SWE209
Object Oriented Analysis and Design

Object Design - 2 (Interface Specification)

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

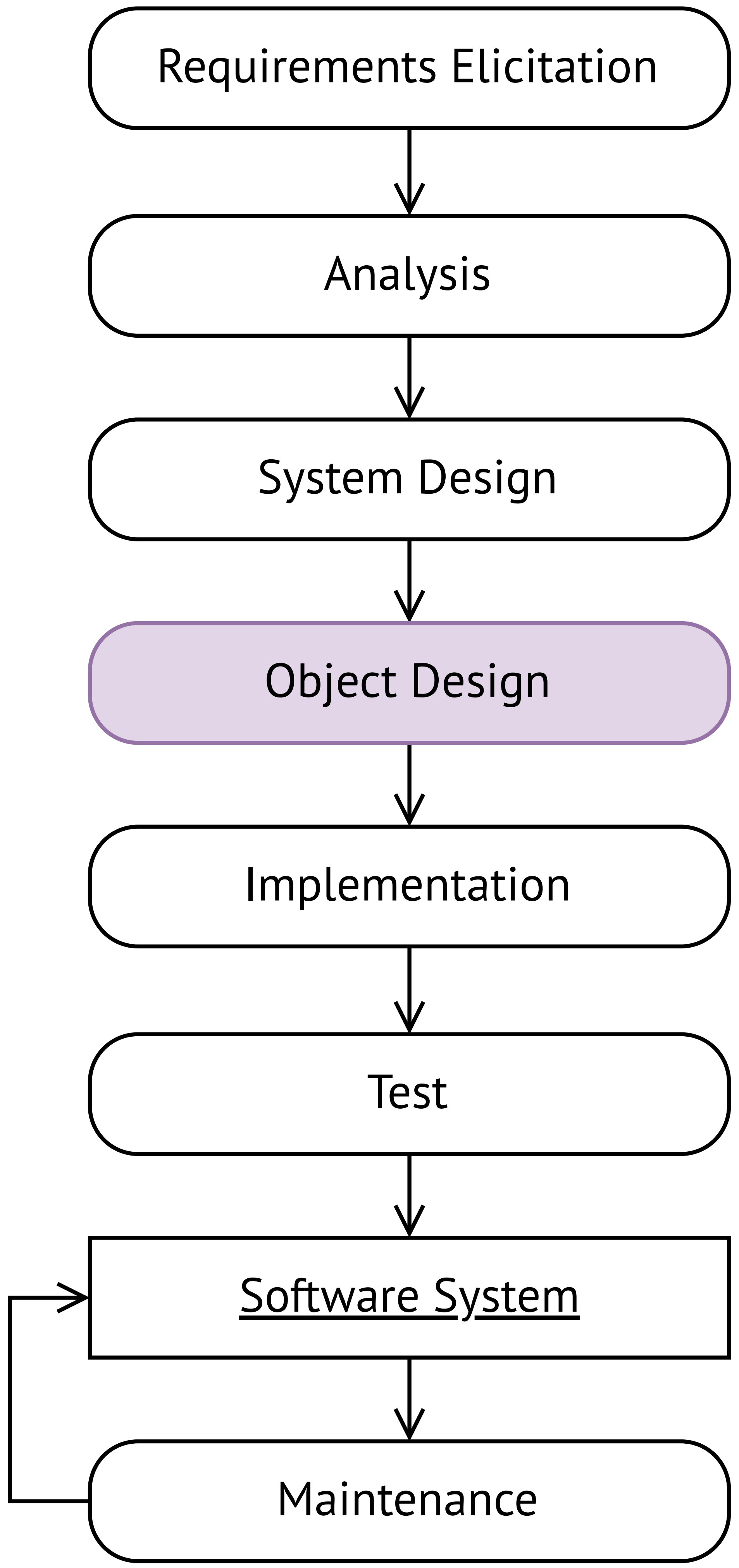
Introduction

Concepts of Interface Specification

Interface Specification Activities

Documenting Interface Specification

Big Picture



Requirements Elicitation

Analysis

System Design

Object Design

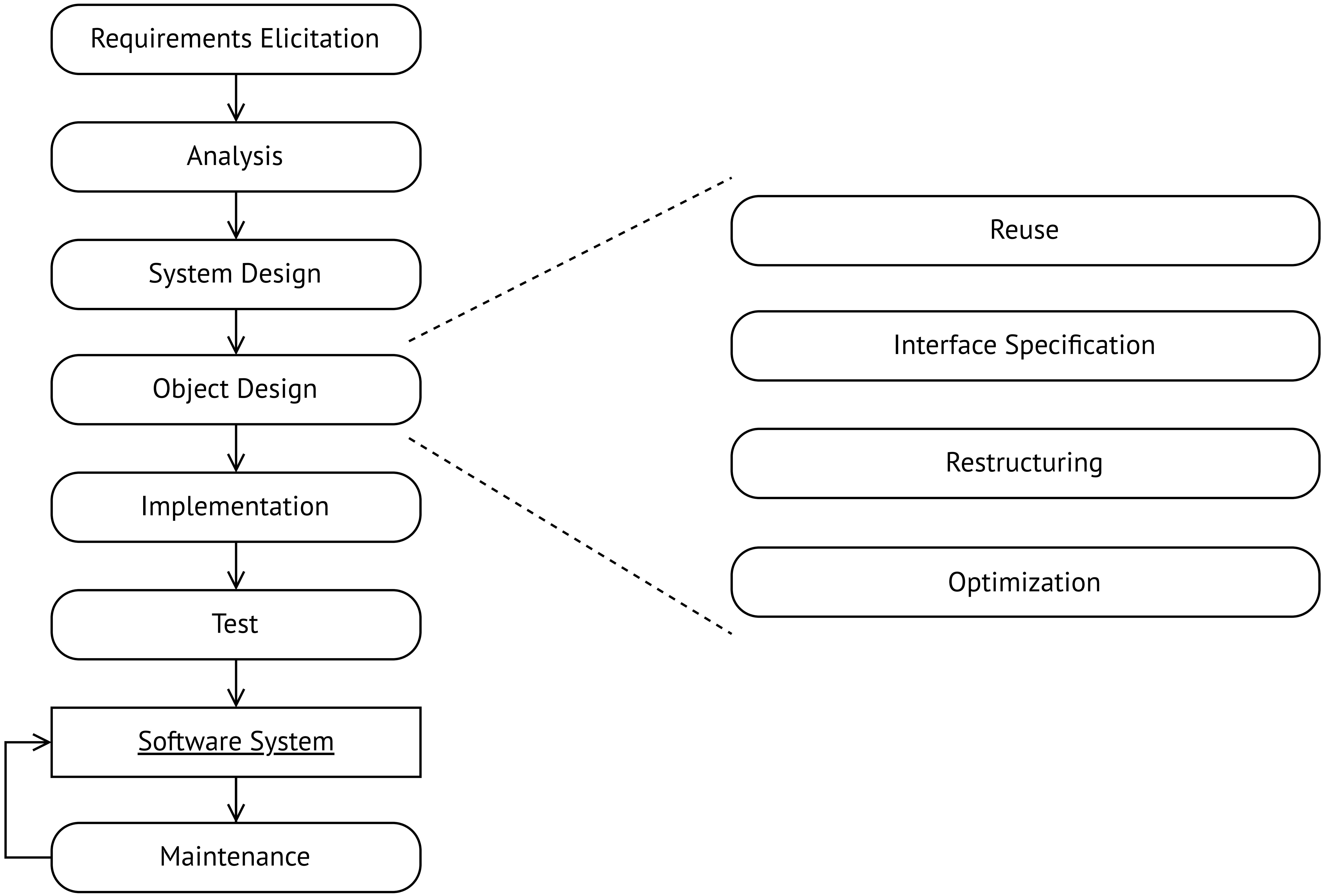
Implementation

Test

Software System

Maintenance

Introduction



Introduction

- In this slide we are going to learn **interface specification** activity of object design.
- Object design → Identify and refine solution objects to realize the subsystems defined during system design.
- Focus of system design → Identify large chunks of work that could be assigned to individual teams or developers
- Focus of object design → Specify the boundaries between objects.

Introduction

- Object design
  - Large number of developers concurrently refines and changes many objects and their interfaces.
  - The opportunity to introduce new, complex faults into the design is still there.
- Focus of interface specification → Communicate clearly and precisely about increasingly lower-level details of the system.

Introduction

Interface Specification

- Main goal:
  - Describe the interface of each object precisely
- Sub activities of interface specification:
  - Identifying missing attributes and operations
  - Specifying type signatures and visibility
  - Specifying invariants
  - Specifying preconditions and postconditions

Introduction

Interface Specification

- Identifying missing attributes and operations
  - Examine each subsystem service and each analysis object.
  - Identify missing operations and attributes.
  - Refine the current object design model and augment it with these operations.

