SWE209
Object Oriented Analysis and Design

System Design - 1

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

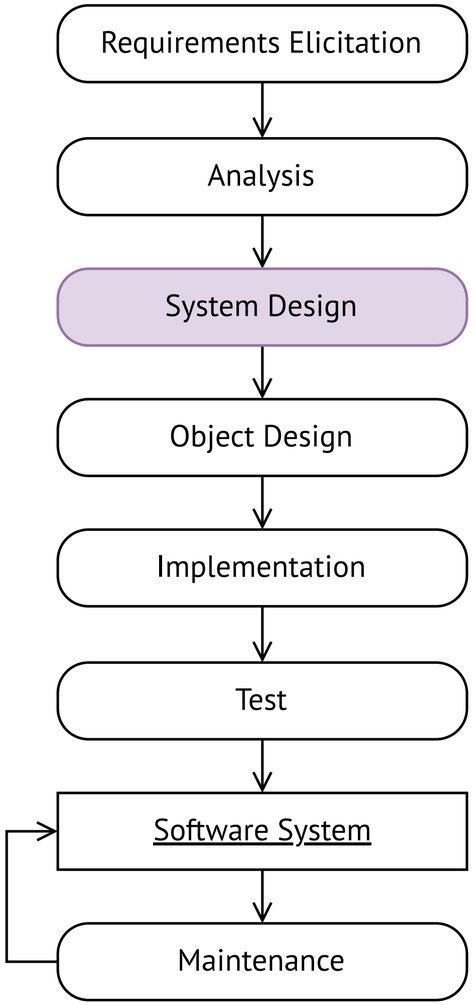
Introduction

System Design Concepts

Architectural Styles

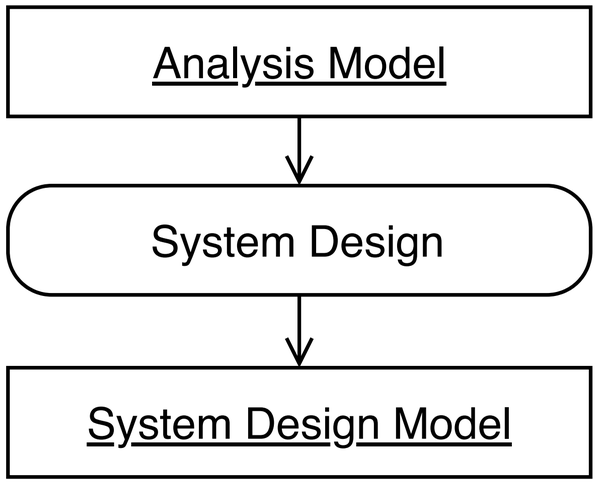
System Design Activities

Big Picture

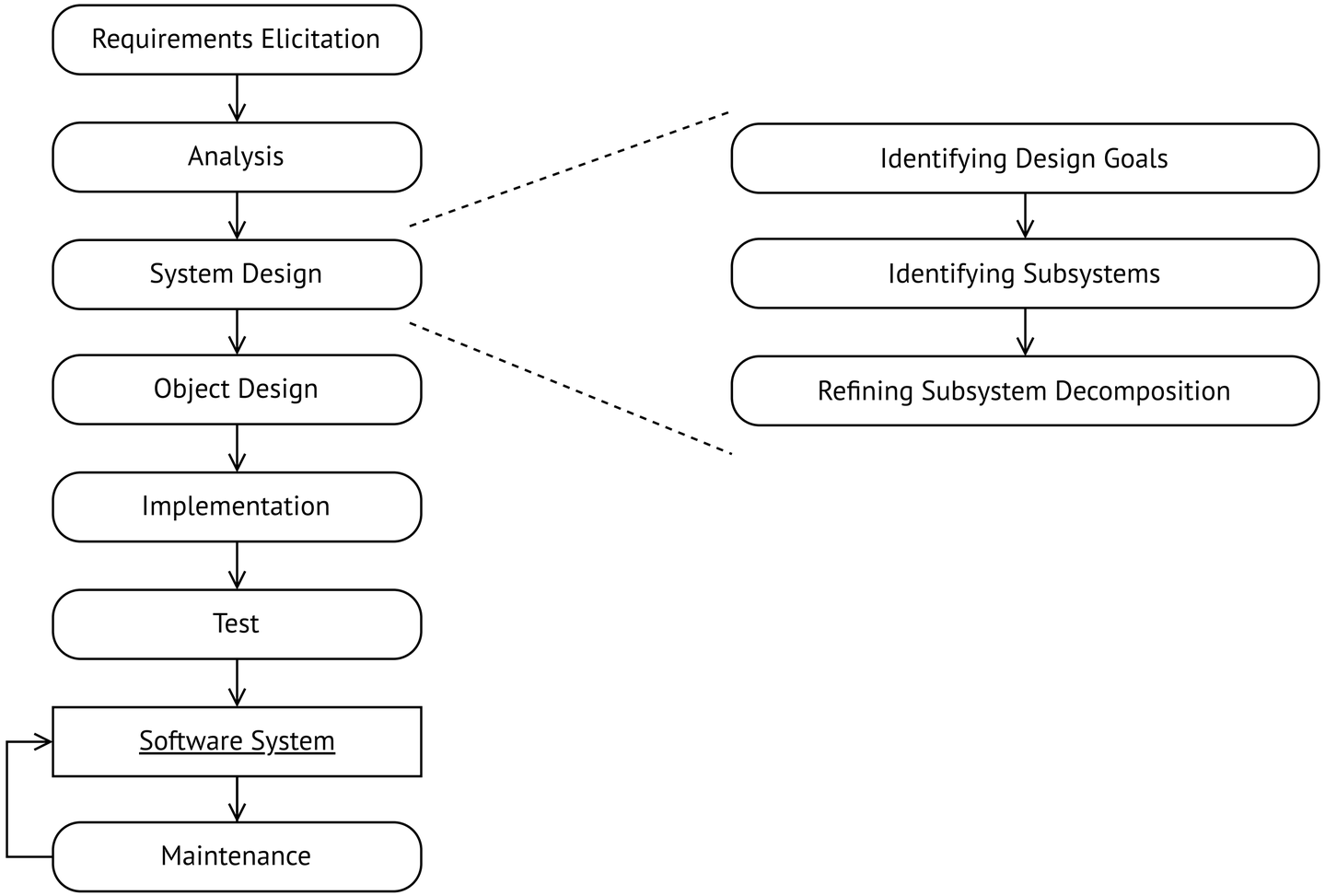

Requirements Elicitation
Analysis
System Design
Object Design
Implementation
Test
Software System
Maintenance

Introduction

- System design → Transform analysis model into system design model.



