SWE209
Object Oriented Analysis and Design

System Design - 2

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

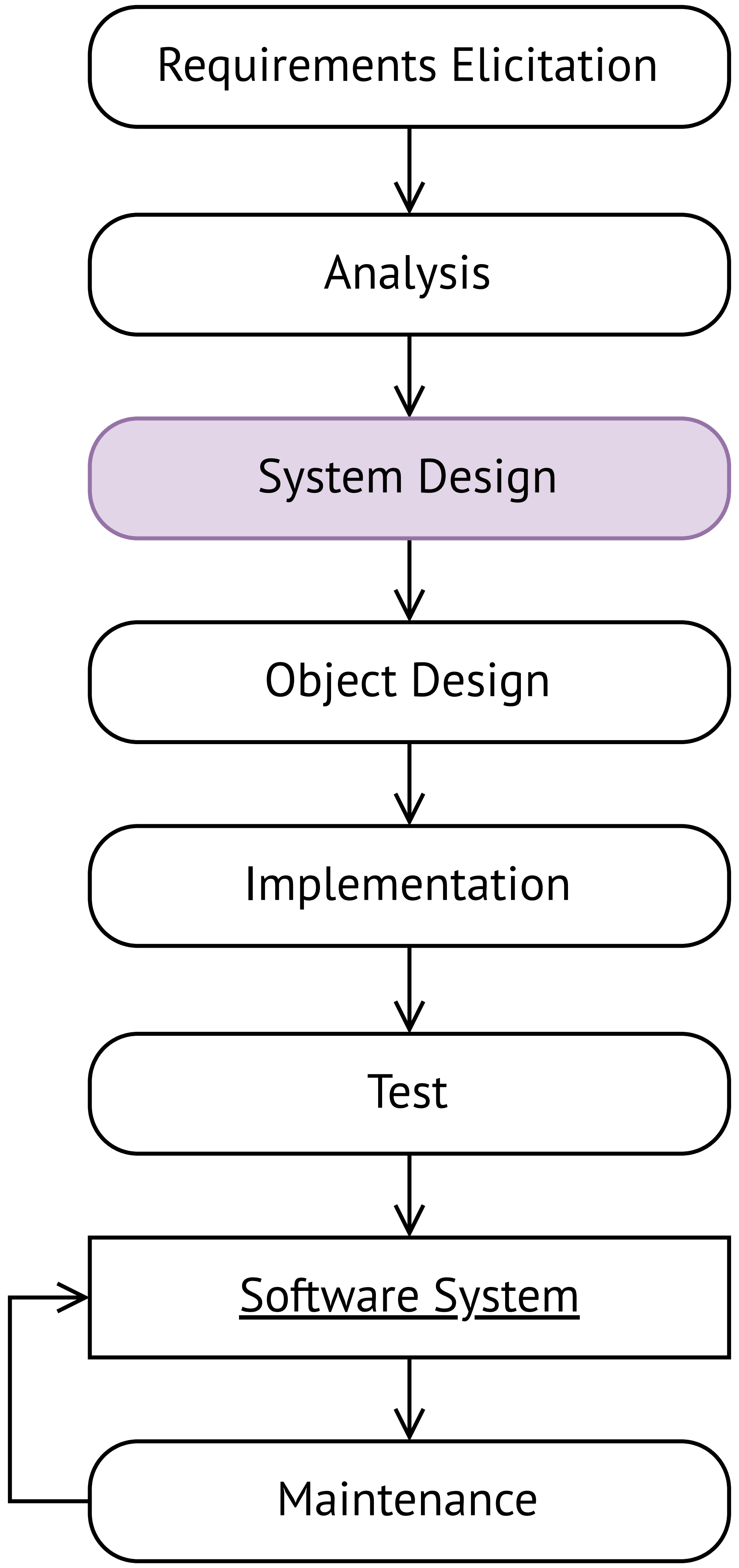
UML Deployment Diagrams

System Design Activities

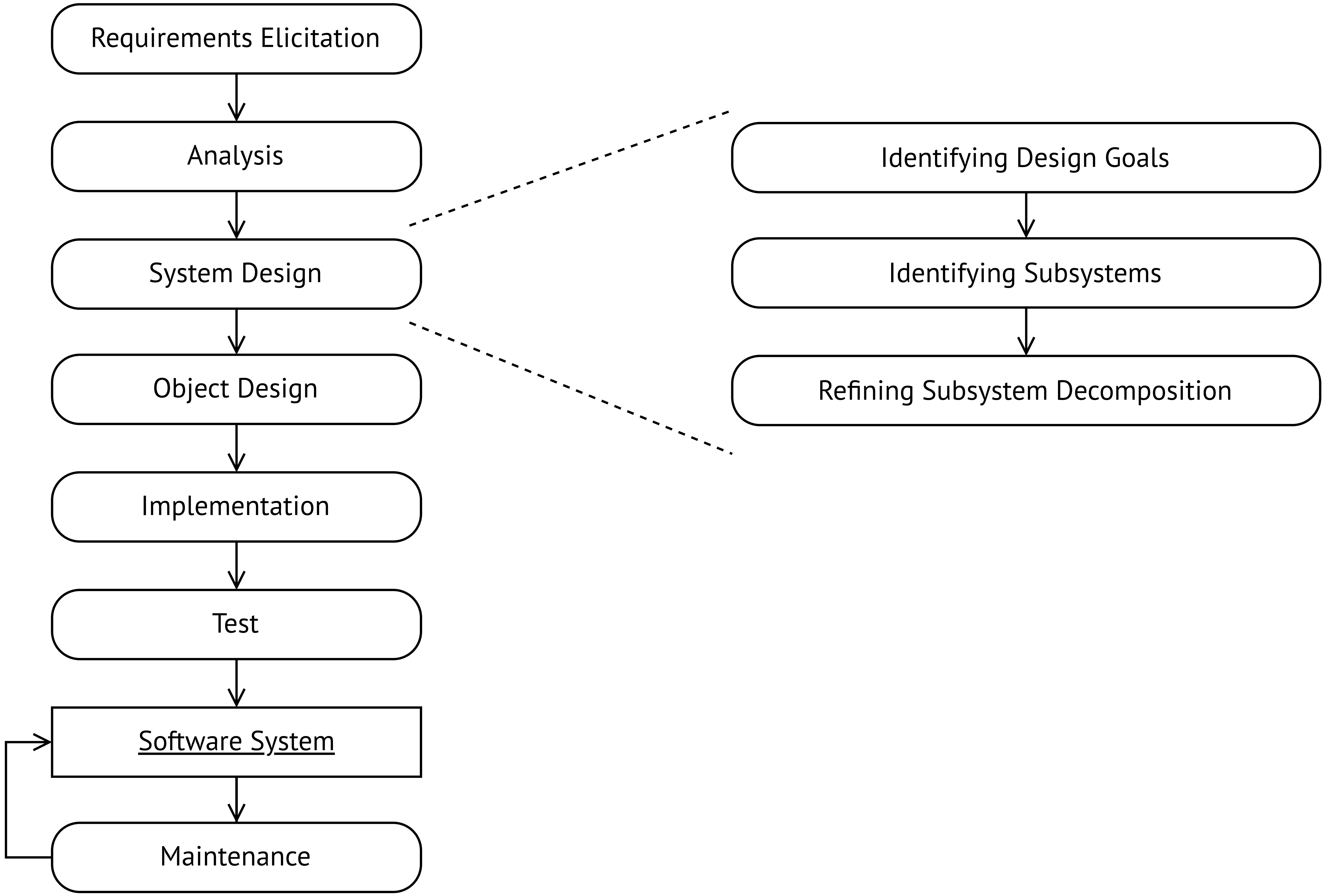
Reviewing System Design

Documenting System Design

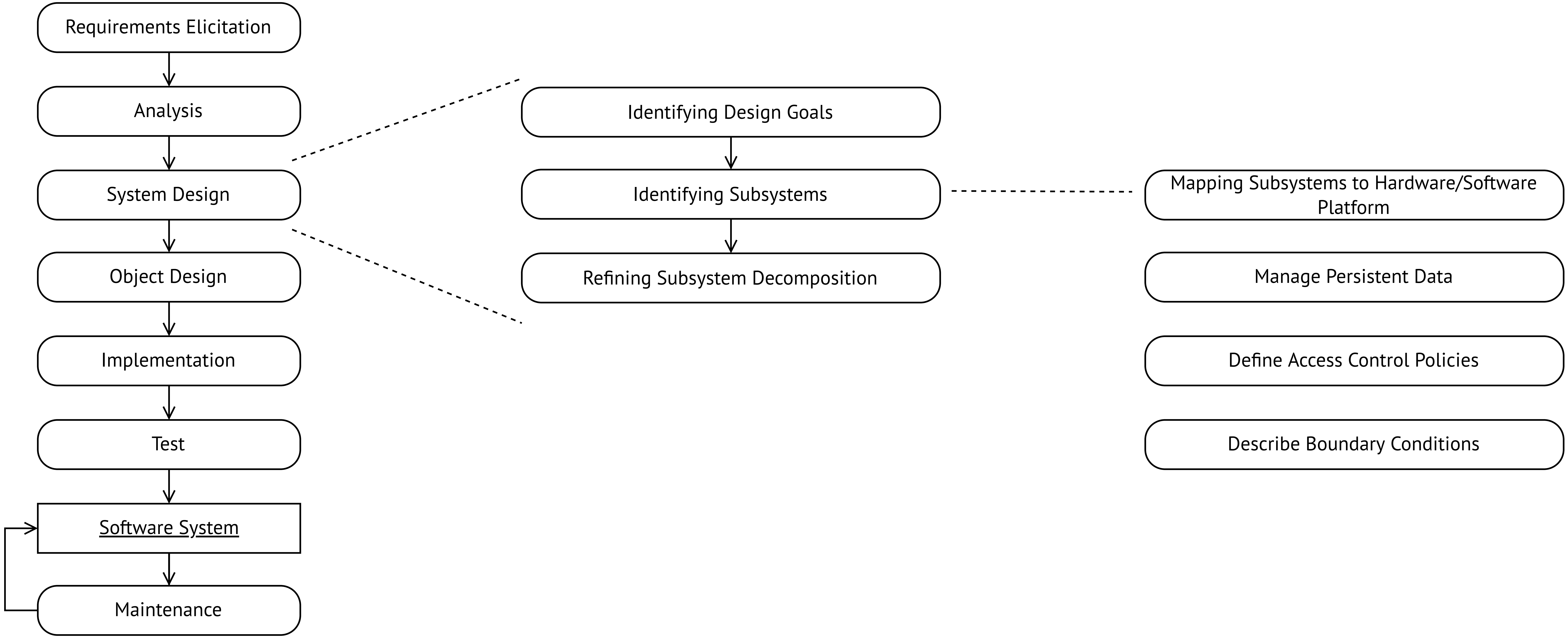
Big Picture


