Software Process

Software Engineering 1

Soon Phei Tin

Objectives

- understand the concepts of software processes and software process models;
- introduced to three generic software process models and when they might be used;
- know about the fundamental process activities of software requirements engineering, software development, testing, and evolution;

Introduction

- A software process is a set of related activities that leads to the production of a software product. These activities may involve the development
  - of software from scratch
  - by extending and modifying existing systems
  - by configuring and integrating off-the-shelf software or system components

Introduction

- A **software process** refers to the actual set of activities, methods, practices, and transformations that people use to develop and maintain software and its associated products (such as project plans, design documents, code, test cases, and user manuals). It is the real-world implementation of the steps involved in creating software.

Software Process Activities

- There are many different software processes, but all must include four activities that are fundamental to software engineering:
  - Software specification
  - Software design and implementation
  - Software validation
  - Software evolution
- In practice, they are complex activities in themselves and include sub-activities (Refer to appendix 1 for the sub-activities)
- There are also supporting process activities such as documentation and software configuration management (Refer to appendix 2 for other supporting process activities)

Question

Which of the below fundamental activities comprises of programming/coding?

A. Software specification
B. Software design and implementation
C. Software validation
D. Software evolution

Types of Software Processes

- Software processes are complex and, there is no ideal process
- Most organizations have developed their own software processes that take advantage of the capabilities of the people in an organization and the specific characteristics of the systems that are being developed
  - For a critical systems, a very structured development process is required.
  - For non-critical systems, with rapidly changing requirements, a less formal, flexible process is likely to be more effective

Categories of Software Processes

- There are many software processes introduced to the industry. To better understand them, we categorized the software processes as follows.
  - **Linear and Sequential Models:** Waterfall, V-Model
  - **Iterative and Incremental Models:** Incremental, Iterative, Spiral
  - **Agile Models:** Scrum, Kanban, Extreme Programming (XP)
  - **Prototyping Models:** Throwaway Prototyping, Evolutionary Prototyping
  - **Component-Based Models:** Component-Based Development (CBD)
  - **Formal Methods:** Formal Specification, Model Checking
  - **Hybrid Models:** Agile-Waterfall Hybrid, DevOps
  - **Rapid Application Development (RAD) Models:** RAD Model, DSDM
  - **Lean Models:** Lean Software Development

Software Process Models

Software Engineering 1

Soon Phei Tin

Software Process Models

- A software process model is a simplified representation of a software process.
- Discuss very general process models and present these from an architectural perspective, framework of the process rather than the details of specific activities.
- These generic models are not definitive descriptions of software processes, they are abstractions of the process that can be used to explain different approaches to software development
- You can think of them as process frameworks that may be extended and adapted to create more specific software engineering processes.

Software Process Models

- While the software process is the actual execution of tasks and activities, the software process model provides the structured framework that guides this execution.
- The model ensures that best practices are followed, risks are managed, and quality is maintained, leading to more predictable, efficient, and successful software development projects.

Software Process Models

- The models in our discussion: -
  - The waterfall model - This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing, and so on.
  - Incremental development - This approach interleaves the activities of specification, development, and validation. The system is developed as a series of versions (increments), with each version adding functionality to the previous version.
  - Reuse-oriented software engineering - This approach is based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them from scratch.

Software Process Models

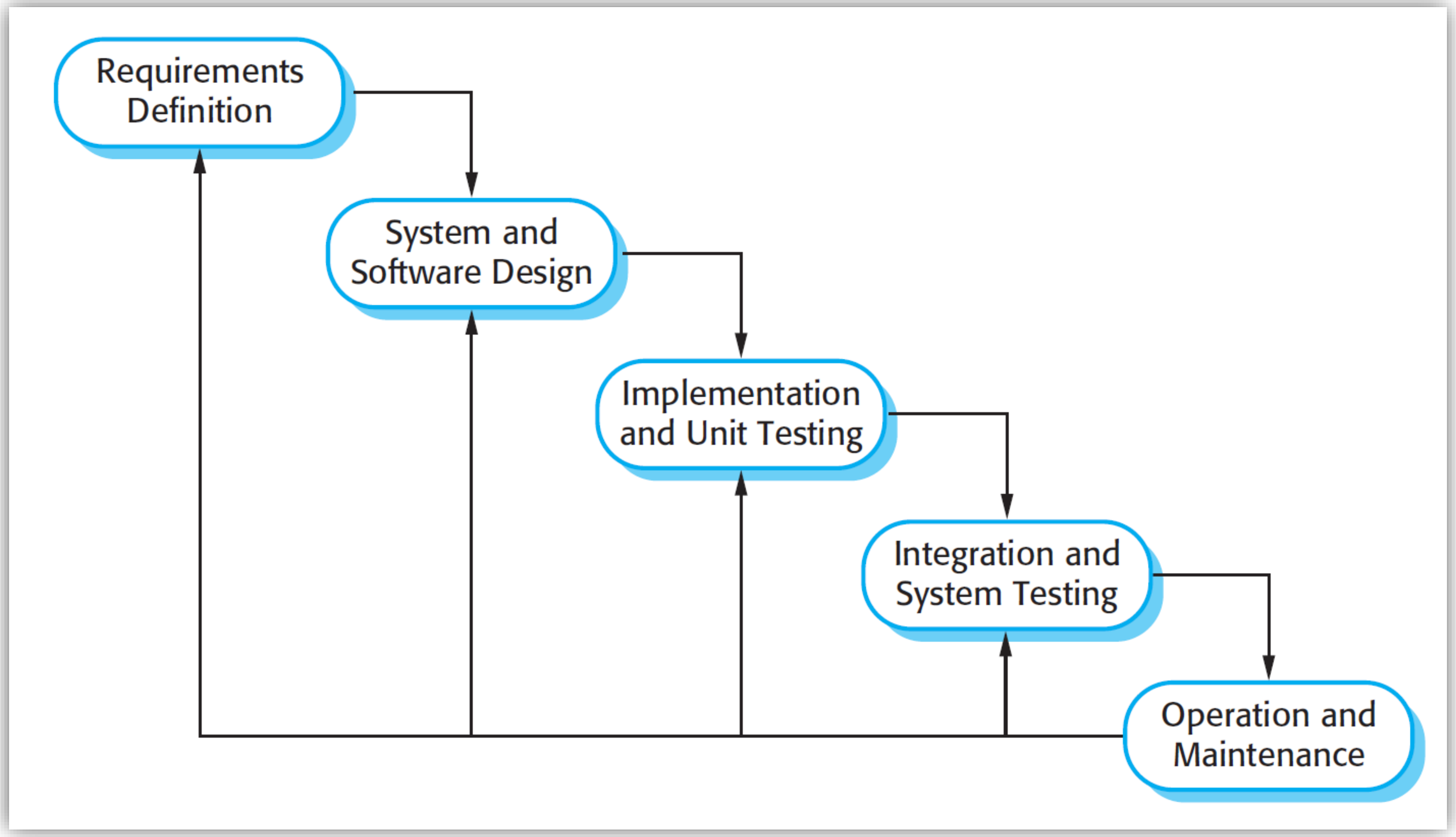
- These models are not mutually exclusive and are often used together, especially for large systems development.