Introduction



Introduction

- Requirements elicitation and analysis
  - The purpose and the functionality of the system
- System design, object design, implementation → Construction of the system
- System design focuses on;
  - Decomposing the system into manageable parts
  - Processes, data structures, and software and hardware components

Introduction

- Analysis model
  - Description of the system from the actors’ point of view
  - No information about the internal structure, hardware configuration etc.
- System design
  - Information about the realization of the system

Introduction

- Products of system design:
  - Design goals → Qualities of the system
  - Software architecture → Subsystem decomposition
  - Boundary use cases → System configuration

System Design Concepts

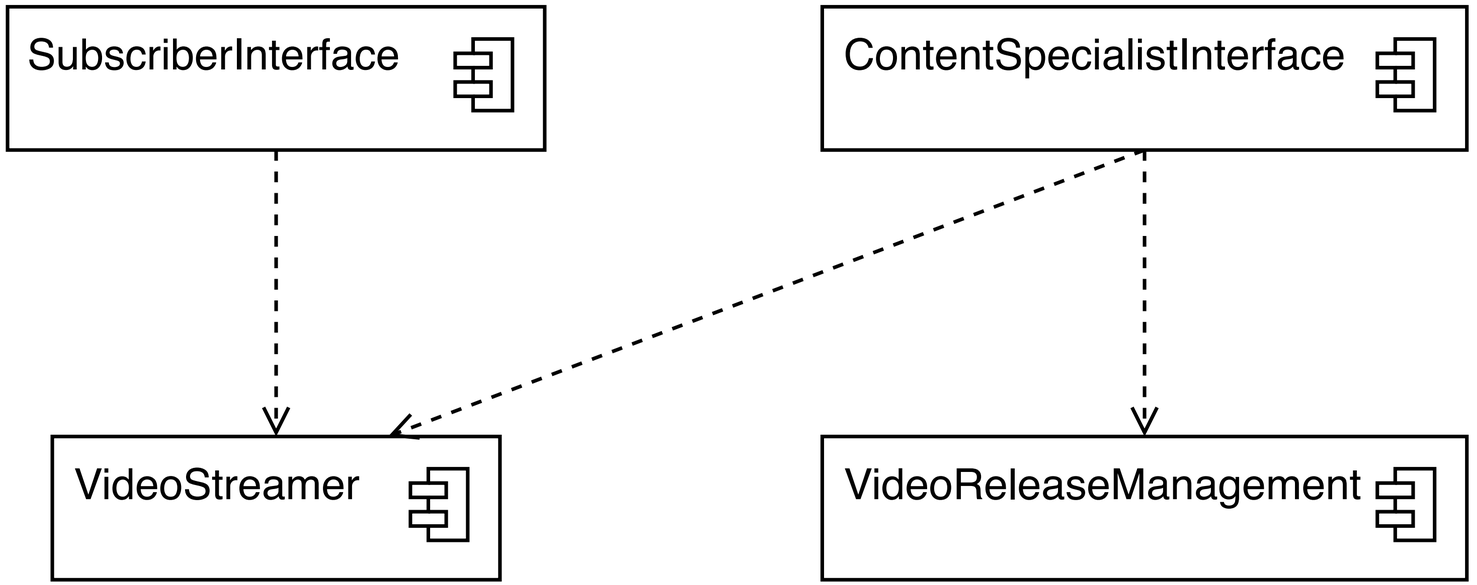
System Design Concepts

- Subsystem
- Service and Subsystem Interface
- Coupling and Cohesion
- Layer

Subsystem

- Subsystem → Replaceable part of the system
  - Well-defined interfaces
    - Encapsulation of the state and behavior of the contained classes
- Amount of work can be handled by a single developer or a single development team.
- Recursively decompose system into subsystems.

Subsystem


SubscriberInterface
ContentSpecialistInterface
VideoStreamer
VideoReleaseManagement

Subsystem

- UML components
  - Logical component → Subsystem that has no explicit run-time equivalent
  - Physical component → Subsystem that as an explicit run-time equivalent
- Programming languages may provide constructs for modeling subsystems
  - Kotlin Packages
  - Java Packages
  - C# Namespaces

Service and Subsystem Interface

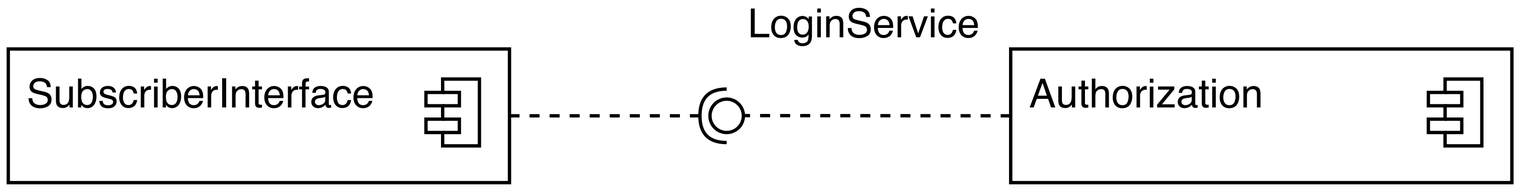
- Subsystems provide services to other subsystems.
- **Service** → A set of related operations that share a common purpose
- **Subsystem interface** → The set of operations of a subsystem that are available to other subsystems.
  - Name of the operations
  - Parameters, types, return values

Service and Subsystem Interface

- System design → Define the services provided by each subsystem
  - Enumerate the operations, their parameters, and their high-level behavior.
- Object design → Application programming interface (API)
  - Refinement, extension of the subsystem interfaces.
  - The type of the parameters and the return value of each operation.

Service and Subsystem Interface

- Provided and required interfaces in UML;
  - Assembly connectors, ball-and-socket connectors
  - Lollipop notation
- Provided interface → Ball icon
- Required interface → Socket icon



