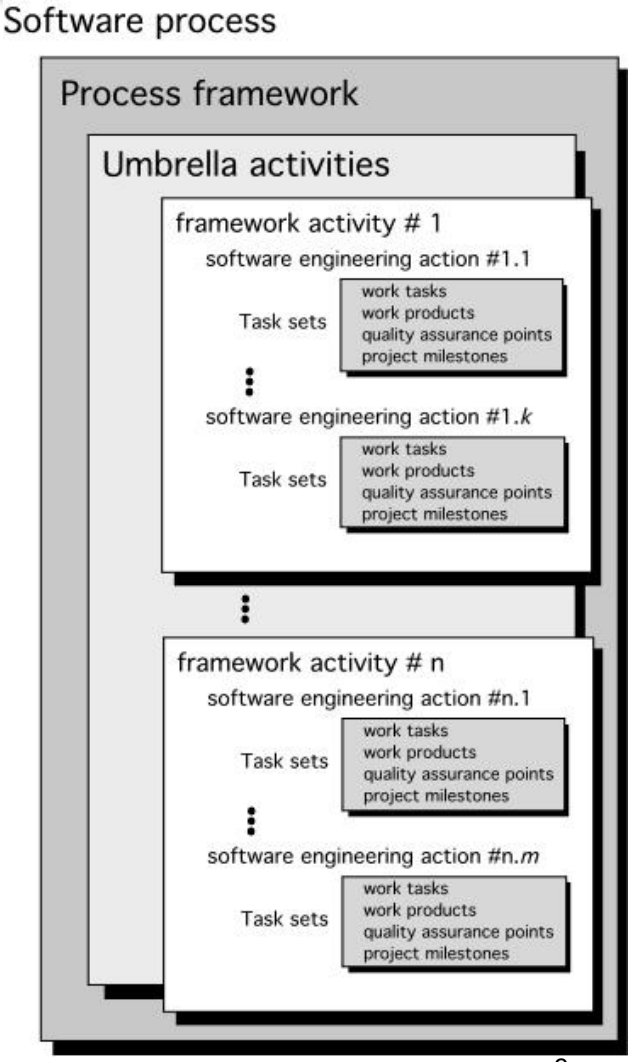
A Generic Process Model



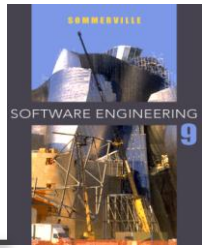
■As we discussed before, a generic process framework for software engineering defines five framework activities-communication, planning, modeling, construction, and deployment.

■In addition, a set of umbrella activities-project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.

■Next question is: how the framework activities and the actions and tasks that occur within each activity are organized with respect to sequence and time?



Software process models (SommerVille)



✧ 1 The waterfall model

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

✧ 2 Incremental (exploratory) development

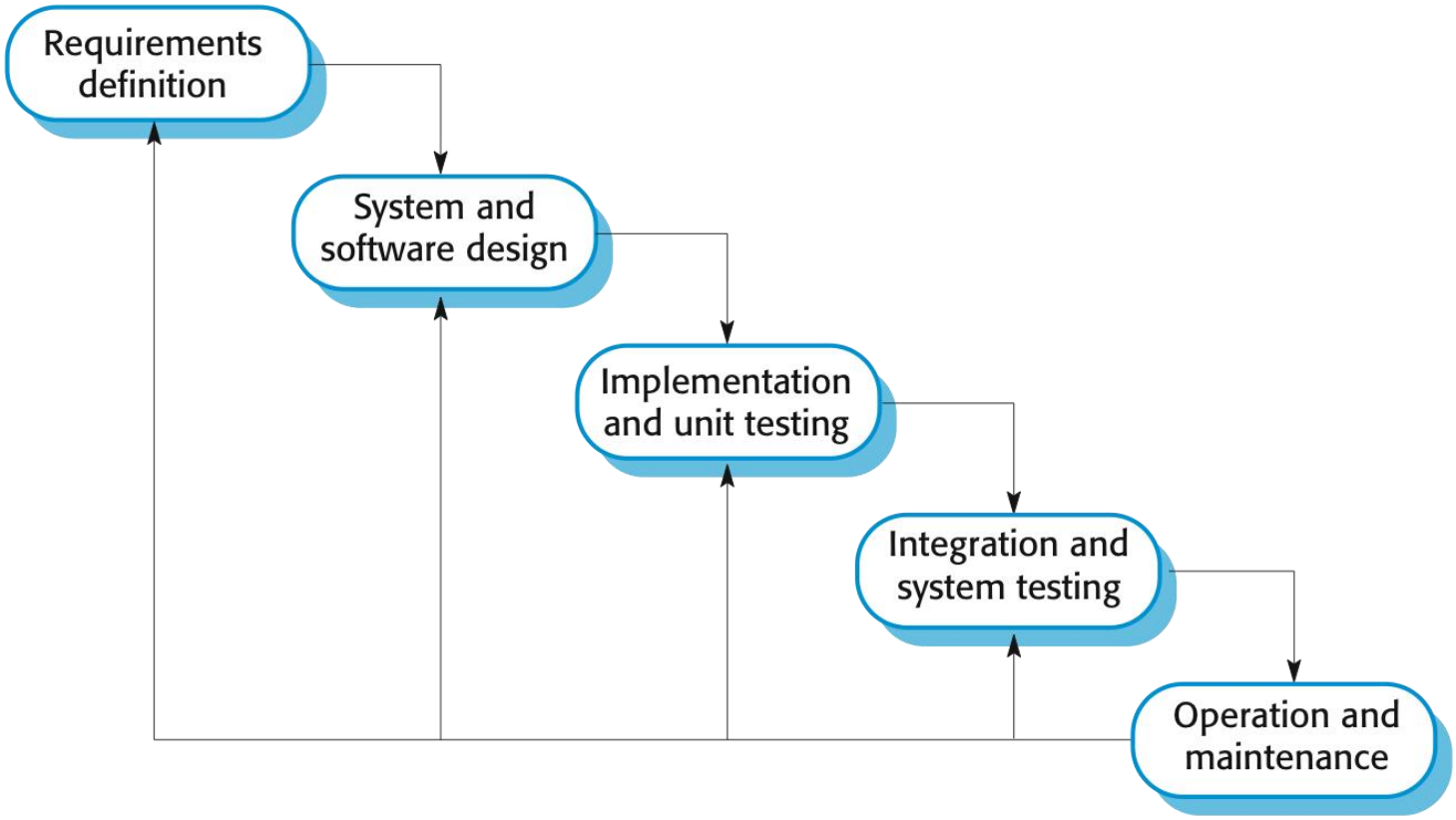
- Specification, development and validation are interleaved. May be plan-driven or agile.

✧ 3 Reuse-oriented software engineering

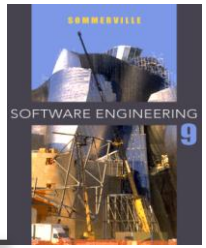
- The system is assembled from existing components. May be plan-driven or agile.

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

1 The waterfall model

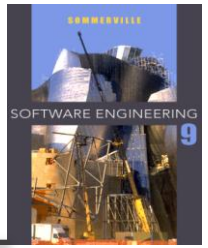


Waterfall model phases



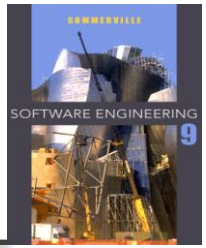
- There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

Waterfall model phases



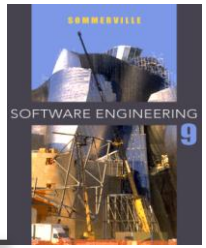
- Requirements analysis and definition
  - The system's services, constraints and goals are established by consultation with system users. They are then defined a detail system specification.
- System and software design
  - Systems design process partitions the requirements to either hardware or software systems. It establishes an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.
- Implementation and unit testing
  - Software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

Waterfall model phases



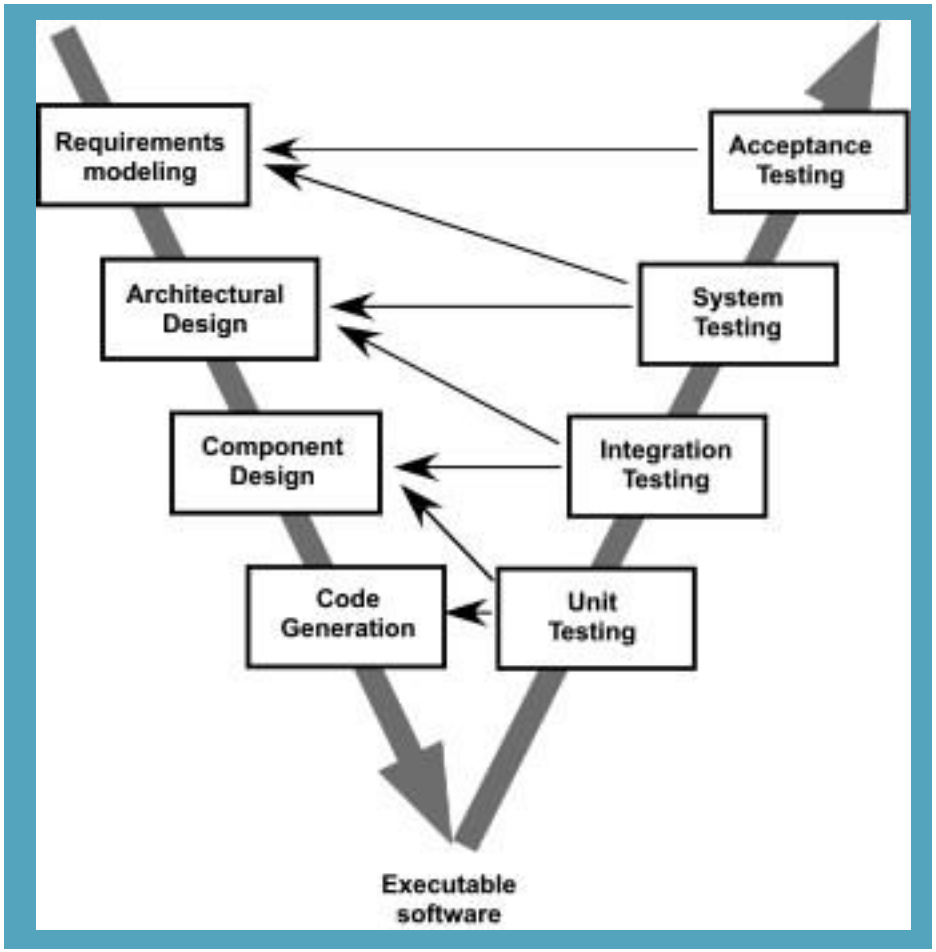
- ✧ Integration and system testing
  - Individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
- ✧ Operation and maintenance
  - This is longest life-cycle 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.