The Waterfall Model

- **Definition**: The Waterfall Model is a linear and sequential approach to software development. It progresses through distinct phases, each with specific deliverables and review processes.

- **Historical Context**: Introduced by Dr. Winston W. Royce in a 1970 paper, it was one of the first formalized software development methodologies.

The Waterfall Model


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

Phases of The Waterfall Model

- Because of the cascade from one phase to another, this model is known as the ‘waterfall model’ or software life cycle. The waterfall model is an example of a plan-driven process
- The principal stages of the waterfall model directly reflect the fundamental development activities:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance

Phases of the Waterfall Model

- **Requirements analysis and definition**
  - **Objective:** Gather and document all functional and non-functional requirements.
  - **Deliverables**: Requirements Specification Document.
  - **Activities**: Stakeholder interviews, surveys, and requirement workshops.
- **System and software design**
  - **Objective**: Create a detailed design based on the requirements.
  - **Deliverables**: System Architecture Document, Design Specifications.
  - **Activities**: Architectural design, database design, user interface design.
- **Implementation and unit testing**
  - **Objective**: Translate design documents into actual code.
  - **Deliverables**: Source code, code documentation.
  - **Activities**: Coding, unit testing, code reviews.
- **Integration and system testing**
  - **Objective**: Integrate all modules and test the complete system.
  - **Deliverables**: Test plans, test cases, test reports.
  - **Activities**: Integration testing, system testing, user acceptance testing.
- **Operation and maintenance**
  - **Objective**: Deploy the system to the production environment and provide ongoing support and enhancements.
  - **Deliverables**: Deployment plan, user manuals, maintenance reports, update patches.
  - **Activities**: Installation, configuration, user training, bug fixes, performance improvements, feature updates.

Characteristics of the Waterfall Model

- **Linear and Sequential**: Each phase must be completed before the next one begins.
- **Documentation-Driven**: Extensive documentation is produced at each stage.
- **Phase-Specific Deliverables**: Each phase has specific deliverables and milestones.
- **Review and Approval**: Each phase requires review and approval before proceeding to the next.

Characteristics of the Waterfall Model

- 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
- 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

Characteristics of the Waterfall Model

- 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 development tasks may mean that the system won’t deliver what the users are expecting.

Characteristics of the Waterfall Model

- During the final life cycle phase (operation and maintenance) the software is put into use.
- Errors and omissions in the original software requirements are discovered.
- Program and design errors emerge and the need for new functionality is identified.
- The system must therefore evolve to remain useful. Making these changes (software maintenance) may involve repeating previous process stages.

Characteristics of the Waterfall Model

- In the Waterfall Model, documentation is produced at each phase. This makes the process visible so managers can monitor progress against the development plan.
- Its major problem is the inflexible partitioning of the project into distinct stages.
- Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
- In principle, the waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development.

When you are developing a software following waterfall model, what should you do if you realize a missing requirement during the software validation stage?

Strengths and Benefits

1. **Simplicity and Ease of Use**
   - **Clear Structure**: The Waterfall Model is straightforward and easy to understand, making it simple to implement and manage.
   - **Well-Defined Phases**: Each phase of the Waterfall Model has a defined start and end point, which helps in understanding the project’s progress and managing the workflow.
2. **Predictability and Planning**
   - **Detailed Planning**: The model allows for detailed planning at the beginning of the project, which can lead to more accurate estimates of costs, timelines, and resources.
   - **Predictable Outcomes**: With well-defined phases and deliverables, the outcomes are more predictable, which is beneficial for stakeholders and project managers.
3. **Documentation**
   - **Comprehensive Documentation**: Each phase requires thorough documentation, which can be useful for future maintenance and knowledge transfer. This documentation provides a clear record of the project’s development.
   - **Traceability**: The documentation provides clear traceability of decisions and changes, making it easier to track the project’s history and understand the rationale behind design choices.

Strengths and Benefits

4. **Discipline and Control**
   - **Structured Approach**: The linear and sequential nature of the Waterfall Model enforces a disciplined approach to software development
   - **Phase Completion**: The requirement to complete each phase before moving on ensures that all aspects of the project are thoroughly reviewed and approved.
5. **Resource Management**
   - **Resource Allocation**: The clear phases allow for better planning and allocation of resources.
   - **Skill Utilization**: Teams can be organized based on the specific skills required for each phase.
6. **Milestones and Deliverables**
   - **Clear Milestones**: Clear milestones and deliverables for each phase, making it easier to track progress and manage project timelines.
   - **Client Approval**: Each phase can be reviewed and approved by the client, providing opportunities for feedback and ensuring that the project stays aligned with client expectations.
7. **Risk Management**
   - **Controlled Changes**: Changes are managed in a controlled manner, and any changes are thoroughly reviewed and documented.

Weaknesses and Limitations

1. **Inflexibility**
   - **Rigid Structure**: The linear nature of the Waterfall Model makes it difficult to accommodate changes once a phase is completed.
   - **Sequential Progression**: Each phase must be completed before moving on to the next, which means that any delays or issues in one phase can impact the entire project timeline.
2. **Late Testing**
   - **Late Problem Detection**: Testing is conducted late in the development process, typically after the implementation phase. This means that defects and issues are often discovered late, making them more expensive and difficult to fix.
   - **Limited Iterative Feedback**: Since testing occurs at the end, there are limited opportunities for iterative feedback and improvements during the development process.
3. **Risk Management**
   - **High Risk**: The Waterfall Model does not inherently include risk management activities, making it less suitable for projects with high uncertainty or risk.
4. **Customer Involvement**
   - **Limited Feedback**: Customers typically do not see the product until the end of the development cycle, which can lead to a final product that does not fully meet their needs or expectations.
   - **Assumption of Perfect Requirements**: The model assumes that all requirements can be clearly defined at the beginning, which is often not the case in real-world projects. This can result in a misalignment between the final product and user needs.