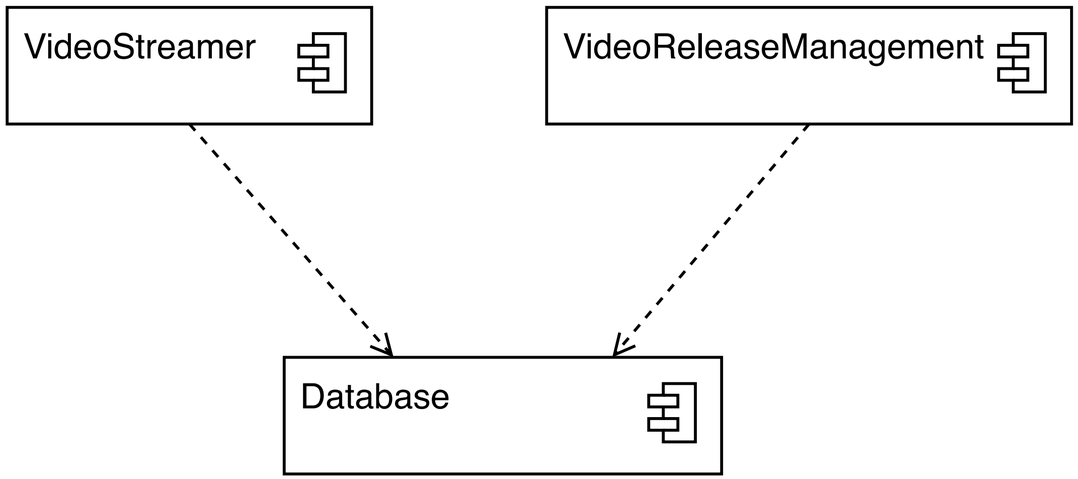
Coupling and Cohesion

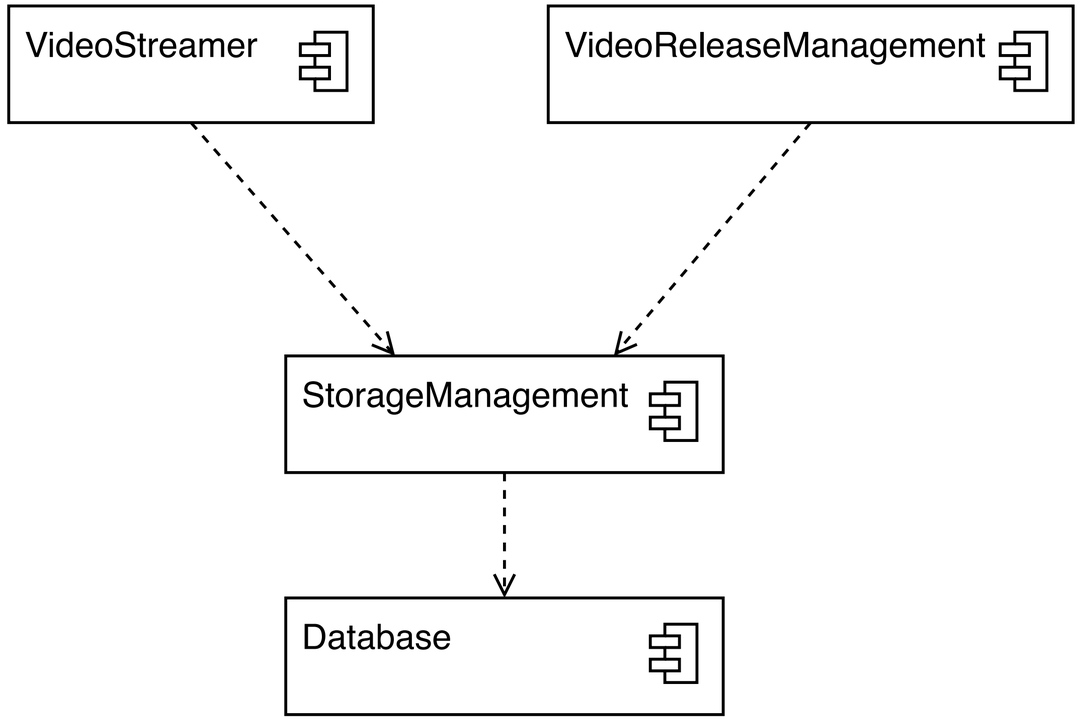
Coupling

- Coupling → The number of dependencies between subsystems.
- Loosely coupled → Modifications to one of the subsystems will have little impact on the other.
- Strongly coupled → Modifications to one subsystem is likely to have impact on the other.
- Desirable property of a subsystem decomposition
  - Create subsystems as loosely coupled as reasonable.

Coupling and Cohesion

Coupling





Coupling and Cohesion

Cohesion

- Cohesion → The number of dependencies within a subsystem.
- High cohesion → Subsystem contains many objects that are related to each other and perform similar tasks.
- Low cohesion → Subsystem contains a number of unrelated objects.
- Desirable property of a subsystem decomposition
  - Leading to subsystems with high cohesion

Coupling and Cohesion

Coupling vs Cohesion

- Trade-off between cohesion and coupling
  - Increasing cohesion increases coupling.
- Good heuristic → 7 ± 2 concepts at any one level of abstraction
- Good systems design can often be accomplished with just three layers.

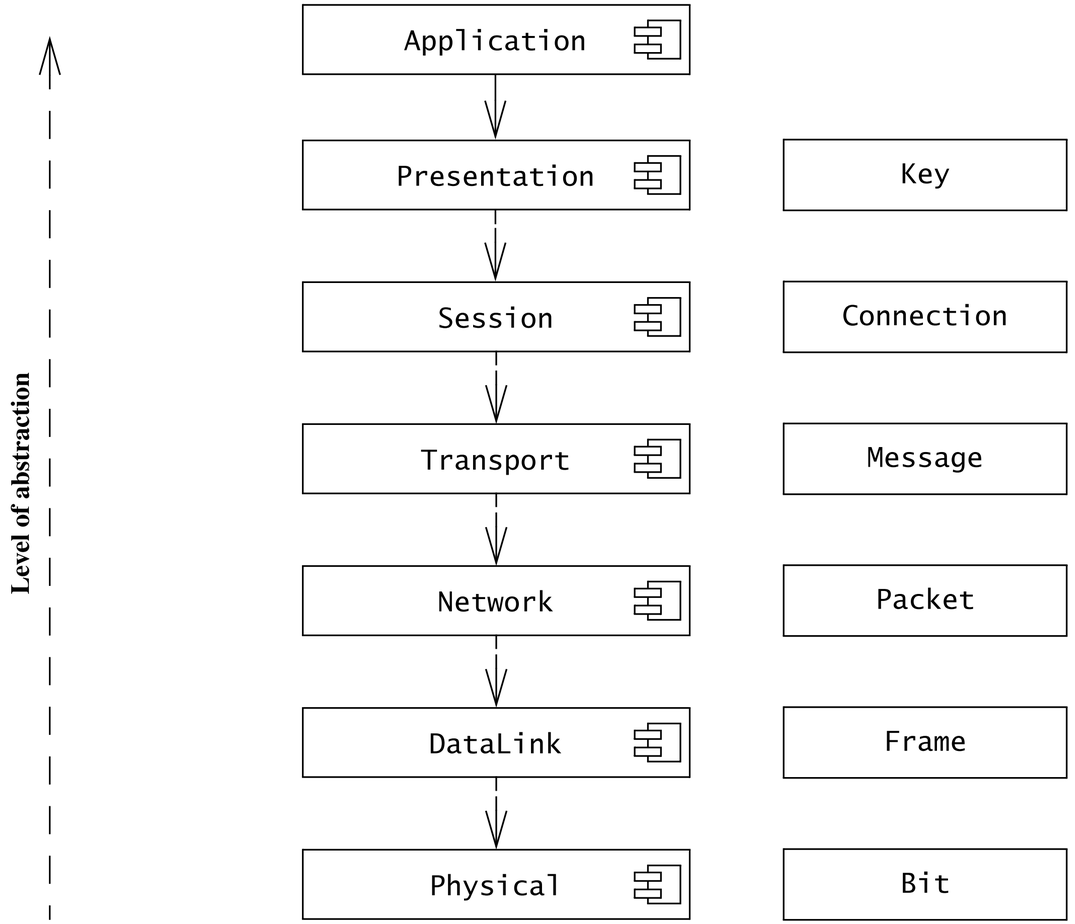
Layer

- Hierarchical decomposition → Ordered set of layers
- Layer → Grouping of subsystems providing related services
- Layers ordering
  - A layer can depend only on lower level layers
  - A layer has no knowledge of the layers above it
- Bottom layer → Layer that does not depend on any other layer
- Top layer → Layer that is not used by any other layer

