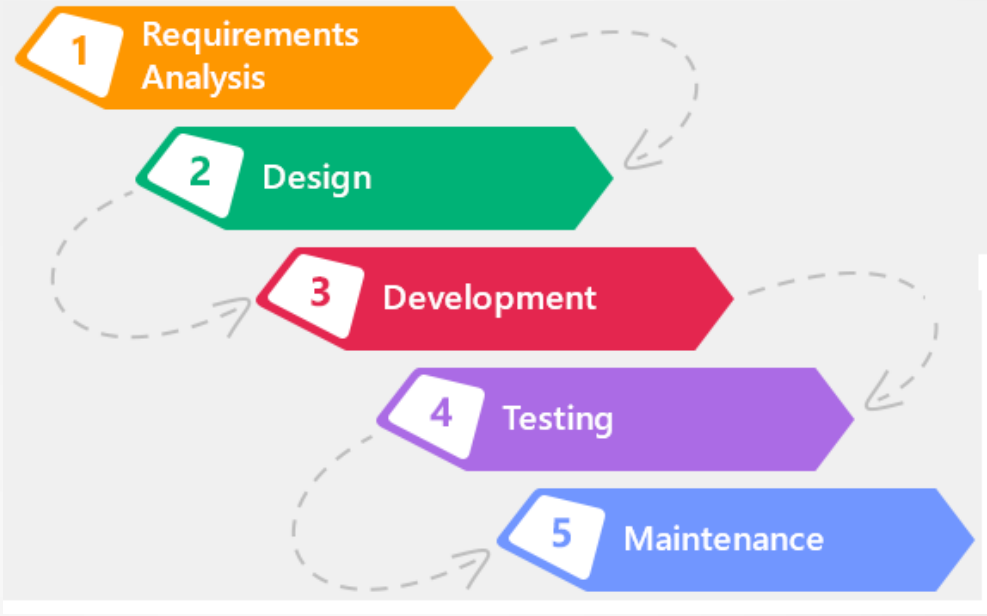


Chapter 2: Software Processes

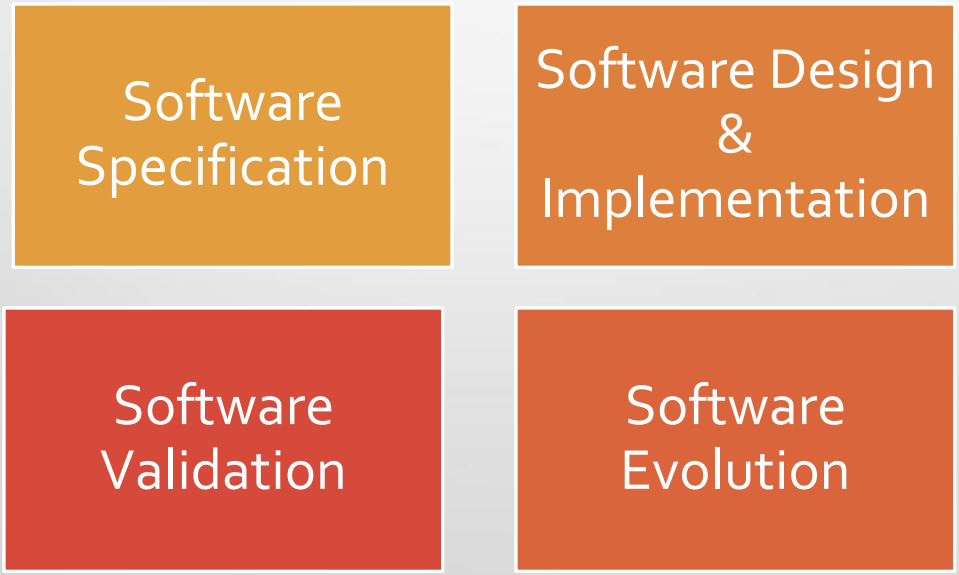
Objectives

When you have read this chapter, you will:

- understand the concepts of software processes and software process models;
- have been introduced to three general software process models and when they might be used;
- know about the fundamental process activities of software requirements engineering, software development, testing, and evolution;
- understand why processes should be organized to cope with changes in the software requirements and design;

Software Processes: Introduction

- A software process is a set of related activities that leads to the production of a software product.
- There are many different software processes, but all must include four activities that are fundamental to software engineering are:



