# Software Engineering

**Dr. Fatma ElSayed**

Computer Science Department
fatma.elsayed@fci.bu.edu.eg

## Questions on Lecture_01

1. Compare between the two types of software products?
2. A software process is a sequence of activities, mention these activities?
3. Mention the key challenges that faces software engineering?
4. What are the key ethical responsibilities of software engineers?

# Course Information

## Contents

# Course Information

## Contents

# Chapter 2: Software Processes

## The Software Process

- 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:
  1. **Software specification:** The functionality of the software and constraints on its operation must be defined
  2. **Software design and implementation:** The software to meet the specification must be produced.
  3. **Software validation (testing) :** The software must be validated to ensure that it does what the customer wants.
  4. **Software evolution (maintenance) :** The software must evolve to meet changing customer needs.

## Software Process Descriptions

Process descriptions may also include:

- **Order** of these activities
- **Products,** which are the **outcomes** of a process activity. For example, the outcome of the activity of architectural design may be a model of the software architecture.
- **Roles,** which reflect the **responsibilities** of the people involved in the process. Examples of roles are project manager, configuration manager, programmer, etc.
- **Pre- and post-conditions,** which are statements that are true before and after a process activity has been enacted or a product produced. For example, before architectural design begins, a pre-condition could be customer approval of all requirements; a post-condition might be the review of UML models describing the architecture.

## Software Process Categories

Software processes are categorized as either:

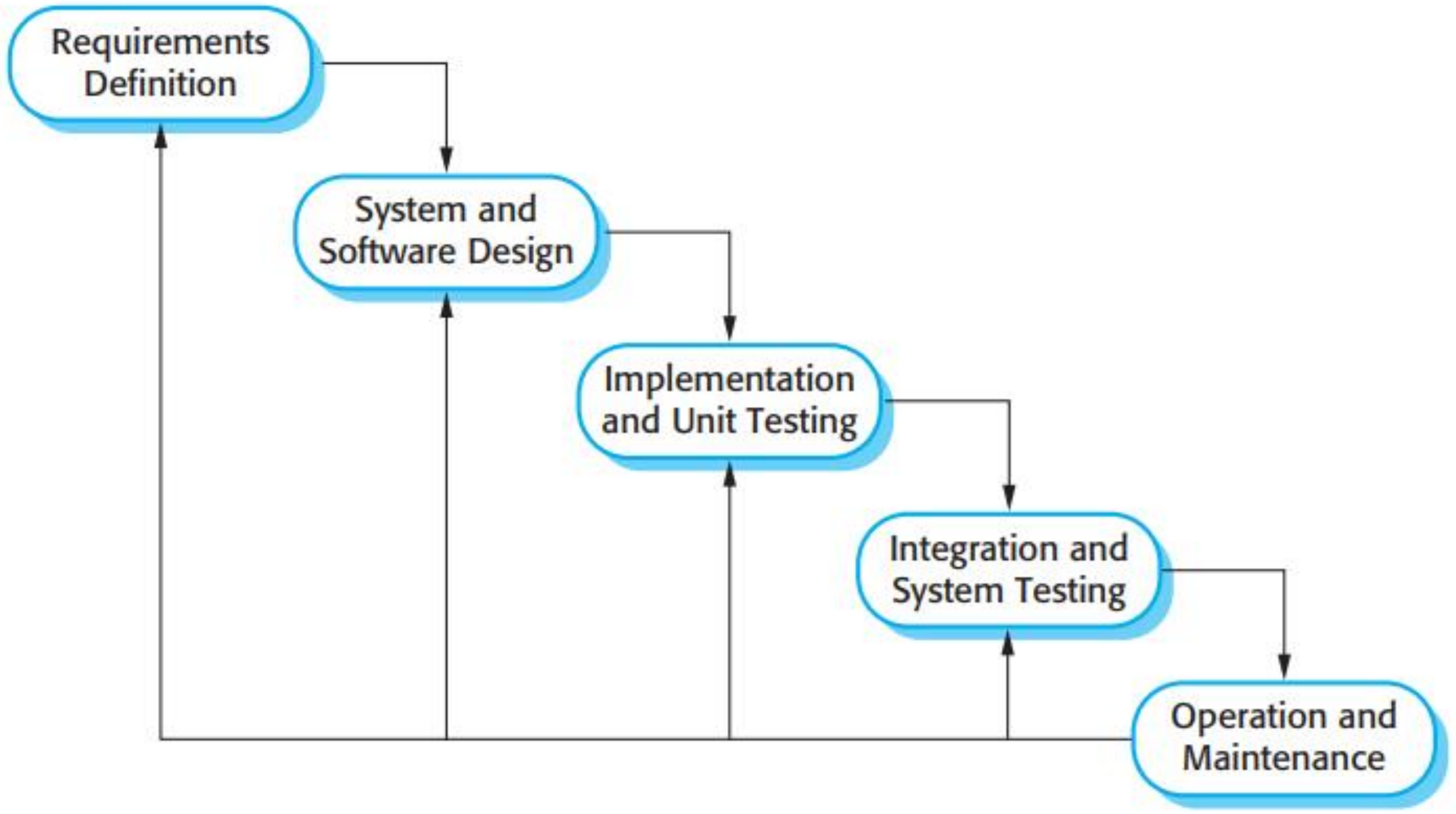
- **Plan-driven processes,** are processes where all of the process activities are planned in **advance** and progress is measured against this plan.
- **Agile processes**, which will be discussed later in details, planning is **incremental** and it is easier to change the process to reflect changing customer requirements.
- In practice, each approach is suitable for different types of software. Generally, you need to find a **balance** between plan-driven and agile processes.
- There are no right or wrong software processes.