Layer

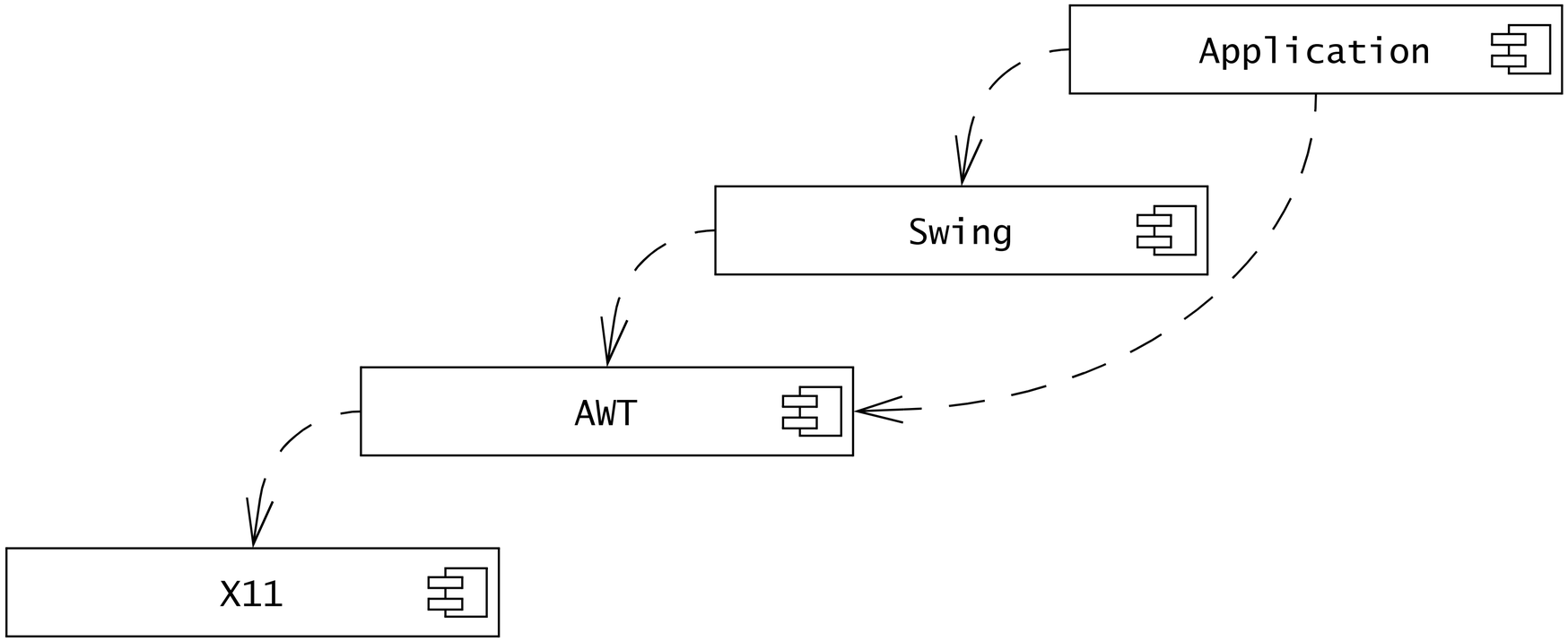
- Closed architecture → A layer can access only the layer just below it.
- Open architecture → Layer can access layers at deeper levels.

Layer



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

Layer



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

Architectural Styles

Architectural Styles

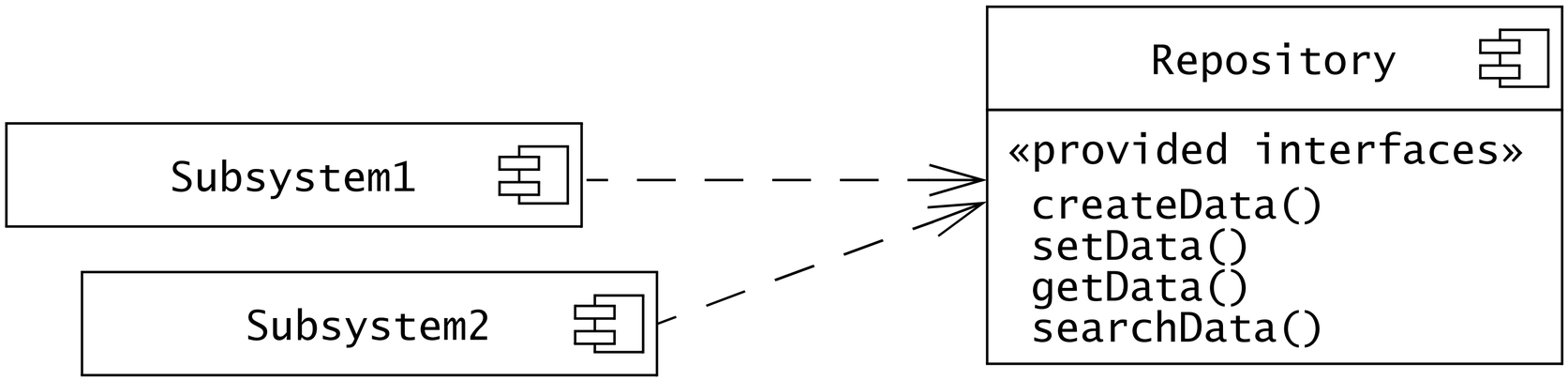
- During system design we start to create the architecture of the system.
- In other words we start define the software architecture.
- Software architecture consists of;
  - Subsystems
  - Control and data flow between subsystems
- Architectural styles help to architect the software.
  - Define the organization of a software system.

Architectural Styles

- Repository
- Model View Controller (MVC)
- Client Server
- Peer-to-Peer
- Three-Tier
- Four-Tier
- Pipe and Filter

Repository

- Subsystems access and modify a single data structure.
- This data structure is called the central repository.



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

Repository

- Subsystems
  - Relatively independent
  - Interact only through the repository.
- Control flow is managed by central repository or subsystems.
- Examples
  - Database management systems
  - Compilers
  - Software Development Environments

Repository

- Well suited for applications with constantly changing data processing tasks.
- Disadvantages
  - Central repository can quickly become a bottleneck.
  - Coupling between each subsystem and the repository is high.

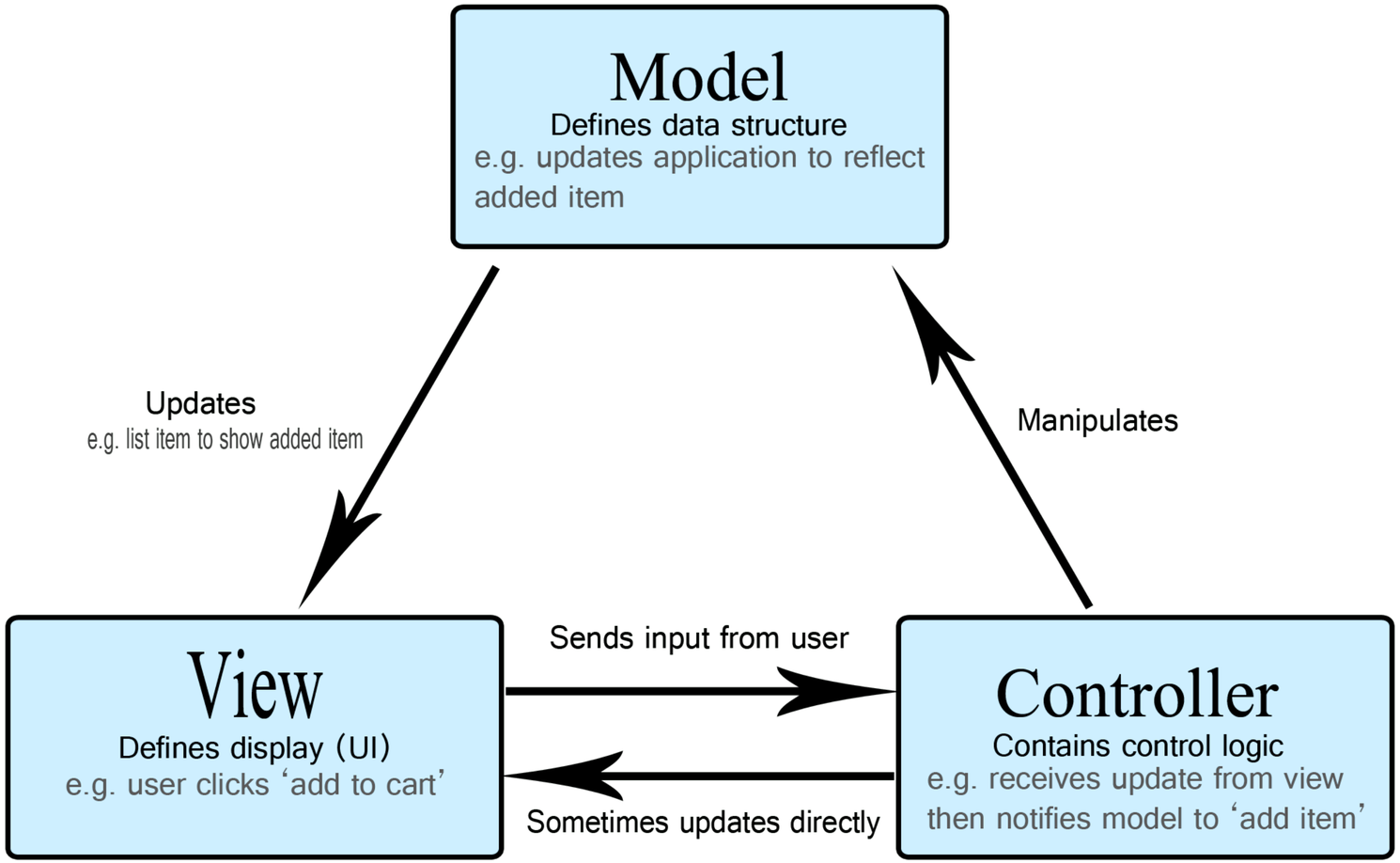
Model View Controller (MVC)