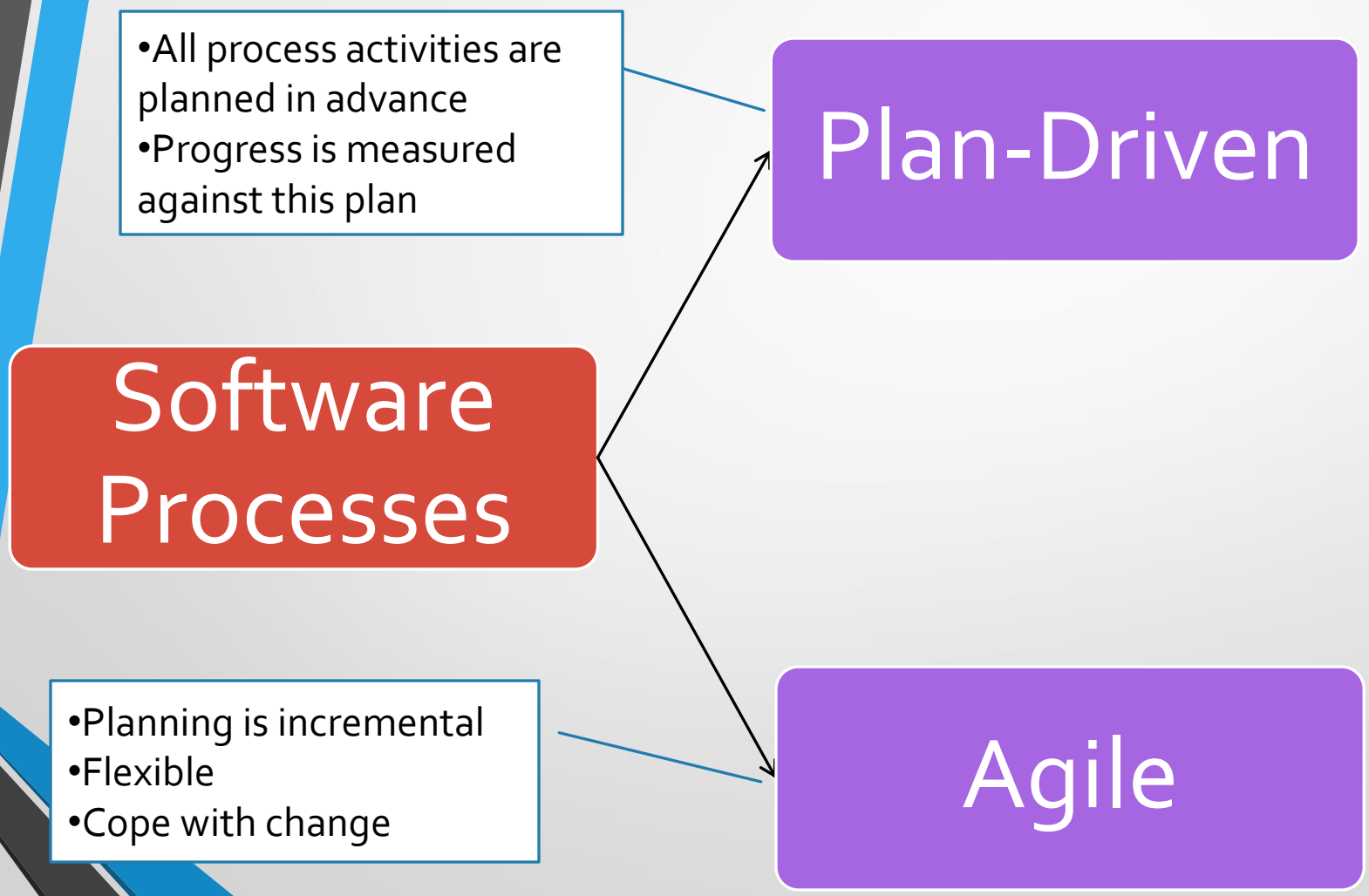
Software Processes: Introduction Cont.

- When we talk about processes, it is not only about the activities and their order. It is also about:
  - **Products: outcomes of each activity, example: Software Architecture Document**
  - **Roles: programmer, designer, analyst, tester, team leader, etc.**
  - **Pre and Post Conditions: example before architectural design begins:**
    - **Software requirements must be ready and approved before building software architecture.**
    - **Postcondition: UML models must be designed and reviewed**.

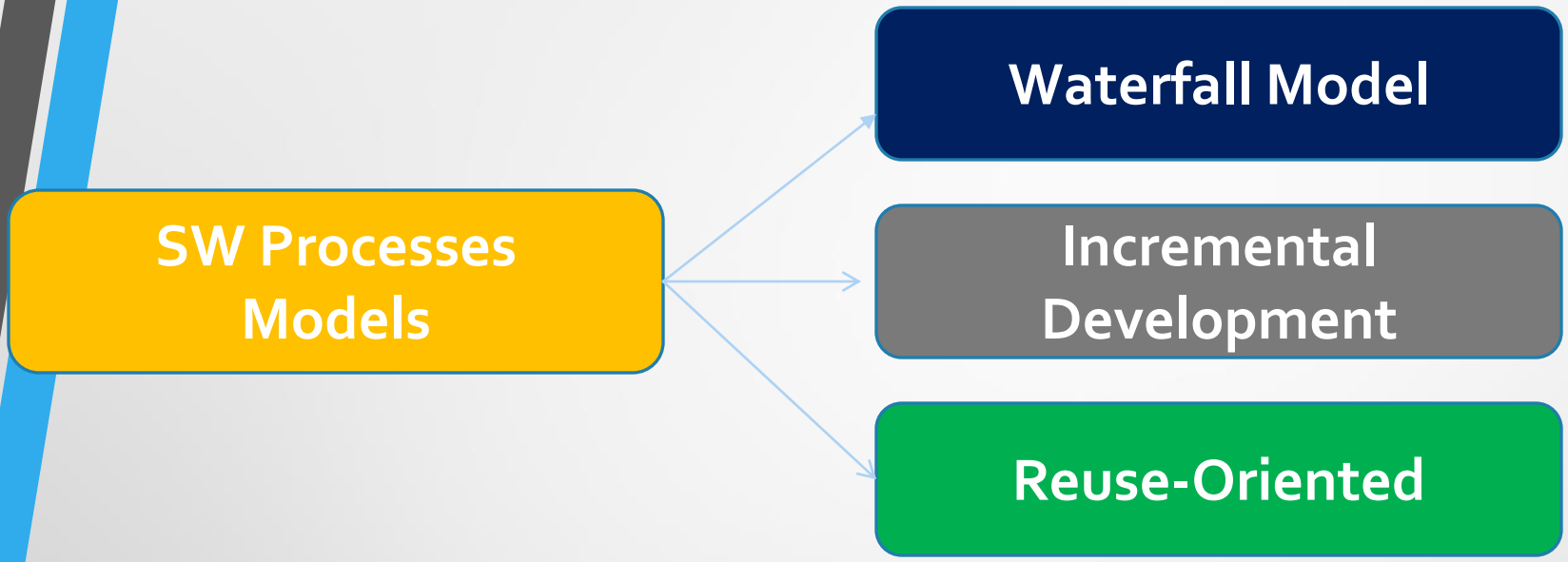
Software Processes: Introduction Cont.

- Software processes **are complex** and, like all intellectual and creative processes, **rely on people making decisions and judgments**.
- There is **no ideal process** and most organizations have developed their **own**.
- Processes have evolved to take advantage of the specific characteristics of the systems that are being developed.
  - **Critical Systems: a very structured development process is required**
  - **A Business System with rapidly changing requirements, a less formal, flexible agile process is likely to be more effective**

Two Categories of Software Processes



2.1 Software process models



- A software process model is a simplified representation of a software process.
- **These models are not mutually exclusive and are often used together, especially for large systems development.**

2.1.1 The waterfall model

- The waterfall model is an example of a plan-driven process.
- You must plan and schedule all of the process activities
- In principle, the result of each phase is one or more documents that are approved ("signed off") before starting work on them
- It is best used for critical systems, where requirements are well understood and will not change (e.g. Aviation and medical systems)
- It can also be used with small systems (mobile apps) with the help of prototypes.