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

Introduction



Introduction

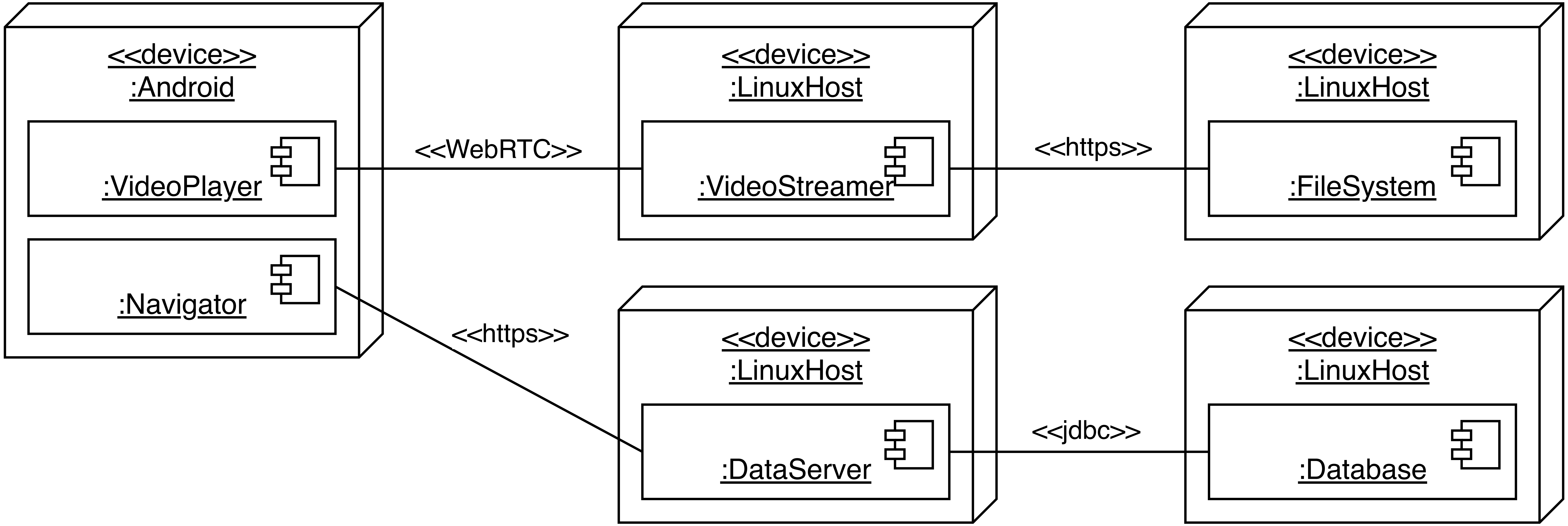


UML Deployment Diagrams

UML Deployment Diagrams

- UML deployment diagrams include components and nodes.
- Components → Self-contained entities that provide services to other components or actors.