- Subsystems are classified into three different types:
  - **Model** subsystems → Maintain domain knowledge
  - **View** subsystems → Display it to the user
  - **Controller** subsystems → Manage the sequence of interactions with the user
- Model subsystems do not depend on any view or controller subsystem.
  - Changes in state are propagated to the view subsystem via a subscribe/notify protocol.

Model View Controller (MVC)

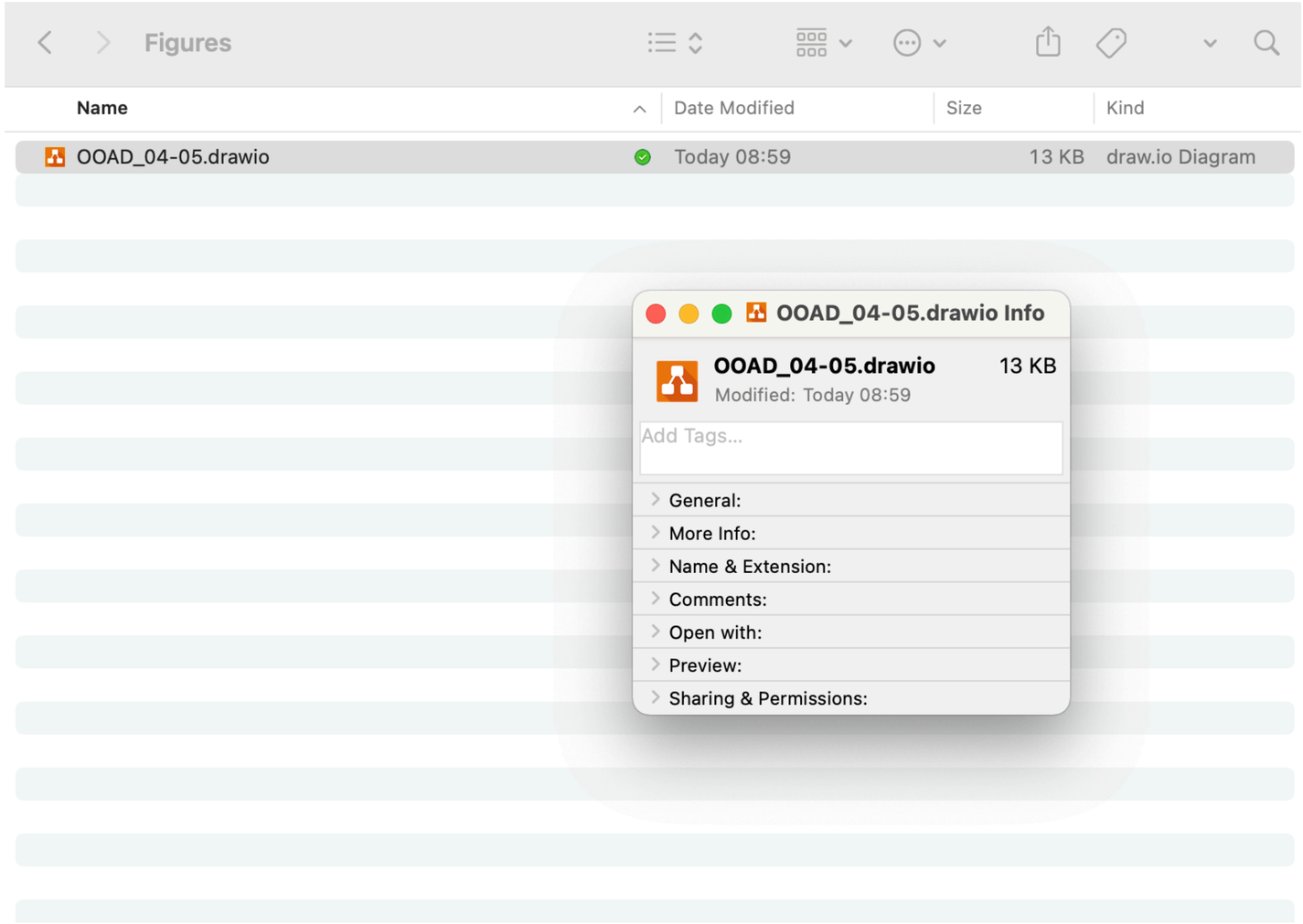
- MVC is a special case of the repository
  - Model implements the central data structure
  - Control objects dictate the control flow

Model View Controller (MVC)



https://developer.mozilla.org/en-US/docs/Glossary/MVC

Model View Controller (MVC)



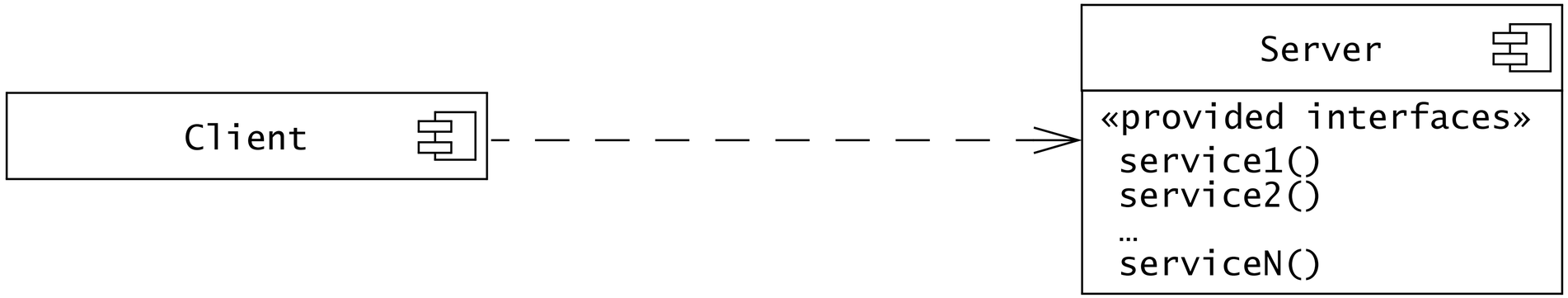
Model View Controller (MVC)

- The rationale of MVC
  - View and Controller, are more subject to change than is Model.
  - Changes in the View do not have any effect on the Model subsystems.
- Well suited for interactive systems.
- Can be used for maintaining consistency across distributed data.
- Same performance bottleneck problem as for other repository styles.

Client Server

- Server provides services to clients.
- Request for a service
  - Remote procedure call
  - Common object request broker (CORBA)
  - HTTP
- Control flow in the clients and the servers is independent except for synchronization to manage requests or to receive results.