2.1.1 The waterfall model Cont.

- Because of the cascade from one phase to another, this model is known as the waterfall model or software life cycle.

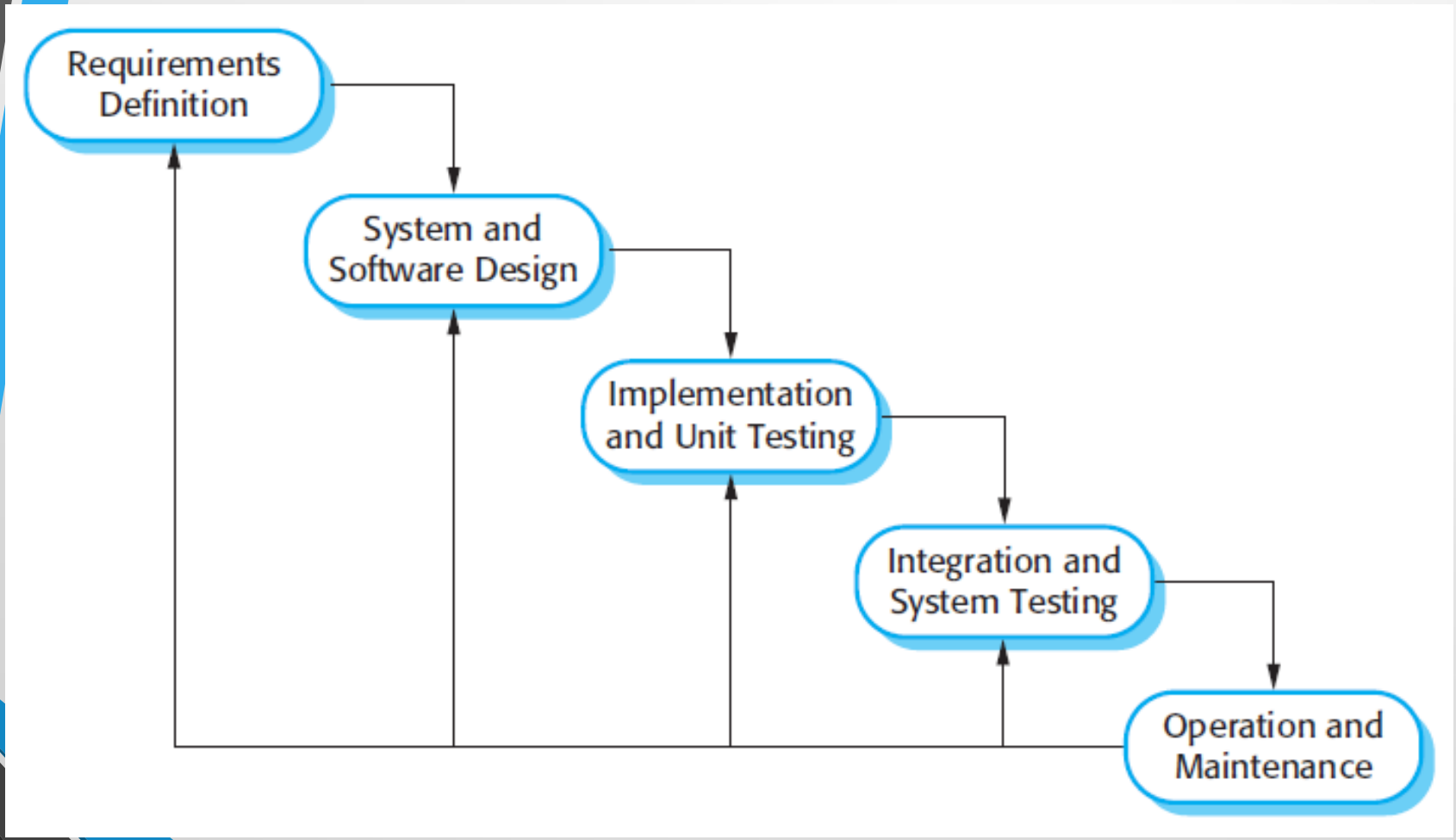


Figure 2.1 The waterfall model

2.1.1 The waterfall model Cont.

- The waterfall model is appropriate for some types of system:
  - Embedded systems: where the software has to interface with hardware systems.
  - Critical systems: where there is a need for extensive safety and security analysis of the software specification and design.
  - Large software systems that are part of broader engineering systems developed by several partner companies.

Advantages of Waterfall Model

- Developers and customers agree on what will be delivered early in the development lifecycle. This makes **planning and designing more straightforward**.
- **Progress** is more **easily monitored and measured**, as the full scope of the work is known in advance.
- Throughout the development effort, it's possible for various members of the team to be involved or to **continue with other work**, depending on the active phase of the project
- Customer presence is **not strictly required** after the requirements phase.
- The software can be designed **completely and more carefully**, based upon a more complete understanding of **all** software deliverables

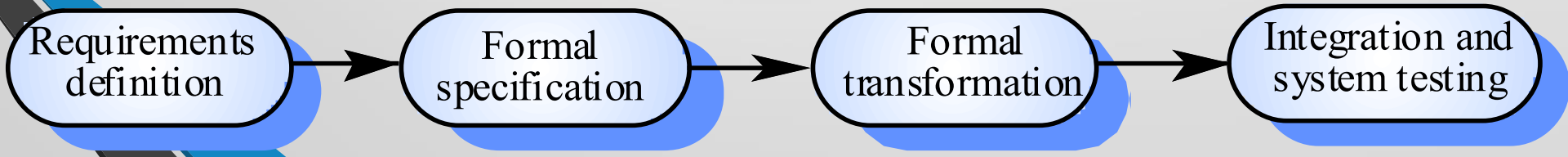
References: http://www.seguetech.com/waterfall-vs-agile-methodology/

Disadvantages of Waterfall Model

- One area which almost always falls short is the **effectiveness of requirements**.
- Gathering and documenting requirements in a way that is meaningful to a customer is often **the most difficult part** of software development.
- **Inflexible partitioning** of the project into distinct stages
- Difficult to **respond to changing** customer requirements
- Fixing **identified problems** during development are **costly** and involve significant rework.
- Another potential drawback of pure Waterfall development is the possibility that the customer will be **dissatisfied** with their delivered software product
- And **Testing** of whole system that only happens at end of project

Formal System Development

- An important variant of the waterfall model is formal system development, where a **mathematical model** of a system specification is created.
- This model is then refined, using mathematical transformations that preserve its consistency, into executable code
- Is particularly suited to the development of systems that have stringent **safety, reliability, or security** requirements.
- The formal approach simplifies the production of a safety or security case.
- Cleanroom software engineering is an example of a formal development process.