Introduction

Interface Specification

- Specifying type signatures and visibility
  - Decide;
    - which operations are available to other objects and subsystems
    - which are used only within a subsystem
  - Specify;
    - the return type of each operation
    - the number and type of its parameters of each operation

Introduction

Interface Specification

- Specifying invariants
  - Describe behavior of the operations provided by each object in terms of constraints.
- Specifying preconditions and postconditions
  - For each operation;
    - describe the conditions that must be met before the operation is invoked
    - describe a specification of the result after the operation returns.

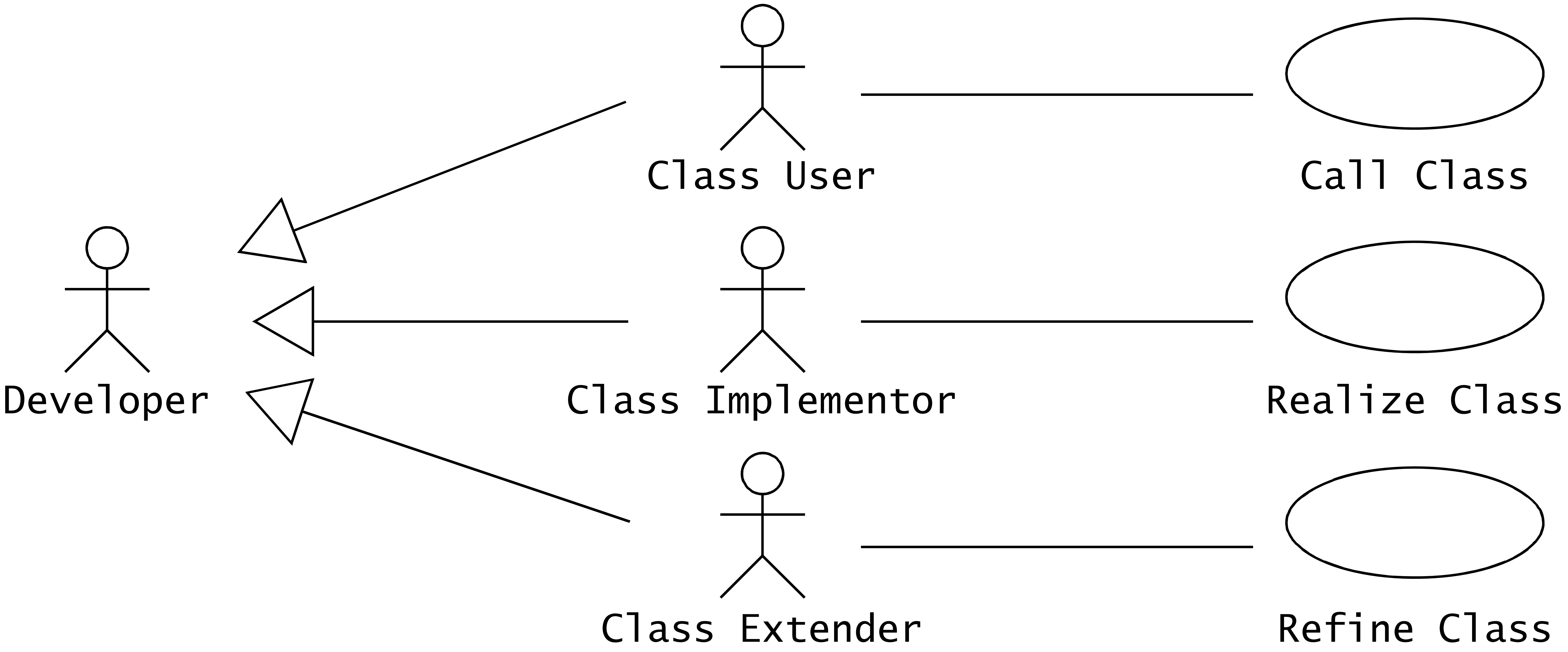
Concepts of Interface Specification

Concepts of Interface Specification

- Class Implementor, Class Extender, and Class User
- Types, Signatures, and Visibility
- Contracts: Invariants, Preconditions, and Postconditions
- Object Constraint Language

Class Implementor, Class Extender, and Class User

- Developers view the specifications from radically different point of views.



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Class Implementor, Class Extender, and Class User

- Class implementor → Realize the class under consideration.
- Class user → Invoke the operations provided by the class under consideration during the realization of another class (client class).
- Class extender → Develop specializations of the class under consideration.

Types, Signatures, and Visibility

- Analysis activity → Identify attributes and operations without necessarily specifying their types or their parameters.
- Object design activity → Refine the analysis and system design models by completing type and visibility information.
- The **type** of an attribute;
    - specifies the range of values the attribute can take.
    - specifies the operations that can be applied to the attribute.

