SWE209
Object Oriented Analysis and Design

Analysis

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

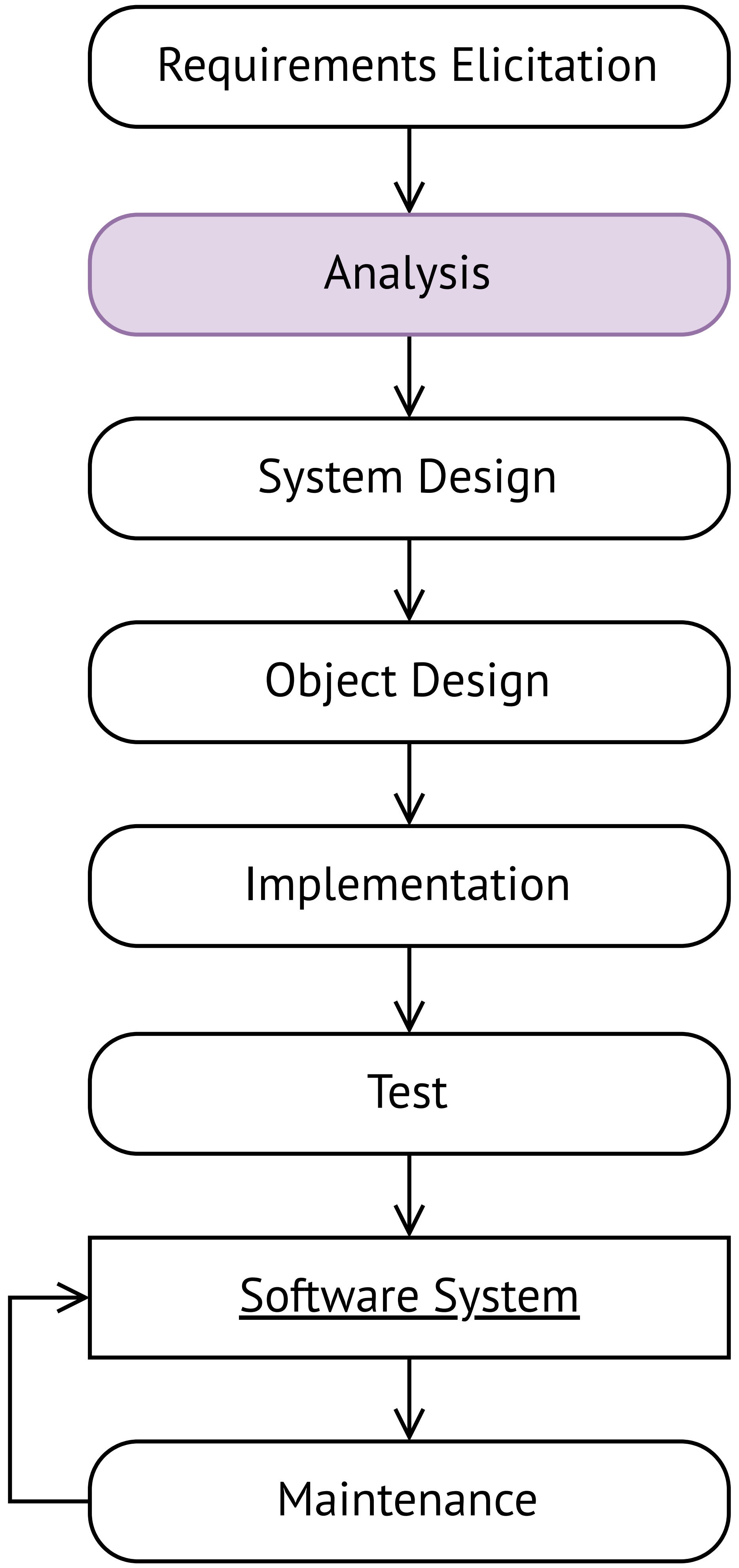
Analysis Concepts

Analysis Activities

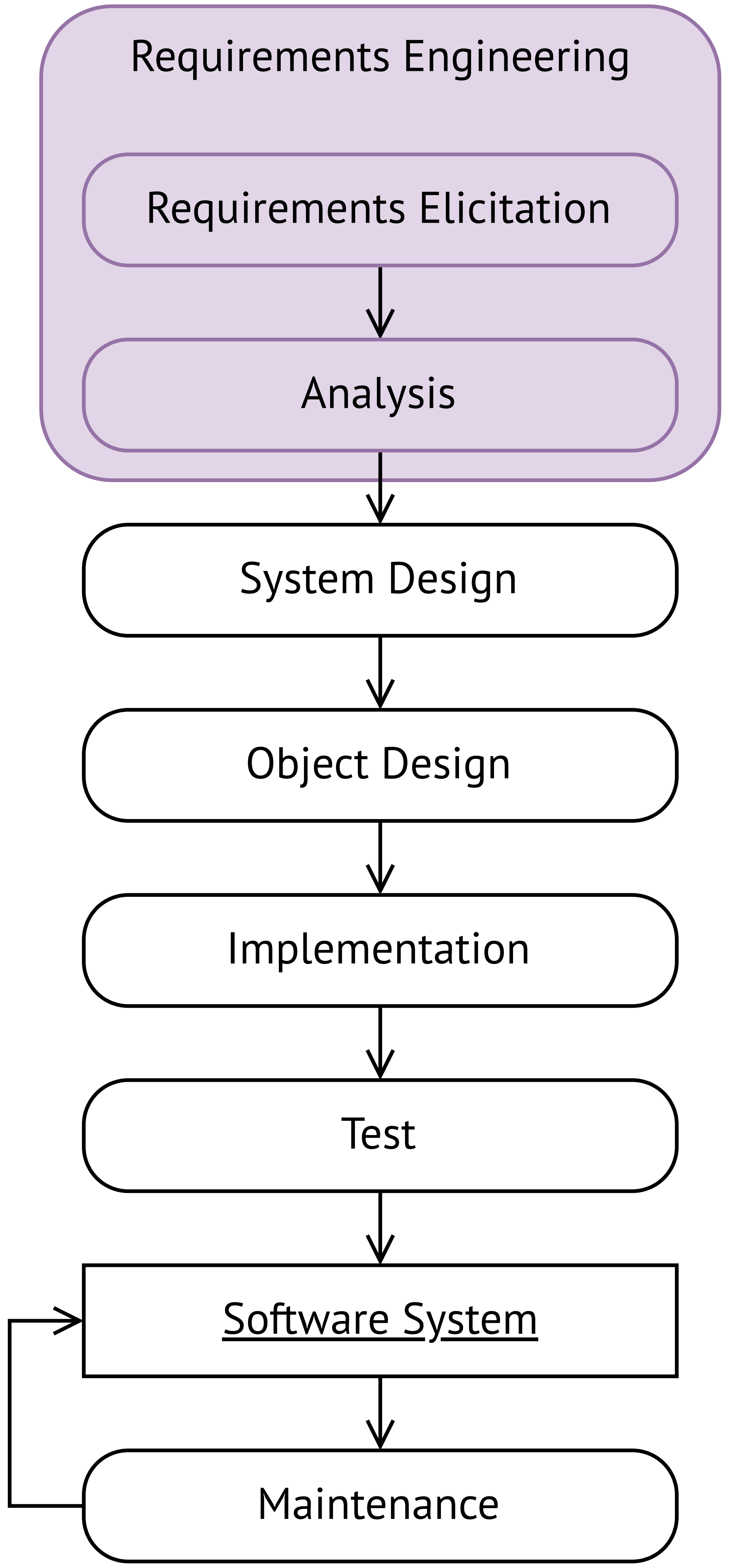
Reviewing the Analysis Model

Documenting Analysis

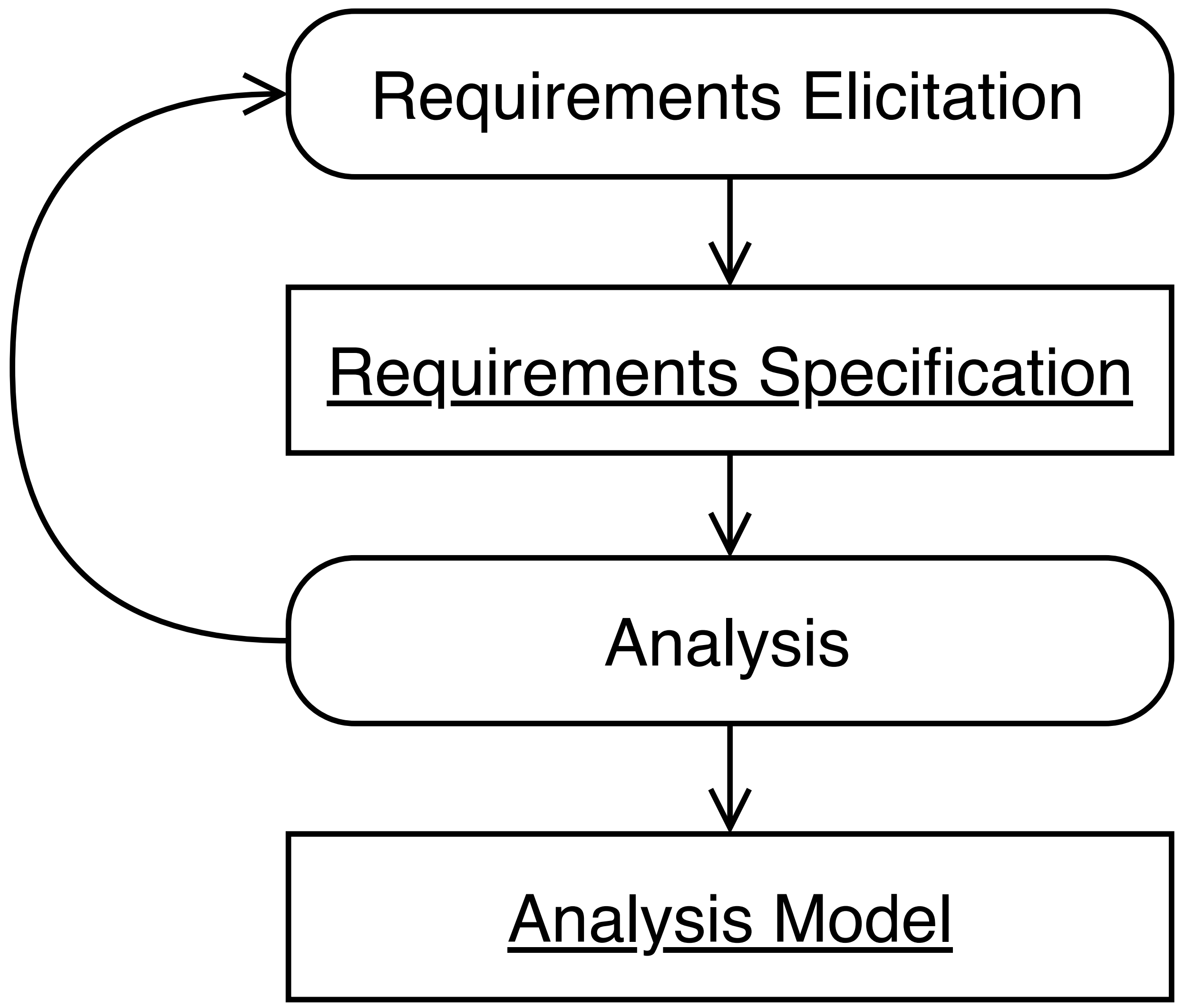
Big Picture


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

Big Picture


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

Introduction



Introduction

- Analysis activity → Analysis model