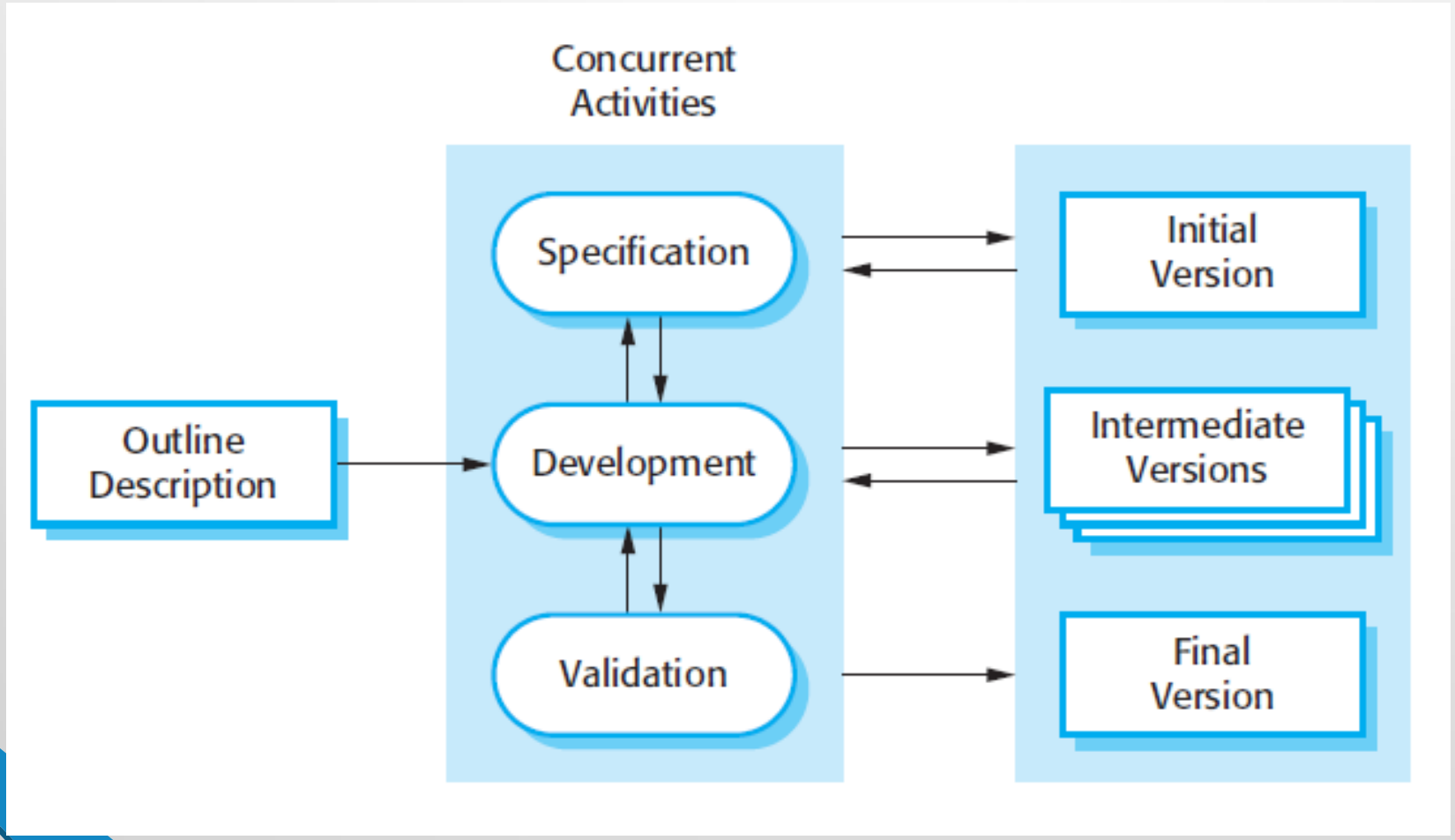


2.1.2 Incremental development

- Incremental development is based on the idea of:
  - A) developing an initial implementation,
  - B) exposing this to user comment and others
  - C) evolving it through several versions until an adequate system has been developed
- fundamental part of **agile** approaches
- Better than waterfall approach for systems whose requirements are likely to change during the development process such as most e-business, e-commerce, and personal systems.
- Can be plan-driven, agile, or a mix of both!
- Applicability:
  - For small or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime systems

Incremental Development Model

- Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities.



Incremental Development Cont.

- Benefits of incremental development
  - The **cost** of accommodating **changing customer requirements** is reduced.
  - It is easier to get customer **feedback** on the development work that has been done.
  - More **rapid delivery** and deployment of useful software to the customer is possible
  - Better fit for short time-to-market
- Problems with incremental development
  - The process is **not visible**. Managers need regular deliverables to measure progress.
  - System **structure** tends to degrade as new increments are added.
  - **Additional unplanned** iterations may be needed.
  - Customer may **not** have the required **free time** to be involved.
  - Team may **not** be located in **same place** (distributed teams)

2.1.3 Reuse-oriented software engineering

- In the majority of software projects, there is some software reuse.