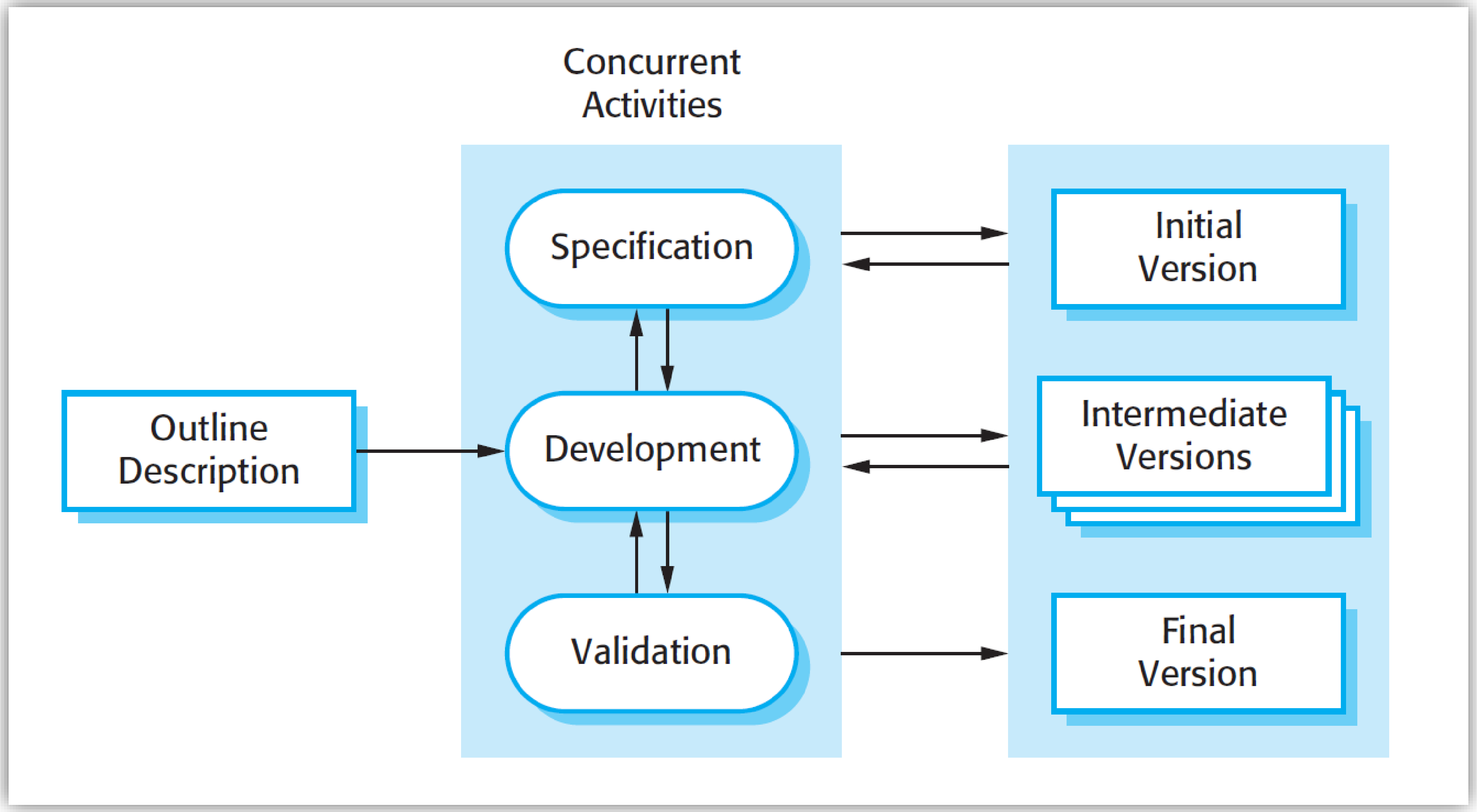
Weaknesses and Limitations

5. **Adaptability**
    - **Poor Adaptability**: The Waterfall Model is not well-suited for projects where requirements are expected to evolve or where iterative development is beneficial.
    - **Scope Creep**: Changes in scope can be difficult to manage and integrate, leading to potential delays and increased costs.
6. **Overhead and Documentation**
    - **Heavy Documentation**: The emphasis on documentation can be time-consuming and may lead to delays, especially in fast-paced projects. The extensive documentation required can also become outdated quickly if changes are made.
    - **Administrative Overhead**: Managing documentation and phase transitions can add administrative overhead, which may not be justified for smaller projects.
7. **Resource Utilization**
    - **Idle Resources**: Resources may be underutilized during certain phases. For example, developers may be idle during the requirements and design phases, while analysts and designers may have less to do during the implementation phase.
    - **Sequential Resource Allocation**: The model does not easily allow for parallel workstreams, which can lead to inefficiencies in resource utilization.
8. **Longer Project Timelines**
    - **Extended Timelines**: The sequential nature of the Waterfall Model can lead to longer project timelines, as each phase must be completed before the next one begins.
    - **Delayed Delivery**: The final product is delivered only at the end of the project, which can delay the realization of value for stakeholders and users

Incremental Model

- **Definition**: The Incremental Model is an iterative approach to software development where the system is built incrementally. Each increment adds functional pieces to the system until the complete product is delivered.
- **Historical Context**: Evolved from the need to address the limitations of the Waterfall Model by allowing more flexibility and iterative development.

Incremental Model



Phases of the Incremental Model

- **Outline Description and Planning**
  - **Objective**: Define the overall project scope and high-level requirements.
  - **Deliverables**: Project plan, high-level requirements document.
  - **Activities**: Stakeholder interviews, requirement workshops, project planning.
- **Specification and Planning**
  - **Objective**: Plan the details for each increment, including specific requirements, design, and tasks.
  - **Deliverables**: Increment plan, detailed requirements for the increment.
  - **Activities**: Requirement analysis, task breakdown, resource allocation.
- **Design and Development**
  - **Objective**: Design and develop the functionality for the specific increment.
  - **Deliverables**: Design documents, source code, unit tests.
  - **Activities**: Architectural design, coding, unit testing, code reviews.
- **Integration and Validation**
  - **Objective**: Integrate the new increment with the existing system and perform testing.
  - **Deliverables**: Integrated system, test cases, test reports.
  - **Activities**: Integration testing, system testing, user acceptance testing.
- **Deployment/Increment**
  - **Objective**: Deploy the increment to the production environment.
  - **Deliverables**: Deployed increment, user manuals.
  - **Activities**: Installation, configuration, user training.
- **Review and Feedback**
  - **Objective**: Gather feedback from users and stakeholders for the deployed increment.
  - **Deliverables**: Feedback reports, updated requirements.
  - **Activities**: User feedback sessions, requirement updates.
- **Next Increment Planning**
  - **Objective**: Plan the next increment based on feedback and updated requirements.
  - **Deliverables**: Updated project plan, detailed requirements for the next increment.
  - **Activities**: Requirement analysis, task breakdown, resource allocation.

Characteristics of the Incremental Model

- **Iterative Development**: The project is divided into smaller, manageable increments.
- **Early Delivery**: Functional pieces of the system are delivered early and frequently.
- **User Feedback**: Regular feedback from users is incorporated into each increment.
- **Flexibility**: Easier to accommodate changes in requirements and feedback.
- **Risk Management**: Early identification and mitigation of risks through iterative development.