Client Server

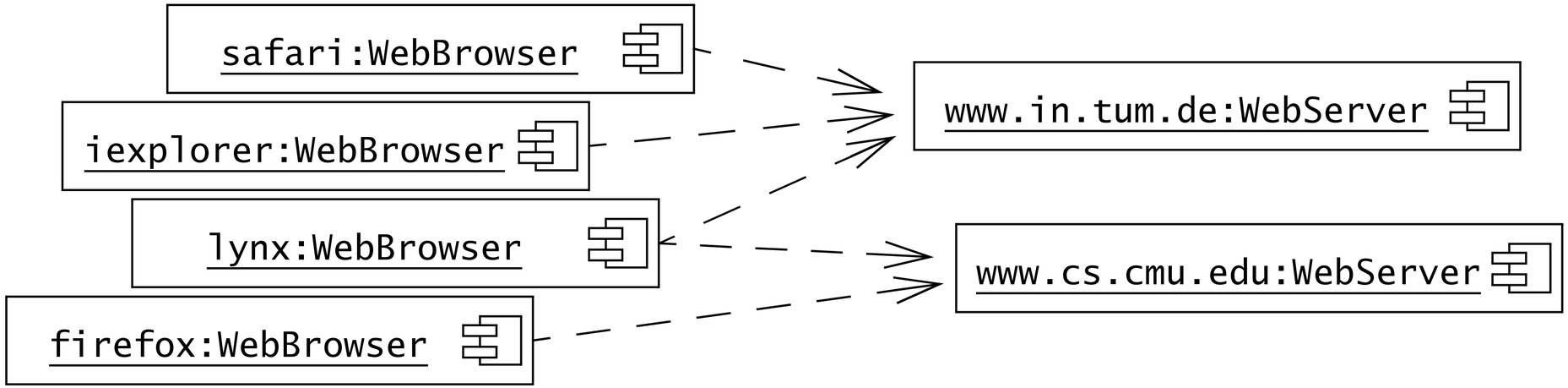


Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Client Server

- Composed of clients and servers.
- Client
  - Interact with users, gather user requests, prepare them for the server and send to the server.
- Server
  - Take the request, perform the necessary operations and return the results to the client.

Client Server



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

Peer-to-Peer

- Generalization of the client server architectural style.
- Subsystems can act both as client or as servers.
  - Each subsystem can request and provide services.
- Independent control flow

Peer-to-Peer

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

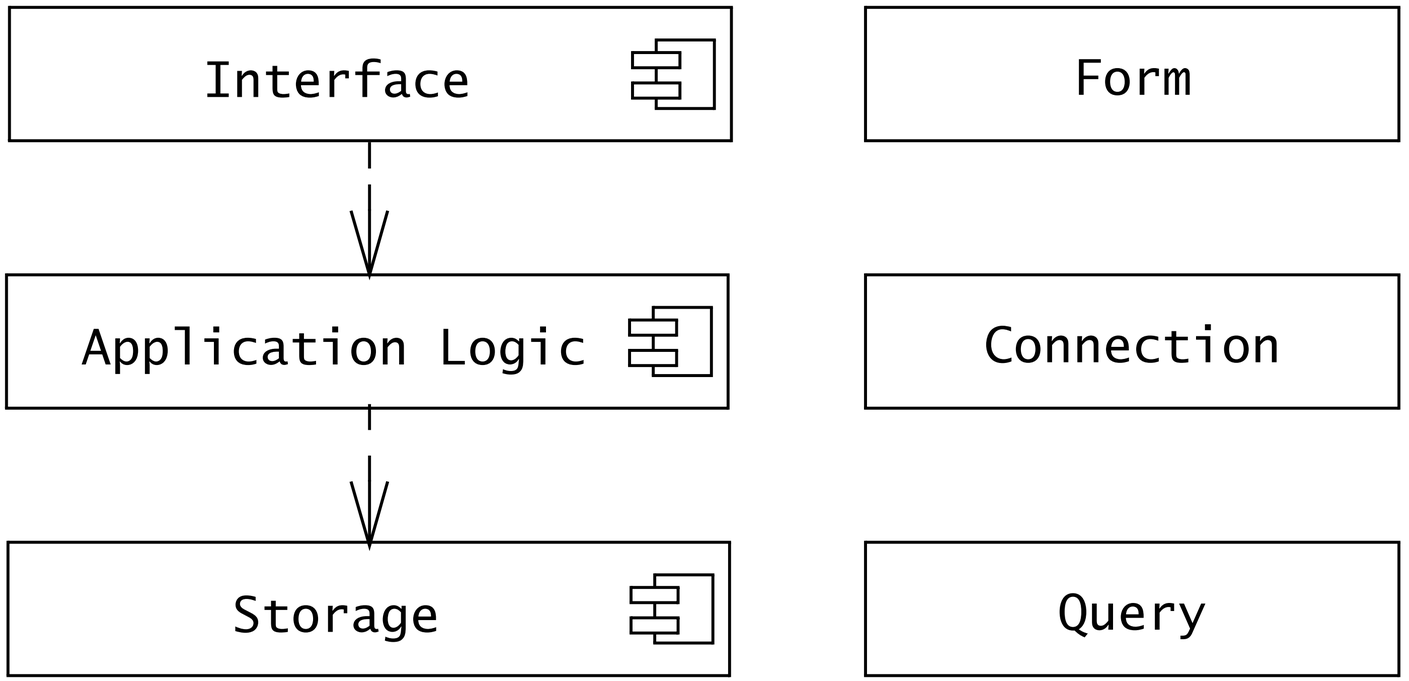
Three-Tier

- The three-tier architectural style organizes subsystems into three layers:
  - Interface layer
  - Application logic layer
  - Storage layer
- Interface layer
  - Boundary objects

Three-Tier

- Application logic layer
  - Control and entity objects
  - Realization of the processing, rule checking, and notification required by the application.
- The storage layer
  - Realization of the storage, retrieval, and query of persistent objects.