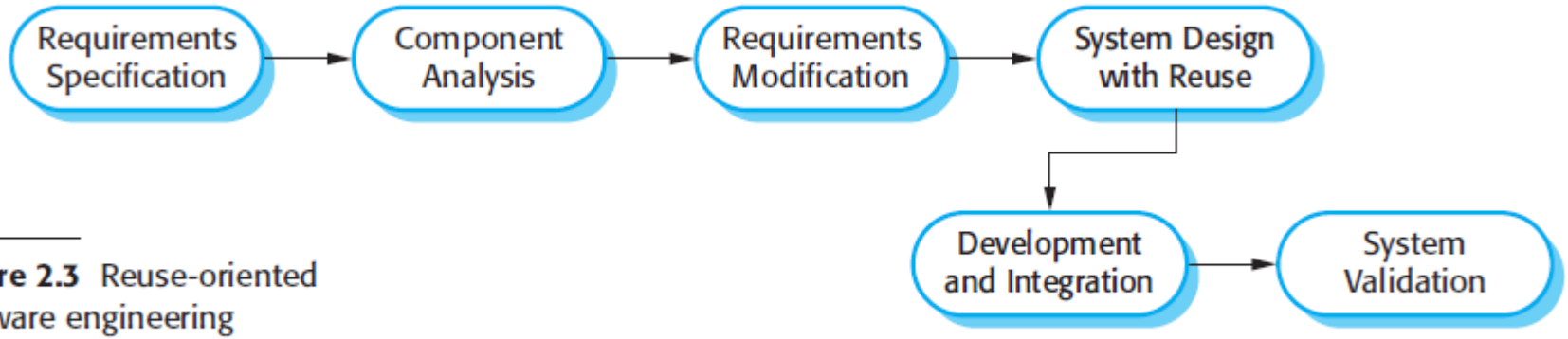


Figure 2.3 Reuse-oriented software engineering

2.1.3 Reuse-oriented software engineering

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

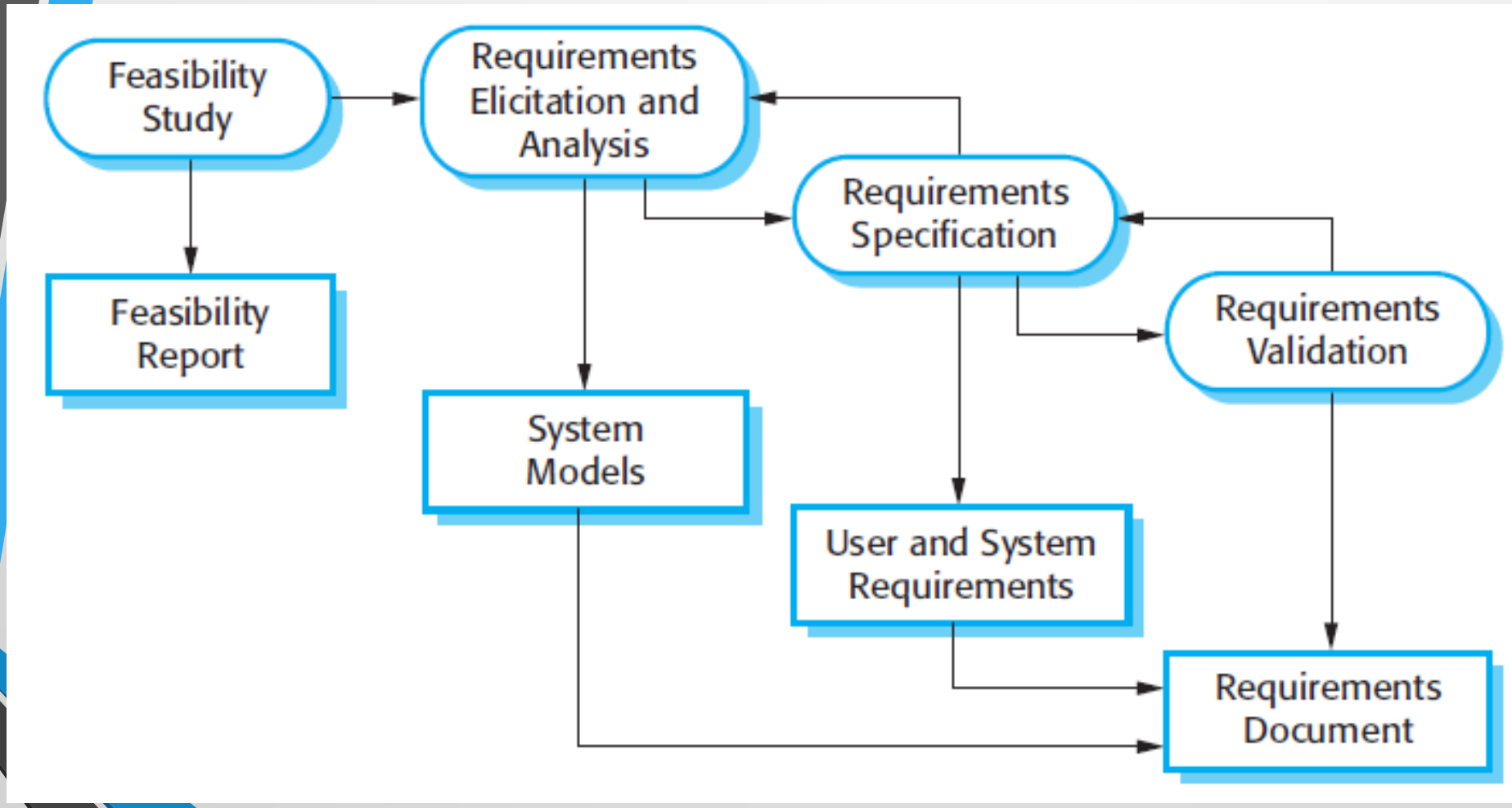
2.1.3 Reuse-oriented software engineering

- Advantages:
  - reducing the amount of software to be developed and so reducing cost and risks.
  - It usually also leads to faster delivery of the software.
- Disadvantages:
  - requirements compromises are inevitable, and this may lead to a system that does not meet the real needs of users.
  - 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.

2.2 Process activities
Software Specifications

- Software specification or requirements engineering is the process of understanding and defining:
  - what services are required from the system
  - and identifying the constraints on the system's operation and development
- Requirements are usually presented at two levels of detail.
  - User requirements: end-users and customers need a high-level statement of the requirements;
  - System requirements: system developers need a more detailed system specification.

Process Activities: Software Specifications Cont.



Process Activities: Software Design & Implementation

- 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.
- The implementation stage of software development is the process of converting a system specification into an executable system for delivery to the customer.
- For critical systems, the outputs of the design process are detailed design documents.
- Generating code from designs and diagrams (software development tools may be used to generate a skeleton program from a design)
- Programming is an individual activity, and there is no general process that is usually followed.