- Node → Physical device or an execution environment in which components are executed.
- System → Interacting run-time components that can be distributed among several nodes.

UML Deployment Diagrams



System Design Activities

System Design Activities

- Additional system design activities
    - Map Subsystems to Hardware/Software Platform
    - Manage Persistent Data
    - Define Access Control Policies
    - Describe Boundary Conditions

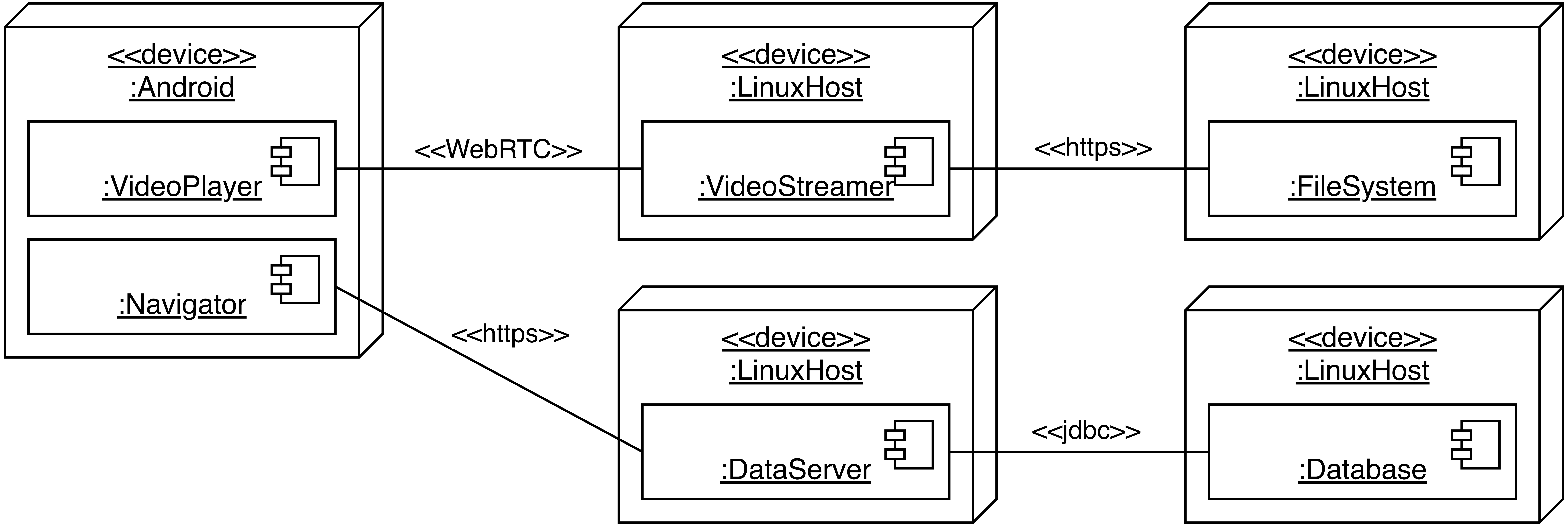
Map Subsystems to Hardware/Software Platform

- Will the software;
  - Run on just a single computer?
  - Run on more than one computer?
- Determine which subsystem will run on which device.
- Determine how subsystems will connect?
- Get help from UML deployment diagrams.