## Software Process Models

1. **The waterfall model**
   - Plan-driven model. Separate and distinct phases of specification and development.
2. **Incremental development**
   - Specification, development and validation are interleaved. May be plan-driven or agile.
3. **Reuse-Orient Software Engineering**
   - The system is assembled from existing configurable components. May be plan-driven or agile.

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

# The Waterfall Model


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

## The Waterfall Model: Advantages & Disadvantages

### Advantages

- Simple and easy to understand and implement.
- Easy to manage: Each phase documents make the process visible so managers can monitor progress against the development plan.
- Phases are clearly defined and completed one at a time.

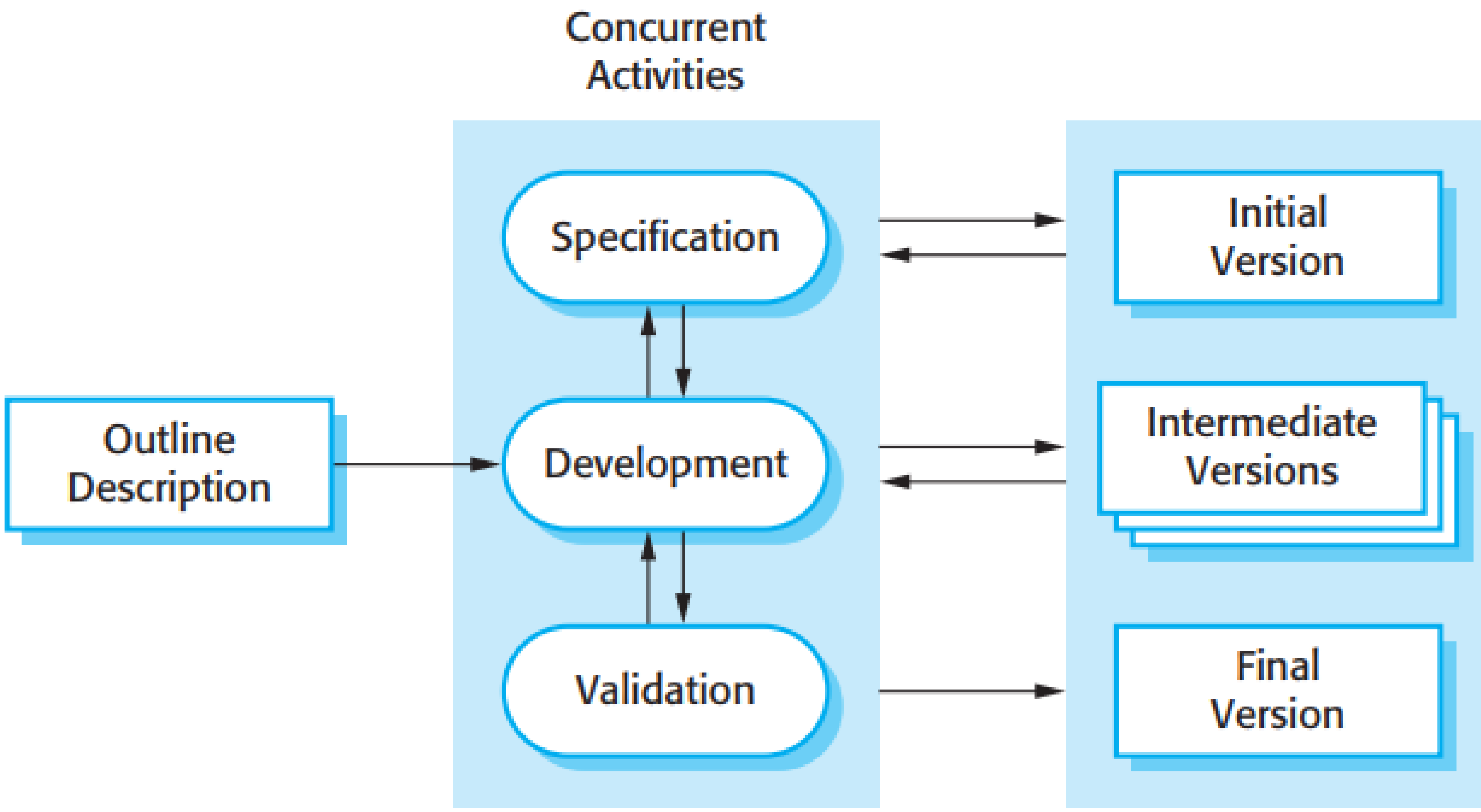
### Disadvantages

- Costs of producing and approving documents.
- No working software is produced until late stages.
- It takes a lot of time to produce the final software product.
- There is no chance to change customer requirements (contract).
- High amount of risk.

## When to Use the Waterfall Model

- 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 waterfall model should only be used when
  - The requirements are very well understood from the beginning.
  - The requirements are unlikely to change *(stable)* during system development.
  - The delivery time is not a problem.

## Incremental Development Model



## Incremental Development Model

- The system is developed as a series of versions **(increments)**, with each version adding functionality to the previous version.
- Incremental development is based on the idea of developing an initial implementation, exposing this to user comment and evolving it through several versions until a suitable system has been develop

## Incremental Development Benefits

Incremental development has **three** important benefits, compared to the waterfall model:

1. 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.
2. More **rapid delivery** and deployment of useful software to the customer is possible, even if all of the functionality has not been included.
   - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