Process Activities: Software Design & Implementation Cont.

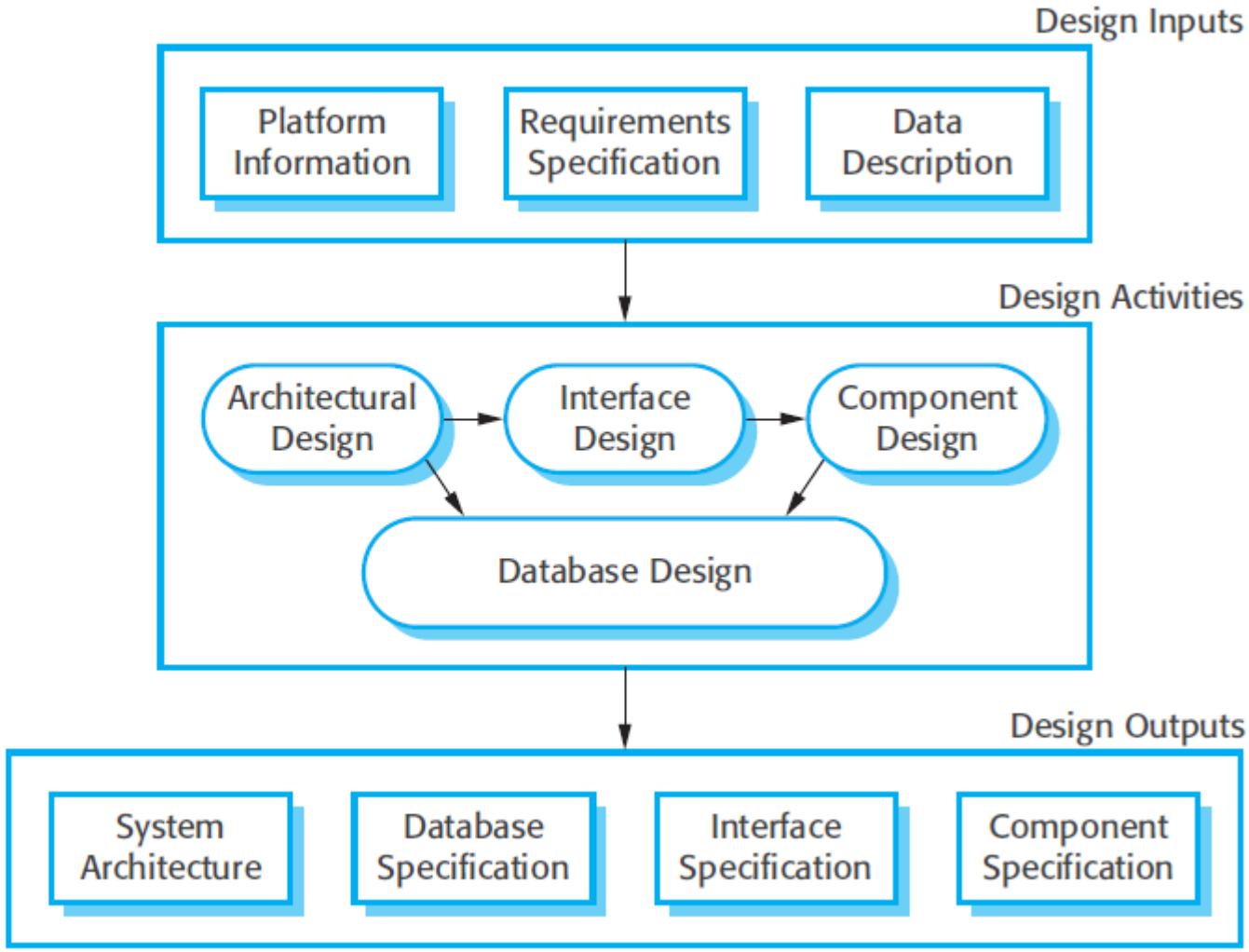


Figure 2.5 A general model of the design process

Process Activities: Software Validation

- Intended to show that a system both conforms to its **specification** and that it meets the **expectations** of the system customer.
- Program testing, where the system is executed using simulated test data, is the principal validation technique.
- **Alpha** Testing VS **Beta** Testing
  - In case of software product, Beta testing involves delivering a system to a number of potential customers/users who agree to use that system.

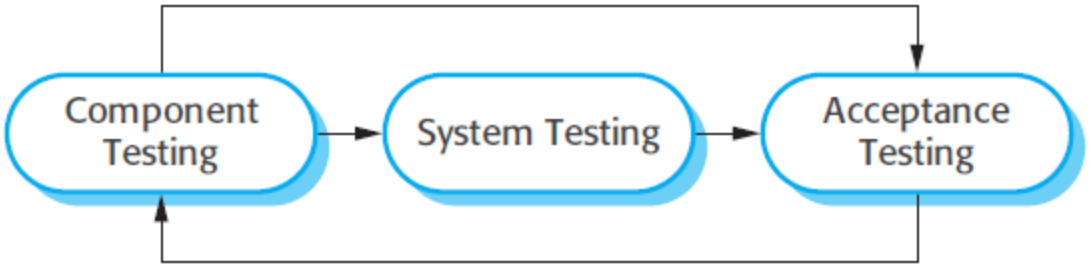


Figure 2.6 Stages of testing

Software Validation: Stages in the testing process

- Development or Component testing:
  - Each component making up the system is tested independently. (Components may be simple entities such as functions or object classes or may be coherent groupings of these entities)
  - Test automation tools, such as JUnit for Java are commonly used.
- System testing:
  - System components are integrated to create a complete system.
  - Focus on finding errors in the interactions between components, and system meets its functional and non-functional requirements
- Acceptance or Customer testing:
  - The system is tested by the system customer using real data.

