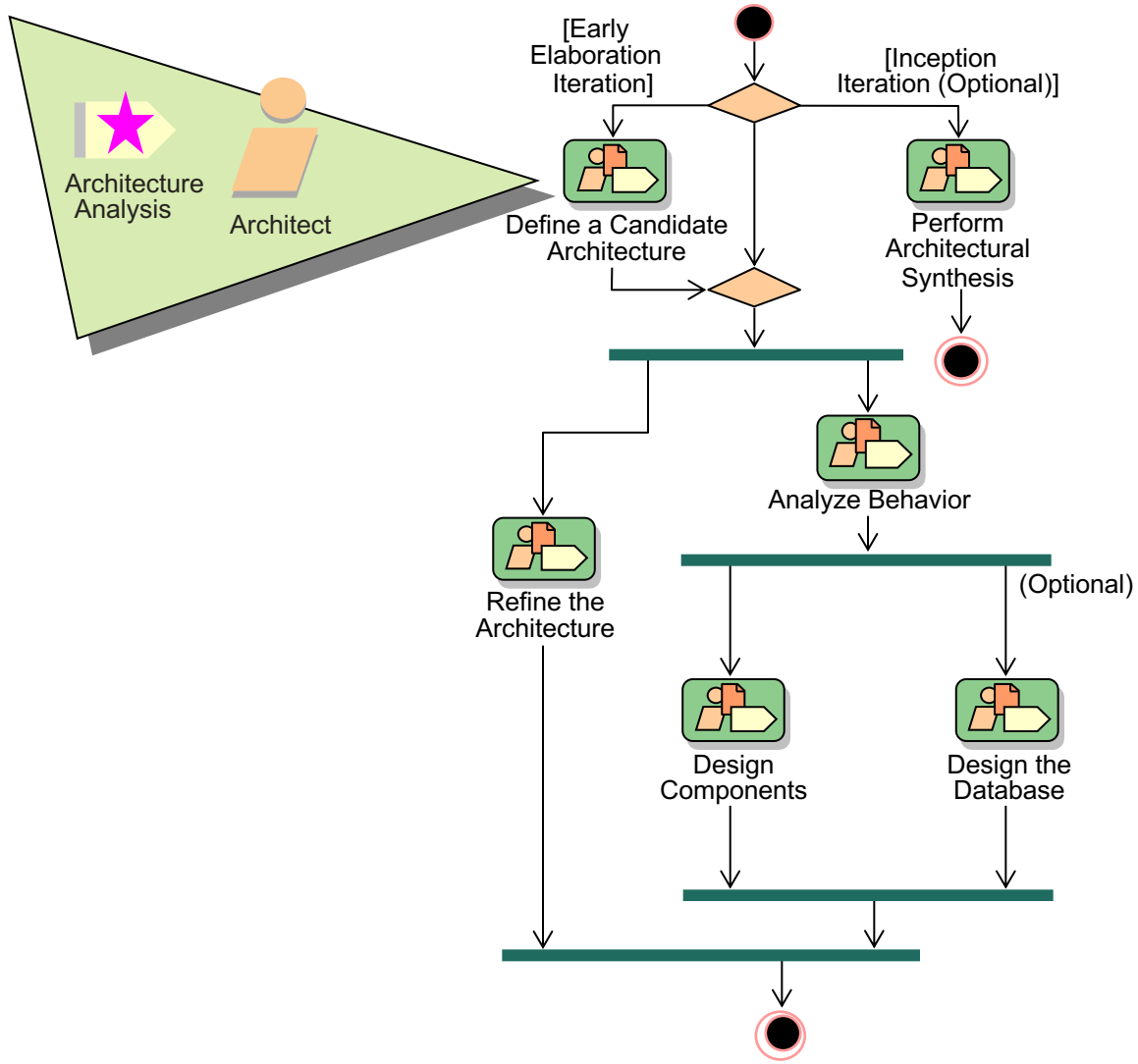
Software analysis and design

Module 9: Architectural Analysis

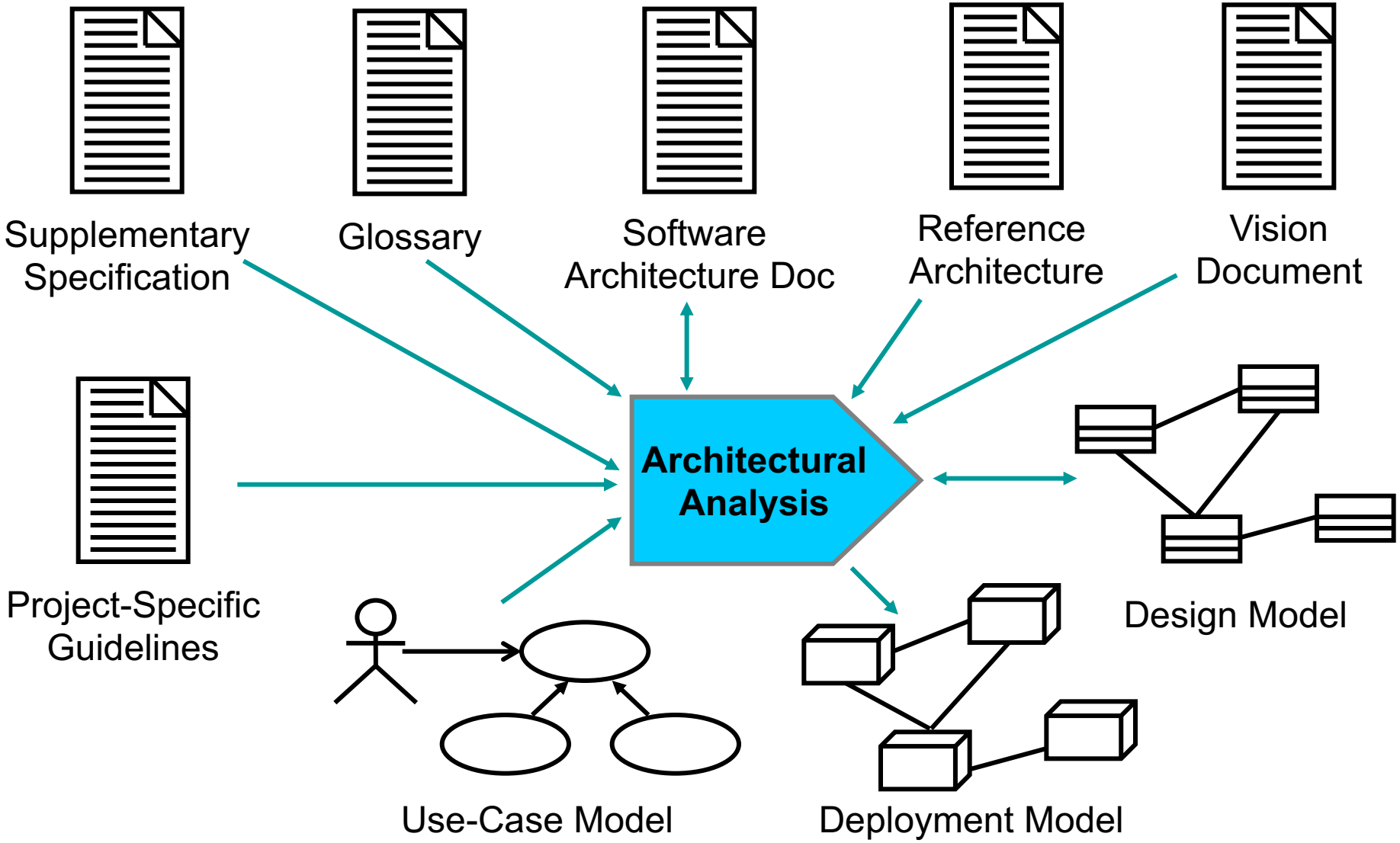
Objectives: Architectural Analysis

- 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
  - Analysis mechanisms

Architectural Analysis in Context