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 and changes will be fairly limited during the design process
  - Few business systems have stable requirements
- 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

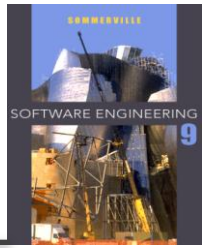
The V-Model



A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activates.

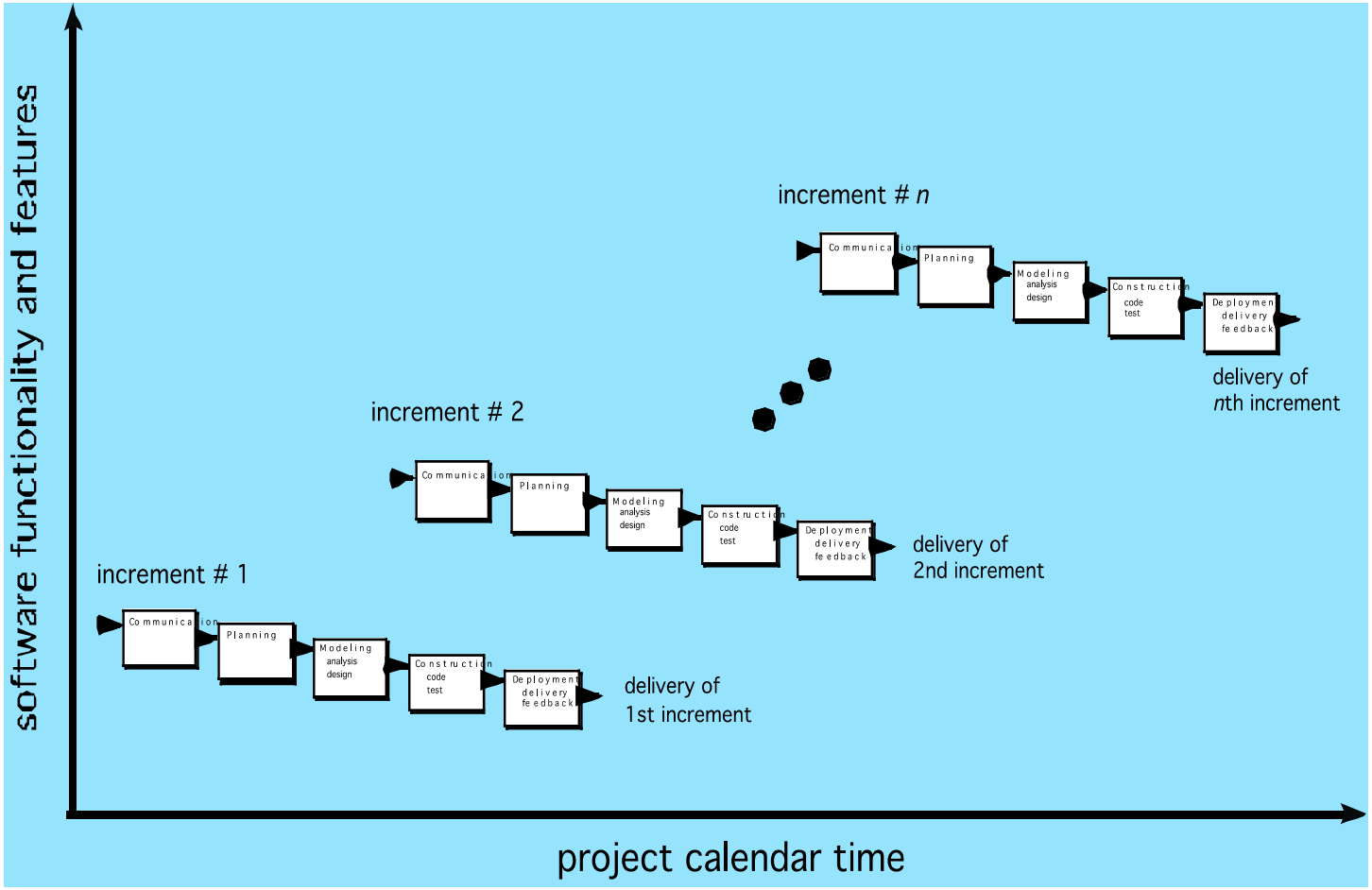
Team first moves down the left side of the V to refine the problem requirements. Once code is generated, the team moves up the right side of the V, performing a series of tests that validate each of the models created as the team moved down the left side.

2 The Incremental Model

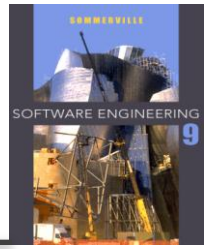


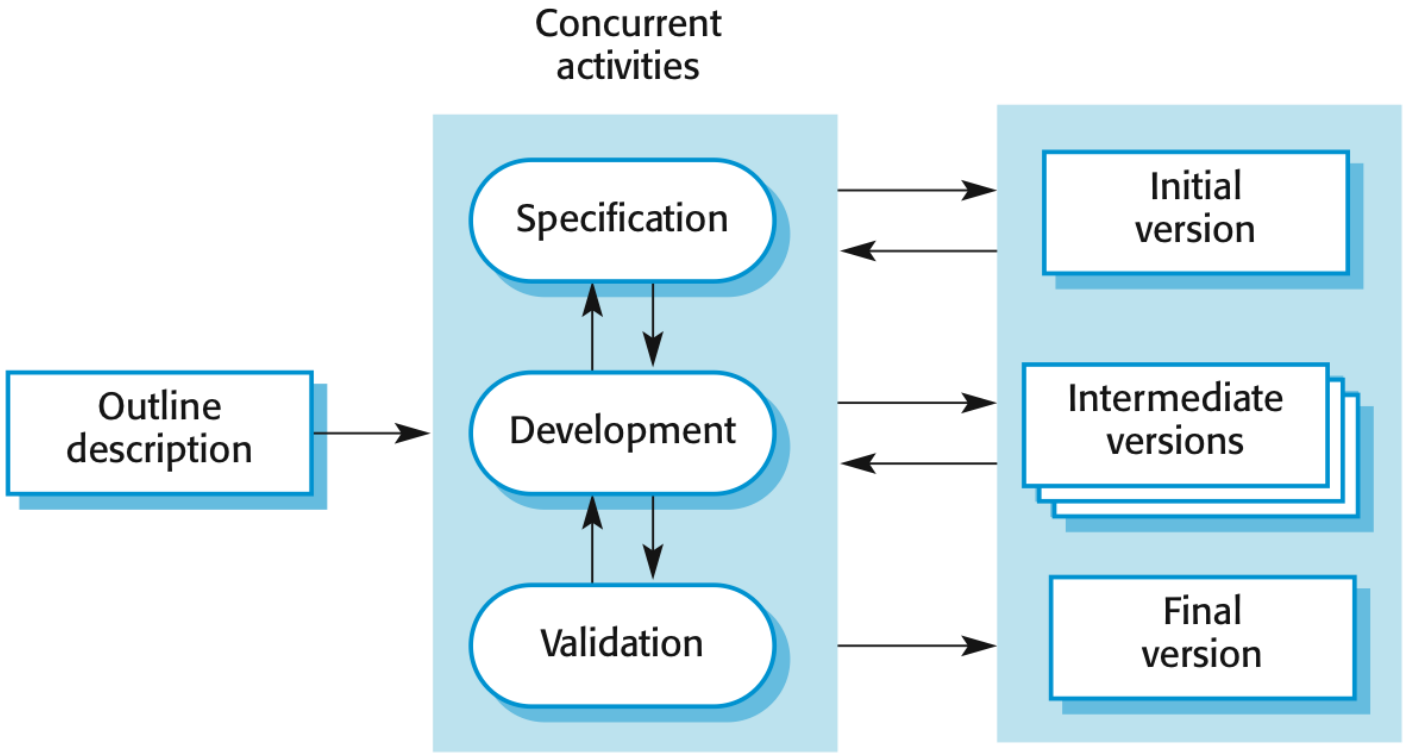
- When initial requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process. A compelling need to expand a limited set of new functions to a later system release.
- It combines elements of linear and parallel process flows. Each linear sequence produces deliverable increments of the software.
- The first increment is often a core product with many supplementary features. Users use it and evaluate it with more modifications to better meet the needs.

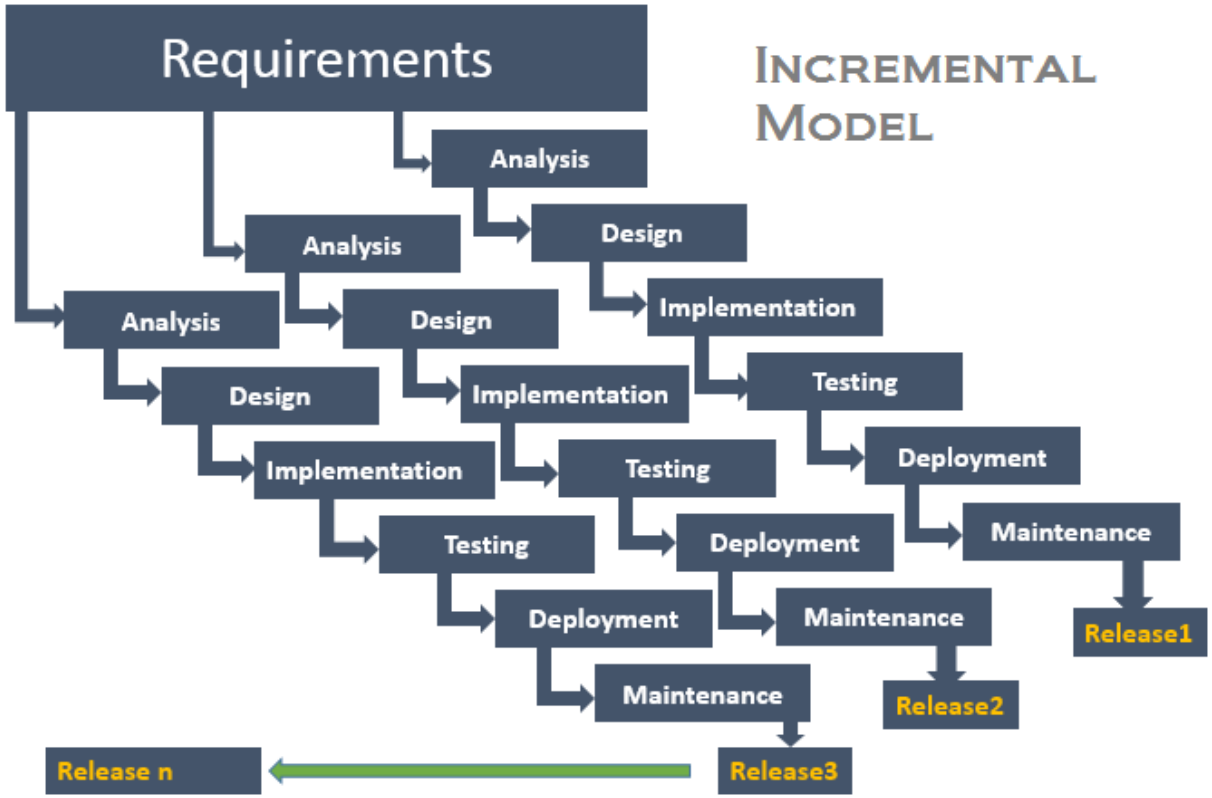
The Incremental Model (Pressman)