Software Validation: Testing phases in a plan-driven software process

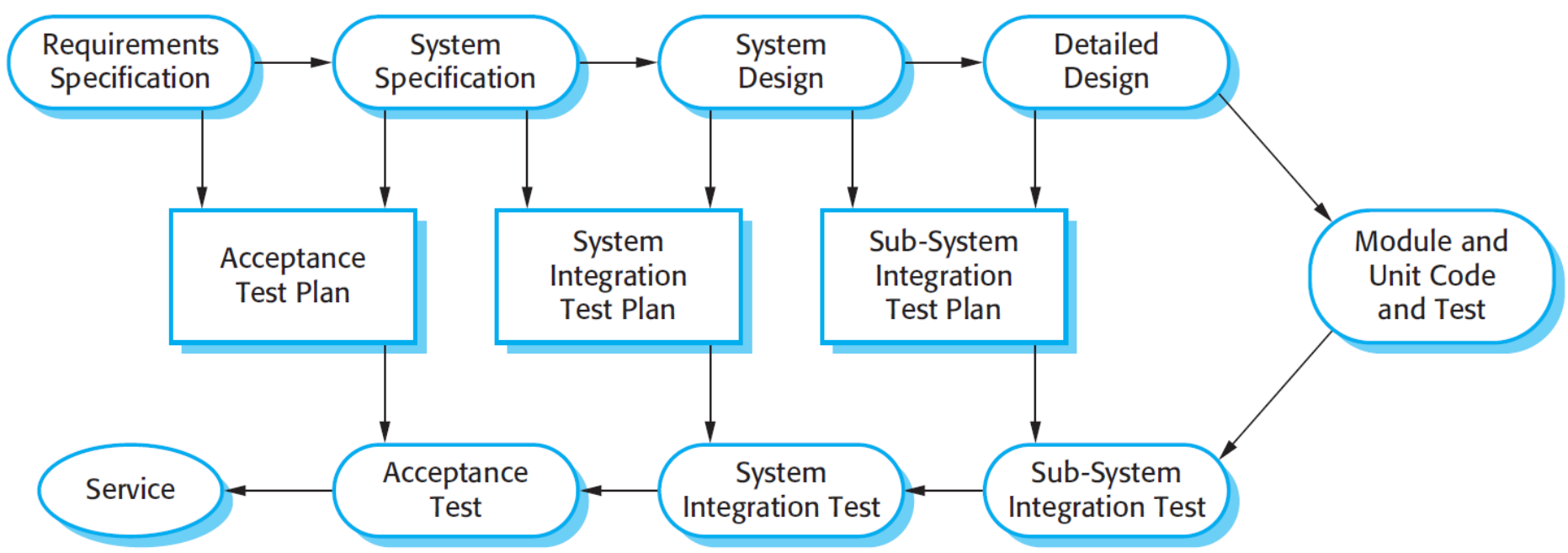


Figure 2.7: Testing phases in a plan-driven software process

2.2.4 Process Activities: Software Evolution

- During software evolution (software maintenance) process the software is continually changed over its lifetime in response to changing requirements, customer and market needs.
- The flexibility of software is one of the main reasons why more and more software is being incorporated into large, complex systems.
- Changes can be made to software at any time during or after the system development, because they are much cheaper than changes to system hardware.
- For most types of systems, the majority of costs are the costs of changing the software after it has gone into use.
- Normally, this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors that 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.

2.3 Coping with change

- Two ways of coping with change and changing system requirements:
  - **System prototyping**: a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This is a method of change anticipation as it allows users to experiment with the system before delivery and so refine their requirements.
  - **Incremental delivery**: where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance/anticipation and change tolerance.

2.3.1 Prototyping

- A prototype is an initial version of a software system that is used to demonstrate:
  - **concepts, try out design options, and find out more about the problem and its possible solutions**
- A software prototype can be used in a software development process to help anticipate changes that may be required:
  - **Requirements engineering:** elicitation and validation of system requirements.
  - **System design, UI design:** help to explore software solutions

2.3.1 Prototyping Cont.

- System prototypes allow users to see how well the system supports their work.
- They may get new ideas for requirements, find **strength** and **weakness** in the software.
- They may then propose new system requirements.
- It may also reveal **errors** and **omissions** in the requirements.
- A function described in a specification may seem useful and well defined. But, when that function is combined with other functions, **users often find that their initial view was incorrect or incomplete**.
- The system specification may then be modified to reflect their changed understanding of the requirements.

2.3.1 Prototyping Cont.

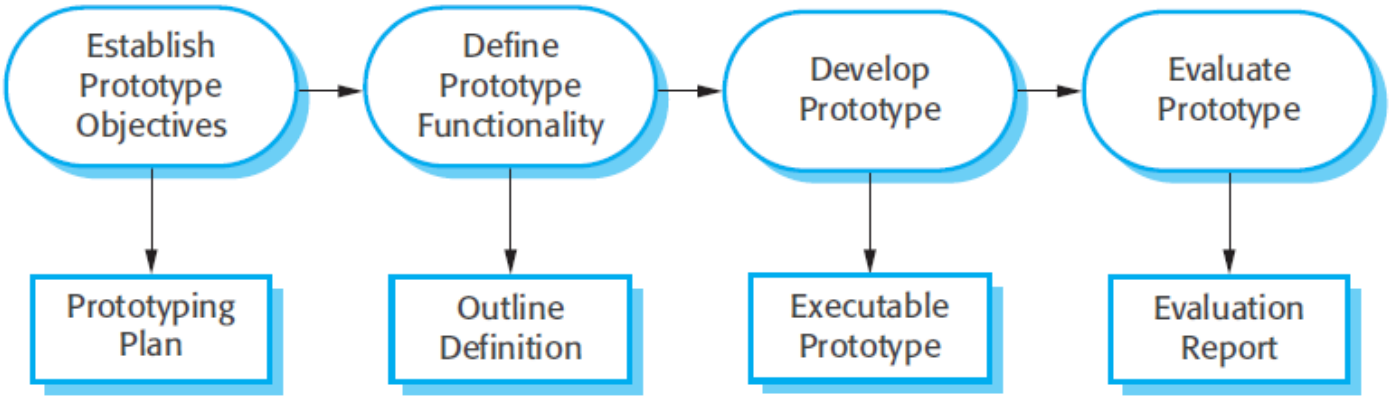


Figure 2.9 The process of prototype development

- Define prototype functionality stage: it is important to decide what to put and what to leave out of the prototype system. To reduce prototyping costs and accelerate the delivery schedule, you may leave some functionality and non-functional requirements such as response time and memory utilization.