Types, Signatures, and Visibility

- Operation parameters and return values are typed in the same way as attributes are.
- **Signature** of the operation → The tuple made out of the types of the parameters and the type of the return value of an operation.
- Sample signatures;
  - `acceptPlayer(Player):void` → Takes one parameter of type Player, does not have a return value.
  - `getMaxNumPlayers():int` → Takes no parameters and returns an int.

Types, Signatures, and Visibility

- The **visibility** of an attribute or an operation → A mechanism for specifying whether the attribute or operation can be used by other classes or not.
- Four levels of visibility according to UML:
  - Private
  - Protected
  - Public
  - Package

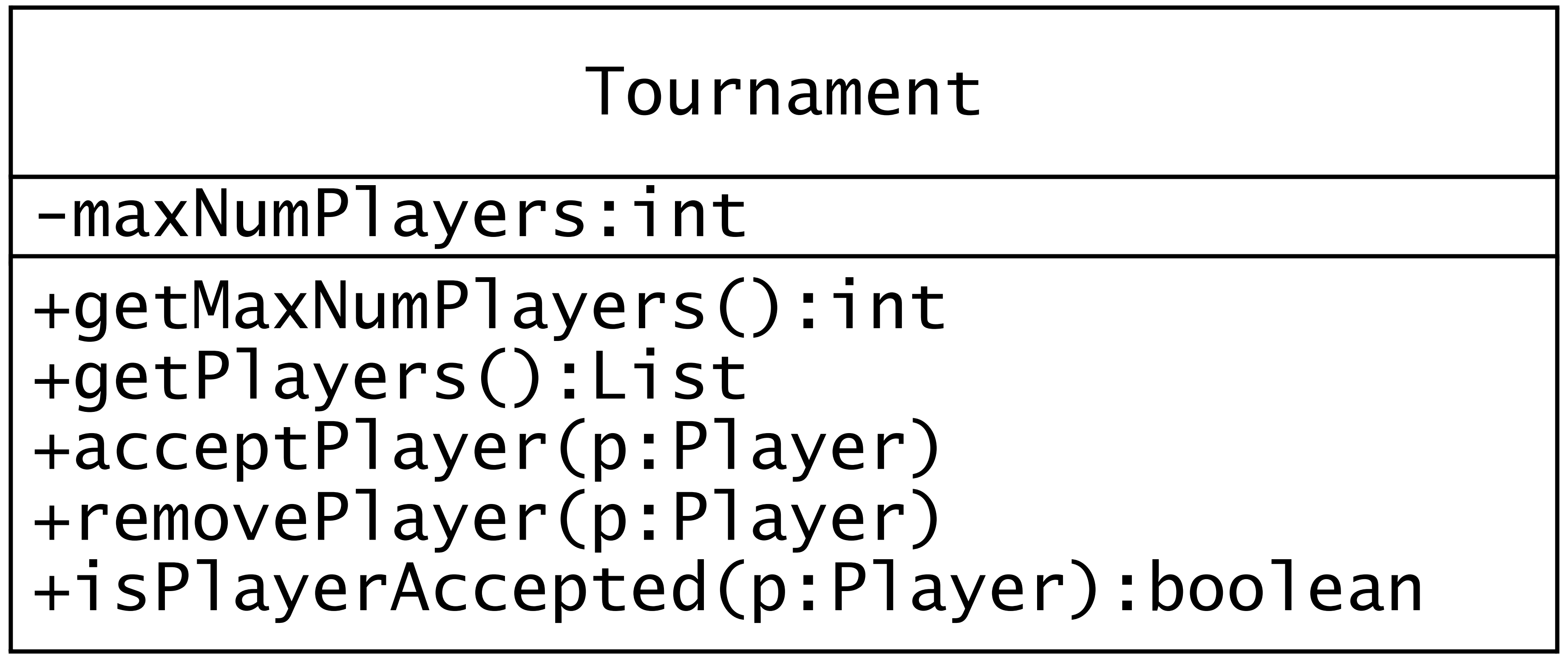
Types, Signatures, and Visibility

- **Private** → Can be accessed only by the class in which it is defined.
  - Intended for the class implementor only.
- **Protected** → Can be accessed by the class in which it is defined and by any descendant of that class.
  - Intended for the class extender.
- **Public** → Can be accessed by any class.
  - Constitute the public interface of the class.
  - Intended for the class user.