Characteristics of the Incremental Model

- 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.
- Incremental software development, which is a fundamental part of agile approaches, is better than a waterfall approach for most business, e-commerce, and personal systems (However, it is different from Agile approach. Check Appendix 3)

Characteristics of the Incremental Model

- By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.
- Each increment or version of the system incorporates some of the functionality that is needed by the customer.
- Generally, the early increments of the system include the most important or most urgently required functionality. This means that the customer can evaluate the system at a relatively early stage in the development to see if it delivers what is required.
- If not, then only the current increment has to be changed and, possibly, new functionality defined for later increments

Case Study: Incremental Model in Action

- **Project**: Development of an E-commerce Platform
- **Phases**:
  - **Initial Planning**: Defined high-level requirements and project scope.
  - **Increment Planning**: Planned the first increment to include basic product listing and user registration features.
  - **Design and Development**: Designed and developed the first increment, performed unit testing.
  - **Integration and Testing**: Integrated the first increment with the existing system, performed system testing.
  - **Deployment**: Deployed the first increment to the production environment.
  - **Review and Feedback**: Gathered feedback from users and stakeholders, updated requirements.
  - **Next Increment Planning**: Planned the next increment to include shopping cart and checkout features.
  - **Iterative Process**: Repeated the cycle for subsequent increments, adding features like payment processing, order tracking, and user reviews.

Strengths and Benefits

1. **Early Delivery and Feedback**
   - **Early Functional Delivery**: The Incremental Model allows for the delivery of functional parts of the system early in the development process.
   - **User Feedback**: Early increments can be reviewed by users, providing valuable feedback that can be used to refine subsequent increments.
2. **Flexibility and Adaptability**
   - **Accommodates Changes**: The model is flexible and can accommodate changes in requirements more easily than linear models.
   - **Iterative Development**: The iterative nature of the Incremental Model allows for continuous improvement and refinement of the system.
3. **Risk Management**
   - **Reduced Risk**: By breaking the project into smaller increments, risks are reduced. Issues can be identified and addressed early in the development process, minimizing the impact on the overall project.
   - **Early Problem Detection**: Testing is performed at the end of each increment, allowing for early detection and resolution of defects and issues.
4. **Improved Resource Utilization**
   - **Parallel Development**: Different increments can be developed in parallel, optimizing the use of resources and reducing development time.
   - **Focused Efforts**: Teams can focus on developing and refining specific features or functionalities in each increment, leading to higher quality outputs.

Strengths and Benefits

5. **Better Planning and Estimation**
   - **Shorter Planning Cycles**: Planning for smaller increments is more manageable and can be done more frequently, improving the accuracy of estimates and schedules.
   - **Milestones and Deliverables**: Each increment provides clear milestones and deliverables, making it easier to track progress and manage the project.
6. **Customer and Stakeholder Satisfaction**
   - **Regular Deliveries**: Regular deliveries of functional increments keep customers and stakeholders engaged and satisfied. They can see tangible progress and provide feedback throughout the development process.
   - **Alignment with Expectations**: Continuous user feedback ensures that the final product is more closely aligned with customer expectations and requirements.
7. **Enhanced Quality**
   - **Continuous Testing**: Each increment undergoes testing, ensuring that defects are identified and addressed early. This leads to a higher quality final product.
   - **Incremental Integration**: Integrating and testing smaller increments reduces the complexity of integration and helps maintain system stability.
8. **Scalability**
   - **Scalable Approach**: The Incremental Model is scalable and can be applied to projects of varying sizes and complexities. It is particularly effective for large-scale projects where delivering the entire system at once is impractical.

Weaknesses and Limitations

- **Complexity in Integration**
  - **Integration Challenges**: As multiple increments are developed and integrated over time, the complexity of integrating these increments can increase. Ensuring that all increments work seamlessly together can be challenging.
  - **Dependency Management**: Managing dependencies between increments can be difficult. Changes in one increment may impact others, requiring careful coordination and communication.
- **Incomplete Systems Early On**
  - **Partial Functionality**: Early increments may deliver partial functionality, which may not provide immediate value to all stakeholders.
  - **Customer Dissatisfaction**: If early increments do not meet user expectations or provide limited functionality, it can lead to dissatisfaction and frustration.
- **Resource Allocation**
  - **Resource Constraints**: Allocating resources effectively across multiple increments can be challenging. Teams may need to balance working on new increments while maintaining and refining previous ones.
  - **Skill Requirements**: Different increments may require different skill sets, making it necessary to have a versatile team or to allocate specialized resources efficiently.
- **Planning and Coordination**
  - **Detailed Planning Required**: While the Incremental Model allows for flexibility, it still requires detailed planning and coordination to ensure that each increment aligns with the overall project goals and objectives.
  - **Complex Scheduling**: Coordinating the development, testing, and integration of multiple increments can create complex scheduling challenges.

Weaknesses and Limitations

- **Risk of Scope Creep**
  - **Uncontrolled Changes**: The flexibility to incorporate changes in each increment can lead to scope creep if changes are not carefully managed and controlled.
  - **Project Overruns**: If changes are frequent and significant, the project may experience cost and schedule overruns.
- **Incremental Delivery**
  - **Interdependent Increments**: Increments that are highly interdependent may require more effort to ensure that changes in one increment do not adversely affect others.
  - **Delayed Full Functionality**: Users may not experience the full functionality of the system until the final increments are delivered, which can delay the realization of complete project benefits.
- **Management Complexity**
  - **Project Management**: Managing an incremental project can be more complex than managing a linear project. It requires careful planning, coordination, and monitoring to ensure that all increments align with the overall project goals.
  - **Stakeholder Management**: Keeping stakeholders engaged and informed throughout the development of multiple increments can be challenging, especially if their expectations are not met in early increments.

Software Characteristics - Incremental Model vs Waterfall Model

- Below is the detailed comparison of the software characteristics suitable for using the Incremental Model versus the Waterfall Model

1. **Requirement Stability**
   - **Incremental Model**: Ideal for projects with evolving and flexible requirements. Allows for iterative refinement based on feedback.
   - **Waterfall Model**: Best for projects with stable and well-defined requirements that are unlikely to change.
2. **Project Size and Complexity**
   - **Incremental Model**: Suitable for large, complex systems that can be broken down into smaller, manageable increments.
   - **Waterfall Model**: More suitable for smaller, less complex projects or those with well-understood complexity.
3. **Early Delivery of Functionality**
   - **Incremental Model**: Enables early delivery of functional parts of the system, providing value to stakeholders sooner.
   - **Waterfall Model**: Full functionality is delivered only at the end of the development cycle.
4. **Risk Management**
   - **Incremental Model**: High-risk projects benefit from early identification and mitigation of risks through iterative development.
   - **Waterfall Model**: Lower-risk projects or those with well-understood risks are more suited to the Waterfall approach.
5. **Stakeholder Involvement**
   - **Incremental Model**: Requires active and continuous stakeholder involvement and feedback throughout the development process.
   - **Waterfall Model**: Limited stakeholder involvement is typically seen, primarily at the beginning and end of the project.