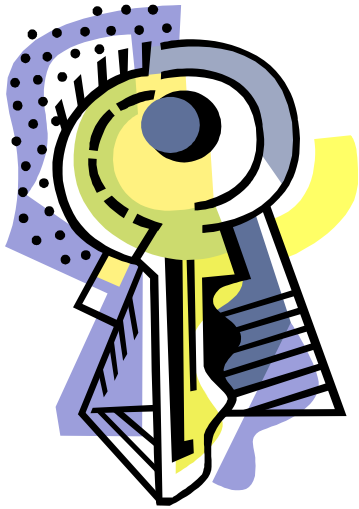


Architectural Analysis Overview

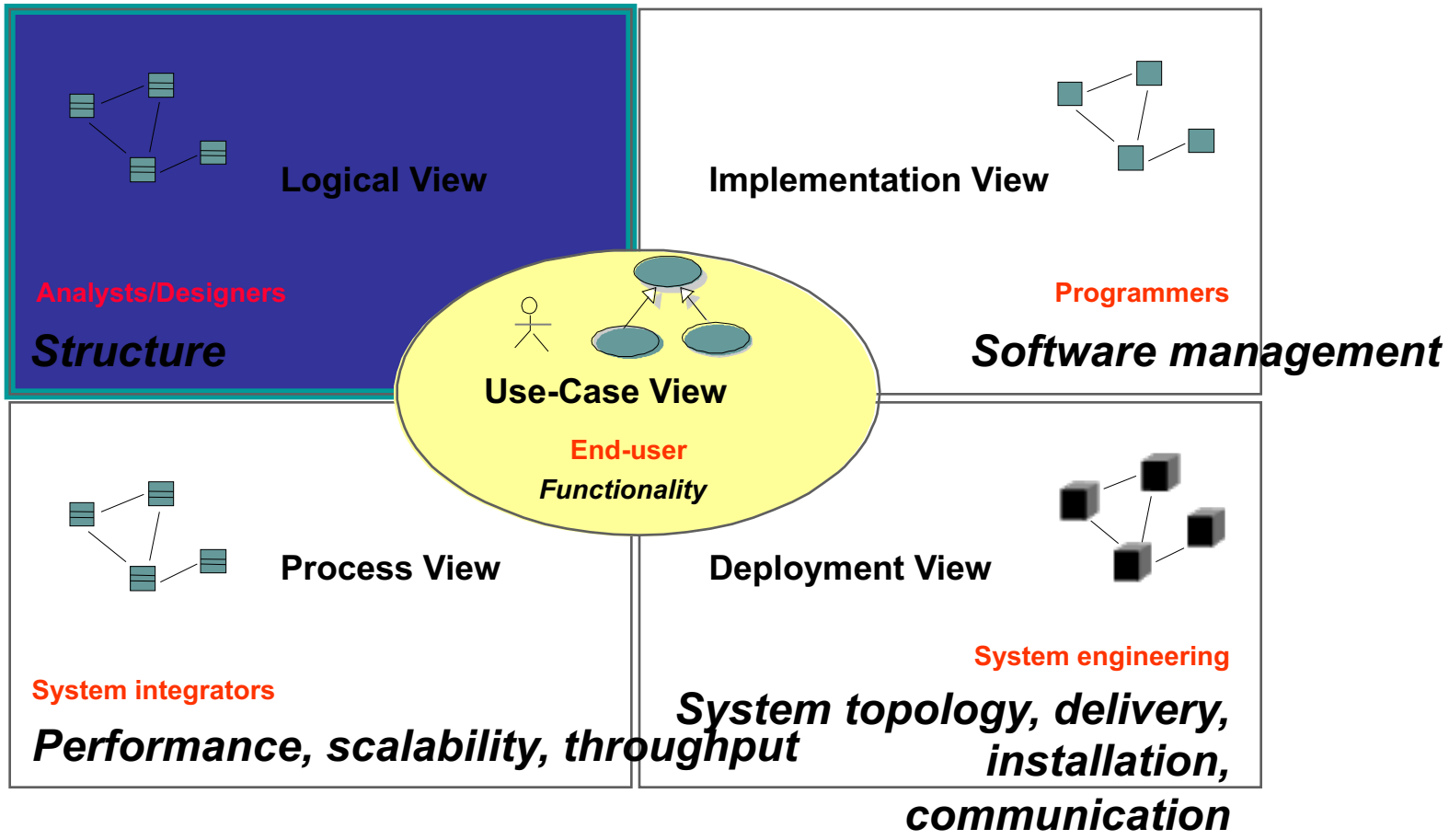


Architectural Analysis Steps

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

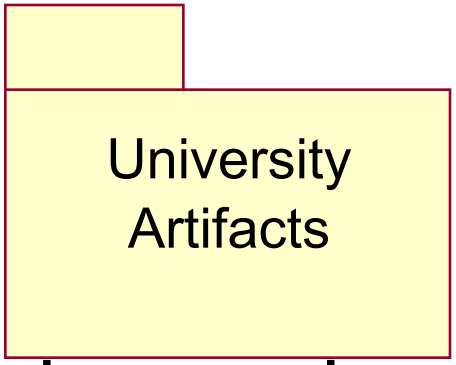


Review: What Is Architecture: The “4+1 View” Model



