SWE209
Object Oriented Analysis and Design

Object Design - 1 (Introduction and Reuse)

Note

- This presentation is based on the slides and content of the course main textbook.
- Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014
- https://ase.in.tum.de/lehrstuhl_1/component/content/article/43-books/217

Agenda

Big Picture

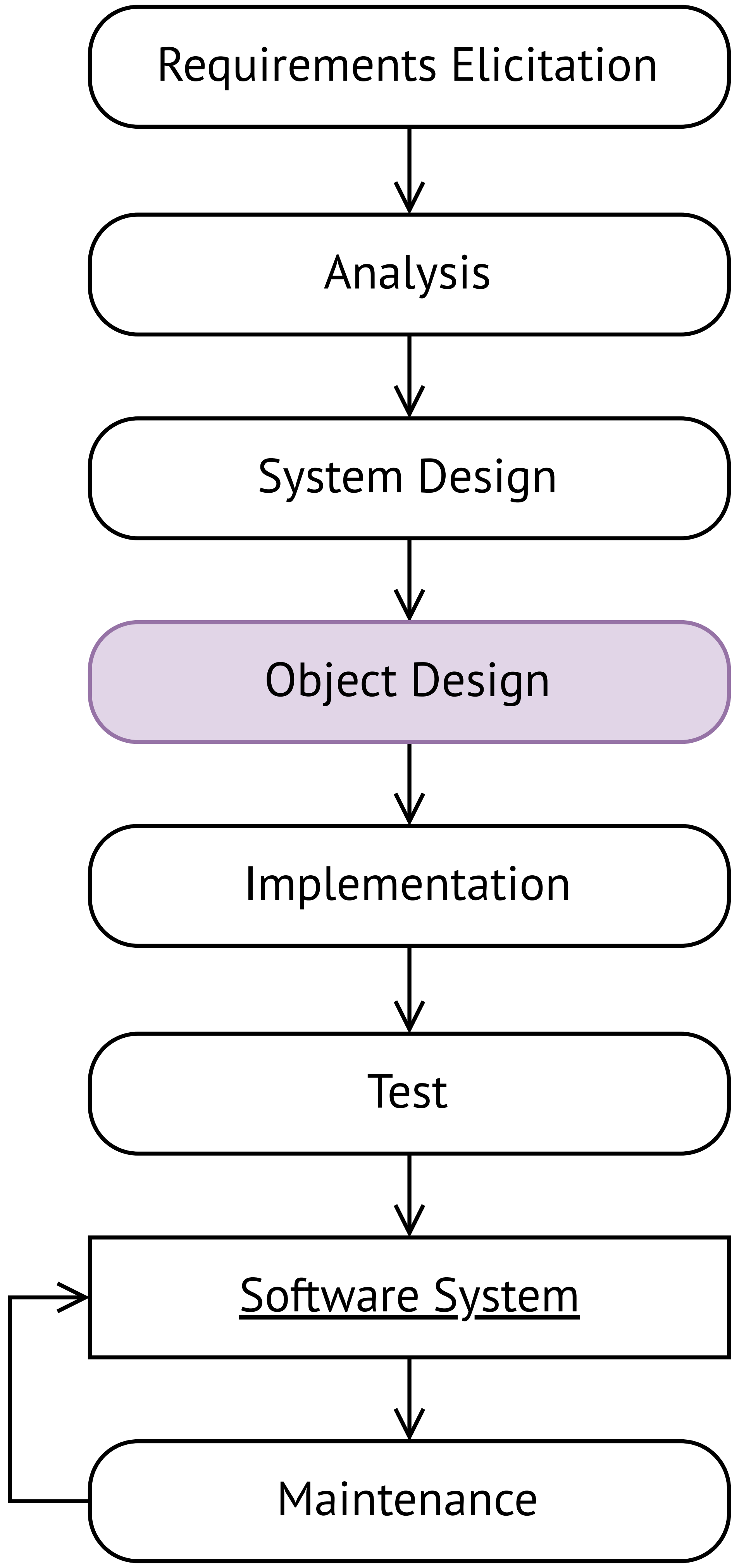
Introduction

Reuse Concepts

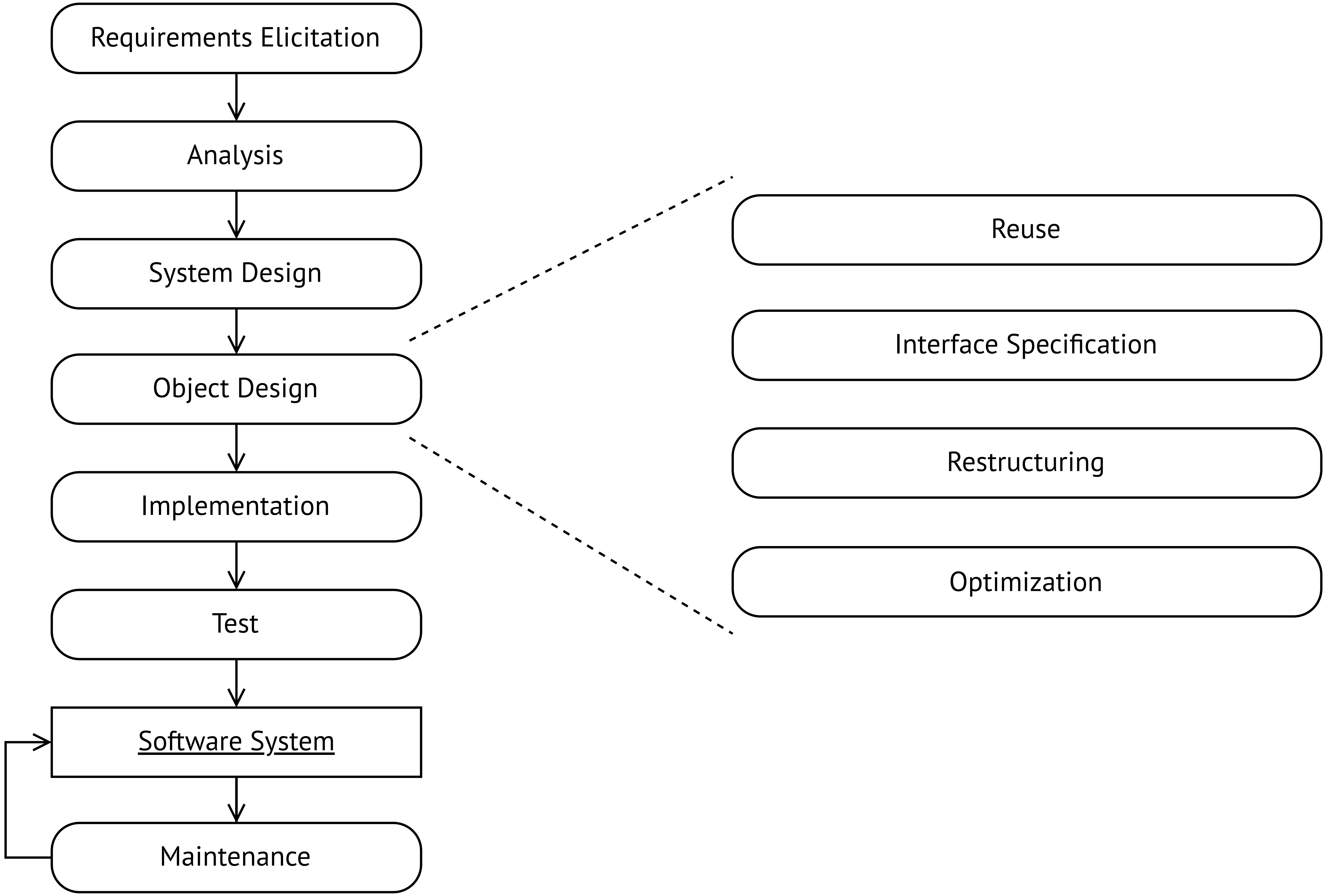
Reuse Activities

Documenting Reuse

Big Picture


Requirements Elicitation
Analysis
System Design
Object Design
Implementation
Test
Software System
Maintenance

Introduction



Introduction

- Reuse
  - Off-the-shelf components identified during system design are used to help in the realization of each subsystem.
  - Class libraries and additional components are selected for basic data structures and services.
  - Design patterns are selected for solving common problems and for protecting specific classes from future change.

Introduction

- Interface specification
  - The subsystem services identified during system design are specified in terms of class interfaces.
  - Additional operations and objects needed to transfer data among subsystems are identified.
  - Result → A complete interface specification for each subsystem.
    - API (Application Programming Interface)

Introduction

- Restructuring
  - Manipulate the system model to increase code reuse or meet other design goals.
  - Restructure classes and their associations.
  - Address design goals such as maintainability, readability, and understandability of the system model.

Introduction

- Optimization
  - Address performance requirements of the system model.