2 Incremental (exploratory) development

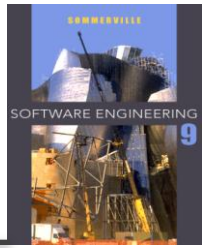






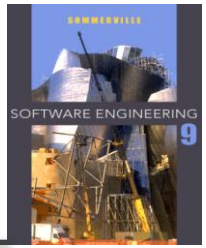
Incremental Model

Incremental development benefits



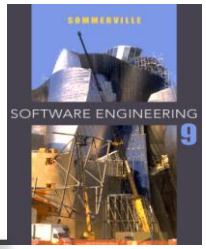
- The cost of accommodating changing customer requirements is reduced
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model
- It is easier to get customer feedback on the development work that has been done
  - Customers can comment 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



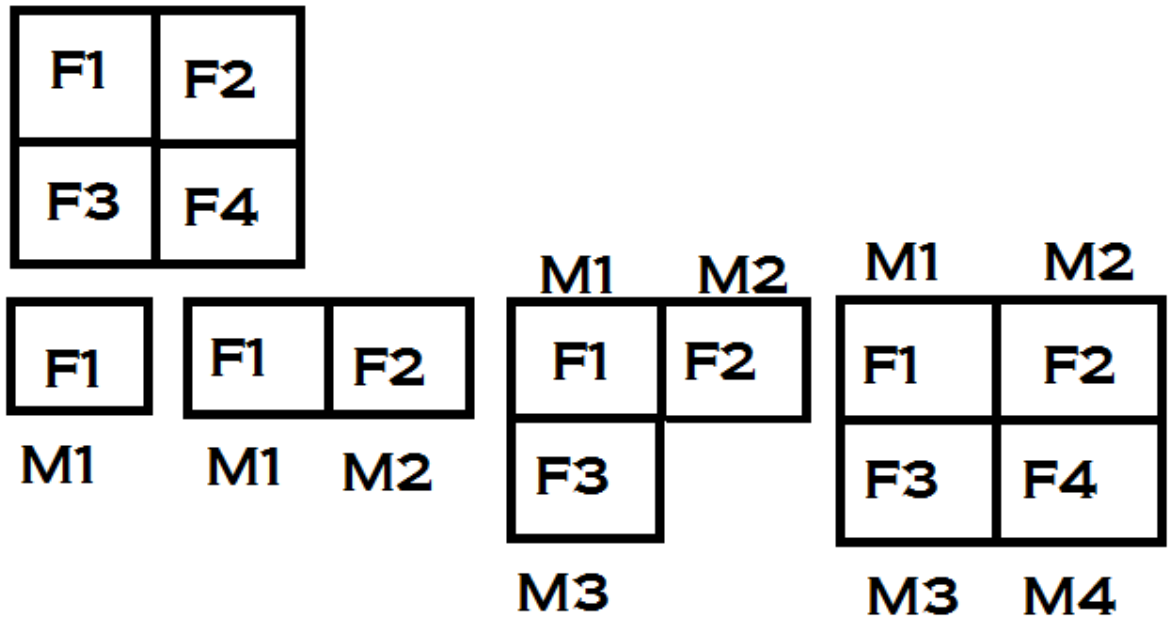
- The process is not visible
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added
  - Unless time and money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

Some case study

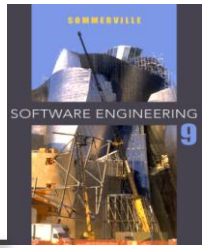


✧ Word Processing:

- Deliver in first increment (core product): file management, editing, and document production.
- more sophisticated editing and document production capabilities in the second increment.

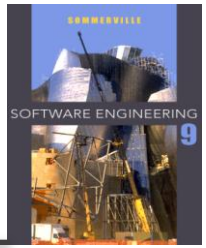


Practice guide



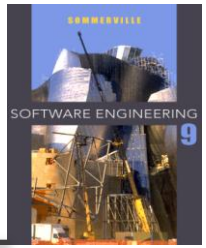
- For small and medium-sized systems (up to 500,000 lines of code). I think that the evolutionary approach is the best approach to development.
- For large, complex, long-life-time systems, where different teams develop different part of the system, it is difficult to establish a stable system architecture using this approach. Which makes it hard to integrate contributions from the teams.

Practice guide



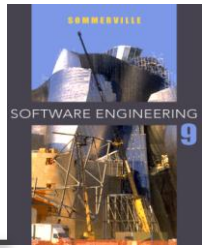
- For large systems, I recommend a mixed process that incorporates the best features of the waterfall and the evolutionary development models.
- This may involve developing a throwaway prototype to resolve uncertainties in the system specification, then re-implement the system using a more structured approach.
- Part of the system that are well understood can be specified and developed using a waterfall-based process.
- Other part of system, such as the user interface, which are difficult to specify in advance, should always be developed using an exploratory programming approach.

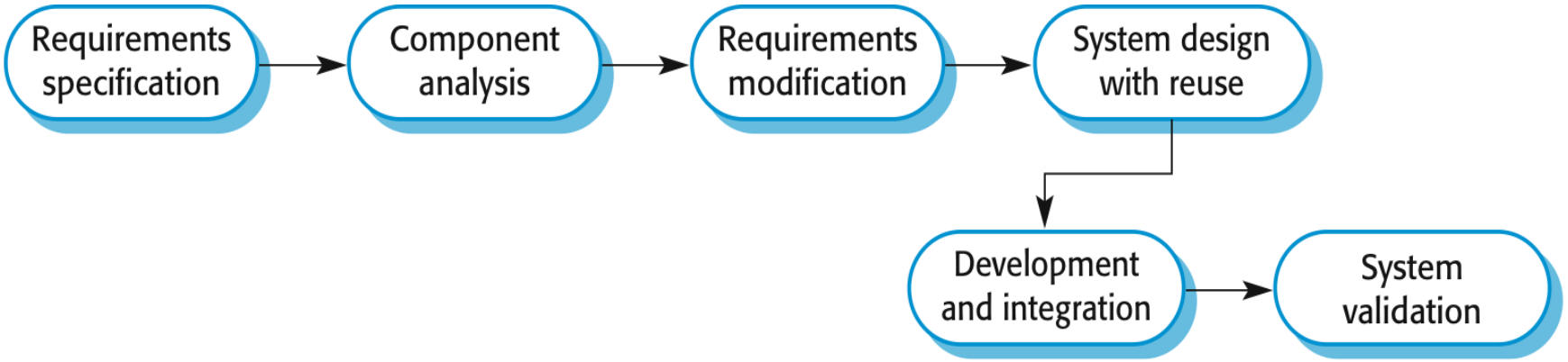
3 Reuse-oriented software engineering



- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems
- Process stages
  - Component analysis
  - Requirements modification
  - System design with reuse
  - Development and integration
- Reuse is now the standard approach for building many types of business system
  - Reuse is covered in more depth in Chapter 16

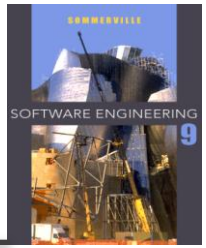
Reuse-oriented software engineering





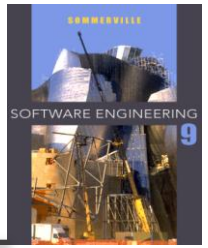
The initial requirements stage is similar to waterfall, but modification occurs later in the process based on what reusable components were discovered

Reuse-oriented Software Engineering Use



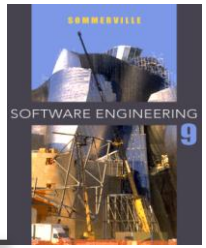
- There are three types of software components that may be used in the reuse-oriented process
  - Web services that are available for remote invocation
  - Collections of objects developed as a package, such as J2EE or .NET
  - Stand-alone software systems configured for use in a particular environment