- Analysis model → Correct, complete, consistent, clear
- Analysis activity → Validation, correction, clarification and formalization of requirements specification.
- Recall
  - Object-oriented analysis → Modeling the application domain
  - Object-oriented design → Modeling the solution domain

Introduction

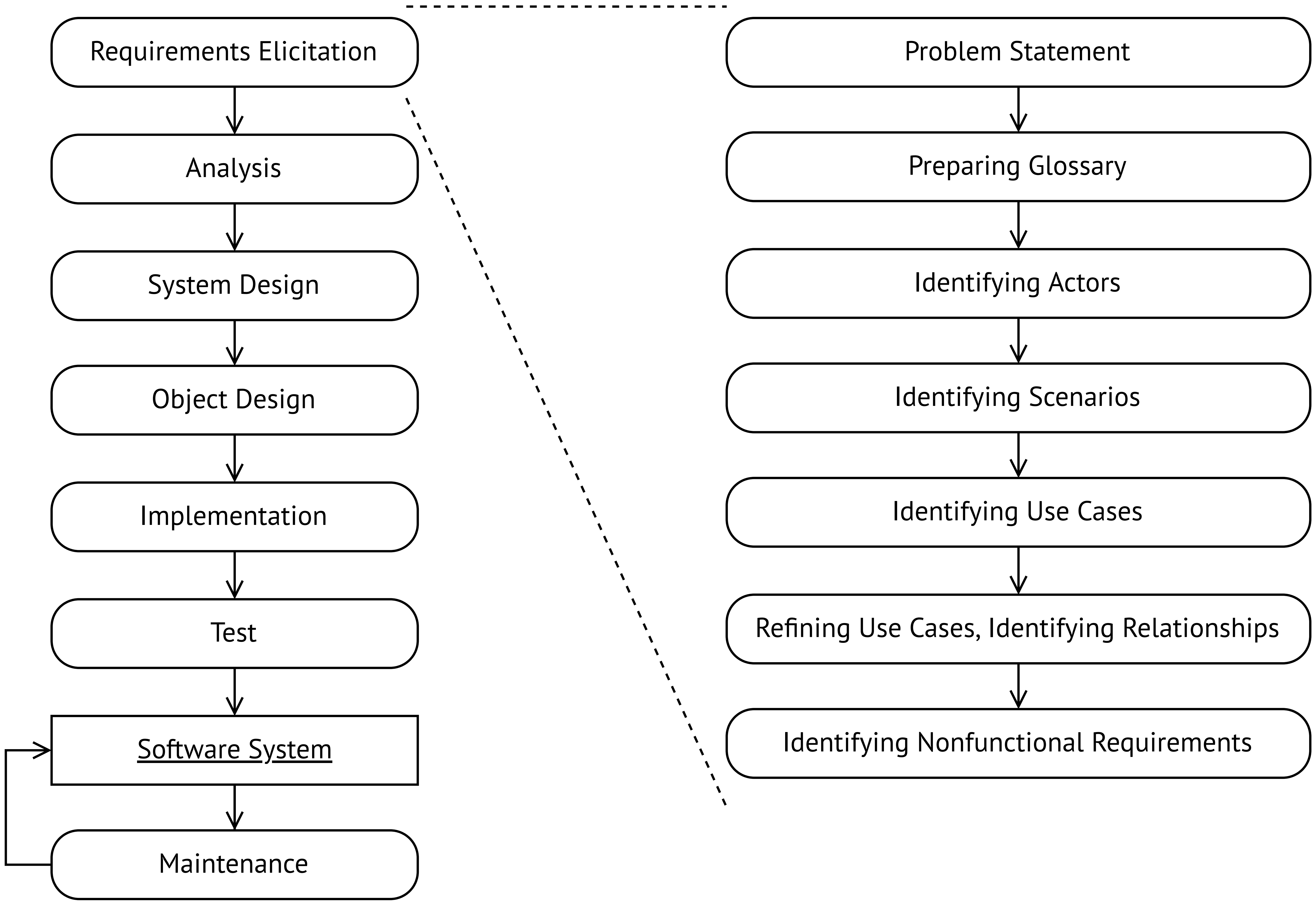
- Requirements specification → Should be understandable to the users and the client.
- Analysis model → May not be understandable to the users and the client.
- Analysis model → Represent the system from the user’s point of view.