Three-Tier



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

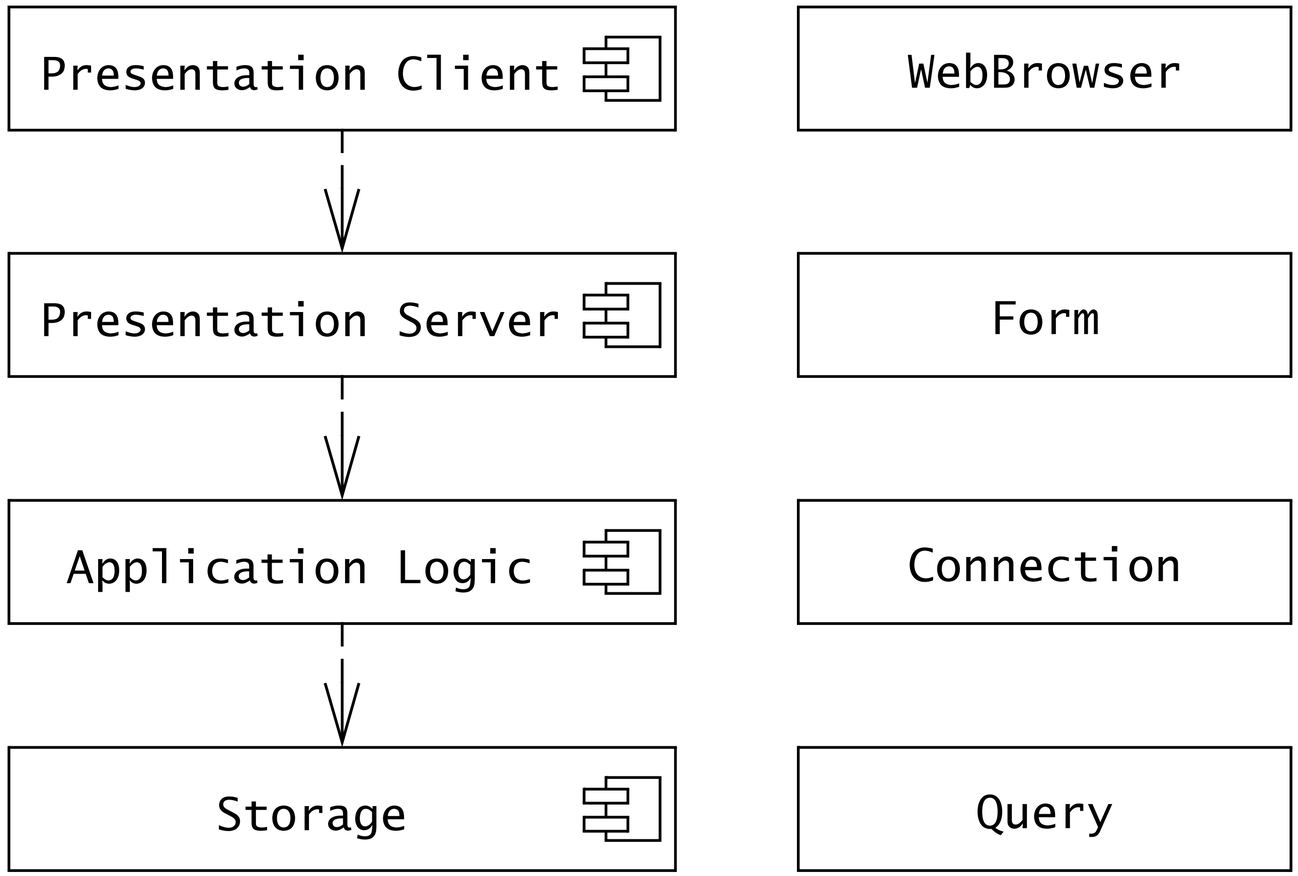
Three-Tier

- Storage layer can be shared by several different applications operating on the same data.
- Separation between the interface layer and the application logic layer
  - Enable the development or modification of different user interfaces for the same application logic.

Four-Tier

- A type of three-tier architecture
- Interface layer = Presentation Client layer + Presentation Server layer
- Presentation Client layer is located on the user machines.
- Presentation Server layer can be located on one or more servers.
- Enables a wide range of different presentation clients in the application.

Four-Tier



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

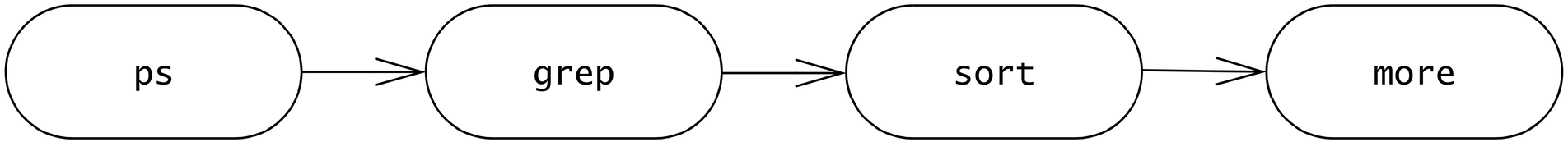
Pipe and Filter

- Subsystems process data received from a set of inputs and send results to other subsystems via a set of outputs.
- The subsystems are called filters.
- The associations between the subsystems are called pipes.
- Each filter;
  - Knows only the content and the format of the data received on the input pipes.
  - Is executed concurrently, and synchronization is accomplished via the pipes.

Pipe and Filter

```
% ps auxwww | grep dutoit | sort | more

dutoit   19737  0.2  1.6 1908 1500 pts/6    O 15:24:36  0:00 -tcsh
dutoit   19858  0.2  0.7  816  580 pts/6    S 15:38:46  0:00 grep dutoit
dutoit   19859  0.2  0.6  812  540 pts/6    O 15:38:47  0:00 sort
```



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

Pipe and Filter

Pipe and Filter

- Well suited for systems that apply transformations to streams of data.
- Not suited for systems that require more complex interactions between components.

System Design Activities

System Design Activities

- Identifying Design Goals
- Identifying Subsystems
- Refining Subsystem Decomposition

Identifying Design Goals

- The definition of design goals is the first step of system design.
- Identification of the qualities that our system should focus on.
- Many design goals can be inferred from
  - Nonfunctional requirements
  - Application domain
  - Client
  - Developer anticipation

Identifying Design Goals

Example - Design Goals for a Sample Application

Nonfunctional requirements for MyTrip

1. `MyTrip` is in contact with the `PlanningService` via a wireless modem. Assume that the wireless modem functions properly at the initial destination.
2. Once the trip has been started, `MyTrip` should give correct directions even if modem fails to maintain a connection with the `PlanningService`.
3. `MyTrip` should minimize connection time to reduce operation costs.
4. Replanning is possible only if the connection to the `PlanningService` is possible.
5. The `PlanningService` can support at least 50 different drivers and 1,000 trips.

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

- Four design goals: Reliability, Fault Tolerance, Security, Modifiability

Identifying Design Goals

Example - Design Goals for a Sample Application

- Reliability
  - MyTrip should be reliable.
  - Generalization of nonfunctional requirement 2.
- Fault Tolerance
  - MyTrip should be fault tolerant to loss of connectivity with the routing service.
  - Rephrased nonfunctional requirement 2.

Identifying Design Goals

Example - Design Goals for a Sample Application

- Security
  - MyTrip should be secure, i.e., not allow other drivers or unauthorized users to access a driver's trips.
  - Deduced from application domain.
- Modifiability
  - MyTrip should be modifiable to use different routing services.
  - Anticipation of change by developers.

Identifying Design Goals

- We define design goals according to different criteria.
- Five categories:
  - Performance, dependability, cost, maintenance, end user
- Performance, dependability, and end user criteria
  - Specified in the requirements or inferred from the application domain.
- Cost and maintenance criteria
  - Dictated by the customer and the supplier.

Identifying Design Goals

Performance Criteria

- Speed and space requirements imposed on the system.
- Response time
  - How soon is a user request acknowledged after the request has been issued?
- Throughput
  - How many tasks can the system accomplish in a fixed period of time?
- Memory
  - How much space is required for the system to run?

Identifying Design Goals

Dependability Criteria

- How much effort should be expended in minimizing system crashes and their consequences.
- Robustness
  - Ability to survive invalid user input
- Reliability
  - Difference between specified and observed behavior
- Availability
  - Percentage of time that system can be used to accomplish normal tasks

Identifying Design Goals

Dependability Criteria

- Fault tolerance
  - Ability to operate under erroneous conditions
- Security
  - Ability to withstand malicious attacks
- Safety
  - Ability to avoid endangering human lives, even in the presence of errors and failures

Identifying Design Goals

Cost Criteria

- Cost to develop the system, to deploy it, and to administer it.
- Cost criteria = design considerations + managerial costs
- Development cost
  - Cost of developing the initial system
- Deployment cost
  - Cost of installing the system and training the users

Identifying Design Goals

Cost Criteria

- Upgrade cost
  - Cost of translating data from the previous system
- Maintenance cost
  - Cost required for bug fixes and enhancements to the system
- Administration cost
  - Cost required to administer the system

Identifying Design Goals

Maintenance Criteria

- How difficult it is to change the system after deployment?
- Harder to optimize and plan for.
- Extensibility
  - How easy is it to add functionality or new classes to the system?
- Modifiability
  - How easy is it to change the functionality of the system

Identifying Design Goals

Maintenance Criteria

- Adaptability
  - How easy is it to port the system to different application domains?
- Portability
  - How easy is it to port the system to different platforms?
- Readability
  - How easy is it to understand the system from reading the code?