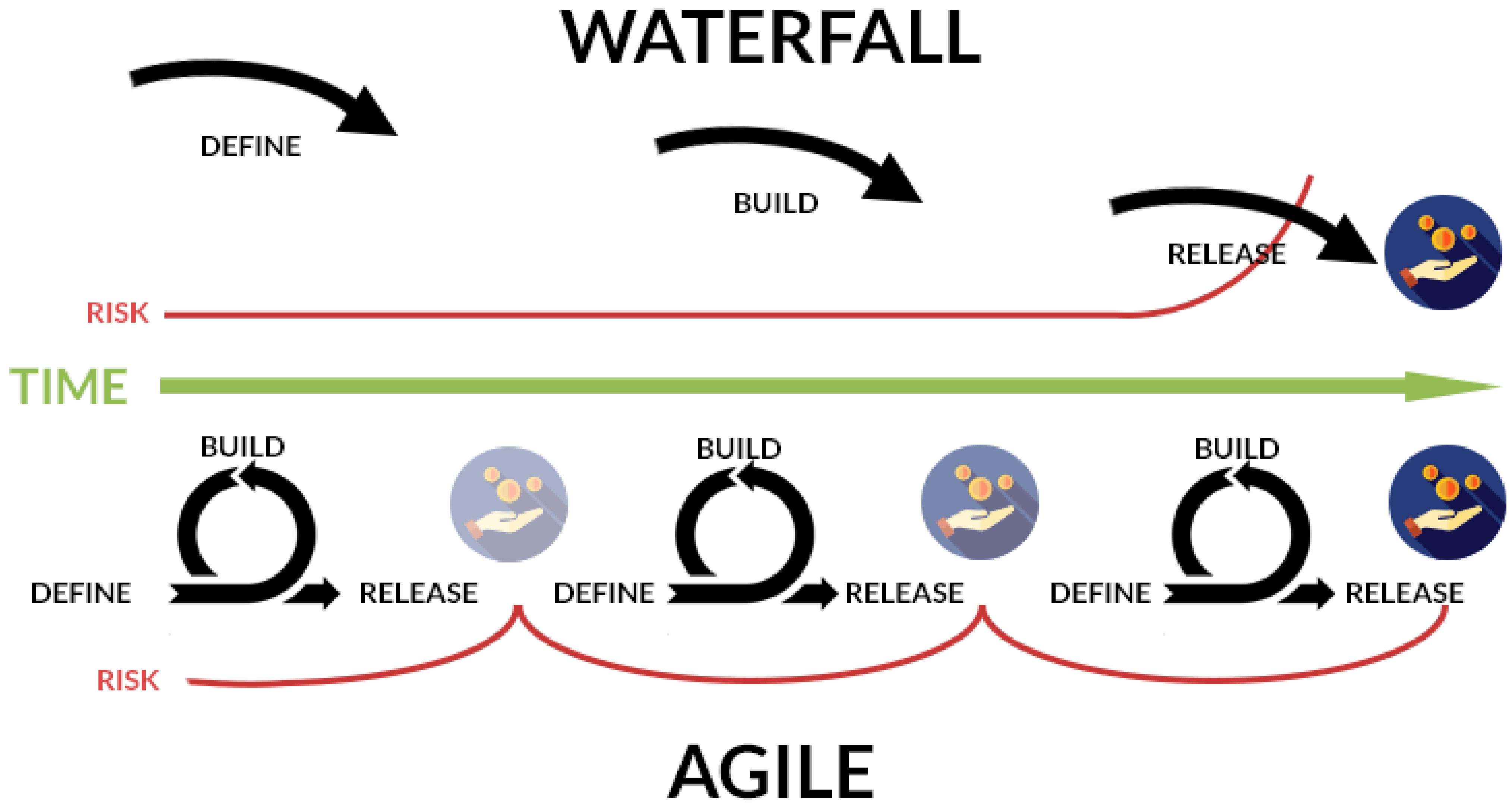
## Incremental Development Benefits

3. It is easier to get customer **feedback** on the development work that has been done.
   - 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.

## 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 are spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

## Waterfall vs Agile [risk & Value]



## 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.
- These components are called (**COTS** or commercial off-the-shelf systems).