Software Characteristics - Incremental Model vs Waterfall Model

6. **Documentation and Audits**
   - **Incremental Model**: Continuous documentation and compliance checks with each increment ensure ongoing alignment with requirements.
   - **Waterfall Model**: Extensive documentation is created upfront and at the end of each phase, which can be less adaptable to changes.
7. **Flexibility and Adaptability**
   - **Incremental Model**: High flexibility to adapt to changes in requirements and feedback, making it suitable for dynamic environments.
   - **Waterfall Model**: Low flexibility; changes are difficult to incorporate once a phase is completed, making it less adaptable.
8. **Resource Availability**
   - **Incremental Model**: Suitable for projects with limited resources that can be allocated incrementally based on project needs.
   - **Waterfall Model**: Best for projects with consistent resource availability throughout the lifecycle.
9. **Maintenance and Enhancement**
   - **Incremental Model**: Ideal for projects requiring ongoing maintenance and enhancements, as each increment can address new needs.
   - **Waterfall Model**: More suitable for projects with minimal maintenance needs post-deployment.
10. **Time-to-Market**
    - **Incremental Model**: Suitable for projects needing a quick time-to-market for core functionalities, allowing for early user adoption.
    - **Waterfall Model**: More suitable for projects where time-to-market is less critical.

Software Characteristics - Incremental Model vs Waterfall Model

11. **Regulatory and Compliance Requirements**
    - **Incremental Model**: Projects in regulated industries benefit from continuous compliance verification and adjustments.
    - **Waterfall Model**: Suitable for projects with well-understood regulatory requirements that are stable and unlikely to change.
12. **Parallel Development**
    - **Incremental Model**: Supports parallel development of different increments or modules, enabling faster overall progress.
    - **Waterfall Model**: Emphasizes sequential development with minimal parallel activities.
13. **User-Centric Development**
    - **Incremental Model**: Ideal for projects where user feedback is critical for refining requirements and functionalities.
    - **Waterfall Model**: Best for projects with clearly defined user requirements from the outset.
14. **Integration Requirements**
    - **Incremental Model**: Suitable for projects requiring frequent integration of new features and components.
    - **Waterfall Model**: More suitable for projects with minimal integration needs until the final stages.
15. **Market-Driven Development**
    - **Incremental Model**: Competitive markets requiring regular feature updates and improvements benefit from incremental development.
    - **Waterfall Model**: More suitable for stable markets with less frequent need for updates.

Real-world Examples of Software Projects

Aerospace and Defense Systems
Example: Software for avionics systems in aircraft.
Reason: These projects require extensive upfront planning, stringent regulatory compliance, and thorough documentation. The Waterfall Model’s structured approach ensures that all requirements are met before moving to the next phase.

Medical Device Software
Example: Software for MRI machines or patient monitoring systems.
Reason: Medical device software must adhere to strict regulatory standards and undergo rigorous testing and validation. The Waterfall Model’s sequential phases support comprehensive documentation and compliance checks.

Real-world Examples of Software Projects

E-commerce Platforms
Example: Amazon, eBay.
Reason: E-commerce platforms need to continuously evolve to meet changing market demands and user expectations. The Incremental Model allows for regular updates and new feature releases.

Banking and Financial Systems
Example: Core banking systems or financial transaction processing systems.
Reason: These systems require high reliability, security, and regulatory compliance. The Waterfall Model ensures that all requirements are well-defined and met before the system goes live.

Real-world Examples of Software Projects

Social Media Applications
Example: Facebook, Twitter.
Reason: Social media platforms require frequent updates and new features to keep users engaged. The Incremental Model supports iterative development and continuous user feedback.

Mobile Applications
Example: Fitness apps like Fitbit, or productivity apps like Evernote.
Reason: Mobile apps often start with core functionalities and then add new features based on user feedback and market trends. The Incremental Model allows for early releases and iterative improvements.

Real-world Examples of Software Projects

Government Projects
Example: National tax management systems or public safety systems.
Reason: Government projects often have fixed requirements and budgets, with a need for extensive documentation and compliance with regulatory standards. The Waterfall Model’s linear approach aligns well with these needs.

Healthcare Management Systems
Example: Electronic Health Record (EHR) systems like Epic or Cerner.
Reason: These systems need to adapt to changing healthcare regulations and user needs. The Incremental Model allows for iterative updates and compliance adjustments.

Real-world Examples of Software Projects

Educational Software
Example: Online learning platforms like Coursera or Khan Academy.
Reason: Educational software needs to continuously add new courses, features, and improvements based on user feedback. The Incremental Model allows for regular updates and enhancements.

Infrastructure Software
Example: Software for power grid management or water treatment facilities.
Reason: These projects require detailed upfront planning and rigorous testing to ensure reliability and safety. The Waterfall Model provides a structured approach to meet these requirements.

Reuse-oriented Software Engineering

- In the majority of software projects, there are some forms of informal software reuse.
- This informal reuse takes place irrespective of the development process that is used.
- 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.

Reuse-oriented Software Engineering

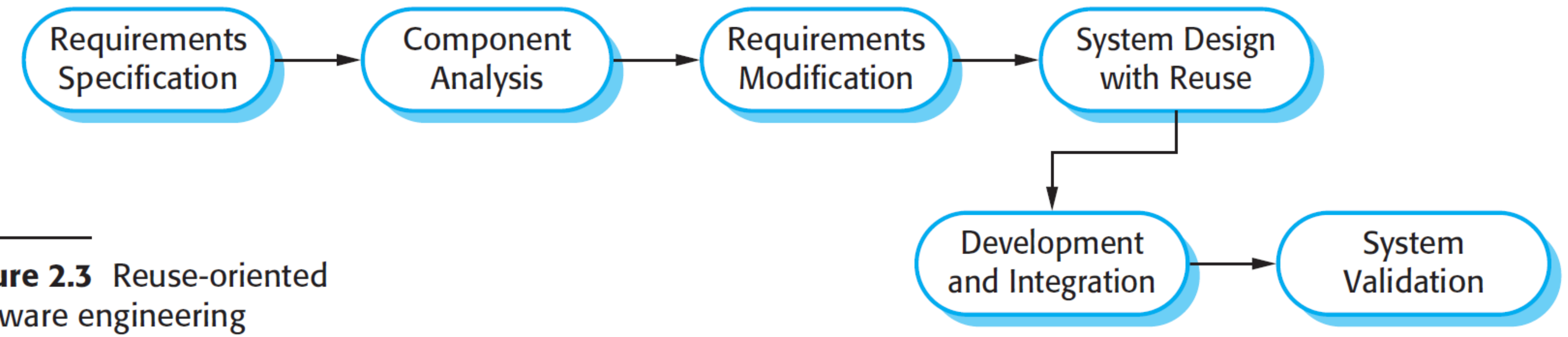

Requirements Specification
Component Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

Figure 2.3 Reuse-oriented software engineering

Reuse-oriented Software Engineering

- There are three types of software component that may be used in a reuse-oriented process:
  - <u>Web services</u> that are developed according to service standards and which are available for remote invocation.
  - <u>Collections of objects</u> that are developed as a package to be integrated with a component framework such as .NET or J2EE.
  - <u>Stand-alone software systems</u> that are configured for use in a particular environment.