Introduction

- Analysis model is composed of;
    - Functional model → use cases and scenarios
    - Analysis object model → class and object diagrams
    - Dynamic model → state machine and sequence diagrams
- Analysis
    - Refine the functional model
    - Derive the object and the dynamic model
    - More precise and complete specification

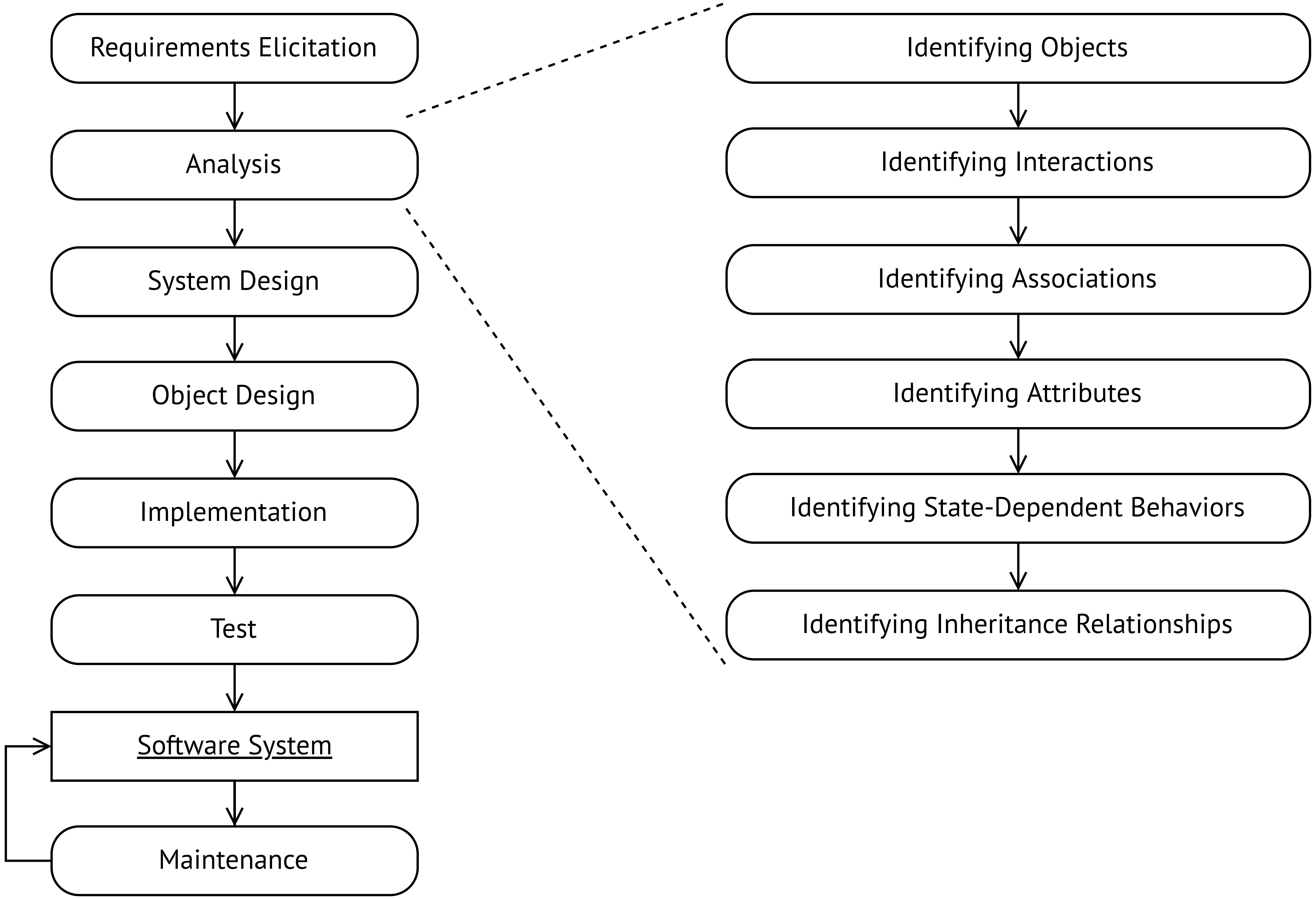
Overview of Analysis

Requirements Elicitation Activities Recap



Introduction

Analysis Activities



Analysis Concepts

Analysis Concepts

- Analysis Object Model and Dynamic Model
- Entity, Boundary, and Control Objects
- Generalization and Specialization

Analysis Object Model, Dynamic Model

- Analysis object model
  - Focus on structure of the system
  - Depicted with class diagrams.
- Dynamic model
  - Focus on the behavior of the system.
  - Depicted with sequence and state-machine diagrams.

Analysis Object Models, Dynamic Models

- Analysis object model and dynamic model:
  - Represent user-level concepts.
  - Do not represent actual software classes or components.
- For example;
  - Subscriber e-mail vs UserID

Entity, Boundary, Control Objects

- Analysis object model → Entity, boundary, and control objects
- Entity objects → Persistent information tracked by the system
- Boundary objects → Interactions between the actors and the system
- Control objects → Realization of use cases
- Helps to distinguish different, but related concepts.