Types, Signatures, and Visibility

- **Package** → Can be accessed by any class in the nearest enclosing package.
- Visibility representation in UML
  - Prefix the name of the attribute or the operation with a character symbol.
  - – → private
  - # → protected
  - + → public
  - ~ → package

Types, Signatures, and Visibility



Tournament

-maxNumPlayers:int

+getMaxNumPlayers():int
+getPlayers():List
+acceptPlayer(p:Player)
+removePlayer(p:Player)
+isPlayerAccepted(p:Player):boolean

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Contracts: Invariants, Preconditions, and Postconditions

- Contract → Constraint on a class that enable developers to share the same assumptions about the class.
- Three types of constraints:
  - Invariant → A predicate that is always true for all instances of a class.
  - Precondition → A predicate that must be true before an operation is invoked.
  - Postcondition → A predicate that must be true after an operation is invoked.

Contract Inheritance

- In a polymorphic language, a class can be substituted by any of its descendants.
- A class user invoking operations on a class could be invoking instead a subclass.
- The class user expects that a contract that holds for the superclass still holds for the subclass.
    - Contract inheritance

Contract Inheritance

- Contracts are inherited in the following manner:
- Preconditions.
    - A method of subclass is allowed to weaken the preconditions of the method it overrides.
    - An overwritten method can handle more cases than its superclass.
- Postconditions.
    - Methods must ensure the same postconditions as their ancestors or stricter ones.

Contract Inheritance

- Invariants.
  - A subclass must respect all invariants of its superclasses.
  - A subclass can strengthen the inherited invariants.

Object Constraint Language

- A constraint can be expressed in natural language or in a formal language.
- Object Constraint Language (OCL) is a language that allows constraints to be formally specified on model elements.
- Model elements can be;
    - single model elements (e.g., attributes, operations, classes) or
    - groups of model elements (e.g., associations and participating classes).

Interface Specification Activities

Interface Specification Activities

- Identifying Missing Attributes and Operations
- Specifying Type Signatures and Visibility
- Specifying Preconditions and Postconditions
- Specifying Invariants

Identify Missing Attributes and Operations

- Examine the service description of the subsystem and identify missing attributes and operations.
- During analysis, we may have missed many attributes. Why?
  - Focus on the application domain when constructing the object model.
  - Ignorance of the details related to the system that are independent of the application domain.

Specify Type Signatures and Visibility

- Specify the types of the attributes, the signatures of the operations, and the visibility of attributes and operations.
- Specifying types refines the object design model in two ways.
    - Add detail to the model by specifying the range of each attribute.
    - Map classes and attributes of the object model to built-in types provided by the development environment.
- Consider the relationship between the classes identified and the classes from existing components.
- Determine the visibility of each attribute and operation during this step.

Specify Preconditions and Postconditions

- Define contracts for each public operation of each class.
- Contract → Agreement between the class user and the class implementor.
- The preconditions of an operation describe the part of the contract that the class user must respect.
- The postconditions describe what the class implementor guarantees in the event the class user fulfilled her part of the contract.
- When refining a class, class extenders inherit the contract from the original class implementor.

Specify Invariants

- Invariants provide an overview of the essential properties of the class.
- Invariants constitute a permanent contract that extends and overwrites the operation-specific contracts.
- Some invariants are obvious and can be written from the start.
- Some invariants can be identified by extracting common properties from operation-specific contracts.

Documenting Object Design

Documenting Object Design

- Object design is documented in the Object Design Document (ODD).
- Describe
  - object design trade-offs made
  - guidelines followed for subsystem interfaces
- Used to exchange interface information among teams and as a reference during testing.
- The audience for the ODD → System architects, developers, and testers.

Documenting Object Design

- There are three main approaches to documenting object design:
- Self-contained ODD generated from model.
  - Write and maintain a UML model and generate the document automatically.
- ODD as extension of the RAD.
  - Treat the object design model as an extension of the analysis model.
- ODD embedded into source code.
  - Embed the ODD into the source code.

Documenting Object Design

- Currently generating the ODD from source code and focusing the RAD on the application domain is the most practical.
- Reduces the amount of redundant information to be maintained.
- Locates the object design information where it is the most accessible.
- The consistency between the source code and the analysis model must still be maintained manually.

Documenting Object Design

- Outline of the object design document
  - 1. Introduction
    - 1.1 Object design trade-offs
    - 1.2 Interface documentation guidelines
    - 1.3 Definitions, acronyms, and abbreviations
    - 1.4 References
  - 2. Packages
  - 3. Class interfaces
  - Glossary

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.