Introduction

- Object design is not sequential.
- Usually, interface specification and reuse activities occur first.
- Restructuring and optimization activities occur next.
- Activities of object design occur iteratively.

Concepts of Reuse

Reuse Concepts

- Application Objects and Solution Objects
- Specification Inheritance and Implementation Inheritance
- Delegation
- Design Pattern
- Framework

Application Objects and Solution Objects

- Class diagrams → Model both the application domain and the solution domain.
- Application objects
  - Represent concepts of the domain that are relevant to the system.
- Solution objects
  - Represent components that do not have a counterpart in the application domain.

Application Objects and Solution Objects

- Analysis and system design → Identify application and solution objects.
- Object design
  - Refine and detail both application and solution objects.
  - Identify additional solution objects needed to bridge the object design gap.

Specification Inheritance and Implementation Inheritance

- Implementation inheritance → The use of inheritance for the sole purpose of reusing code.
  - Reuse code quickly by subclassing an existing class and refining its behavior.
- Specification inheritance (interface inheritance) → The classification of concepts into type hierarchies.

Delegation

- Alternative to implementation inheritance that should be used when reuse is desired.
- Implement an operation by resending a message to another class.
- Preferable mechanism to implementation inheritance.
    - It does not interfere with existing components and leads to more robust code.