Entity, Boundary, Control Objects

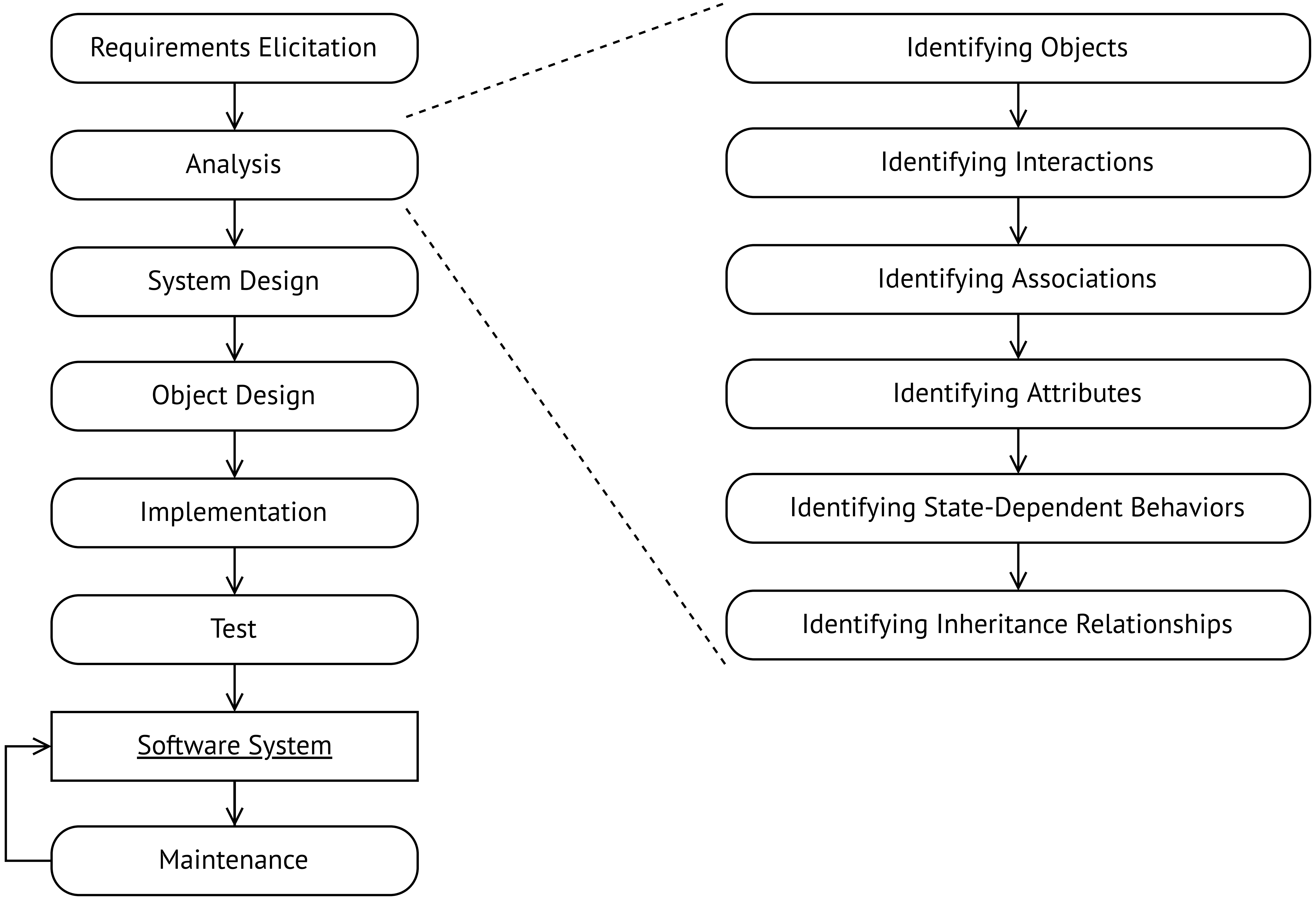
- University Information System
  - Course → Entity object
  - ShowCourseListButton → Boundary object
  - EnrollCourseControl → Control object

Generalization, Specialization

- Generalization → Identify abstract concepts from lower-level ones.
- Specialization → Identify specific concepts from a high-level one.
- Generalization and specialization → Specification of inheritance relationship
- Inheritance → Relationship
- Generalization, specialization → Activities that find inheritance relationships

Analysis Activities

Analysis Activities



Analysis Activities

Use case name	WatchVideo
Participating actors	Initiated by the Subscriber
Flow of events	1. The Subscriber clicks the videos button. 2. Albatros shows the list of videos to the subscriber. 3. The Subscriber clicks on the icon of a video. 4. Albatros opens the viewer. 5. Albatros streams the video.
Entry conditions	The Subscriber is logged into Albatros.
Exit conditions	The Subscriber closed the viewer, OR Albatros ended streaming.
Quality requirements	The streaming starts within 30 seconds after the Subscriber clicks the video icon.

Analysis Activities

Abbott's Heuristics

Part of speech	Model component	Examples
Proper noun	Instance	Alice
Common noun	Class	Field officer
Doing verb	Operation	Creates, submits, selects
Being verb	Inheritance	Is a kind of, is one of either
Having verb	Aggregation	Has, consists of, includes
Modal verb	Constraints	Must be
Adjective	Attribute	Incident description

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

Identifying Objects

- Identifying Entity Objects
- Identifying Boundary Objects
- Identifying Control Objects

Identifying Objects

Identifying Entity Objects

- Entity objects represent the persistent information tracked by the system.
- Start with the names used by end users and application domain specialists.
- A tentative name and a brief description for each object is sufficient.
- Document attributes and responsibilities of objects if they are not obvious.
- Remember there will be plenty of iterations.

Identifying Objects

Identifying Entity Objects

- Heuristics
  - Terms that developers or users need to clarify in order to understand the use case.
  - Recurring nouns in the use cases.
  - Real-world entities that the system needs to track.
  - Real-world activities that the system needs to track.
  - Data sources or sinks.

Identifying Objects

Identifying Entity Objects

- Entity objects of WatchVideo use case
  - Subscriber
    - A person who is registered to Albatros and paying for the service.
  - Video
    - A multimedia content which subscribers watch.
  - Viewer - ? (Control or entity object?)
    - An object that streams video to the user.

Identifying Objects

Identifying Boundary Objects

- Boundary objects represent the system interface with the actors.
- Boundary object;
    - Collect the information from the actor
    - Translate it into a form that can be used by both entity and control objects.
- Boundary objects model the user interface at a coarse level.

Identifying Objects

Identifying Boundary Objects

- Heuristics
  - Identify user interface controls that the user needs to initiate the use case.
  - Identify forms the users needs to enter data into the system.
  - Identify notices and messages the system uses to respond to the user.
  - When multiple actors are involved in a use case, identify actor terminals to refer to the user interface under consideration.
  - Always use the end user’s terms for describing interfaces.

Identifying Objects

Identifying Boundary Objects

- Boundary objects of WatchVideo use case
  - VideoListButton
    - Button used by Subscribers to see the list of videos.

Identifying Objects

Identifying Control Objects

- Control objects are responsible for;
    - Coordinating boundary and entity objects.
    - Collecting information from the boundary objects and dispatching it to entity objects.
- Usually do not have a concrete counterpart in the real world.
- Close relationship between a use case and a control object.
    - A control object is usually created at the beginning of a use case and ceases to exist at its end.

Identifying Objects

Identifying Control Objects

- Control objects of WatchVideo use case
    - WatchVideoControl
        - Manages the video watching activity of the subscriber. It is created when the subscriber clicks VideoListButton.

