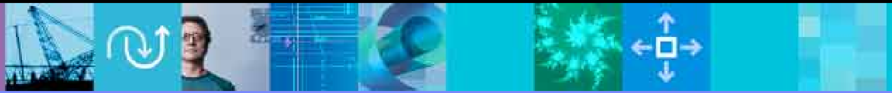
IBM Software Group

Mastering Object-Oriented Analysis and Design with UML

Module 4: Architectural Analysis

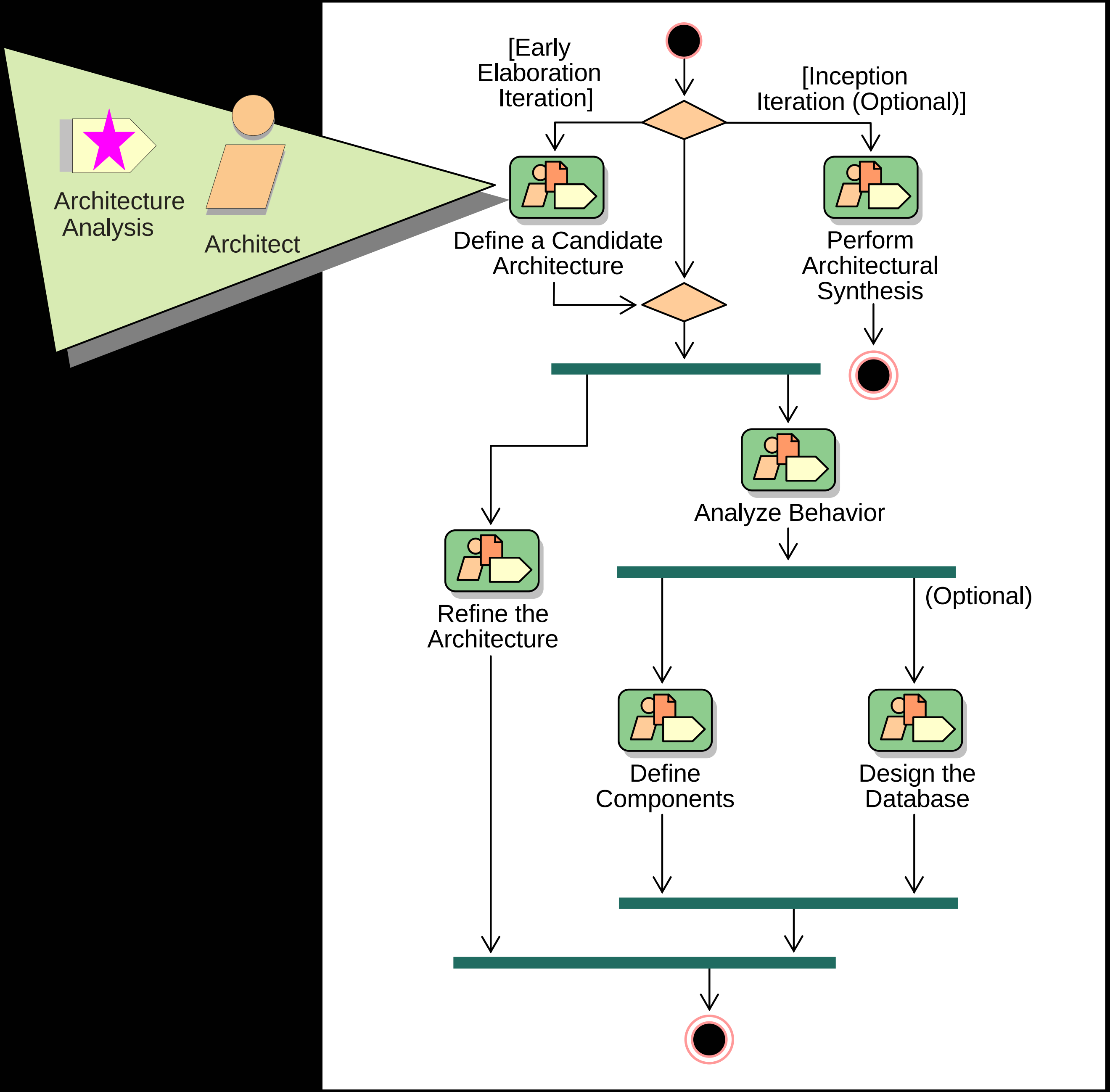
Rational. software



Objectives: Architectural Analysis

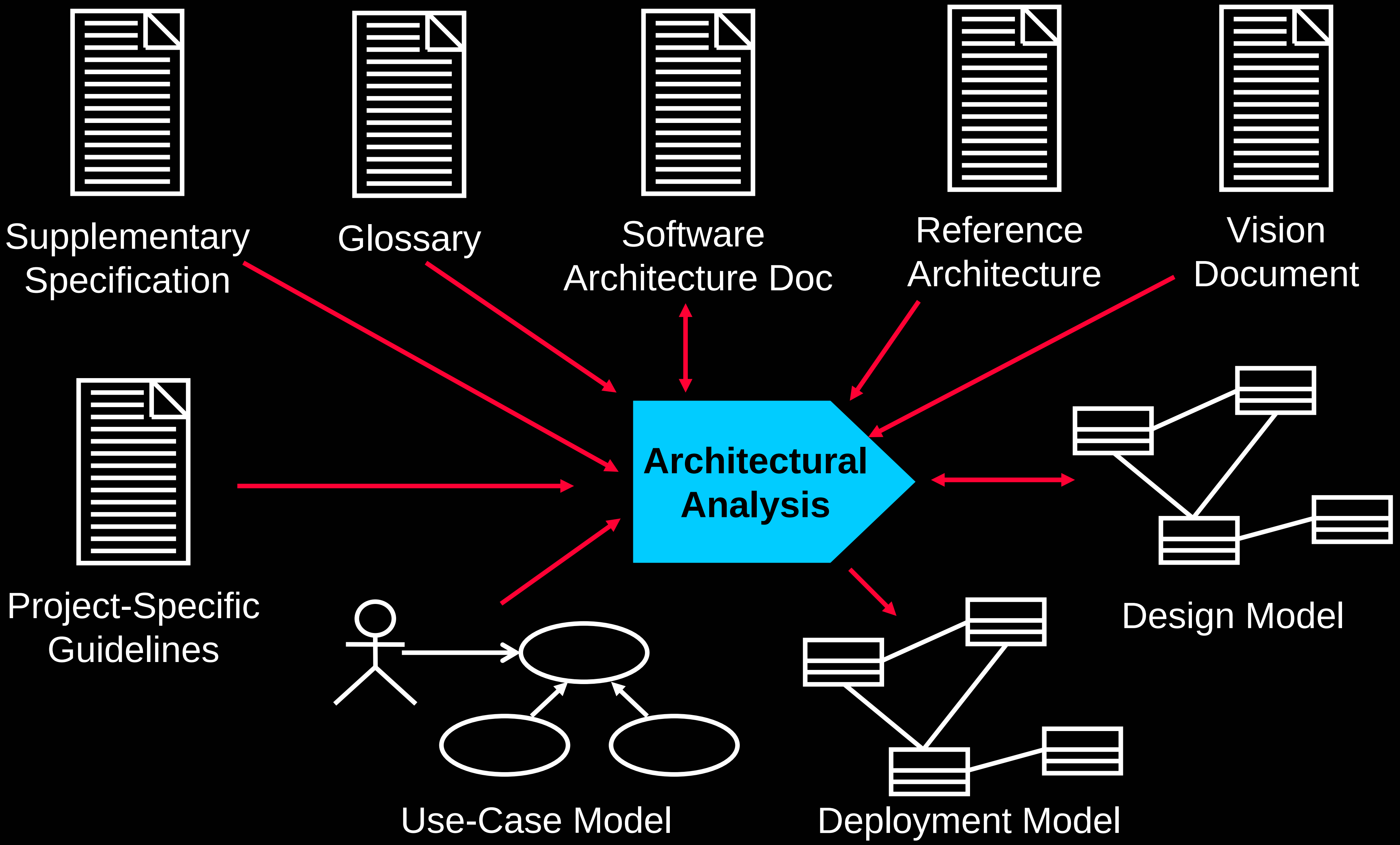
- Explain the purpose of Architectural Analysis and where it is performed in the lifecycle.
- Describe a representative architectural pattern and set of analysis mechanisms, and how they affect the architecture.
- Describe the rationale and considerations that support the architectural decisions.
- Show how to read and interpret the results of Architectural Analysis:
  - Architectural layers and their relationships
  - Key abstractions