Map Subsystems to Hardware/Software Platform

- Decide additional software for the system.
  - Map them to appropriate devices.
- Decide virtual machine usage.
  - On which device virtual machine(s) will be placed?
  - How many virtual machines will be used?
  - Which components will be mapped to which virtual machines?

Map Subsystems to Hardware/Software Platform



Manage Persistent Data

- Systems commonly use persistent data.
- Three sub activities:
    - Identify Persistent Data
    - Decide Storage Management Strategy
    - Decide Storage Location
- Performed concurrent with the previous activity.

Manage Persistent Data

Identify Persistent Data

- Identify which data must be persistent.
- Obvious candidates → Entity objects
- Not all entity objects must be persistent.
- Persistent objects are not limited to entity objects.

Manage Persistent Data

Identify Persistent Data

- How can we identify persistent objects?
    - Examine the classes that must survive during shutdown.
- The system must be able to restore these long-lived objects by retrieving their attributes from storage during system initialization.

Manage Persistent Data

Decide Storage Management Strategy

- Decide how persistent objects should be stored.
  - Should the objects be retrieved quickly?
  - Must the system perform complex queries to retrieve these objects?
  - Do objects require a lot of memory or disk space?

Manage Persistent Data

Decide Storage Management Strategy

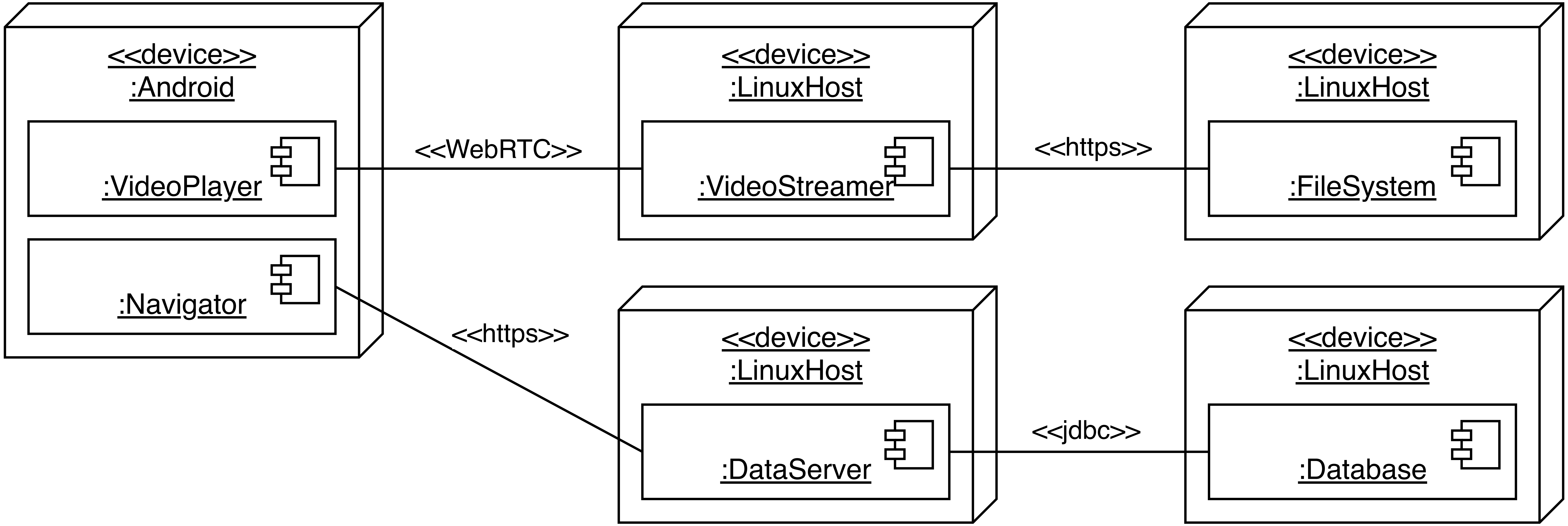
- Main options for storage management:
  - Classical File Structure
    - Binary file, CSV files, XML files, etc.
  - Database Management System
    - Relational, NoSQL, etc.