Identifying Interactions

- Sequence Diagrams
- CRC Cards
  - CRC → Class, Responsibilities, Collaborators

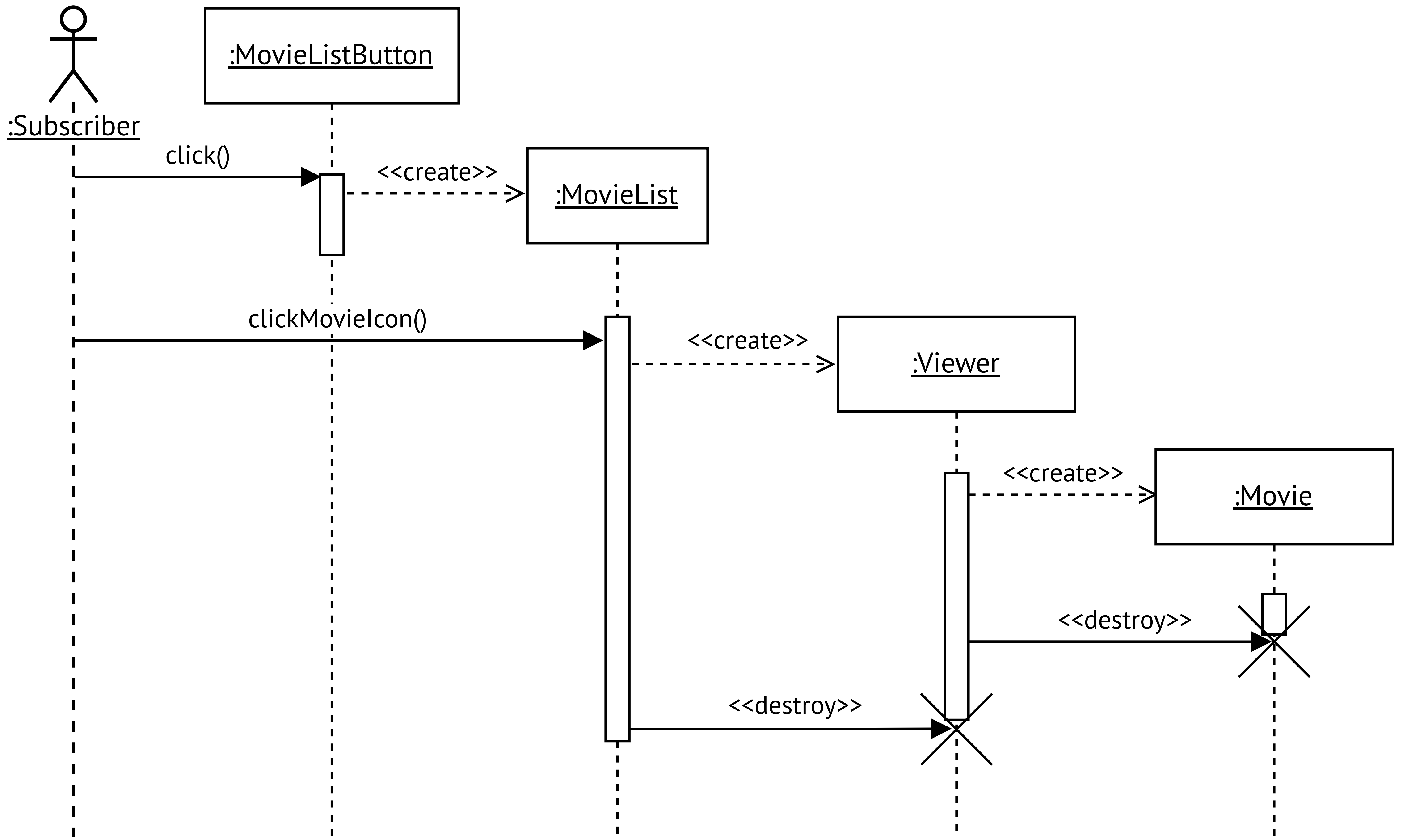
Identifying Interactions

Sequence Diagrams

- Tie use cases with objects.
- Model the sequence of interactions among objects.
- How the behavior of a use case (or scenario) is distributed among its participating objects?
- Allow the developers to find missing objects or grey areas in the requirements specification.

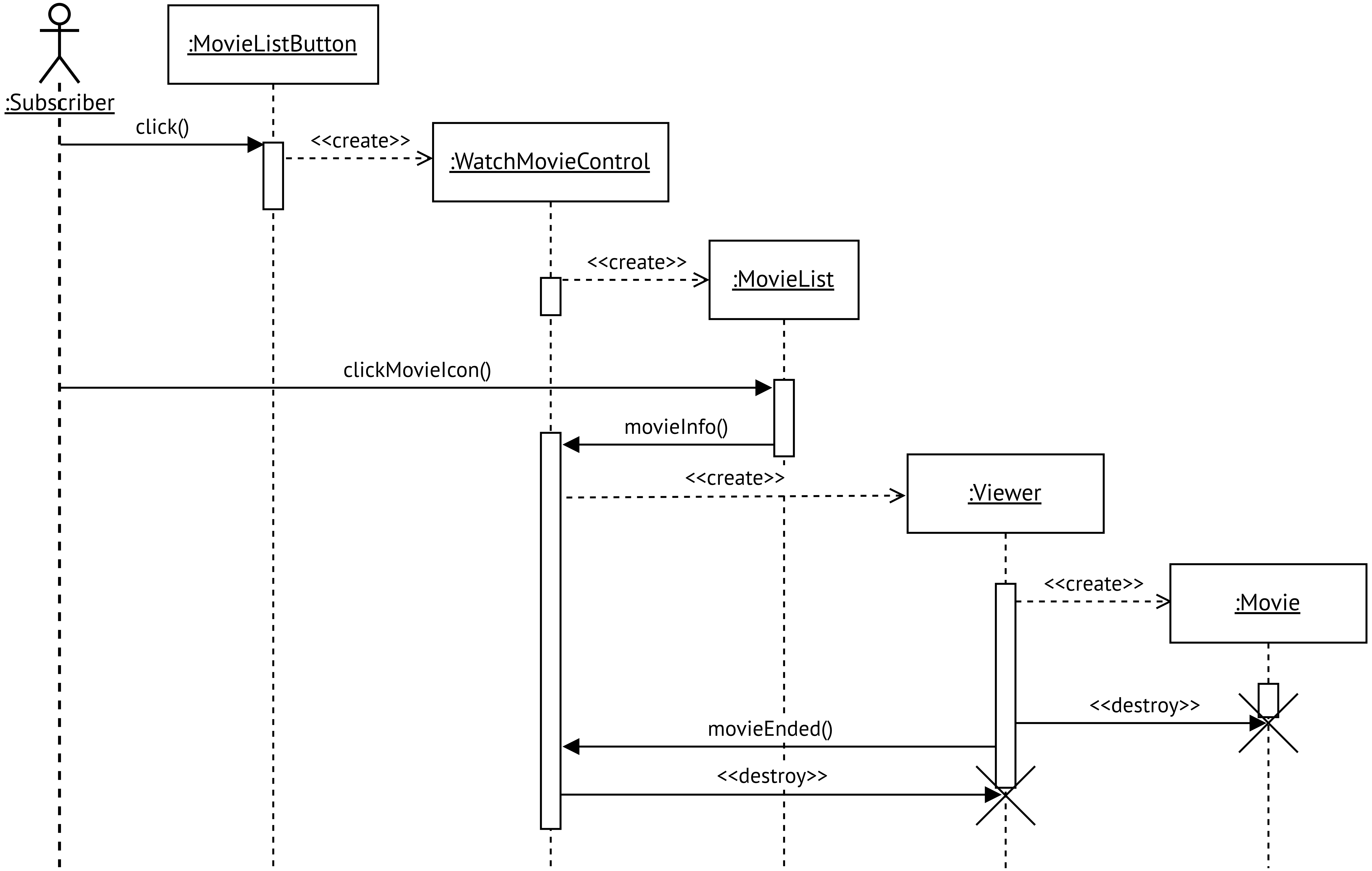
Identifying Interactions

Sequence Diagrams



Identifying Interactions

Sequence Diagrams



Identifying Interactions

Sequence Diagrams

- Sequence diagrams help us to;
  - Understand the system better.
  - Identify new objects.
- Don’t draw sequence diagrams of all the parts of the system.
  - No need to draw the parts that are well understood and simple.