Architectural Analysis in Context



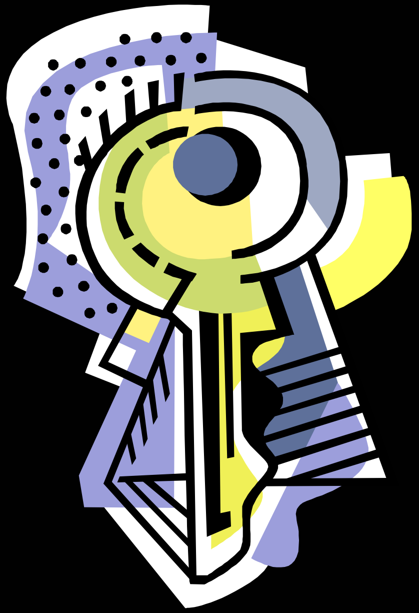
Architecture Analysis
Architect
[Early Elaboration Iteration]
[Inception Iteration (Optional)]
Define a Candidate Architecture
Perform Architectural Synthesis
Analyze Behavior
Refine the Architecture
(Optional)
Define Components
Design the Database

Architectural Analysis Overview



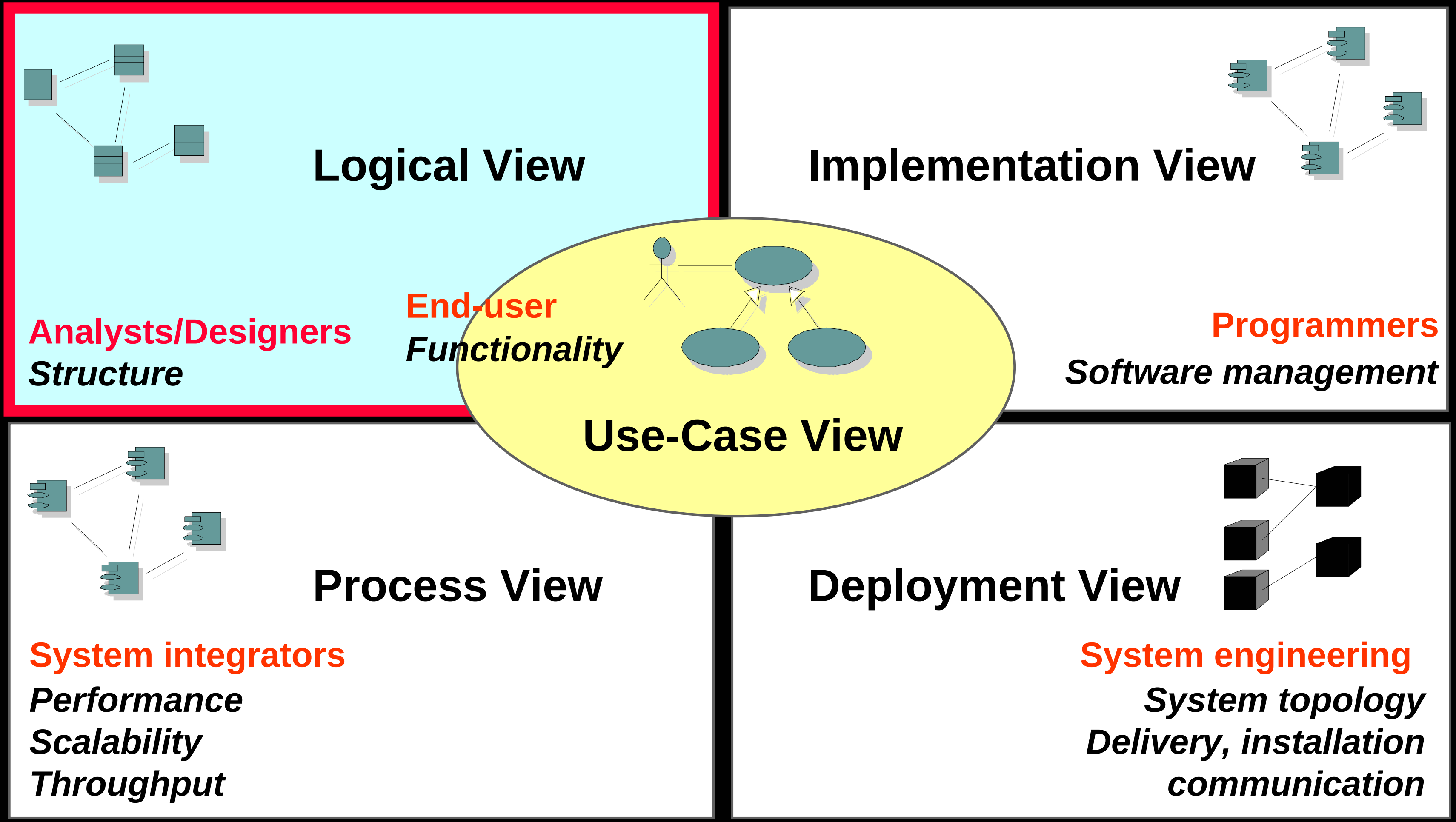
Architectural Analysis Steps

- ★ Key Concepts
- Define the High-Level Organization of Subsystems
- Identify Key Abstractions
- Create Use-Case Realizations
- Checkpoints