## Reuse-Oriented Software Engineering

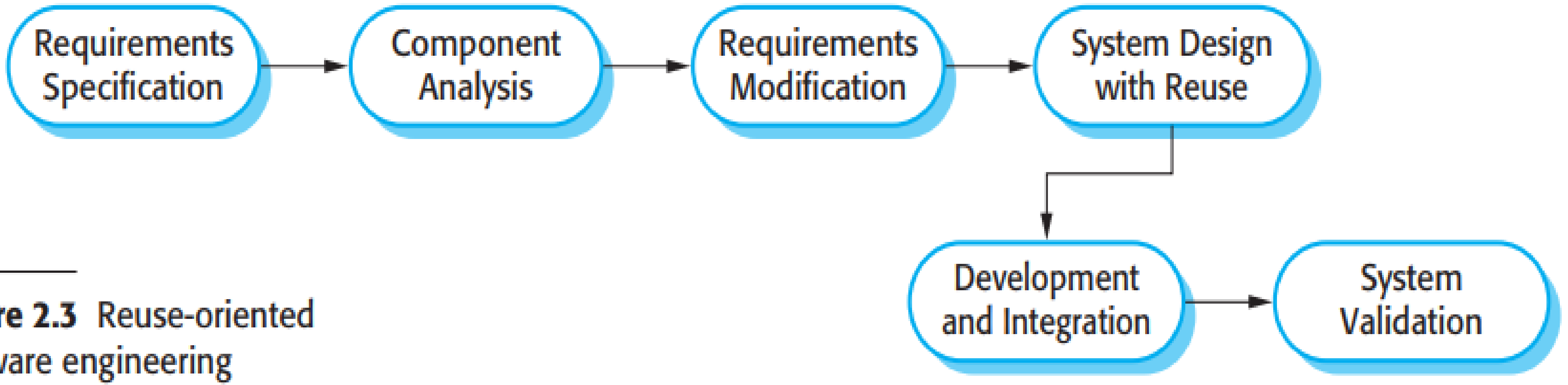

Requirements Specification
Component Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

**Figure 2.3** Reuse-oriented software engineering

## Types of Reusable Components

There are **three** types of software component that may be used in a reuse-oriented process:

1. Stand-alone application systems (sometimes called COTS) that are configured for use in a particular environment.

2. Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.