Identifying Interactions

CRC Cards

- CRC Cards → An alternative for identifying interactions among objects.
- CRC → class, responsibilities, collaborators
- Initially introduced as a tool for teaching object-oriented concepts.
- Each class is represented with an index card → CRC card.

Identifying Interactions

CRC Cards

WatchMovieControl	
Responsibilities	**Collaborators**
Gets the video information from subscriber. Manages the movie watching activity.	MovieListButton

Identifying Interactions

CRC Cards

- Can be used during modeling sessions with teams.
- Participants go through a scenario and identify the classes.
- Put one card per class on the table.
  - Analyze scenario. → Assign responsibilities to each class.
  - Identify dependencies with other cards. → Fill collaborators column.
- The collaborators column is filled as the dependencies with other cards are identified.

Identifying Associations

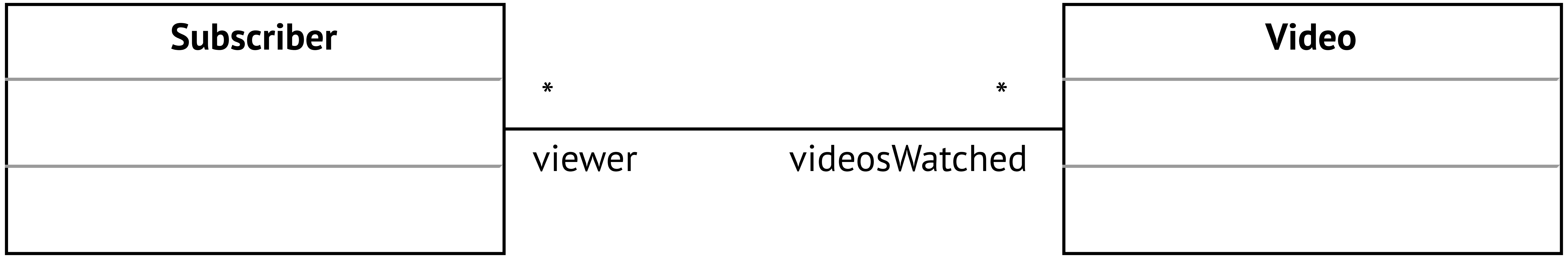
- Use of class diagrams for representing associations among objects.
- Association → Relationship between two or more classes
- Two advantages:
  - Clarification of the analysis model
  - Enables the developer to discover boundary cases associated with links.
- Properties of associations → Name, role, multiplicity

Identifying Associations

- Every association should be named.
- Roles should be assigned to each end.
- Abbott’s heuristics
  - Verbs and verb phrases denoting a state
- Too many associations make a model unreadable.

Identifying Associations

- Detected problems:
  - Viewer role name?
  - Video, movie, tv show?



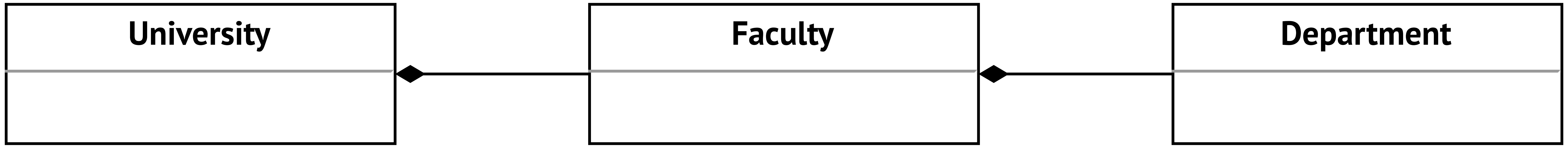
Identifying Associations

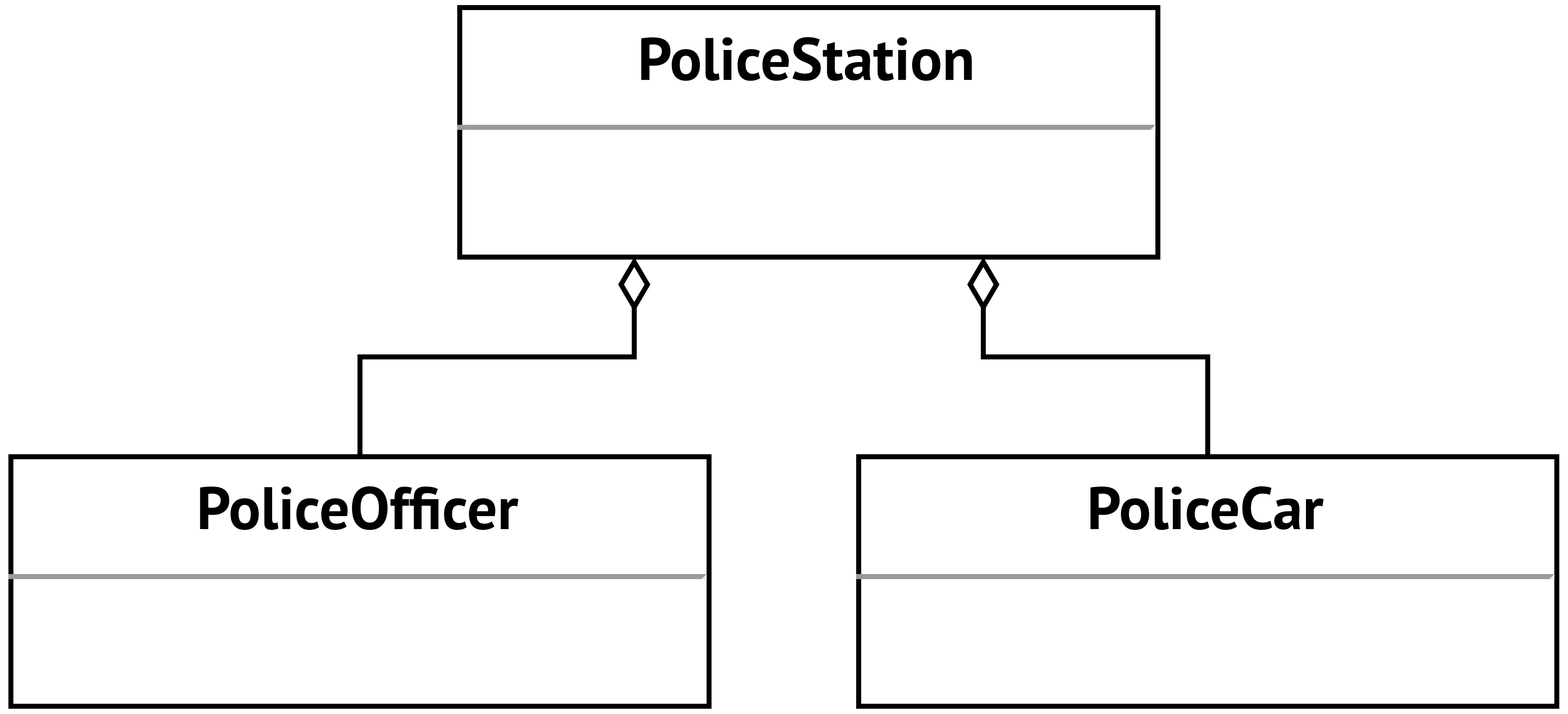
Identifying Aggregates

- Aggregation → Special types of association.
    - Contain, compose relationship
    - Whole–part relationship
- There are two types of aggregation
    - Composition → Solid diamond
    - Shared → Hollow diamond