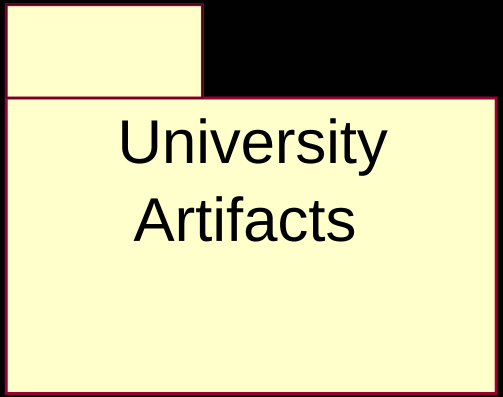
IBM

Review: What Is Architecture: The “4+1 View” Model



Review: What Is a Package?

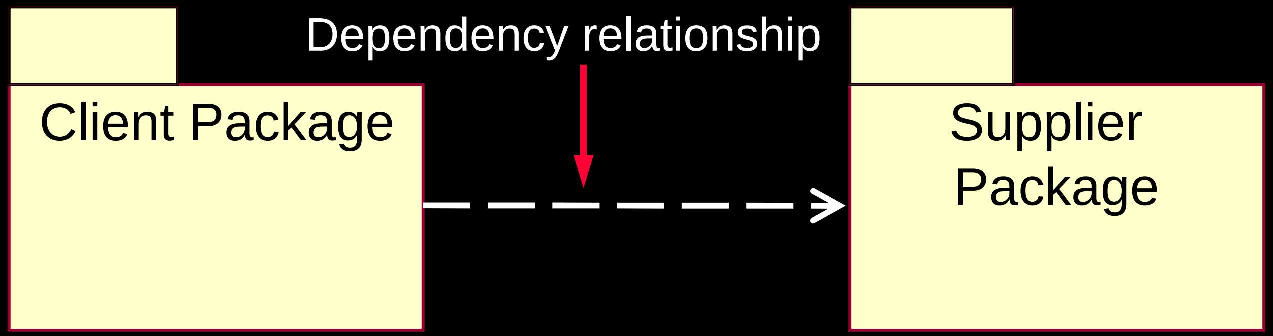
- A package is a general-purpose mechanism for organizing elements into groups.
- It is a model element that can contain other model elements.



- A package can be used
  - To organize the model under development.
  - As a unit of configuration management.

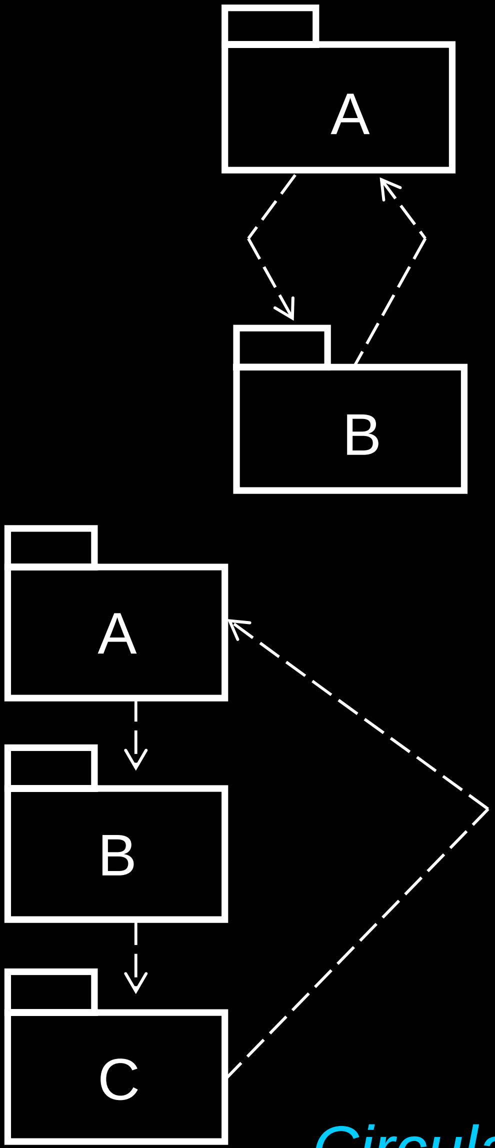
Package Relationships: Dependency

- Packages can be related to one another using a dependency relationship.

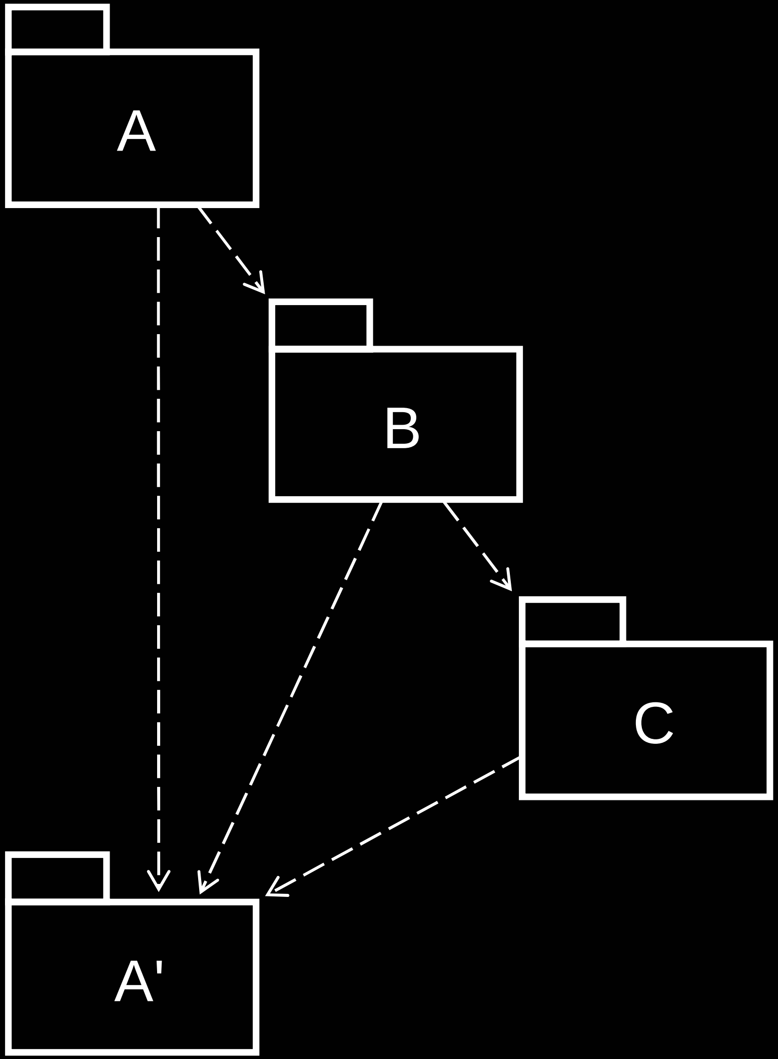


- Dependency Implications
  - Changes to the Supplier package may affect the Client package.
  - The Client package cannot be reused independently because it depends on the Supplier package.