2.3.1 Prototyping Cont.

- A system prototype may be used while the system **is being designed** to carry out design **experiments** to check the **feasibility** of a proposed design.
- For example, a **database design** may be prototyped and tested to check that it supports **efficient data access** for the most common user queries.
- A general problem with prototyping is that users may not use the prototype in the same way as they use the final system.

2.3.2 Incremental Delivery

- Is an approach to software development where **some of the developed increments** are delivered to the customer and deployed for use in an operational environment.
- In an incremental delivery process, customers identify, in outline, the services to be provided by the system.
- They identify which of the **services** are **most important** and which are least important to them.
- A number of delivery increments are then defined, with each increment providing a sub-set of the system functionality.
- With the **highest-priority** services implemented and delivered first.

Advantages of Incremental Delivery

- Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments. Unlike prototypes, these are part of the real system so there is no re-learning when the complete system is available.
- Customers do not have to wait until the entire system is delivered before they can gain value from it. The first increment satisfies their most critical requirements so they can use the software immediately.
- The process maintains the benefits of incremental development in that it should be relatively easy to incorporate changes into the system.
- As the highest-priority services are delivered first and increments then integrated, the most important system services receive the most testing.

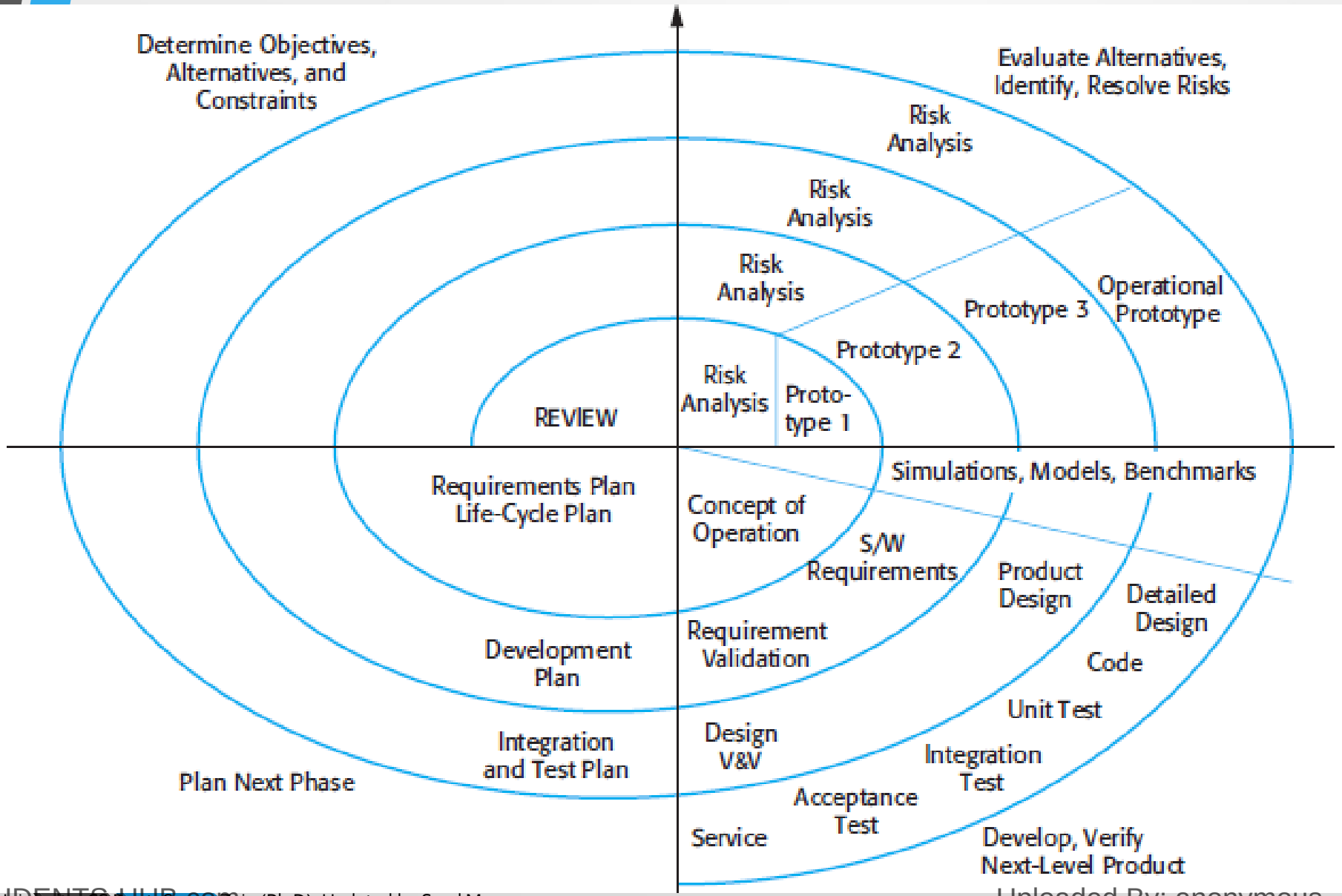
Problems with incremental delivery

- Iterative development can also be difficult when the new system is intended to replace an existing system. Users want all of the functionality of the old system and are often unwilling to experiment with an incomplete new system. Therefore, getting useful customer feedback is difficult.
- 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. In the incremental approach, there is no complete system specification until the final increment is specified. This requires a new form of contract, which large customers such as government agencies may find difficult to accommodate.

Boehm’s spiral model

- A **risk-driven** software process framework
- Risk simply means something that **can go wrong**.
- Each loop in the spiral represents a phase of the software process.
- Thus, the innermost loop might be concerned with system **feasibility**, the next loop with **requirements** definition, the next loop with system **design**, and so on.
- The spiral model combines change avoidance with change tolerance. It assumes that changes are a result of project risks and includes **explicit risk management** activities to reduce these risks.

Spiral Model of Boehm



Key points

- Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes. Examples of these general models include the waterfall model, incremental development, and reusable component configuration and integration.
- Requirements engineering is the process of developing a software specification. Specifications are intended to communicate the system needs of the customer to the system developers.
- 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. Changes are continuous, and the software must evolve to remain useful.
- 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.