Identifying Associations

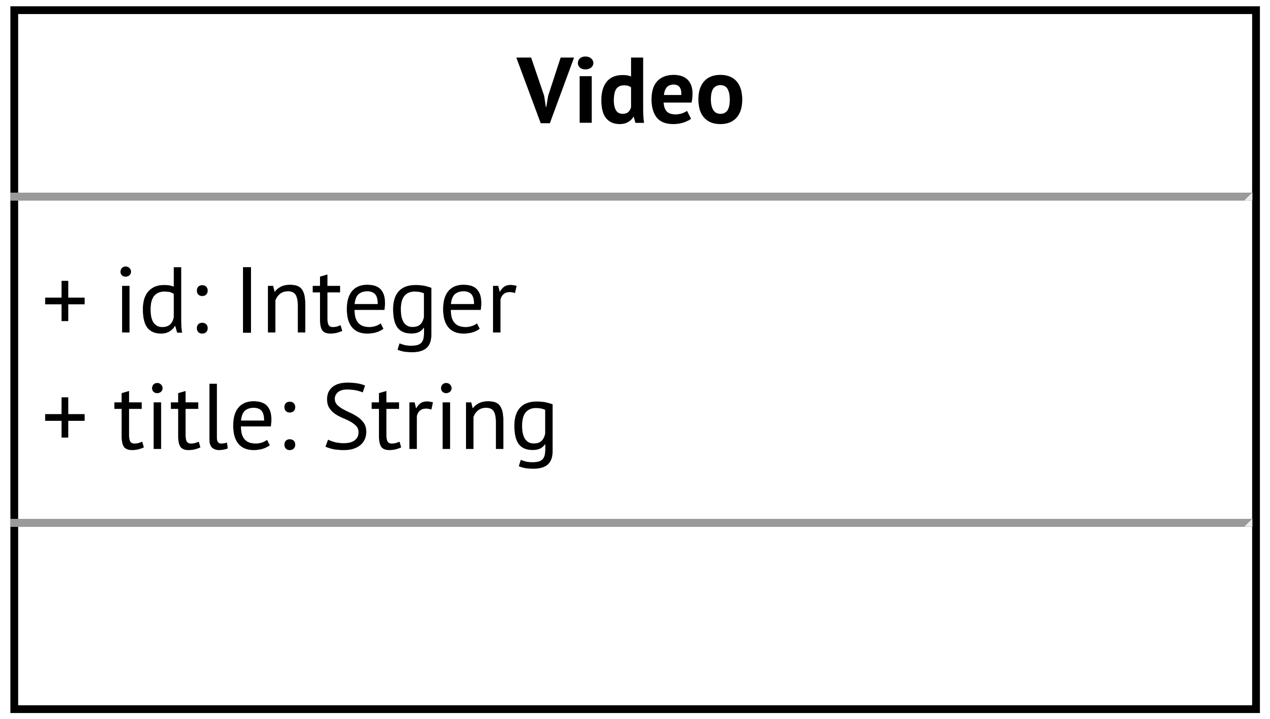
Identifying Aggregates





Identifying Attributes

- Attributes → Properties of objects
- Consider only the attributes relevant to the system.
- Attribute → name, description, type



Video

+ id: Integer
+ title: String

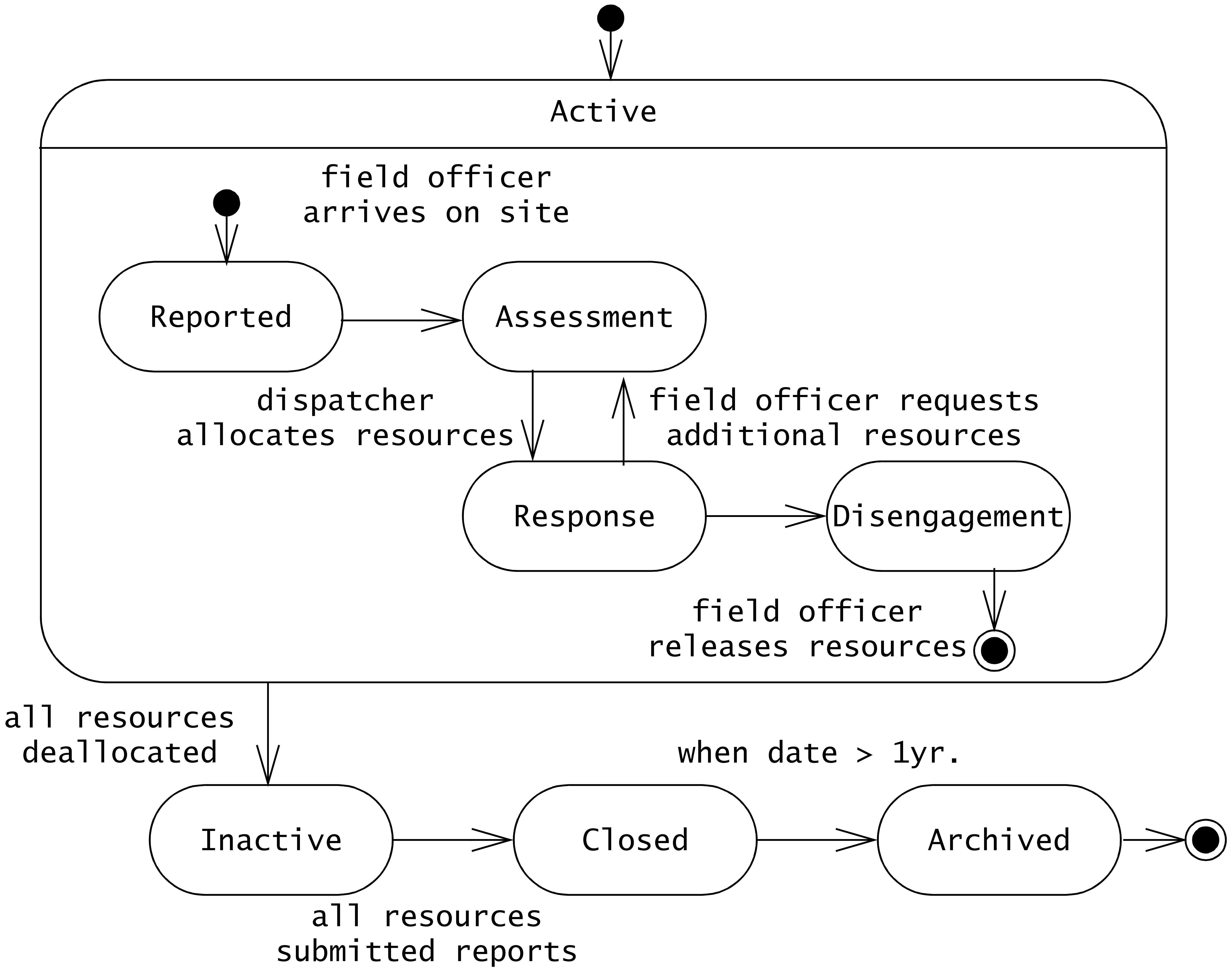
Identifying Attributes

- Abbott’s heuristics
  - A noun phrase followed by;
    - A possessive phrase or
    - An adjective phrase
- In entity objects, any property that must be stored by the system.
- Attributes are added and changed in the later stages of the development.