Avoiding Circular Dependencies



Hierarchy should be acyclic



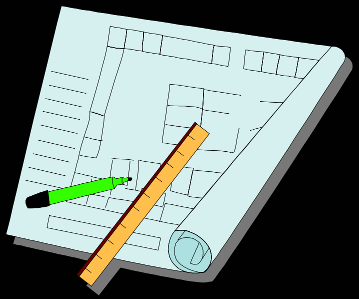
Circular dependencies make it impossible to reuse one package without the other.

Architectural Analysis Steps

- Key Concepts
- ★ Define the High-Level Organization of Subsystems
- Identify Key Abstractions
- Create Use-Case Realizations
- Checkpoints

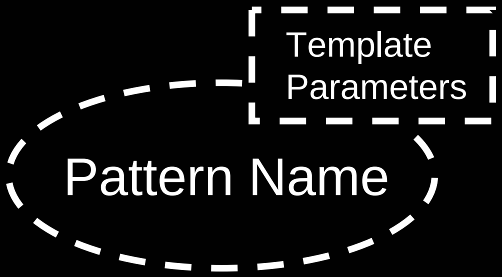
Patterns and Frameworks

- Pattern
  - Provides a common solution to a common problem in a context
- Analysis/Design pattern
  - Provides a solution to a narrowly-scoped technical problem
  - Provides a fragment of a solution, or a piece of the puzzle
- Framework
  - Defines the general approach to solving the problem
  - Provides a skeletal solution, whose details may be Analysis/Design patterns

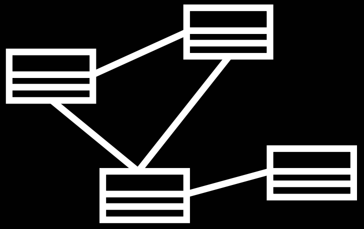


What Is a Design Pattern?

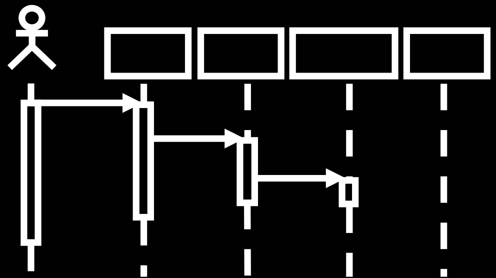
- A design pattern is a solution to a common design problem.
  - Describes a common design problem
  - Describes the solution to the problem
  - Discusses the results and trade-offs of applying the pattern
- Design patterns provide the capability to reuse successful designs.



Parameterized Collaboration



Structural Aspect

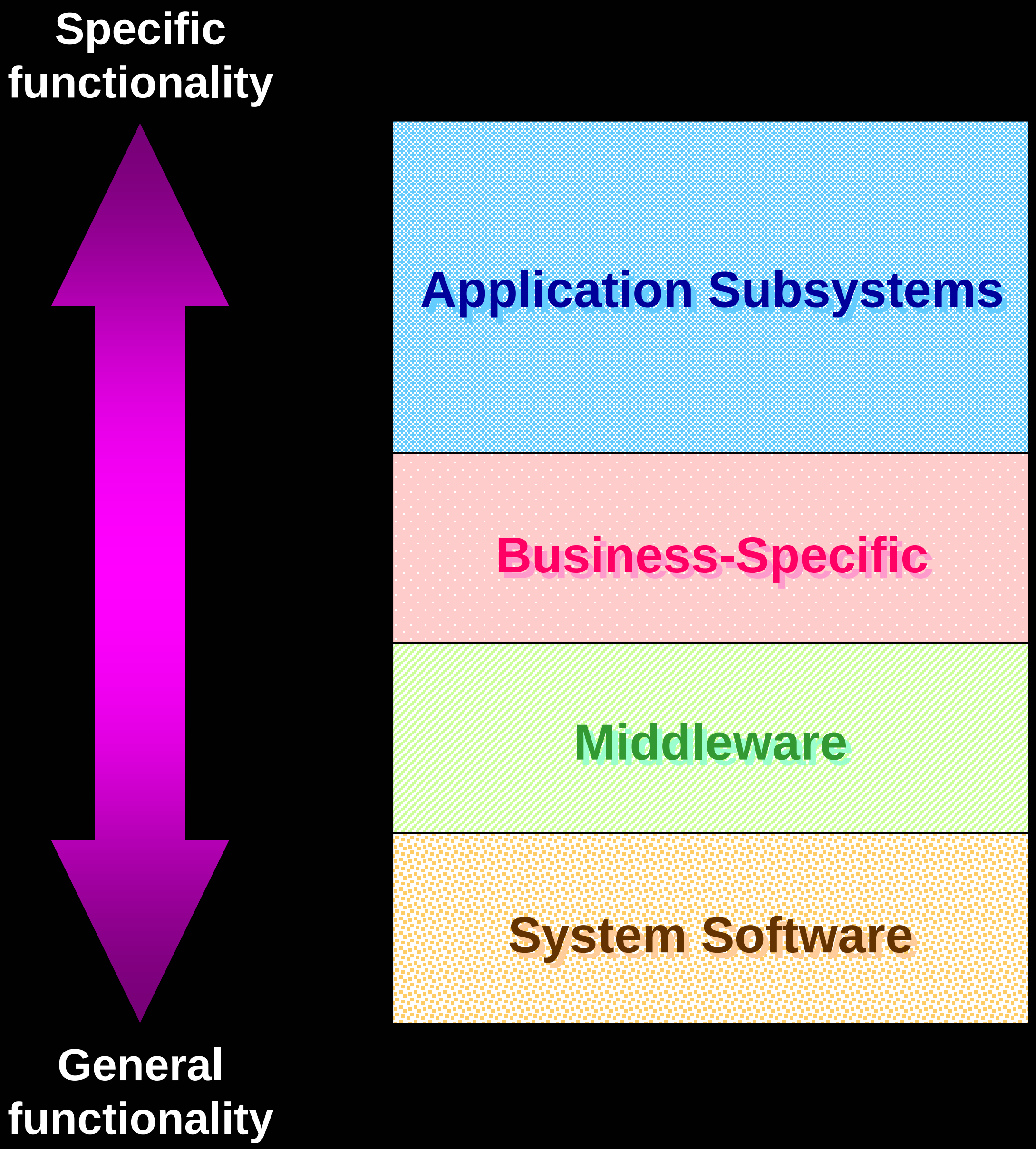


Behavioral Aspect

What Is an Architectural Pattern?