Advantage and short-point of Component-based software engineering

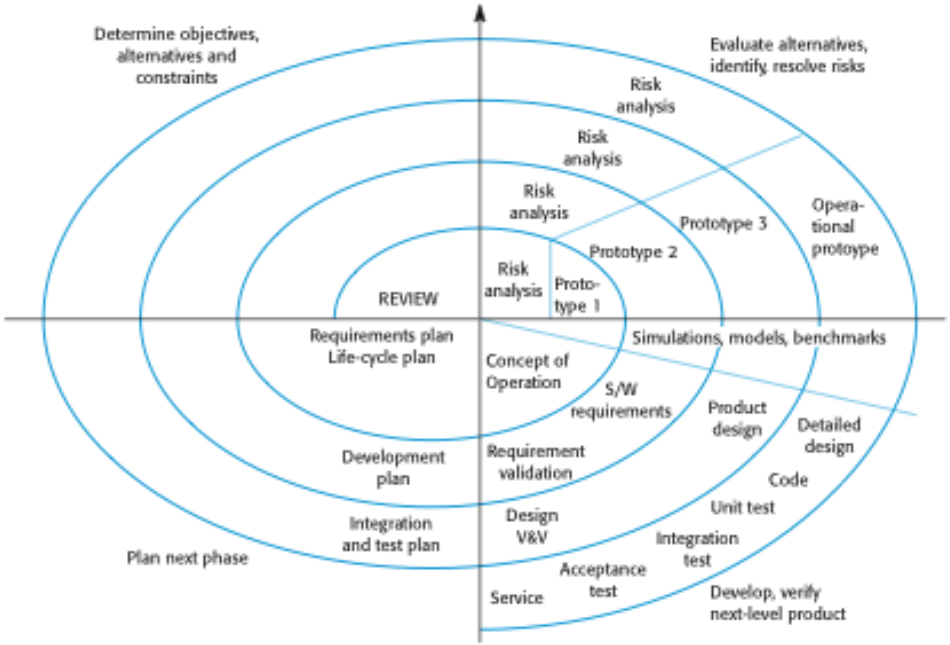


- ✧ Has obvious advantage of reducing the amount of software to be developed and so reducing cost and risks.
- ✧ Leads to faster delivery of the software.
- ✧ However, requirements compromises are inevitable and this may lead to a system that does not meet the real needs of users.
- ✧ Furthermore, some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

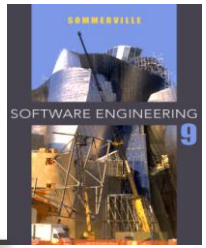
Spiral Model



- Developed by Barry Boehm of USC in 1988
- Process is represented as a spiral with no fixed phases
  - Loops are chosen based on what is required and could represent different parts of the process
  - Risks are explicitly assessed and resolved throughout the process

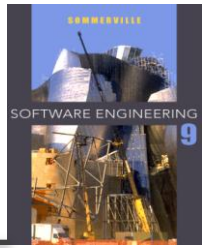


Process activities



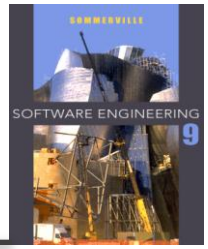
- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system
- The four basic process activities of specification, development, validation and evolution are organized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

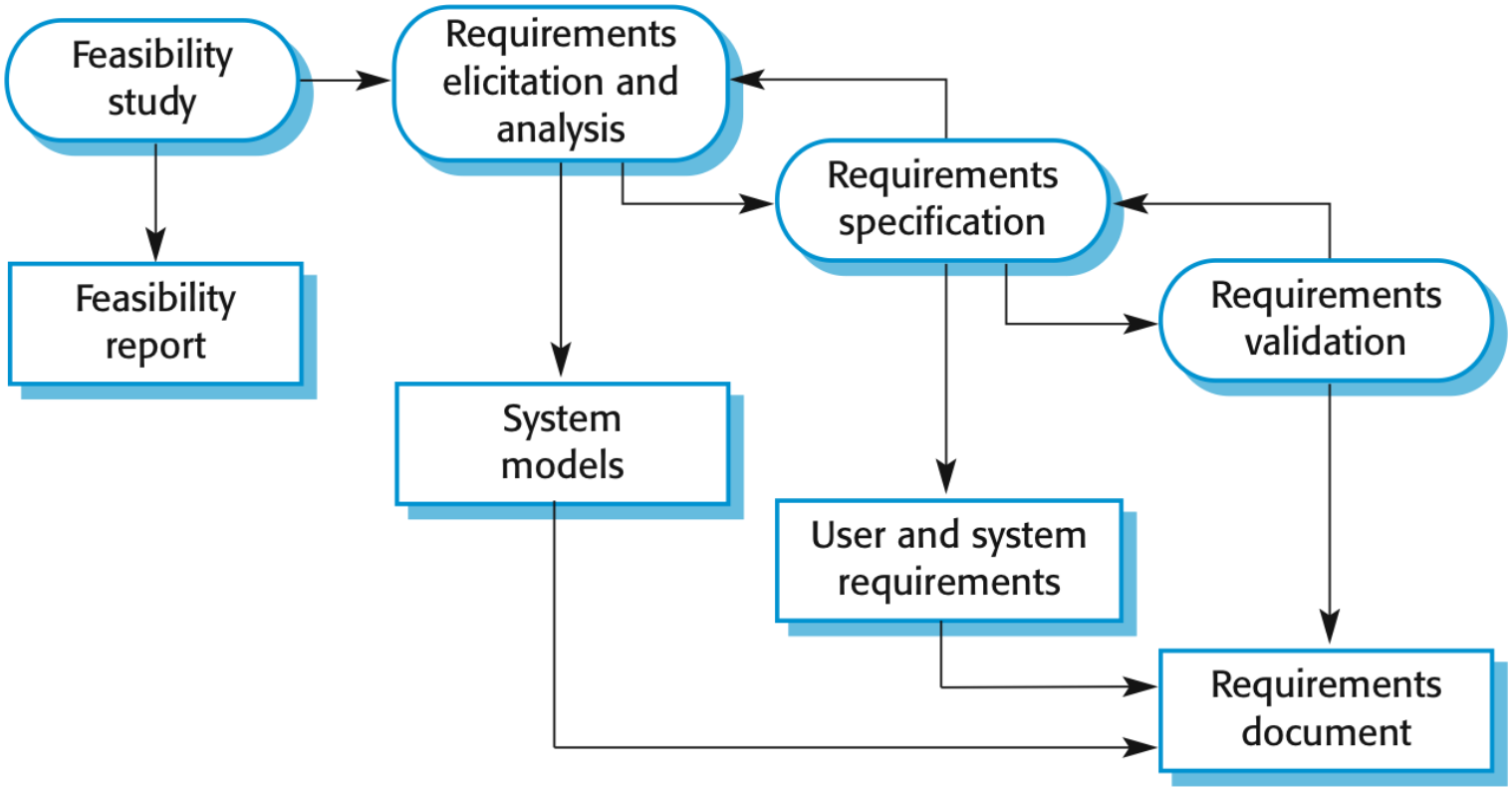
Software specification



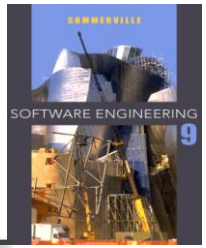
- The process of establishing what services are required and the constraints on the system's operation and development
- Requirements engineering process
  - Feasibility study
    - Is it technically and financially feasible to build the system?
  - Requirements elicitation and analysis
    - What do 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



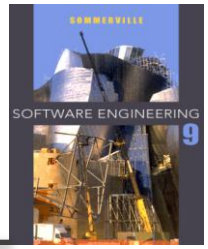


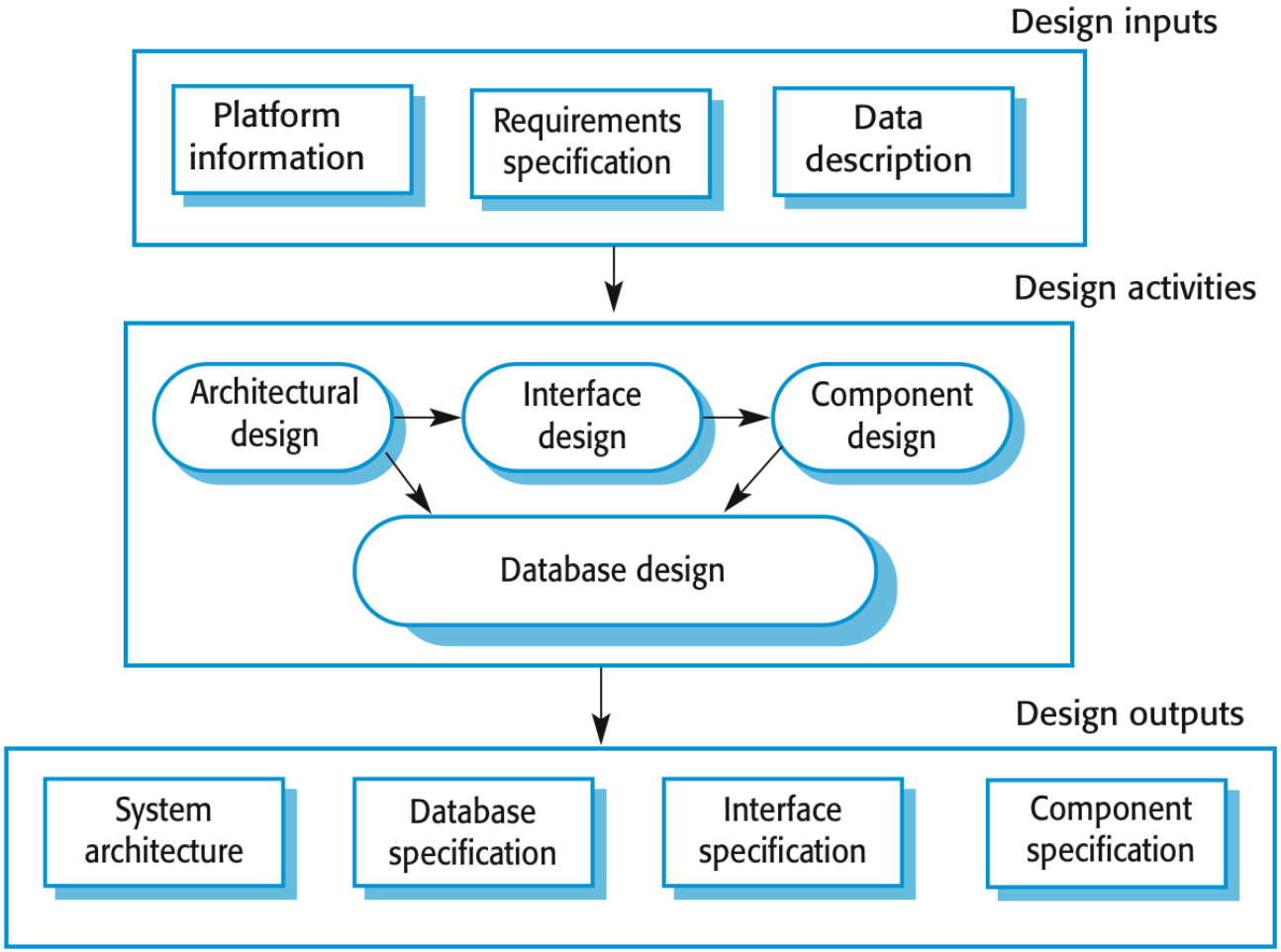
Software design and implementation



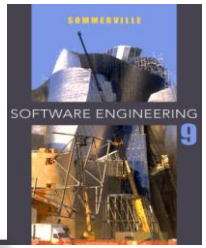
- The process of converting the system specification into an executable system. Design is about *how* to build a system.
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;
- The activities of design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

A general model of the design process



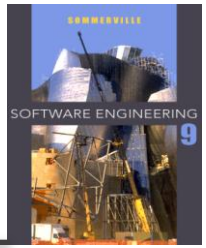


Design activities



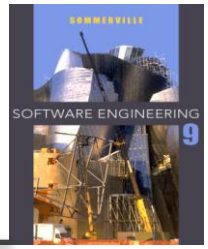
- *Architectural design*, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- *Interface design*, where you define the interfaces between system components.
- *Component design*, 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.