Delegation vs. Inheritance

- When to use delegation or inheritance?
    - Requires some experience and judgement on the part of the developer.
- Inheritance and delegation, used in different combinations, can solve a wide range of problems:
    - Decoupling abstract interfaces from their implementation
    - Wrapping around legacy code
    - Decoupling classes that specify a policy from classes that provide mechanism.

Design Pattern

- Design patterns → Template solutions that developers have refined over time to solve a range of recurring problems.
- Four elements:
    - Name
    - Problem Description
    - Solution
    - Consequences

Design Pattern

- Terms to denote different classes participating in the pattern:
- Client class → The class that accesses the pattern.
- Pattern interface → The part of the pattern that is visible to the client class.
    - Abstract class, interface
- Implementor class → The class that provides the lower-level behavior of the pattern.
- Extender class → The class that specializes an implementor class.

Design Pattern

- Evolve and refine design patterns for maximizing reuse and flexibility.
- Design patterns are usually not solutions that programmers would initially think of.
- Design patterns capture a great deal of knowledge.
- Design patterns constitute a source of guidance about when to use inheritance and delegation.

Design Pattern

- Three main categories:
  - Creational Patterns
  - Structural Patterns
  - Behavioral Patterns
- https://refactoring.guru/design-patterns/

Framework

- Framework → Software framework
- Framework → A framework is a reusable and extensible partial software that can be used to create software systems.
- Some types of frameworks:
    - Web frameworks
        - Spring MVC, ASP.NET Core, Next.js - Server side
        - React - Client side
    - Object relational mapping (ORM) framework
        - Hibernate, Entity Framework (EF)

Framework

- Classification by the techniques used to extend them.
- Whitebox frameworks
    - Rely on inheritance and dynamic binding for extensibility.
- Blackbox frameworks
    - Support extensibility by defining interfaces for components that can be plugged into the framework.

Reuse Activities

Reuse

- Reuse → Examine the use of design patterns and frameworks for solving a range of common object design problems.
- The goals of the system design and object design conflict:
- System Design → Define a stable architecture to deal with complexity.
- Object Design → Allow flexibility to deal with change later in the development process.

Reuse

- Solve the conflict by anticipating change and designing for it.
- Sources of later changes:
  - New vendor, new technology
  - New implementation
  - New views
  - New complexity
  - Errors

Reuse

- Reuse → Reuse existing code and solution blueprints with the help of design patterns, class libraries, components, and frameworks.
- Reuse Activity → Decide which design patterns, class libraries, components and frameworks to use, and where to use them.
- Aim of reuse activity → Handle future changes, harmonize new and legacy, current systems, create flexible systems.

Reuse

- Sub activities of reuse activity:
  - Identify class libraries
  - Identify components
  - Identify design patterns
  - Identify frameworks

Comparison of Reuse Entities

Design Patterns, Frameworks, Class Libraries, Components

- Design patterns vs. Frameworks
    - Frameworks focus on reuse of concrete designs, algorithms, and implementations in a particular programming language.
    - Design patterns focus on reuse of abstract designs and small collections of cooperating classes.
    - Frameworks focus on a particular application domain
    - Design patterns can be viewed more as building blocks of frameworks.

Comparison of Reuse Entities

Design Patterns, Frameworks, Class Libraries, Components

- Class Libraries vs. Frameworks
    - Classes in a framework cooperate to provide a reusable architectural skeleton for a family of related applications.
    - Class libraries are less domain specific and provide a smaller scope of reuse.
    - Class libraries are typically passive; that is, they do not implement or constrain the control flow.
    - Frameworks, however, are active; that is, they control the flow of control within an application.
    - In practice, developers often use frameworks and class libraries in the same system.

Comparison of Reuse Entities

Design Patterns, Frameworks, Class Libraries, Components

- Components vs. Frameworks
  - Compared with frameworks, components are less tightly coupled and can even be reused on the binary code level.
  - Frameworks can be used to develop components, where the component interface provides a facade pattern for the internal class structure of the framework.
  - Components can be plugged into blackbox frameworks.
  - Frameworks are used to simplify the development of infrastructure and middleware software
  - Components are used to simplify the development of end-user application software.

Documenting Reuse

Documenting Reuse

- Two types of documentation:
  - The documentation of the reusable solutions (design pattern, framework, component etc.)
  - The documentation of the system that is reusing the solution

Documenting Reuse

- The documentation of the reusable solutions include;
  - Description of the solution
  - Description of the class of problems it addresses
  - Trade-offs faced by the developer
  - Alternative implementations, and examples of use.
- The documentation of the reusable solutions is;
  - Difficult to produce.
  - Usually generic and abstract.
  - Documentation of a reusable solution is usually not ideal.

Documenting Reuse

- The documentation of the reusable solutions can be improved by adding the following:
  - Reference to a system using the solution
  - Example of use
  - Alternative solutions considered
  - Encountered trade-offs

References

- Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.
- Object Management Group, OMG Unified Modeling Language Superstructure, Version 2.2., http://www.omg.org/2009.

Thank you.