- An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them – *Buschman et al, "Pattern-Oriented Software Architecture — A System of Patterns"*
  - Layers
  - Model-view-controller (M-V-C)
  - Pipes and filters
  - Blackboard

Typical Layering Approach



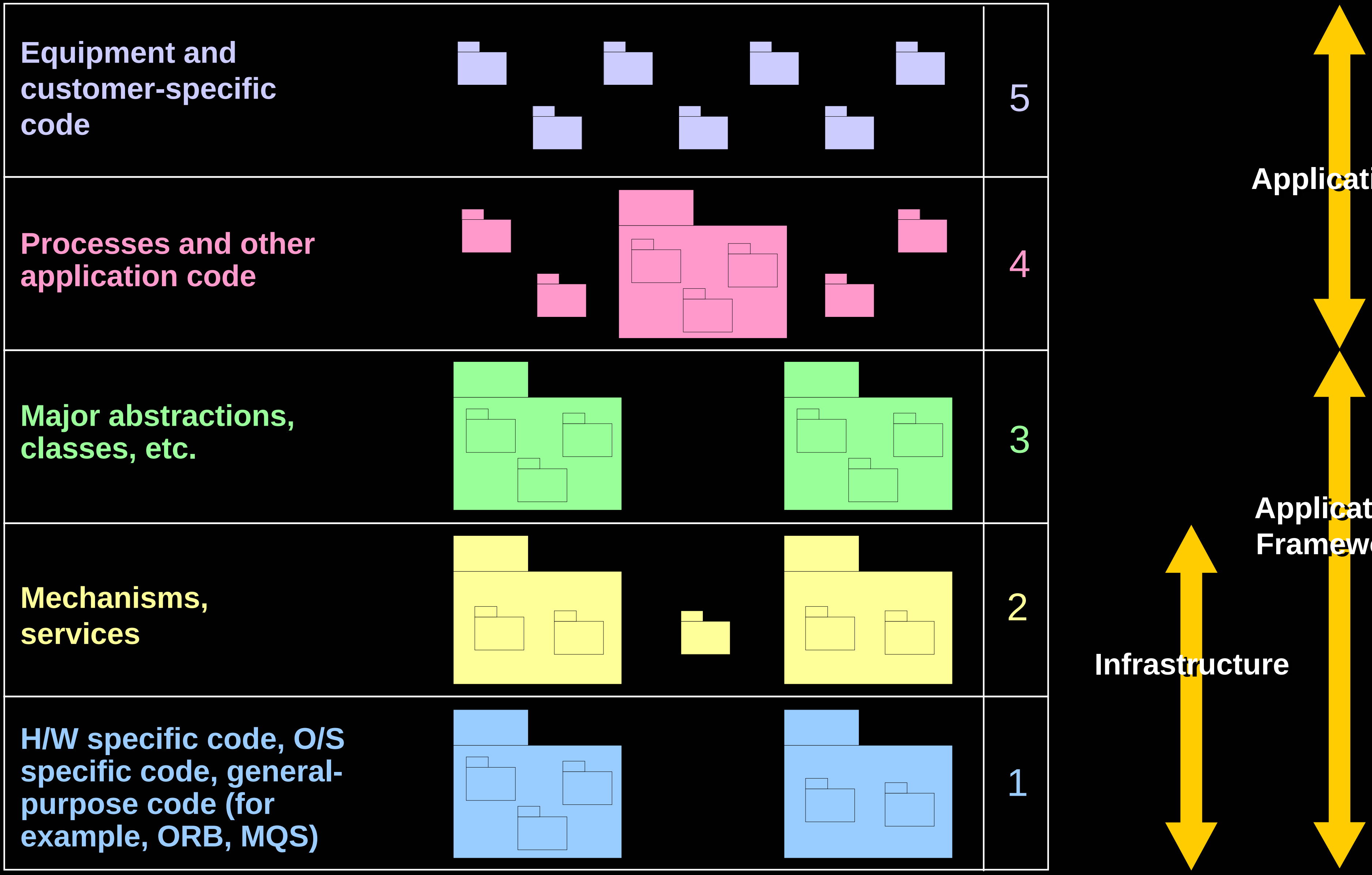
Distinct application subsystems that make up an application — contains the value adding software developed by the organization.

Business specific — contains a number of reusable subsystems specific to the type of business.

Middleware — offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.

System software — contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers, and so on.

Architectural Pattern: Layers

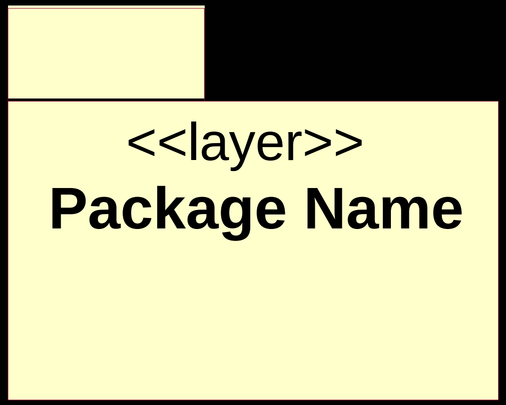

Equipment and customer-specific code
5
Processes and other application code
4
Major abstractions, classes, etc.
3
Mechanisms, services
2
H/W specific code, O/S specific code, general-purpose code (for example, ORB, MQS)
1
Application
Application Framework
Infrastructure