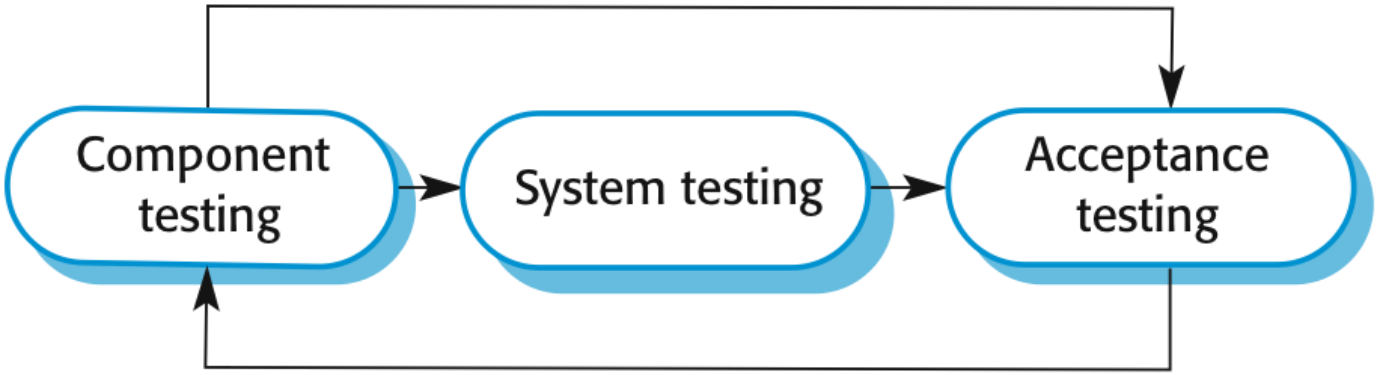
Software validation



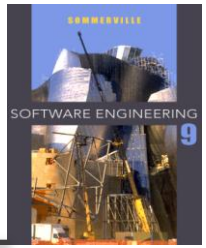
- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer
- Involves checking and review processes and system testing
- System testing involves executing the system with test cases that are derived from the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

Stages of testing



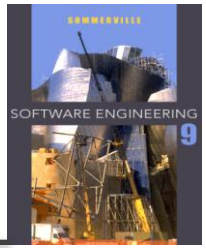


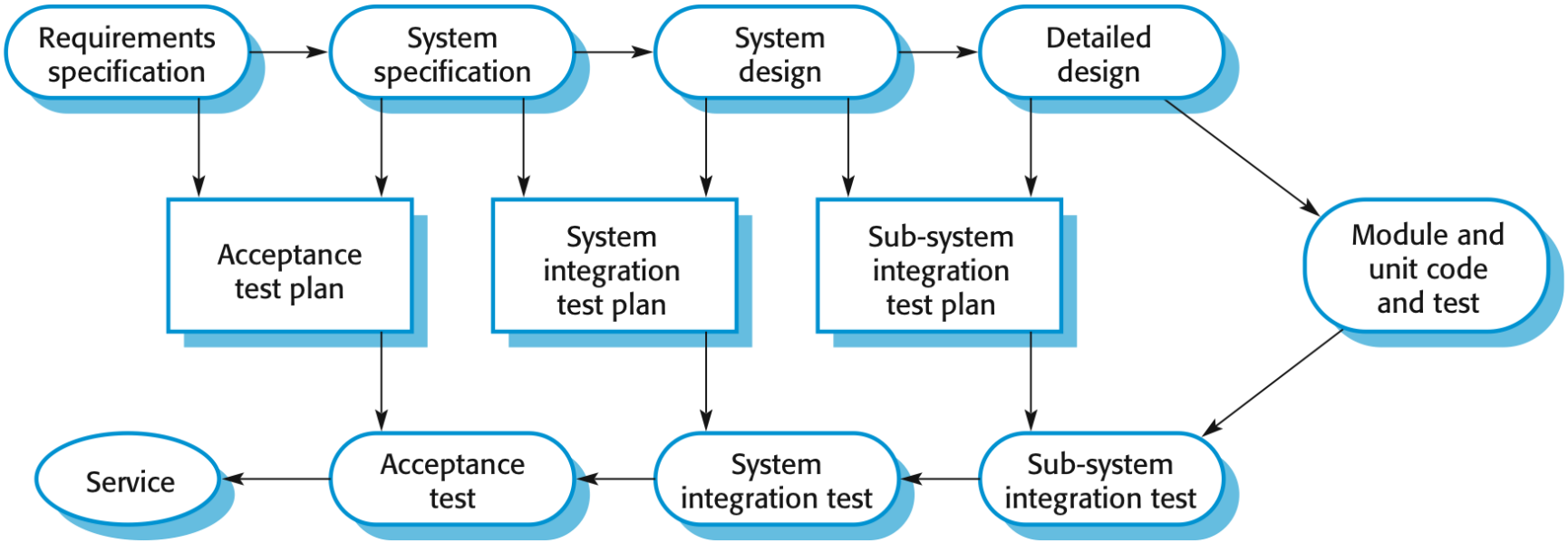
Testing stages



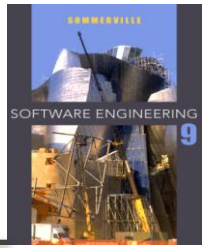
- Development or component testing
  - Individual components are tested independently
  - Components may be functions or objects or coherent groupings of these entities
- System testing
  - Testing of the system as a whole. Testing of emergent properties is particularly important.
- Acceptance testing
  - Testing with customer data to check that the system meets the customer's needs

Testing phases in a plan-driven software process



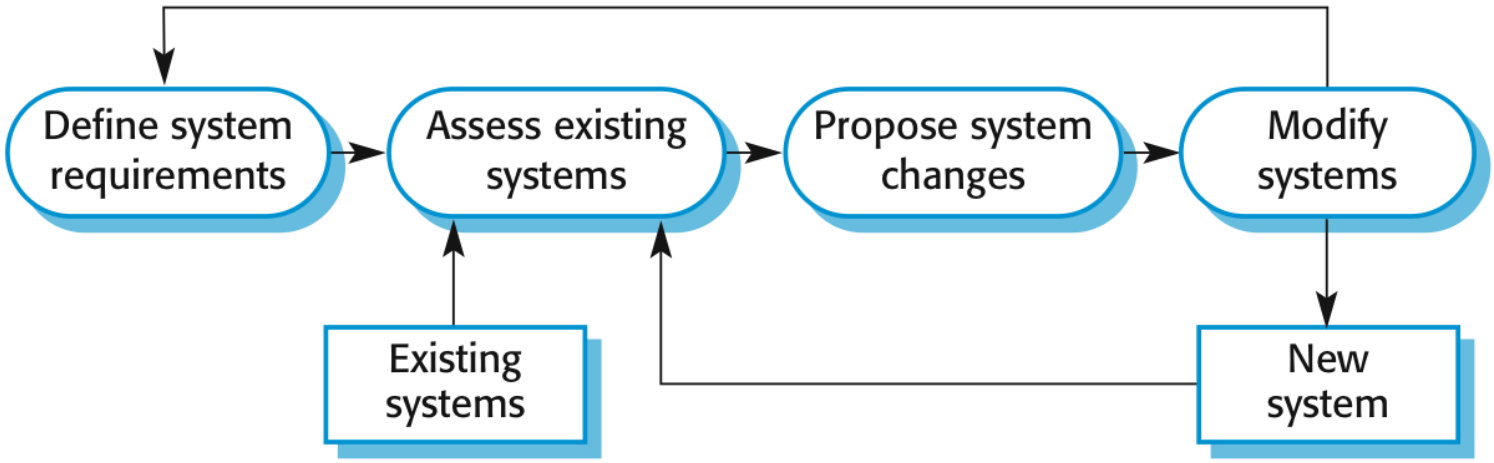


Software evolution

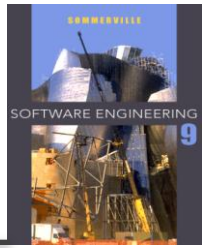


- Software is inherently flexible and can change
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant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

System evolution

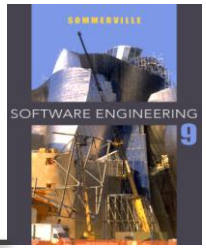


Coping with change



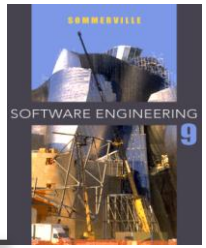
- Change is inevitable in all large software projects
  - Business changes lead to new and changed system requirements
  - New technologies open up new possibilities for improving implementations
  - Changing platforms require application changes
- Change leads to rework so the costs of change include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

Reducing the costs of rework



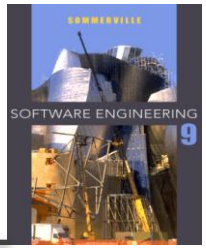
- Change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required
  - For example, a prototype system may be developed to show some key features of the system to customers
- Change tolerance, where the process is designed so 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