Manage Persistent Data

Decide Storage Location

- Where to store the data?
- Concurrent with/part of
    - Mapping subsystems to hardware/software platform activity

Manage Persistent Data



Define Access Control Policies

- Modern software systems → Multi-user systems
- Which actors can access and perform which operations on which objects?
  - Permissions, access rights
- Policies → Determine permissions, access rights
- Mechanisms → Manage the access to the objects and functionalities of the system.

Define Access Control Policies

- Four sub activities
  - Prepare access matrix
  - Choose access control approach
  - Choose authentication mechanism
  - Choose encryption

Define Access Control Policies

Prepare Access Matrix

Objects Actors	Corporation	LocalBranch	Account

Teller			postSmallDebit()
			postSmallCredit()
			examineBalance()

Manager		examineBranchStats()	postSmallDebit()
			postSmallCredit()
			postLargeDebit()
			postLargeCredit()
			examineBalance()
			examineHistory()

Analyst	examineGlobalStats()	examineBranchStats()	

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

Define Access Control Policies

Choose Access Control Approach

- Realization of access matrix:
    - Access Control List (ACL)
    - Role-Based Access Control (RBAC)

Define Access Control Policies

Choose Authentication Mechanism

- Authentication → Identify actor
- Authentication mechanisms:
    - Single-factor authentication (SFA)
        - User name and password
        - Find user name
        - Check whether passwords match
        - Passwords are transmitted and stored encrypted

Define Access Control Policies

Choose Authentication Mechanism

- Two-factor authentication (2FA)
  - SFA + Ask verification of the user using a device the user owns (smartphone, smart card etc.)
- Multi-factor authentication (MFA)
  - 2FA + Ask verification of the user using personal property (fingerprint, voice, etc.)
  - 2FA + Biometric verification

Define Access Control Policies

Choose Encryption

- Encryption of password → password hashing
  - Cryptographic hash algorithms
    - MD5
    - SHA-1, SHA-2, SHA-3
    - BLAKE, BLAKE2, BLAKE3

Describe Boundary Conditions

- Decide how the system is started, initialized, and shut down.
- Define how we deal with major failures.
- Examine each subsystem and each persistent object, and determine boundary use cases.
- Perform this activity under three titles:
    - Configuration, start-up and shutdown, exception handling

Describe Boundary Conditions

- Configuration
    - For each persistent object, examine in which use cases it is created or destroyed (or archived).
    - For objects that are not created or destroyed in any of the common use cases, add a use case invoked by a system administrator.
- Start-up and shutdown
    - For each component, add three use cases to start, shutdown, and configure the component.