Layering Considerations

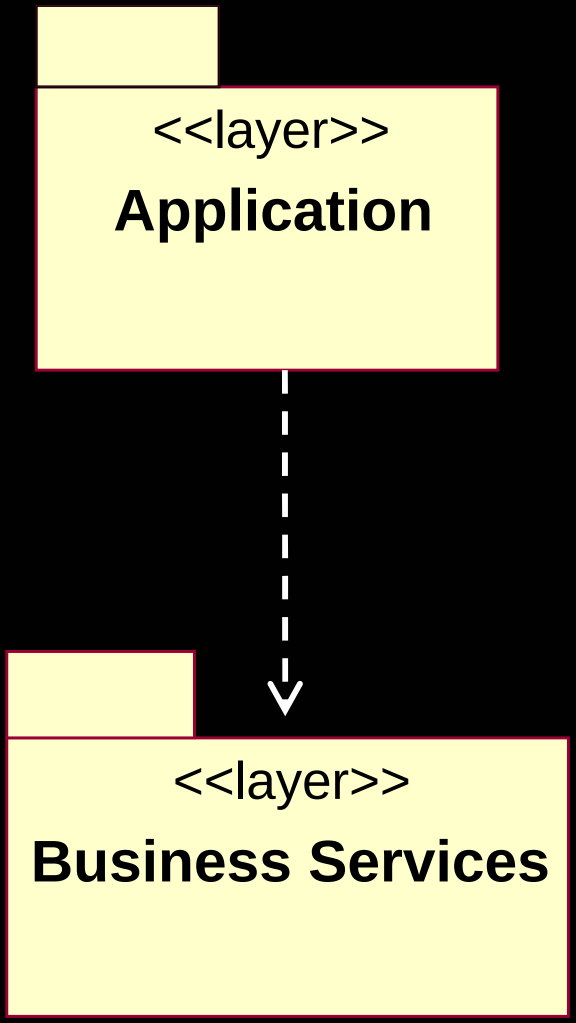
- Level of abstraction
  - Group elements at the same level of abstraction
- Separation of concerns
  - Group like things together
  - Separate disparate things
  - Application vs. domain model elements
- Resiliency
  - Loose coupling
  - Concentrate on encapsulating change
  - User interface, business rules, and retained data tend to have a high potential for change

Modeling Architectural Layers

- Architectural layers can be modeled using stereotyped packages.
- <<layer>> stereotype

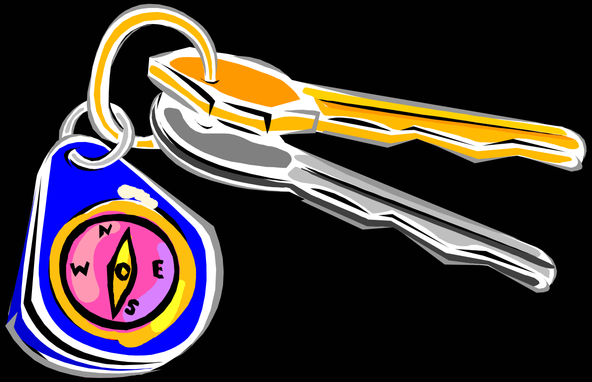


Example: High-Level Organization of the Model



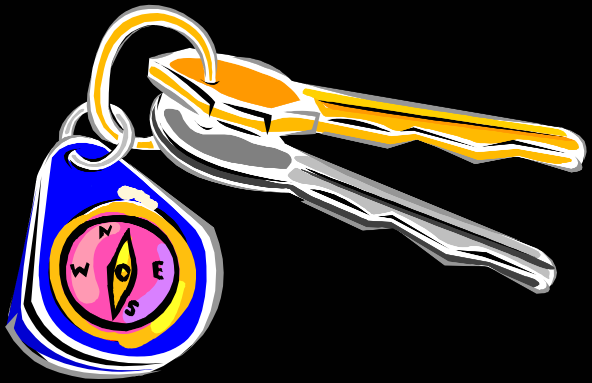
Architectural Analysis Steps

- Key Concepts
- Define the High-Level Organization of Subsystems
- Identify Key Abstractions
- ★ Create Use-Case Realizations
- Checkpoints



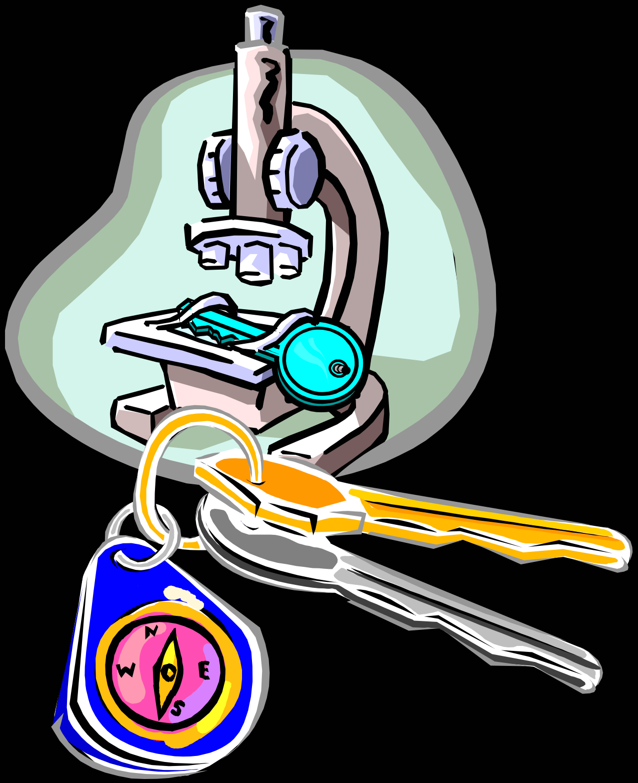
What Are Key Abstractions?

- A key abstraction is a concept, normally uncovered in Requirements, that the system must be able to handle
- Sources for key abstractions
  - Domain knowledge
  - Requirements
  - Glossary
  - Domain Model, or the Business Model (if one exists)