Software prototyping



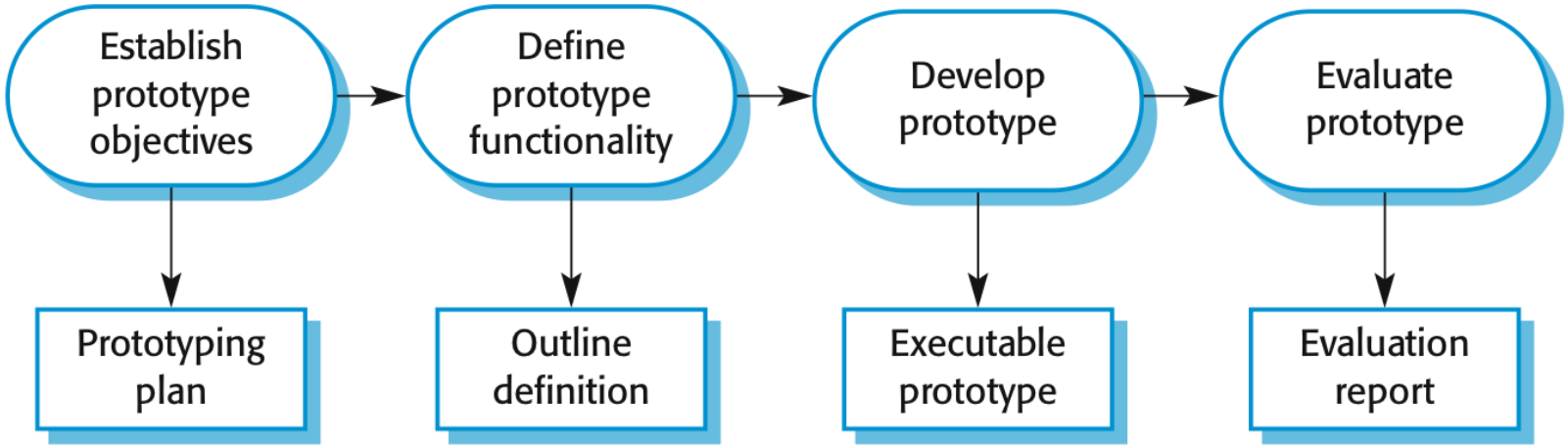
- A prototype is an initial version of a system used to demonstrate concepts and try out design options
- A prototype can be used in:
  - The requirements engineering process to help with requirements elicitation and validation
  - In design processes to explore options and develop a UI design
  - In the testing process to run back-to-back tests

Benefits of prototyping

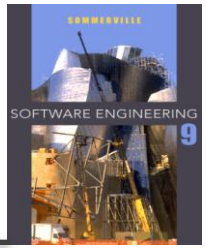


- ✧ Improved system usability
- ✧ A closer match to users' real needs
- ✧ Improved design quality
- ✧ Improved maintainability
- ✧ Reduced development effort

The process of prototype development

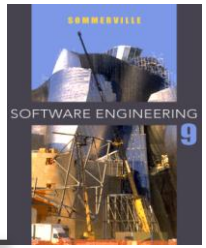


Prototype development



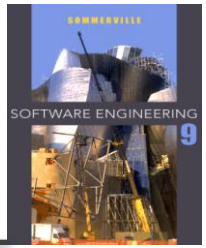
- ✧ May be based on rapid prototyping languages or tools
- ✧ May involve leaving out functionality
  - Prototype should focus on areas of the product that are not well-understood
  - Error checking and recovery may not be included in the prototype
  - Focus on functional rather than non-functional requirements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

Throw-away prototypes



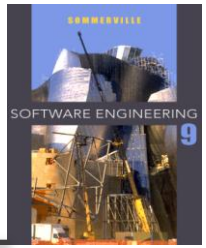
- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system to meet non-functional requirements
  - Prototypes are normally undocumented
  - The prototype structure is usually degraded through rapid change
  - The prototype probably will not meet normal organisational quality standards

4 Incremental delivery



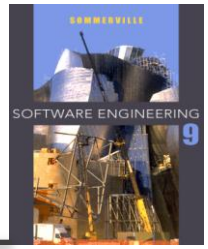
- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality
- User requirements are prioritized and the highest priority requirements are included in early increments
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve

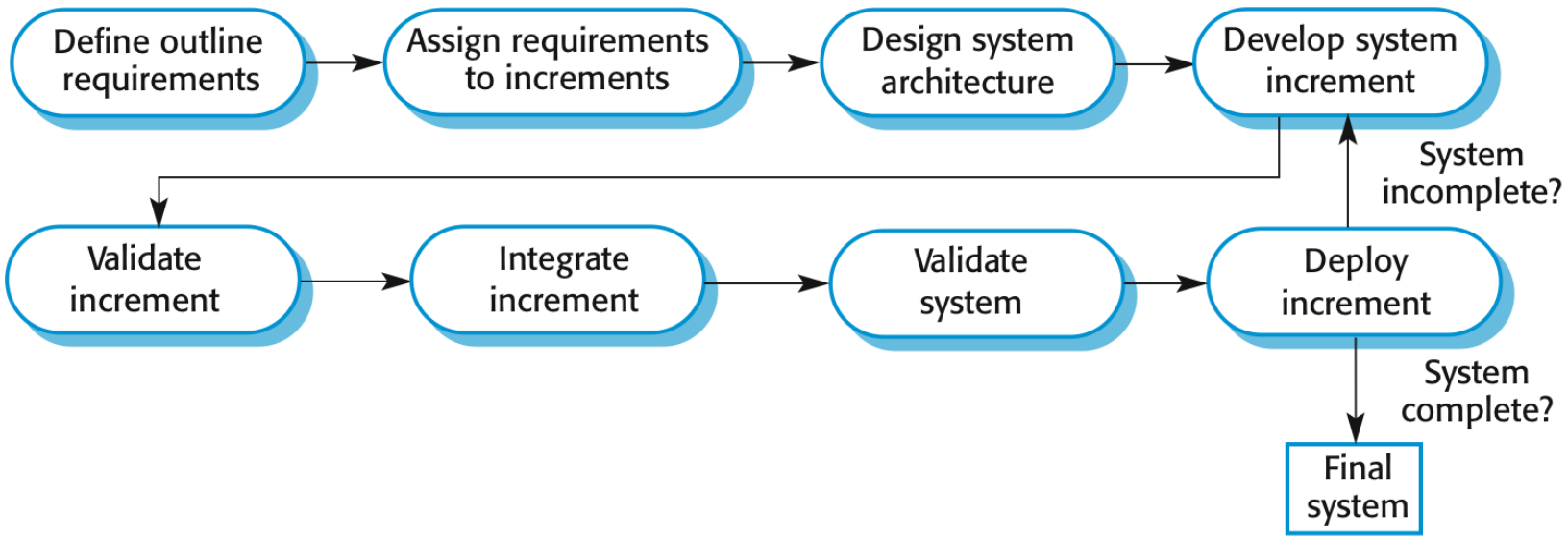
Incremental delivery



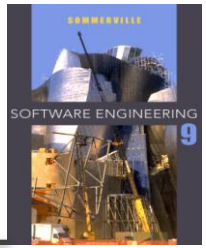
- Incremental development
  - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment
  - Normal approach used in agile methods
  - Evaluation done by user/customer proxy
- Incremental delivery
  - Deploy an increment for use by end-users
  - More realistic evaluation about practical use of software
  - Difficult to implement for replacement systems as increments have less functionality than the system being replaced

Incremental delivery



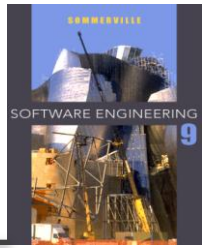


Incremental delivery advantages



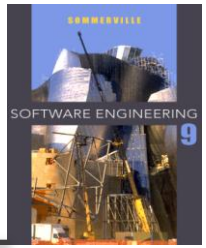
- Customer value can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing

Incremental delivery problems



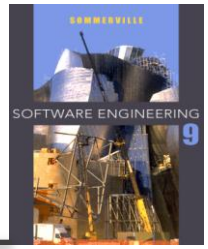
- Most systems require a set of basic facilities that are used by different parts of the system
  - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments
- The essence of iterative processes is 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