Identifying State-Dependent Behaviors

- State machine diagram
- Behavior from the perspective of a single object.
- Build a more formal description of the behavior of the object.
- Identify missing use cases.
- Identify new behavior.
- It is not necessary to build state machines for every class in the system.

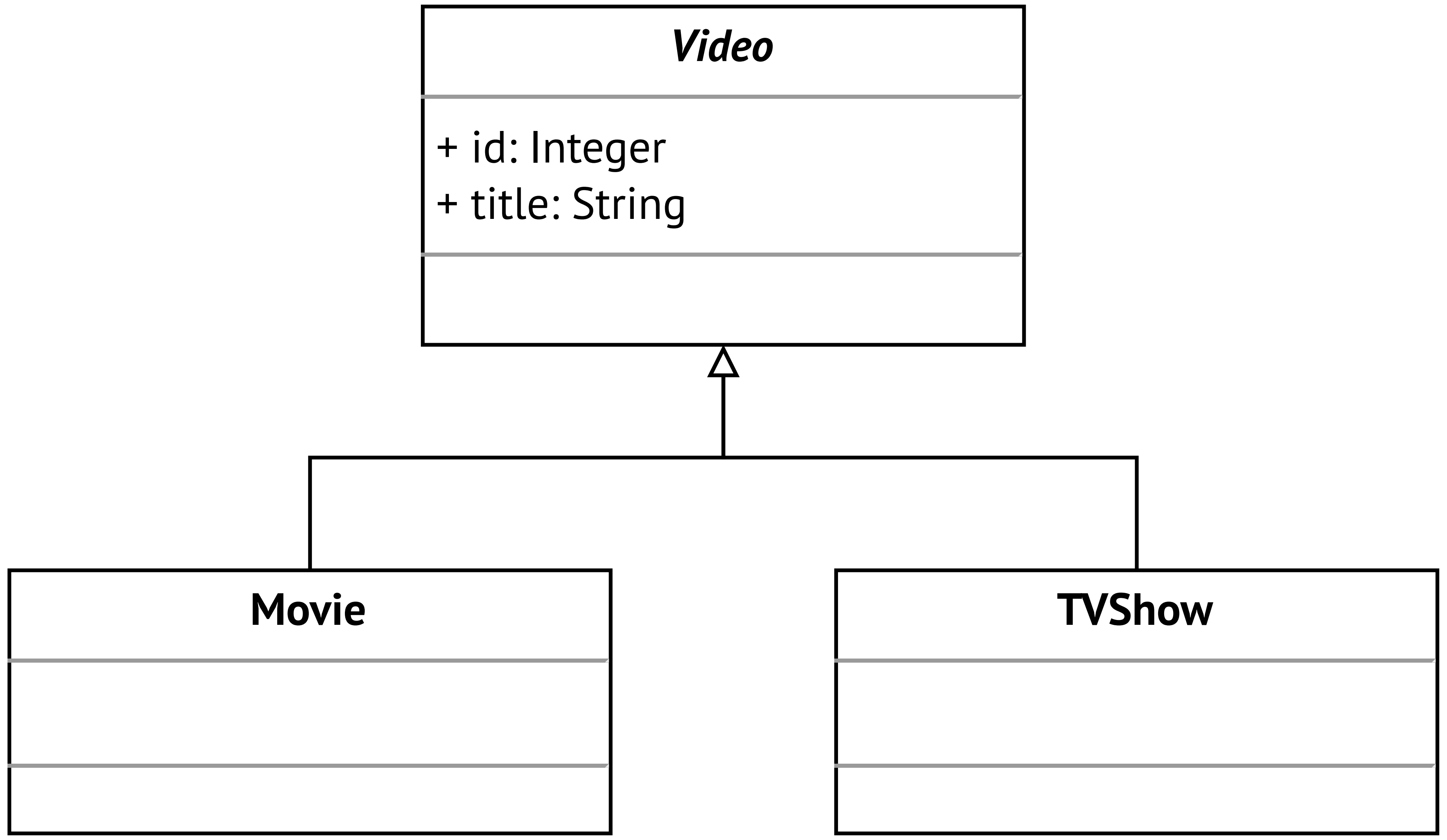
Identifying State-Dependent Behaviors


Active
field officer arrives on site
Reported
Assessment
dispatcher allocates resources
field officer requests additional resources
Response
Disengagement
field officer releases resources
all resources deallocated
when date > 1yr.
Inactive
Closed
Archived
all resources submitted reports

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

Identifying Inheritance Relationships

- Identify inheritance relationships by generalization and specialization.
- Eliminate redundancy.



Reviewing the Analysis Model

Reviewing the Analysis Model

- Incrementally and iterative approach.
- On the first pass the analysis model is not correct or complete.
- Several iterations are necessary.
- Number of changes ↓ - Stable analysis model ↑
- Then the analysis model is reviewed, by the developers and the client.

Reviewing the Analysis Model

- The goal of the review;
  - The requirements specification is
    - correct, complete, consistent, and clear.
  - The requirements are realistic and verifiable.

Reviewing the Analysis Model

Review Checklist - Correctness

- Is the glossary of entity objects understandable by the user?
- Do abstract classes correspond to user-level concepts?
- Are all descriptions in accordance with the users' definitions?
- Do all entity and boundary objects have meaningful noun phrases as names?
- Do all use cases and control objects have meaningful verb phrases as names?
- Are all error cases described and handled?

Reviewing the Analysis Model

Review Checklist - Completeness

- Object questions:
  - Is it needed by any use case?
  - In which use case is it created, modified, destroyed?
  - Can it be accessed from a boundary object?
- Attribute questions:
  - When is it set?
  - What is its type?
  - Should it be a qualifier?

Reviewing the Analysis Model

Review Checklist - Completeness

- Association questions:
    - When is it traversed?
    - Why was the specific multiplicity chosen?
    - Can associations with one-to-many and many-to-many multiplicities be qualified?
- Control object question
    - Does it have the necessary associations to access the objects participating in its corresponding use case?

Reviewing the Analysis Model

Review Checklist - Consistency

- Are there multiple classes or use cases with the same name?
- Do entities with similar names denote similar concepts?
- Are there objects with similar attributes and associations that are not in the same generalization hierarchy?

Reviewing the Analysis Model

Review Checklist - Realistic

- Are there any novel features in the system?
  - Were any studies or prototypes built to ensure their feasibility?
- Can the performance and reliability requirements be met?
  - Were these requirements verified by any prototypes running on the selected hardware?

Documenting Analysis

Documenting Analysis

RAD

1. Introduction
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Objectives
2. Current system
3. Proposed system
    - 3.1 Overview
    - 3.2 Functional requirements
    - 3.3 Nonfunctional requirements
    - 3.4 System models
        - 3.4.1 Scenarios
        - 3.4.2 Use case model
        - 3.4.3 Object model
        - 3.4.4 Dynamic model
        - 3.4.5 User interface
4. Glossary

Documenting Analysis

- Object models
  - Document the objects with textual definitions.
  - Denote the relationships among objects with class diagrams.
- Dynamic models
  - Document the behavior of the object model
    - State machine diagrams and sequence diagrams

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.