Reuse-oriented Software Engineering

- Advantages
  - reducing the amount of software to be developed and so reducing cost and risks.
  - 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.

Software Process Activities

Software Engineering 1

Soon Phei Tin

Software Process Activities

- Real software processes are interleaved 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, design and implementation, validation, and evolution are organized differently in different development processes.
- In the waterfall model, they are organized in sequence
- In incremental development they are interleaved
- How these activities are carried out depends on the type of software, people, and organizational structures involved

Software Process Activities

- Software Specification
- Software Design and Implementation
- Software Validation
- Software Evolution

Software Specification

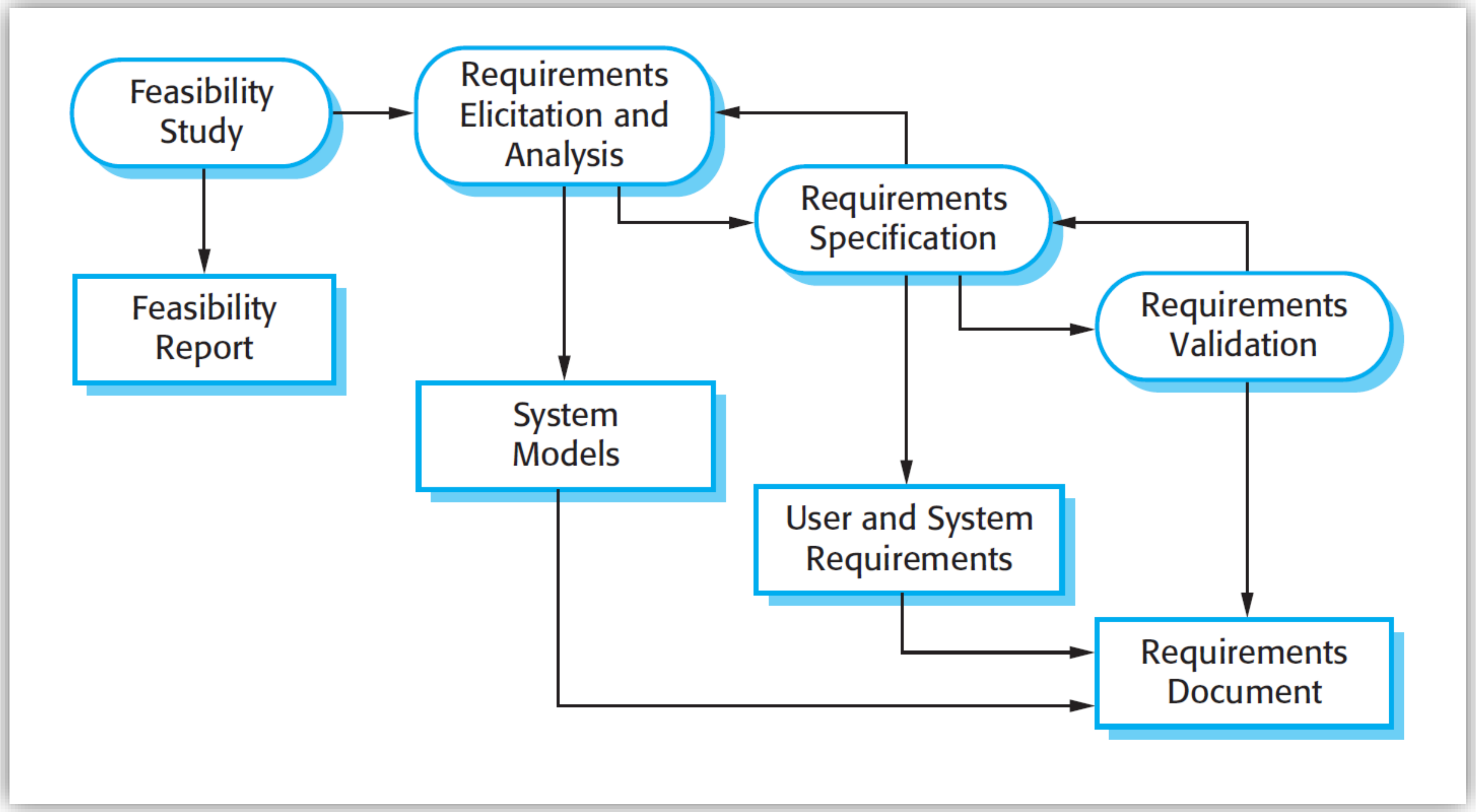
- Software specification or requirements engineering is the process of understanding and defining
  - what services are required from the system
  - identifying the constraints on the system’s operation and development.
- Requirements engineering is a particularly critical stage of the software process as errors at this stage inevitably lead to later problems in the system design and implementation.

question

Which of the following is an invalid software requirement?

A. The cost of developing the software cannot exceed 5 million dollar
B. We want our sales personnel to improve the annual sales by 25%
C. The system must be able to cope with at least 10 million hits per minutes on 24th Dec
D. I want to be able to notify customers upon completion of a process