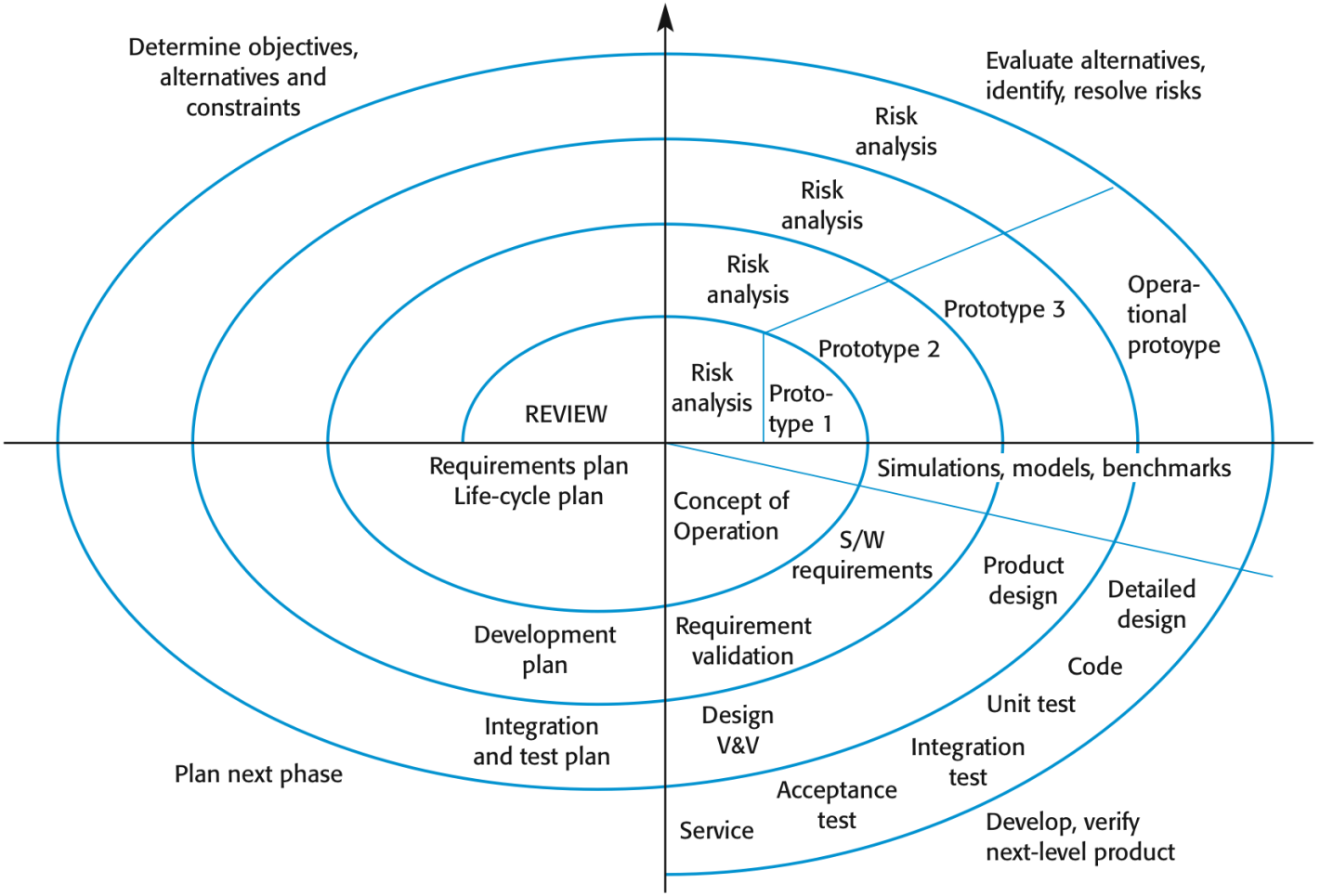
5 Boehm's spiral model



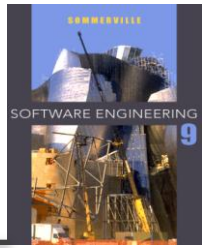
- Process is represented as a spiral rather than as a sequence of activities with backtracking
- Each loop in the spiral represents a phase in the process
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed and resolved throughout the process

Boehm's spiral model of the software process



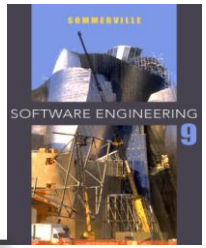


Spiral model sectors



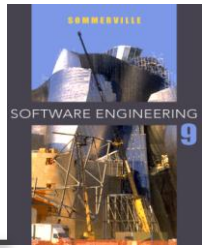
- ✧ Objective setting
  - Specific objectives for the phase are identified
- ✧ Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks
- ✧ Development and validation
  - A development model for the system is chosen which can be any of the generic models
- ✧ Planning
  - The project is reviewed and the next phase of the spiral is planned

Spiral model usage



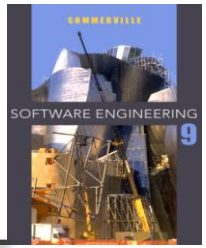
- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development
- In practice, however, the model is rarely used as published for practical software development

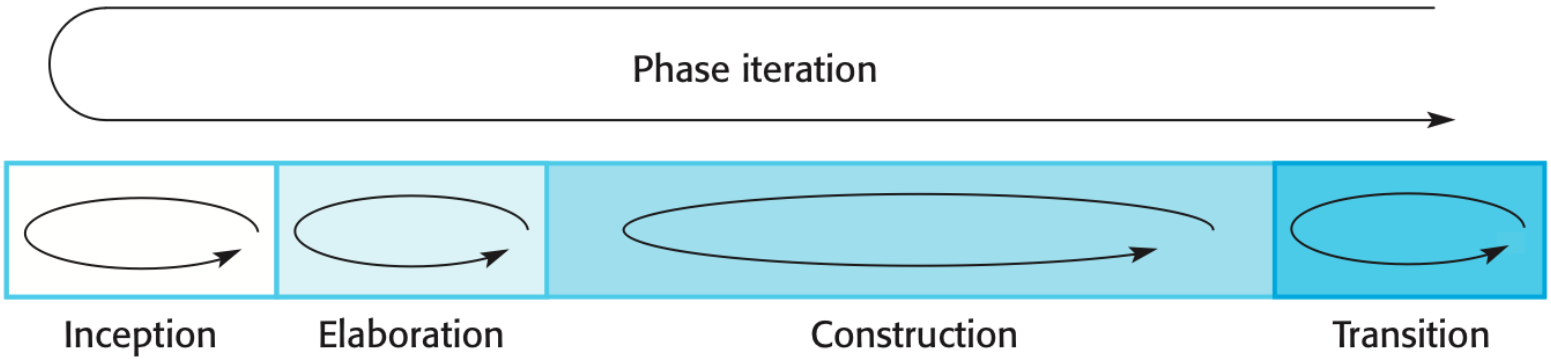
6 The Rational Unified Process



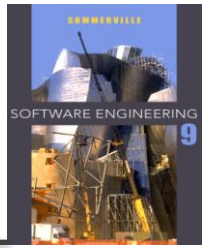
- A modern generic process derived from the work on the UML and associated process
- Brings together aspects of the generic process models discussed previously
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time
  - A static perspective that shows process activities
  - A practice perspective that suggests good practice

Phases in the Rational Unified Process



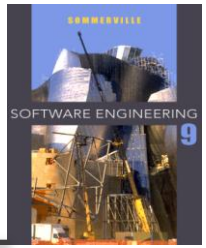


RUP phases



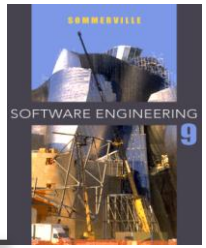
- Inception
  - Establish the business case for the system
- Elaboration
  - Develop an understanding of the problem domain and the system architecture
- Construction
  - System design, programming and testing
- Transition
  - Deploy the system in its operating environment

RUP iteration



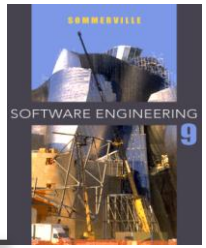
- In-phase iteration
  - Each phase is iterative with results developed incrementally
- Cross-phase iteration
  - As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally

Static workflows in the Rational Unified Process



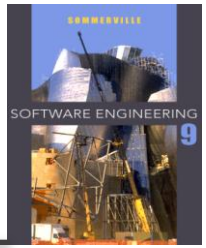
Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural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

Static workflows in the Rational Unified Process



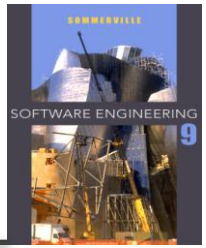
Workflow	Description
Testing	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management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

RUP good practice



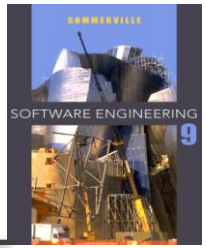
- ✧ Develop software iteratively
  - Plan increments based on customer priorities and deliver highest priority increments first
- ✧ Manage requirements
  - Explicitly document customer requirements and keep track of changes to these requirements
- ✧ Use component-based architectures
  - Organize the system architecture as a set of reusable components

RUP good practice



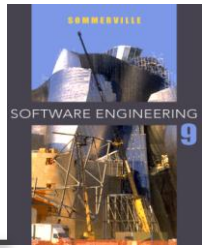
- ✧ Visually model software
  - Use graphical UML models to present static and dynamic views of the software
- ✧ Verify software quality
  - Ensure that the software meet's organizational quality standards
- ✧ Control changes to software
  - Manage software changes using a change management system and configuration management tools

Key points



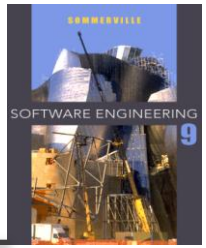
- Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes. Examples of these general models include the 'waterfall' model, incremental (exploratory) development, reuse-oriented development, incremental delivery, Boehm's spiral model, and RUP.

Key points



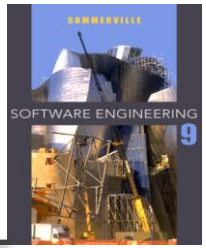
- Requirements engineering is the process of developing a software specification
- Design and implementation processes are concerned with transforming a requirements specification into an executable software system
- Software validation is the process of checking that the system conforms to its specification 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

Key points



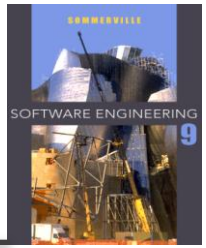
- Processes should include activities to cope with change. This may involve a prototyping phase that helps avoid poor decisions on requirements and design
- Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole
- The Rational 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

Documentation in Waterfall



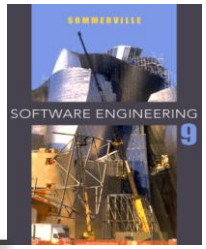
- ✧ Concept
- ✧ High-Level Requirements
- ✧ Technical Specification
- ✧ Detailed Design
- ✧ Testing
- ✧ Implementation
- ✧ Deployment

Concept



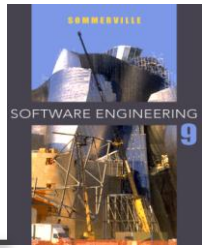
- ✧ Many projects start off with a concept document
- ✧ This would usually consist of 1-2 paragraphs explaining what the stakeholder would like
- ✧ It sometimes also includes motivation for wanting the project implemented

Example – Concept



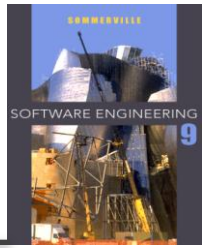
- It would be great if students could submit a programming assignment and have it graded instantaneously in an automated manner. This would reduce time and money paying a grader and provide students an opportunity to submit their program more than once in an effort to improve their score.

High-Level Requirements



- The high-level requirements for a piece of software usually originate from the person, department, or organization requesting the software to be developed
  - The people do not need to be technical
- The requirements should be detailed enough for a programmer (or technical person) to be able to write out detailed technical specifications that will be passed along to the programmers
  - The details of the implementation should not be included in the requirements document
- Conceptual screenshots could be included, though this is not necessary

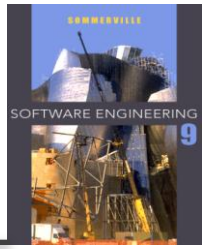
Example – High-Level Requirements



- We need to create a web interface where students can upload their Java source code to a program. The program should be executed on known input and validated against known output. The students will then be given a score representing how much of the output is correct.