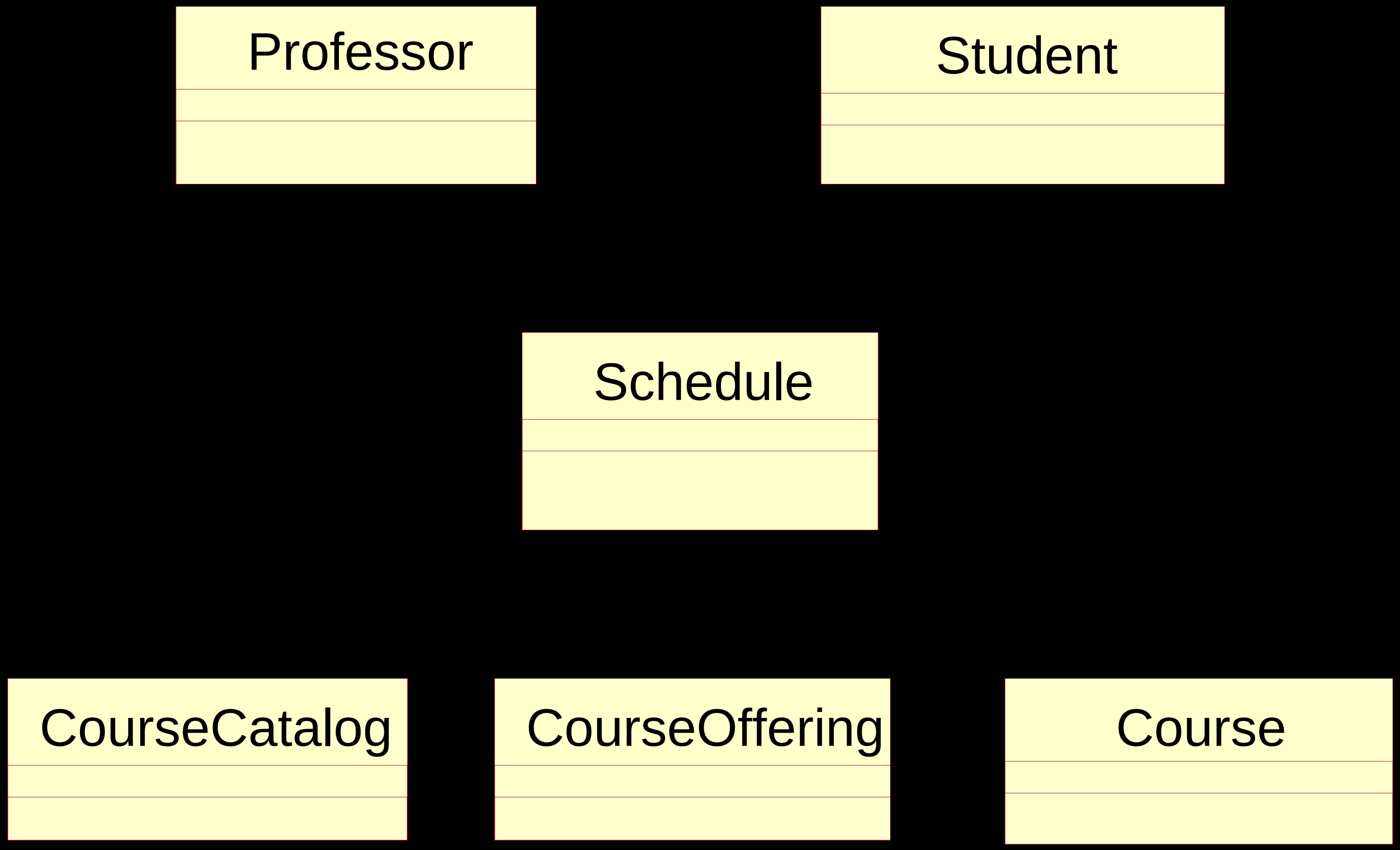


Defining Key Abstractions

- Define analysis class relationships
- Model analysis classes and relationships on class diagrams
  - Include brief description of analysis class
- Map analysis classes to necessary analysis mechanisms



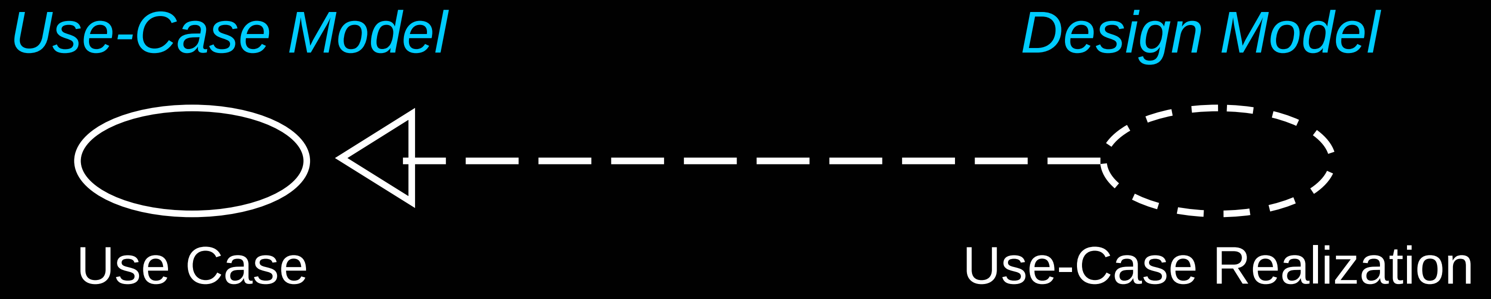
Example: Key Abstractions

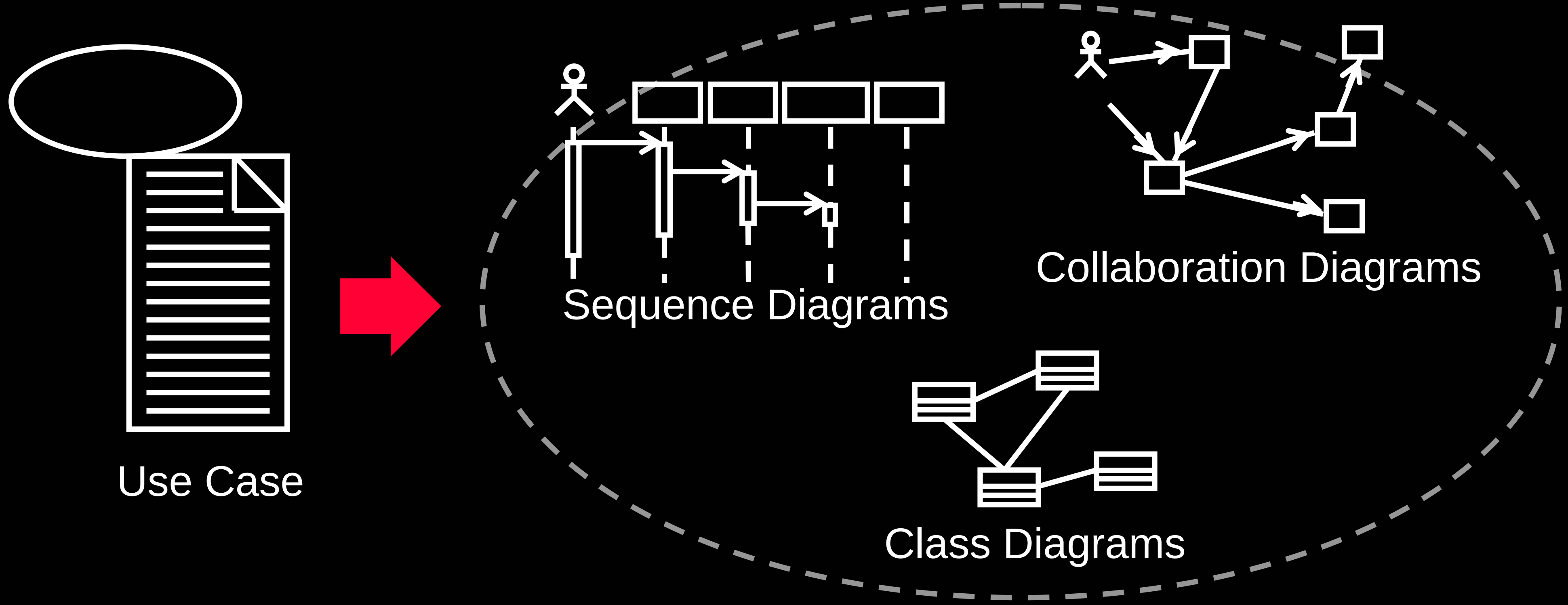


Architectural Analysis Steps

- Key Concepts
- Define the High-Level Organization of Subsystems
- Identify Key Abstractions
- Create Use-Case Realizations
- ★ Checkpoints