Identifying Design Goals

Maintenance Criteria

- Traceability of requirements
  - How easy is it to map the code to specific requirements?

Identifying Design Goals

End User Criteria

- Qualities that are desirable from a users' point of view.
- Utility
  - How well does the system support the work of the user?
- Usability
  - How easy is it for the user to use the system?

Identifying Design Goals

- A small subset of these criteria can be simultaneously taken into account.
- Prioritize design goals and make trade-offs.

Identifying Subsystems

- Similar to identifying objects during analysis.
- Abbotts's heuristics can be applied.
- Subsystem decomposition is also constantly revised:
  - Merge subsystems
  - Split subsystems
  - Add new subsystems
- Brainstorming

Identifying Subsystems

- Derive initial subsystem decomposition from functional requirements.
- Keep functionally related objects together.
- Assign participating objects to subsystems.

Identifying Subsystems

- Heuristics for grouping objects into subsystems:
  - Assign objects identified in one use case into the same subsystem.
  - Create a dedicated subsystem for objects used for moving data among subsystems.
  - Minimize the number of associations crossing subsystem boundaries.
  - All objects in the same subsystem should be functionally related.

Identifying Subsystems

Use case name	`PlanTrip`
Flow of events	1. The `Driver` activates her computer and logs into the trip-planning Web service. 2. The `Driver` enters constraints for a trip as a sequence of destinations. 3. Based on a database of maps, the planning service computes the shortest way of visiting the destinations in the order specified. The result is a sequence of segments binding a series of crossings and a list of directions. 4. The `Driver` can revise the trip by adding or removing destinations. 5. The `Driver` saves the planned trip by name in the planning service database for later retrieval.

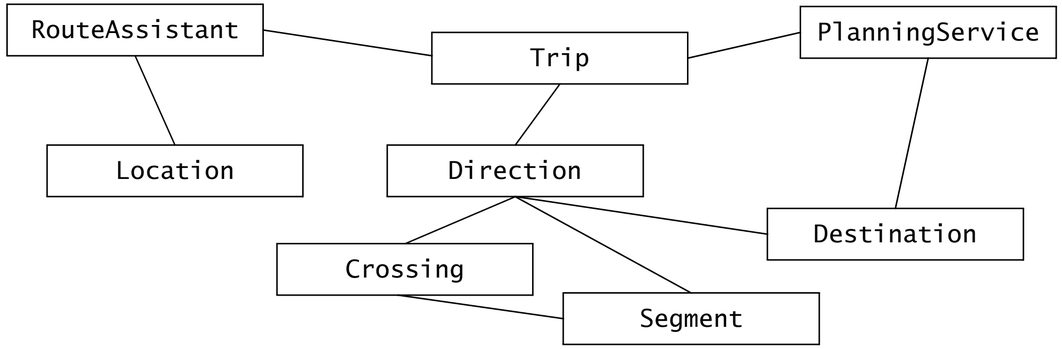
Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

Identifying Subsystems

Use case name	`ExecuteTrip`
Flow of events	1. The `Driver` starts her car and logs into the onboard route assistant. 2. Upon successful login, the `Driver` specifies the planning service and the name of the trip to be executed. 3. The onboard route assistant obtains the list of destinations, directions, segments, and crossings from the planning service. 4. Given the current position, the route assistant provides the driver with the next set of directions. 5. The `Driver` arrives to destination and shuts down the route assistant.

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

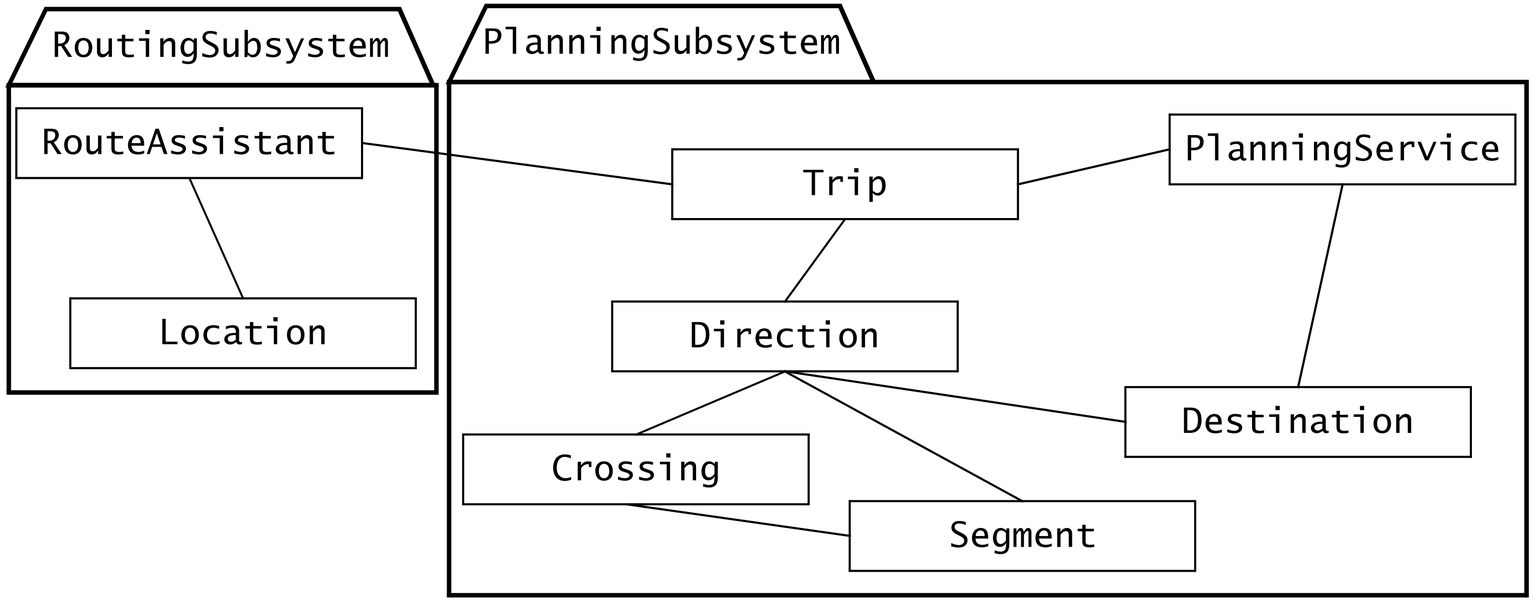
Identifying Subsystems



Crossing	A Crossing is a geographical point where several Segments meet.
Destination	A Destination represents a location where the driver wishes to go.
Direction	Given a Crossing and an adjacent Segment, a Direction describes in natural language how to steer the car onto the given Segment.
Location	A Location is the position of the car as known by the onboard GPS system or the number of turns of the wheels.
PlanningService	A PlanningService is a Web server that can supply a trip, linking a number of destinations in the form of a sequence of Crossings and Segments.
RouteAssistant	A RouteAssistant gives Directions to the driver, given the current Location and upcoming Crossing.
Segment	A Segment represents the road between two Crossings.
Trip	A Trip is a sequence of Directions between two Destinations.

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

Identifying Subsystems



`PlanningSubsystem` The `PlanningSubsystem` is responsible for constructing a `Trip` connecting a sequence of `Destinations`. The `PlanningSubsystem` is also responsible for responding to replan requests from `RoutingSubsystem`.

`RoutingSubsystem` The `RoutingSubsystem` is responsible for downloading a `Trip` from the `PlanningService` and executing it by giving `Directions` to the driver based on its `Location`.

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

Refining Subsystem Decomposition

- Refine the subsystems in an iterative approach.

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.