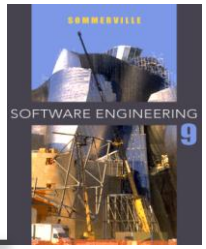
- Example

Technical Specifications



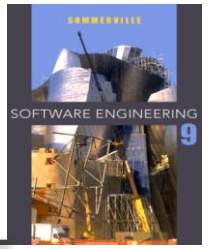
- Whereas the high-level requirements could have come from a non-technical person, the technical specifications are written by a technical person
  - They ultimately will be approved by the person who requested the project to be completed
- The requirements should have been detailed enough for technical specifications to be written
  - The technical specifications include hardware requirements, software requirements, benchmarks to meet, estimate of hours, estimate of cost
  - Language and platform do not need to be included unless they are critical to the project
    - These are typically design decisions

Example – Technical Specifications



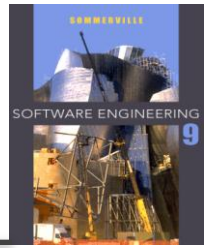
- Web interface (4 hours)
  - The web interface needs to have a login page with a username field, a password field, and a Login button
  - Upon verification, another form should be displayed with a drop-down list specifying the assignment number, a file chooser field that allows the user to select the source code file, and a Submit button
  - The submission page should show the user the final grade based on how much of the output was correct
- Assignment grading (8 hours)
  - When a Java source code file is uploaded, it needs to be compiled and executed
  - The output will be compared against known outputs provided by the instructor
  - The percentage of the output that matches exactly will be returned to the user
- Database (8 hours)
  - The database will consist of two tables – a User table and a Submission table.
  - The User table will be used for validating users and consist of userID, username, password, fname, lname, and timestamp representing the last login time
  - The Submission table will consist of the name of the file uploaded, userID, assignment number, percentage of output that matches, and a timestamp

Detailed Design Document

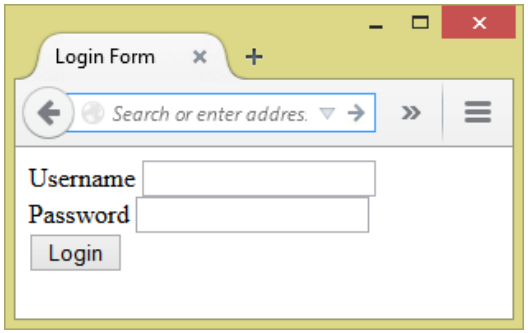


- Now that you have the technical specifications document, you need to start thinking about how you will actually implement them
- The detailed design document will include the specific hardware and software to use
  - This will include programming language
- It will also include the class diagram, inheritance hierarchy, method and variable descriptions, pseudo-code, and algorithms to be used
- After the design document is created and approved, you should be able to start writing test cases and code

Example – Detailed Design



- The login page should look like

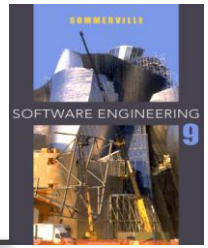


- The UserAuthentication class will access the User table in a MySQL database named AutomateProgram using the jdbc:mysql driver
  - The authenticate() method will take two strings as parameters, representing the username and password
  - It will return a Boolean value specifying whether the user was authenticated or not against the User table

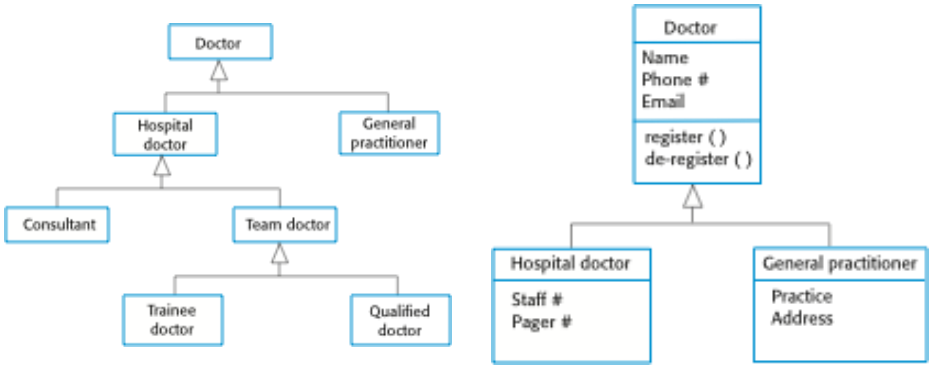
There will be a lot more in the detailed design document, but this gives you a general idea

- Example

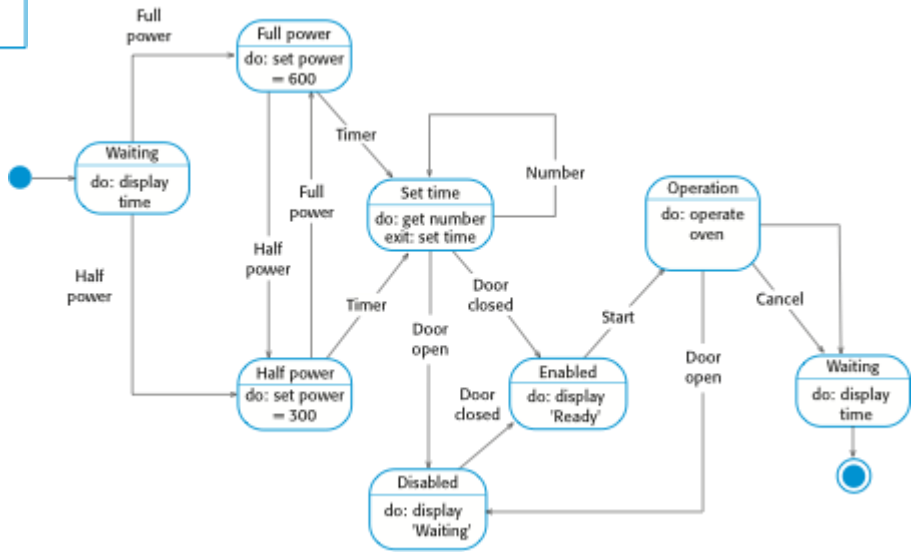
Example – Detailed Design (cont.)



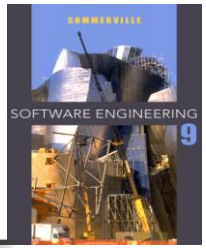
✧ Sample class diagrams



✧ Sample state diagram

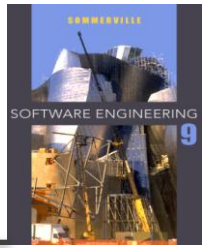


Testing



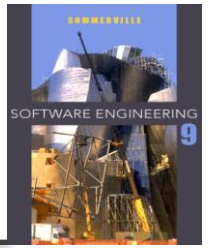
- Now that you have the detailed designed document, you need to start thinking about testing the design
- There are different testing strategies, including
  - Black box testing – test the entire application without looking at the code
  - White box testing – test the entire application while looking at the code
  - Unit testing – test individual functionality of the code by writing customized programming
  - Stress testing – test the extensibility of the program by trying to find the limits
  - Regression testing – when changes are made, make sure the changes don't affect other parts of the program

Example – Testing



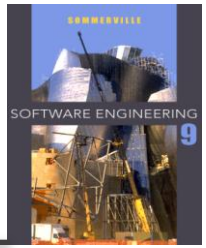
- Test Case 1
  - White Box Test – test the login functionality by specifying a username that exists and a password that does not match. The user should be taken back to the login page with a message "Invalid login."
- Test Case 2
  - White Box Test – test the login functionality by specifying a username that does not exist. The user should be taken back to the login page with a message "Invalid login."
- Test Case 3
  - White Box Test – test the login functionality by specifying a username that exists and a password that matches. The user should be taken to the assignment submission page.
- Test Case 4
  - Unit Test – insert SQL code in the username variable of the UserAuthentication.authenticate() method. Verify that the SQL code specified is not executed against the database.

Implementation



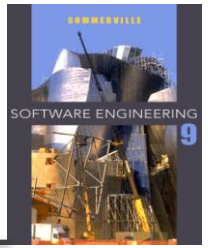
- We have now completed the high-level requirements, technical specifications, detailed design document, and testing strategies
- We are ready to start implementing our design
  - This typically involves setting up servers (development servers, testing servers, QA servers, deployment servers) installing third-party applications, writing code, ensuring the program meetings the specifications, and testing
- Everything else that is required up to actually launching the program goes in the implementation phase

Example - Implementation



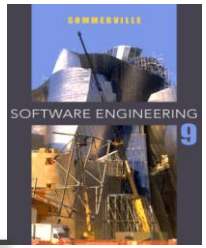
- Set up the servers
- Write the code
- Test the code
- Ensure alignment with specifications
- Prepare to turn over to client for approval before deployment

Deployment



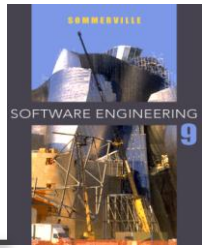
- Deployment is the process of promoting a fully-tested and approved application
- Deployment could require different promotion processes
  - For web applications, it would require promoting the application to a server, testing, redirecting DNS
  - For standalone applications, it would require producing media or downloadable packages
  - For software as a service, there are licensing issues that need to be determined
- Ensure data migration to the live server has occurred completely

Example – Deployment



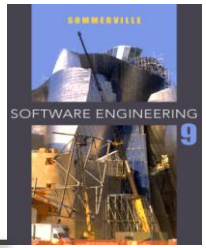
- Step 1 - push the application to a live server
- Step 2 - ensure all of the data was migrated to the live server from the existing system
  - To do this, run the verify.sql script
- Step 3 – perform a full regression test of the application
- Step 4 - notify the users of the updated application and provide them with the URL http://usc.edu/login

Complete Documentation



- After the application has been deployed, provide complete documentation to whoever requested the project to be implemented
- This will include all of the documents you have generated in a single document with a title page, table of contents, page numbers, and any references or related work
- You may also need to help generate documentation for the end users, such as user guides

Homework



- Sommeville: 2.1, 2.2, 2.5.
- Find two SE-related video clips online and write short description for them (120 to 200 words each). Concisely explain what they present, their significance for Software Engineering, and why you think the video clips are worthwhile watching. The videos should have a length of between 2:00 and 10:00 minutes (they can be excerpts from longer videos, in which case indicate your selected start and stop time). Provide the complete and correct links for them (make sure they are clickable and give access to the right videos). For example:
  - Spotify Software Engineering Culture Part 1, 2
  - https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/
  - https://labs.spotify.com/2014/09/20/spotify-engineering-culture-part-2/