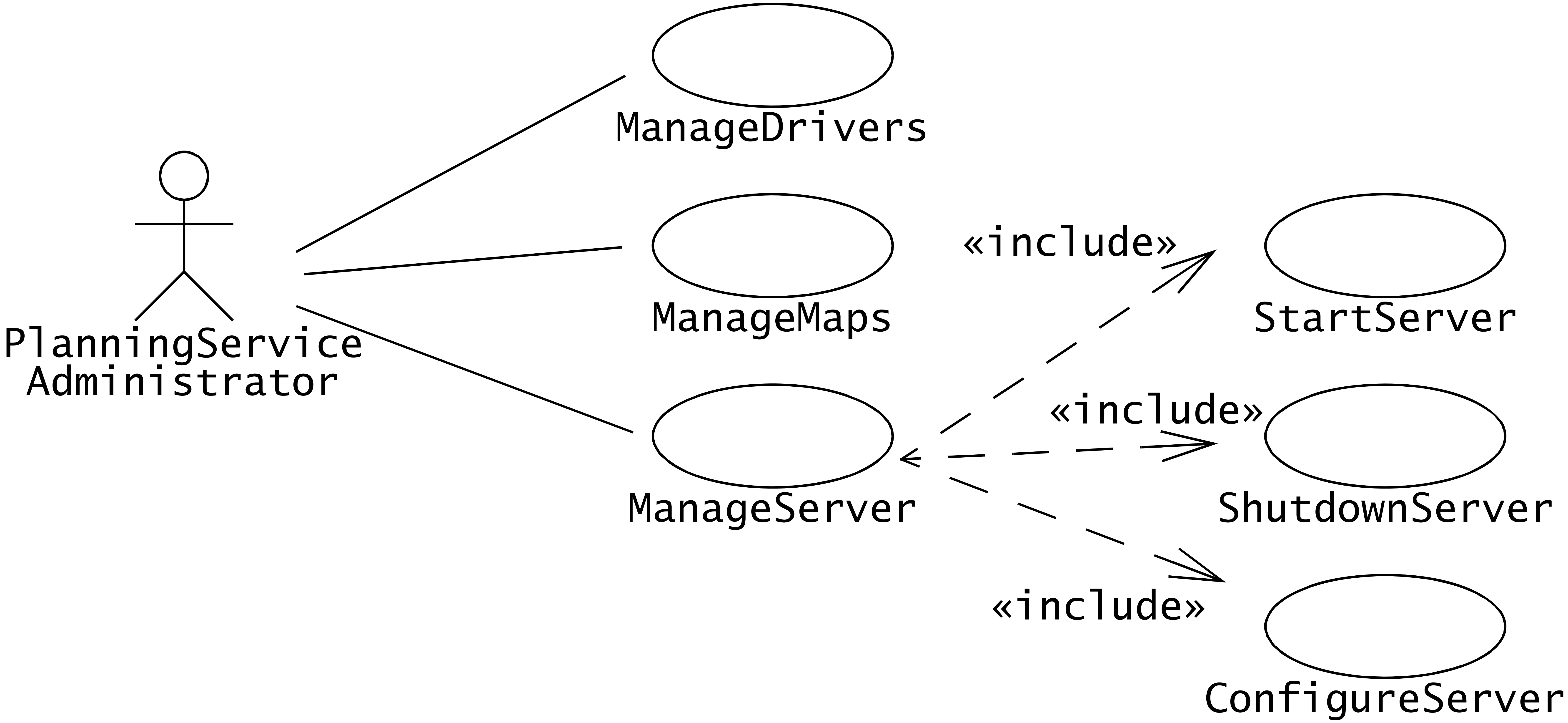
Describe Boundary Conditions

- Exception handling
  - For each type of component failure, decide how the system should react.
  - Add exceptional use cases that extends the relevant common uses cases.

Describe Boundary Conditions

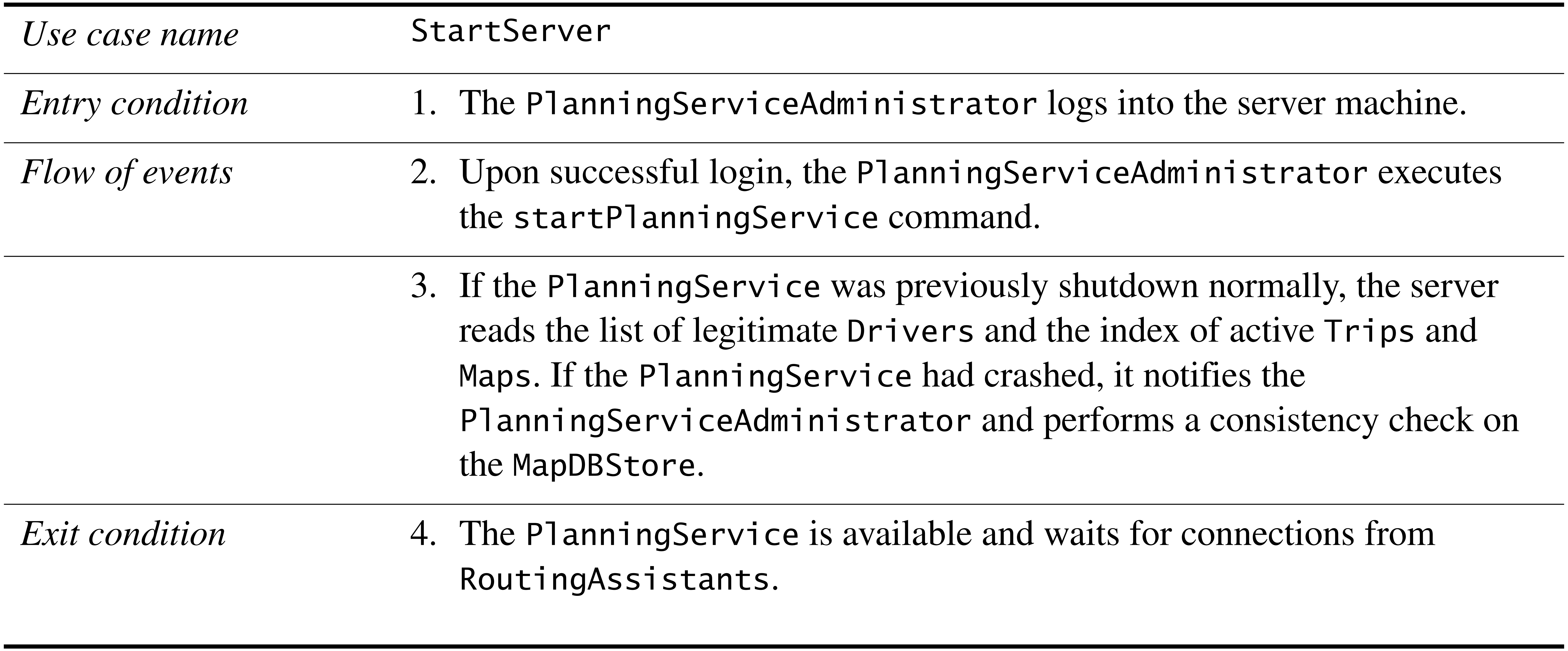
- Exception → An event or error that occurs during the execution of the system.
- Three sources:
  - Hardware failure
  - Changes in the operating environment
  - Software fault
- Exception handling → Mechanism by which a system treats an exception.