Software Specification workflow



The activities of analysis, definition, and specification are interleaved

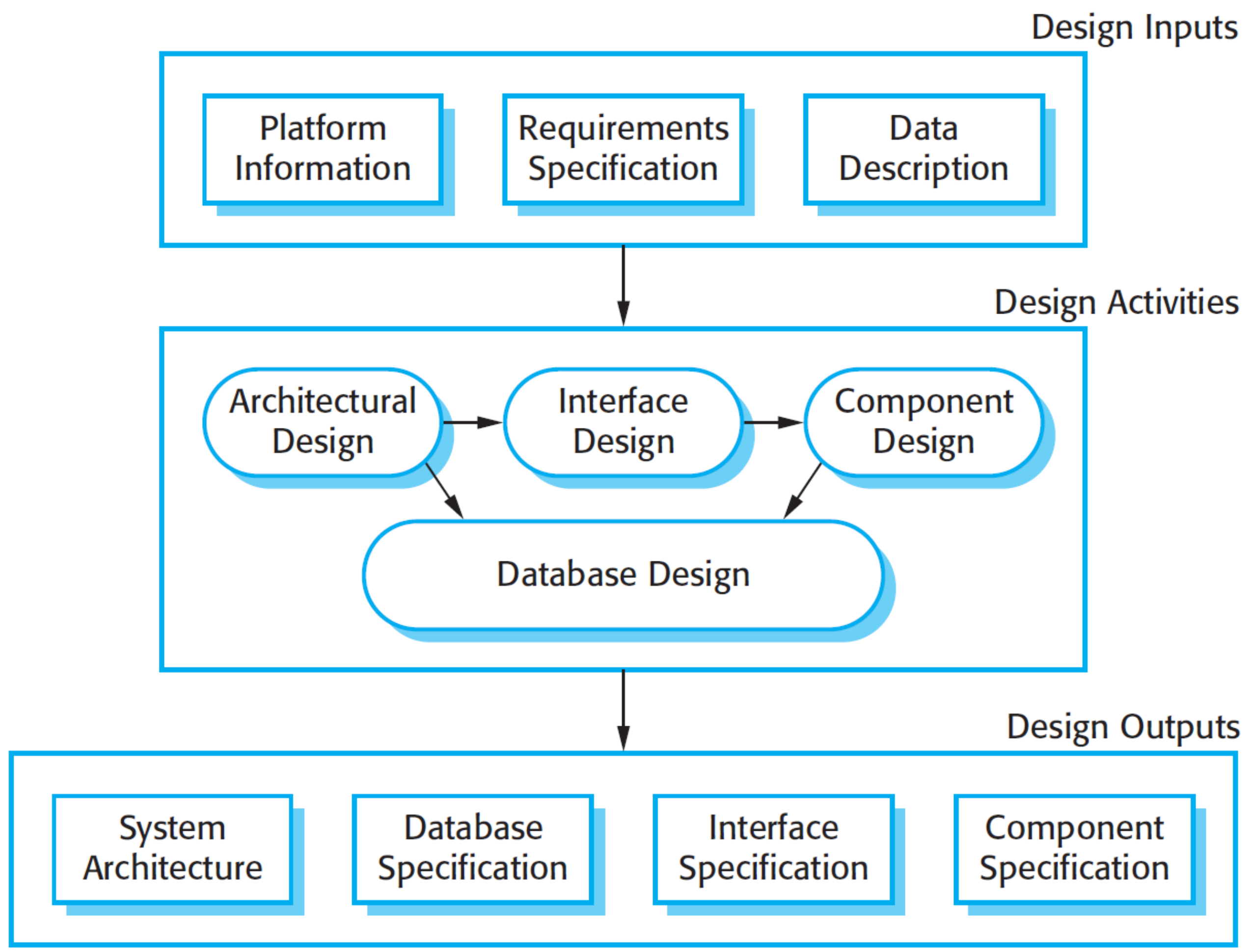
Software Specification

- The software specification process aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements.
- Requirements are usually presented at two levels of detail.
    - End-users and customers need a high-level statement of the requirements;
    - system developers need a more detailed system specification.
- There are four main activities in the requirements engineering process:
    - Feasibility study
    - Requirements elicitation and analysis
    - Requirements specification
    - Requirements validation

Software Design and Implementation

- The design and implementation stage (activity) of the software process is to convert a system specification into an executable system.
- It always involves processes of software design and programming but, if an incremental approach to development is used, may also involve refinement of the software specification.
- A software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used.

Software Design and Implementation workflow



Software Design and Implementation

- Designers do not arrive at a finished design immediately but develop the design iteratively. They add formality and detail as they develop their design with constant backtracking to correct earlier designs.

Software Design and Implementation

- 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. This interface specification must be unambiguous.

Software Design and Implementation

- Design activities
  - Component design, where you take each system component and design how it will operate.
    - a simple statement of the expected functionalities
    - a list of changes to be made to a reusable component
    - a detailed design model (model-driven approach).
  - Database design, where you design the system data structures and how these are to be represented in a database.

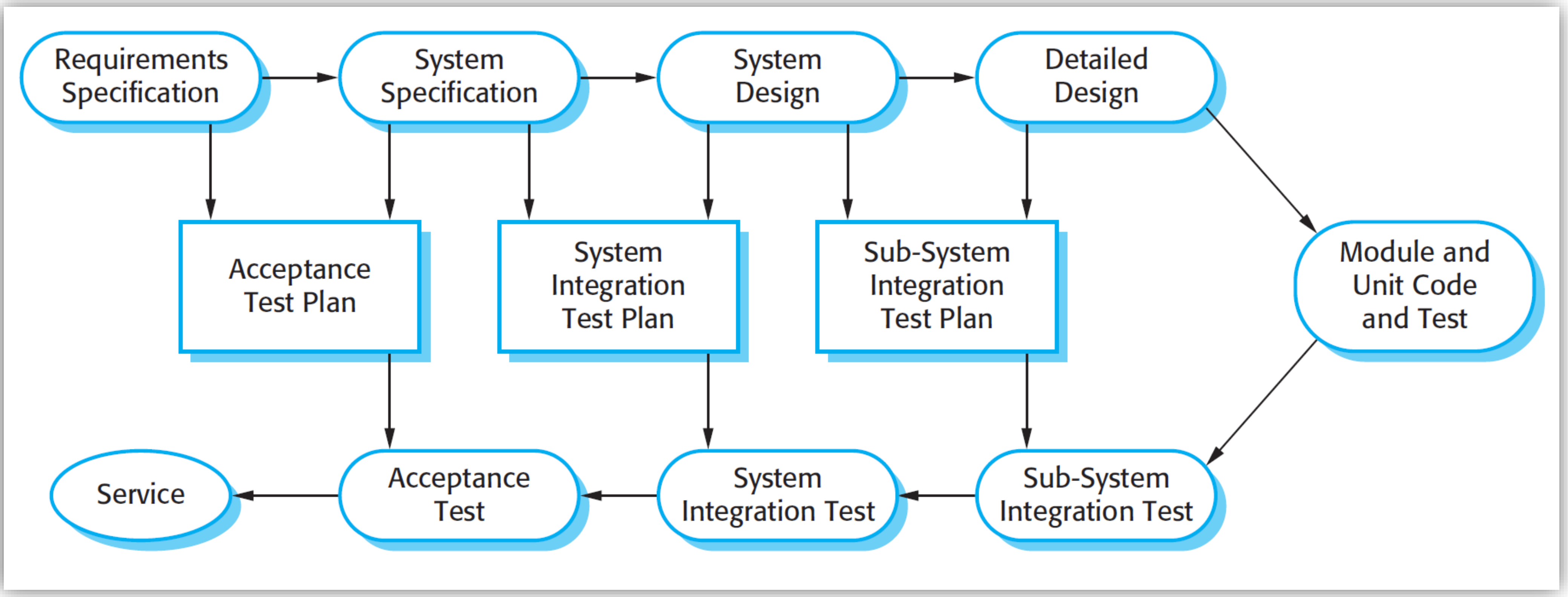
Software Design and Implementation

- The detail and representation of the design output vary considerably.
    - For critical systems, detailed design documents setting out precise and accurate descriptions of the system must be produced.
    - If a model-driven approach is used, these outputs may mostly be diagrams.
    - Where agile methods of development are used, the outputs of the design process may not be separate specification documents but may be represented in the code of the program.