3. Web services that are developed according to service standards and which are available for remote invocation.

## Advantages & Disadvantages

**Advantages**

- Reduces the amount of software to be developed and so reducing cost and risks.
- leads to faster delivery of the software.

**Disadvantages**

- Loss of control over evolution of reused system elements.
- Requirements compromises are inevitable and this may lead to a system that does not meet the real needs of users.

## Coping with Change

## 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

## Reduce the Costs of Rework

- There are two related approaches that may be used to reduce the costs of rework:

**1. Change avoidance:**

- Where the software process includes activities that can predict 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.

**2. Change tolerance:**

- Where the process is designed so that changes can be accommodated at a relatively low cost. This normally involves some form of **incremental development**. Proposed changes may be implemented in increments that have not yet been developed.

## Coping with Changing Requirements

There are two ways of coping with change and changing system requirements.

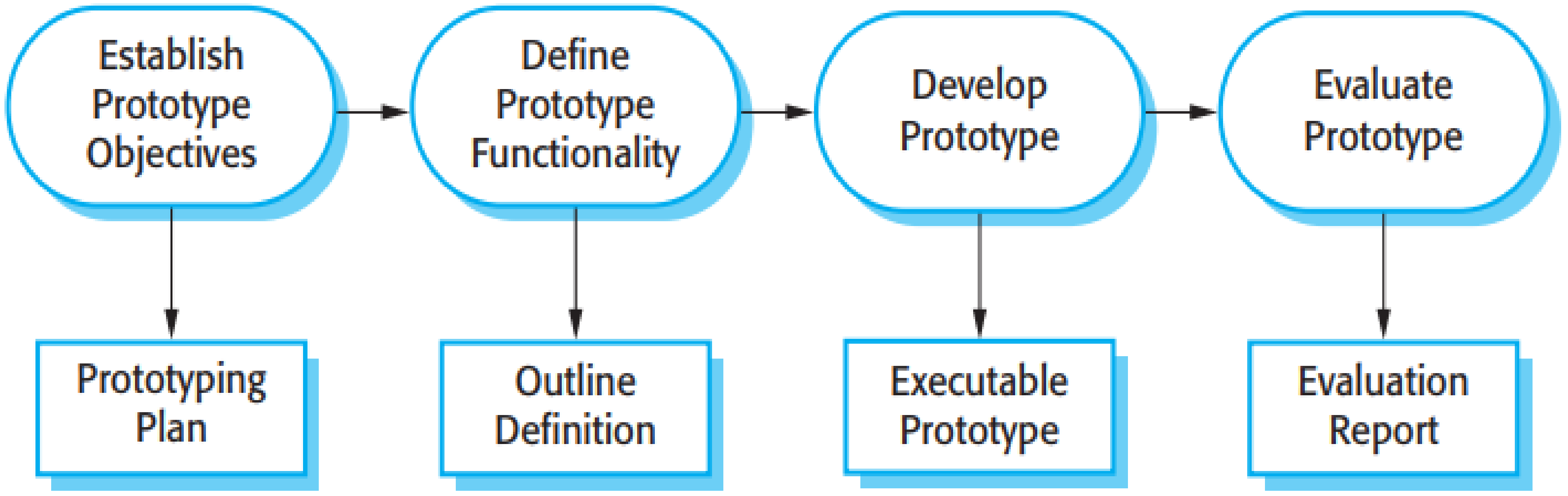
1. **System prototyping:**
   - Where a version of the system or part of the system is developed quickly to check the customer’s requirements and the feasibility of some design decisions. This supports change avoidance.
2. **Incremental delivery:**
   - Where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.