Describe Boundary Conditions



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

Describe Boundary Conditions



Use case name	`StartServer`
Entry condition	1. The `PlanningServiceAdministrator` logs into the server machine.
Flow of events	2. Upon successful login, the `PlanningServiceAdministrator` executes the `startPlanningService` command.
	3. If the `PlanningService` was previously shutdown normally, the server reads the list of legitimate `Drivers` and the index of active `Trips` and `Maps`. If the `PlanningService` had crashed, it notifies the `PlanningServiceAdministrator` and performs a consistency check on the `MapDBStore`.
Exit condition	4. The `PlanningService` is available and waits for connections from `RoutingAssistants`.

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

Reviewing System Design

Reviewing System Design

- Like analysis, system design is an evolutionary and iterative activity.
- Unlike analysis, there is no external agent to review.
- Need to organize a review process.
- Developers who were not involved in system design may review.
- Developers from another project may review.

Reviewing System Design

- Ensure that the system design model is correct, complete, consistent, realistic, and readable.
- The system design model is;
  - Correct if the analysis model can be mapped to the system design model.
  - Complete if every requirement and every system design issue has been addressed.
  - Consistent if it does not contain any contradictions.

Reviewing System Design

- Realistic if the corresponding system can be implemented.
- Readable if developers not involved in the system design can understand the model.

Documenting System Design

Documenting System Design

- System Design Document → SDD
- Describes;
    - Design goals
    - Subsystem decomposition (UML class diagrams)
    - Hardware/software mapping (with UML deployment diagrams)
    - Data management
    - Access control
    - Boundary conditions

Documenting System Design

- SDD is used to define interfaces between teams of developers
- SDD serves as a reference when architecture-level decisions need to be revisited.
- Audience for the SDD;
  - Project managers
  - System architects (developers who participate in the system design)
  - Developers who design and implement each subsystem

Documenting System Design

1. Introduction
   - 1.1 Purpose
   - 1.2 Design Goals
2. Current Software Architecture
3. Proposed Software Architecture
   - 3.1 Overview
   - 3.2 Subsystem Decomposition
   - 3.3 Hardware/Software Mapping
   - 3.4 Persistent Data Management
   - 3.5 Access Control and Security
   - 3.6 Boundary Conditions
4. Subsystem Services
5. Glossary

Documenting System Design

- SDD is written after the initial system decomposition is done
- System architects should not wait until all system design decisions are made before publishing the document.
- SDD, once published, is baselined and put under configuration management.
- The revision history section of the SDD provides a history of changes.

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.