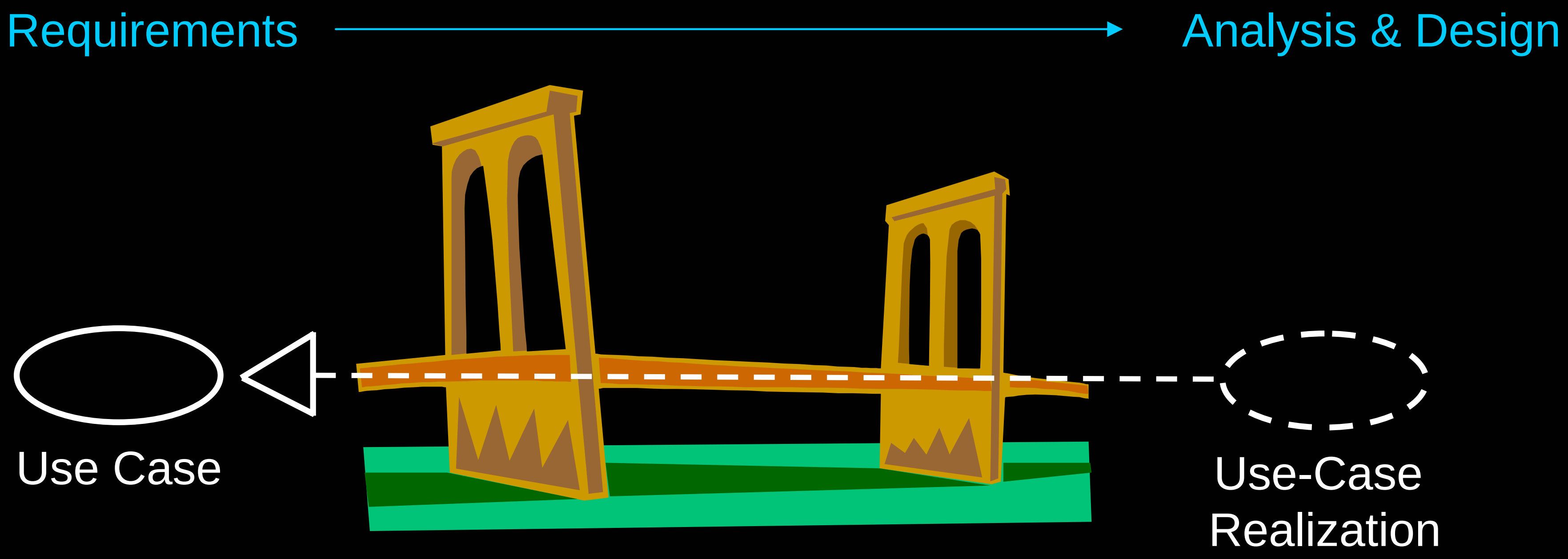
Review: What is a Use-Case Realization?





The Value of Use-Case Realizations

- Provides traceability from Analysis and Design back to Requirements
- The Architect creates the Use-Case Realization



Architectural Analysis Steps

- Key Concepts
- Define the High-Level Organization of Subsystems
- Identify Key Abstractions
- Create Use-Case Realizations
- Checkpoints

★

Checkpoints

- General
  - Is the package partitioning and layering done in a logically consistent way?
  - Have the necessary analysis mechanisms been identified?
- Packages
  - Have we provided a comprehensive picture of the services of the packages in upper-level layers?

(continued)

Checkpoints (cont.)

- Classes
  - Have the key entity classes and their relationships been identified and accurately modeled?
  - Does the name of each class clearly reflect the role it plays?
  - Are the key abstractions/classes and their relationships consistent with the Business Model, Domain Model, Requirements, Glossary, etc.?

Review: Architectural Analysis

- What is the purpose of Architectural Analysis?
- What is a package?
- What key abstractions are identified during Architectural Analysis? Why are they identified here?
- What is a layered architecture? Give examples of typical layers.

Exercise: Architectural Analysis

- Given the following:
  - Some results from the Requirements discipline:
    - Problem statement
    - Use-Case Model main diagram
    - Glossary
  - Some architectural decisions:
    - (textually) The upper-level architectural layers and their dependencies

(continued)

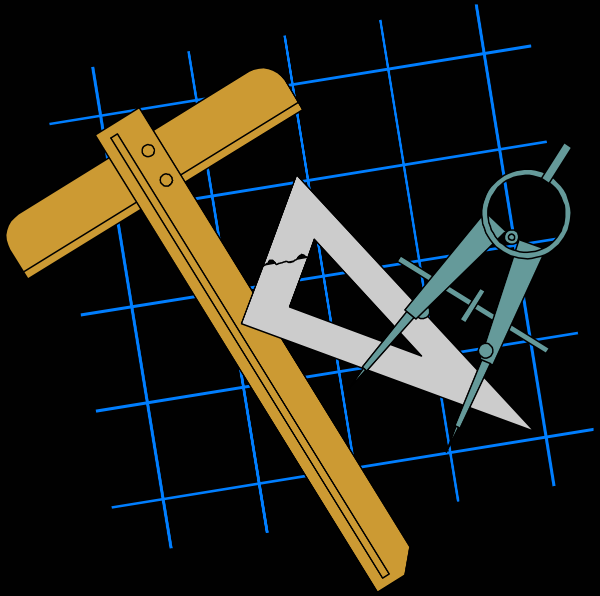
Exercise: Architectural Analysis (cont.)

- Identify the following:
  - The key abstractions

(continued)

Exercise: Architectural Analysis (cont.)

- Produce the following:
  - Class diagram containing the key abstractions
  - Class diagram containing the upper-level architectural layers and their dependencies



Exercise: Review

- Compare your key abstractions with the rest of the class
  - Have the key concepts been identified?
  - Does the name of each class reflect the role it plays?
- Compare your class diagram showing the upper-level layers
  - Do the package relationships support the Payroll System architecture?