Software Validation

- Software validation or, more generally, verification and validation (V&V) is intended to show that
  - a system both conforms to its specification
  - it meets the expectations of the system customer
- Validation techniques: -
  - Program testing, where the system is executed using simulated test data, is the principal validation technique
  - Validation may also involve checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development

Software Validation


Requirements Specification
System Specification
System Design
Detailed Design
Acceptance Test Plan
System Integration Test Plan
Sub-System Integration Test Plan
Module and Unit Code and Test
Service
Acceptance Test
System Integration Test
Sub-System Integration Test

Software Validation

- A three-stage testing process involved testing for system components then the testing for an integrated system and, finally, the testing of the system with the customer’s data

Software Validation

- The stages in the testing process are:
  - Development testing - The components making up the system are tested by the people developing the system. Each component is tested independently, without other system components.
  - System testing - System components are integrated to create a complete system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems.
  - Acceptance testing - This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data.

Software Validation

- Normally, component development and testing processes are interleaved. Programmers make up their own test data and incrementally test the code as it is developed.
- If an incremental approach to development is used, each increment should be tested as it is developed, with these tests based on the requirements for that increment.
- When a plan-driven software process is used (e.g., for critical systems development), testing is driven by a set of test plans. An independent team of testers works from these pre-formulated test plans

Software Evolution

- Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design. However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware
- Historically, there has always been a split between the process of software development and the process of software evolution (software maintenance).
- People think of software development as a creative activity in which a software system is developed from an initial concept through to a working system. However, they sometimes think of software maintenance as dull and uninteresting
- Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process

The end
Thank you

Appendix 1

- Fundamental Process Activities
  - **Specification -** Specification involves defining what the software should do. This phase includes the following sub-activities:
    - **Requirements Elicitation**: Gathering requirements from stakeholders through interviews, surveys, workshops, observation, and other techniques.
    - **Requirements Analysis**: Analyzing the gathered requirements to ensure they are complete, clear, and feasible. This may involve prioritizing requirements and resolving conflicts.
    - **Requirements Documentation**: Documenting the requirements in a structured format, often using requirements specification documents, user stories, or use cases.
    - **Requirements Validation**: Ensuring that the documented requirements accurately reflect the needs and expectations of stakeholders. This may involve reviews, inspections, and prototyping.
    - **Requirements Management**: Managing changes to the requirements as the project progresses, ensuring that all stakeholders are aware of the changes and their implications.
  - **Design and Implementation -** Design and implementation involve creating the software architecture and writing the code. This phase includes the following sub-activities:
    - **Architectural Design**: Defining the high-level structure of the software, including the main components, their interactions, and the technologies to be used.
    - **Detailed Design**: Specifying the internal details of each component, including algorithms, data structures, and interfaces.
    - **Prototyping**: Creating prototypes to explore design options and validate design decisions with stakeholders.
    - **Coding**: Writing the actual source code for the software components.
    - **Code Reviews and Inspections**: Conducting formal and informal reviews of the code to identify and correct issues early in the development process.
    - **Unit Testing**: Testing individual components or units of code to ensure they work as intended.
    - **Integration**: Combining individual components into a cohesive system and ensuring they work together correctly.
    - **Build Management**: Automating the process of compiling and assembling the software.

Appendix 1

- Fundamental Process Activities
  - **Validation -** Validation involves ensuring that the software meets its requirements and is free of defects. This phase includes the following sub-activities:
    - **Test Planning**: Defining the scope, objectives, and approach for testing. Creating test plans that outline the testing strategy, resources, schedule, and deliverables.
    - **Test Case Development**: Creating detailed test cases that specify the inputs, expected results, and execution conditions for each test.
    - **Test Execution**: Running the test cases and recording the results.
    - **Defect Tracking**: Identifying, logging, and tracking defects found during testing. This may involve using defect tracking tools.
    - **Regression Testing**: Re-running previously conducted tests to ensure that changes or fixes have not introduced new defects.
    - **Acceptance Testing**: Conducting tests to ensure that the software meets the acceptance criteria defined by the stakeholders and is ready for deployment.
  - **Evolution & Maintenance -** Maintenance involves making changes to the software after it has been deployed. This phase includes the following sub-activities:
    - **Corrective Maintenance**: Fixing defects and issues identified after the software is deployed.
    - **Adaptive Maintenance**: Modifying the software to adapt to changes in the environment, such as new operating systems, hardware, or third-party software.
    - **Perfective Maintenance**: Enhancing the software to improve performance, usability, or maintainability. This may involve refactoring code, optimizing algorithms, or improving the user interface.
    - **Preventive Maintenance**: Making changes to prevent future issues, such as updating libraries, improving security, or adding logging and monitoring capabilities.
    - **Release Management**: Planning and coordinating the release of updates and new versions of the software.
    - **User Support**: Providing support to users, including troubleshooting issues, answering questions, and providing training.

Appendix 2

- Supporting Process Activities
  - **Project Management**
    - **Planning**: Defining the scope, objectives, and resources required for the project. Creating a project plan with timelines and milestones.
    - **Scheduling**: Allocating time and resources to different tasks and activities.
    - **Risk Management**: Identifying, analyzing, and mitigating risks that could affect the project.
    - **Monitoring and Control**: Tracking progress, ensuring that the project stays on schedule and within budget, and making adjustments as needed.
  - **Configuration Management**
    - **Version Control**: Managing changes to the software code, documents, and other artifacts.
    - **Change Management**: Handling requests for changes to the software and ensuring that changes are implemented in a controlled manner.
    - **Build Management**: Automating the process of compiling and assembling the software.
  - **Deployment**
    - **Release Management**: Planning and coordinating the release of the software to users.
    - **Installation and Configuration**: Ensuring that the software is correctly installed and configured in the target environment.
    - **User Training and Support**: Providing training and support to users to ensure they can effectively use the software.

Appendix 2

- Supporting Process Activities
  - **Documentation**
    - **User Documentation**: Creating manuals, guides, and help systems for end-users.
    - **Technical Documentation**: Documenting the design, architecture, and code for developers and maintainers.
  - **Process Improvement**
    - **Process Assessment**: Evaluating the current software development process to identify strengths and weaknesses.
    - **Process Improvement Initiatives**: Implementing changes to improve the efficiency and effectiveness of the software development process

Appendix 3 – Incremental and Agile

- Agile Methodologies and the Incremental Model are both iterative approaches to software development, but they have distinct philosophies, practices, and focuses.
- **Agile Methodologies** are highly flexible, emphasize customer collaboration, and adapt to changes quickly. They focus on delivering small, functional increments in short, time-boxed iterations with continuous customer feedback.
- The **Incremental Model** is more structured, with a focus on delivering functionality in planned increments. It allows for changes but is less adaptive compared to Agile, with more emphasis on documentation and planning.
- Both approaches aim to deliver functional software incrementally, but Agile methodologies place a stronger emphasis on flexibility, customer involvement, and iterative improvement.