## Software 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 prototype can be used in:
  - The **requirements engineering** process to help with requirements gathering and validation. Allows users to get new ideas for requirement, propose new system requirements
  - In **design processes** to explore options and support user interface design process (For example, a database design may be prototyped and tested to check that it supports efficient data access for the most common user queries)

## The Process of Prototype Development



## Prototype Development

- May involve *leaving out functionality* to reduce prototyping costs and accelerate the delivery schedule,
  - 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 adjust the system to meet **non-functional** requirements;
  - Prototypes are normally **undocumented**;
  - The prototype structure is usually degraded through rapid change;
  - The prototype probably will **not meet** normal organizational quality **standards**.

## Incremental Delivery

- Rather than deliver the system as **a single delivery**, the development and delivery is broken down into **increments** with each increment delivering part of the required functionality.
- User requirements are **prioritised** and the highest priority requirements are included in early increments.
- As new increments are completed, they are **integrated** with existing increments so that the system **functionality improves** with each delivered increment

## Incremental Development and Delivery

- **Incremental development**
  - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment by exposing it to customers for comment, **without** actually **delivering** it and deploying it in the customer's environment.
  - Normal approach used in agile methods;
- **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

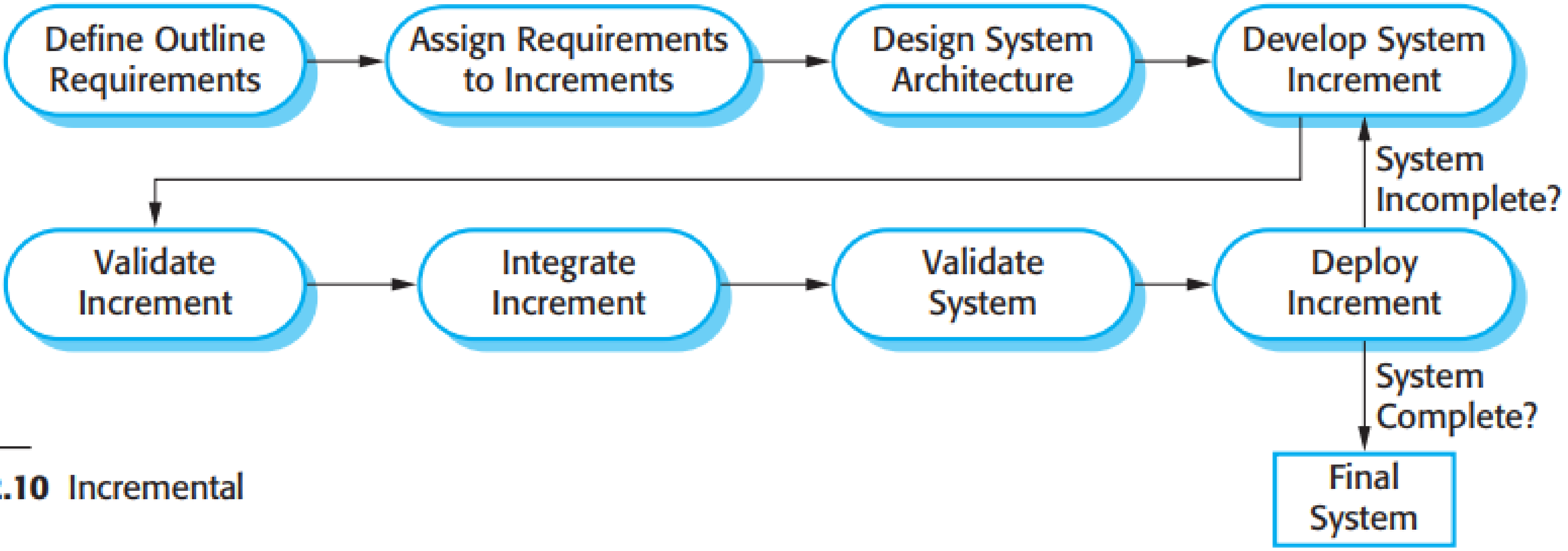


**Figure 2.10** Incremental delivery

## Incremental Delivery Advantages

- 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.
- Early increments serve as prototypes, providing customers with experience that informs future system requirements, but they are part of the real system, eliminating the need for re-learning.
- The priority delivery of services, followed by integration, ensures the most crucial system services undergo the most testing, reducing the likelihood of software failures.
- Lower risk of overall project failure.

## Incremental Delivery Problems

1. 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.
2. The essence of iterative processes is that the specification is developed in conjunction with the software. However, this conflicts with 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.

# Thank You