Review: What Is a Package?

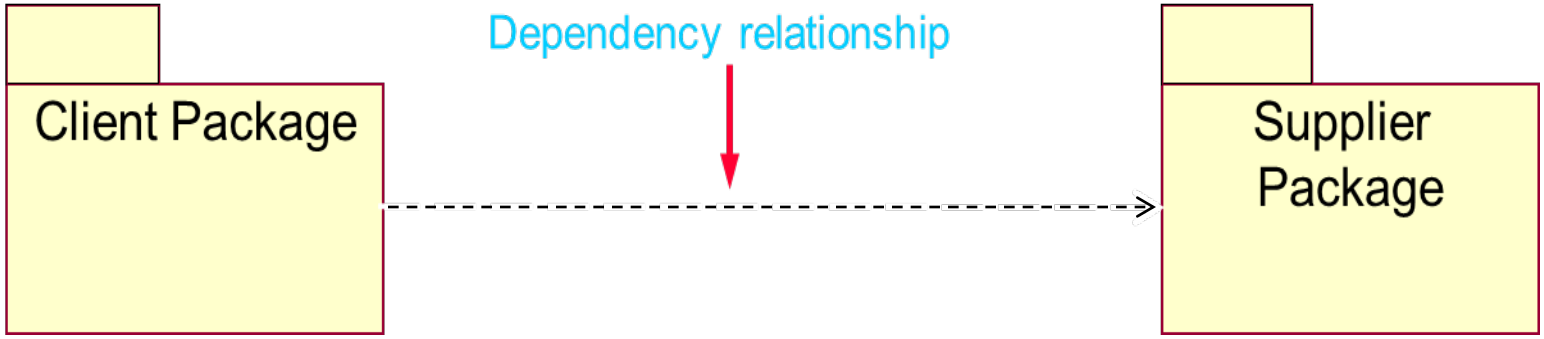
- A package is a general-purpose mechanism for organizing elements into groups.
- It is a model element that can contain other model elements.



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

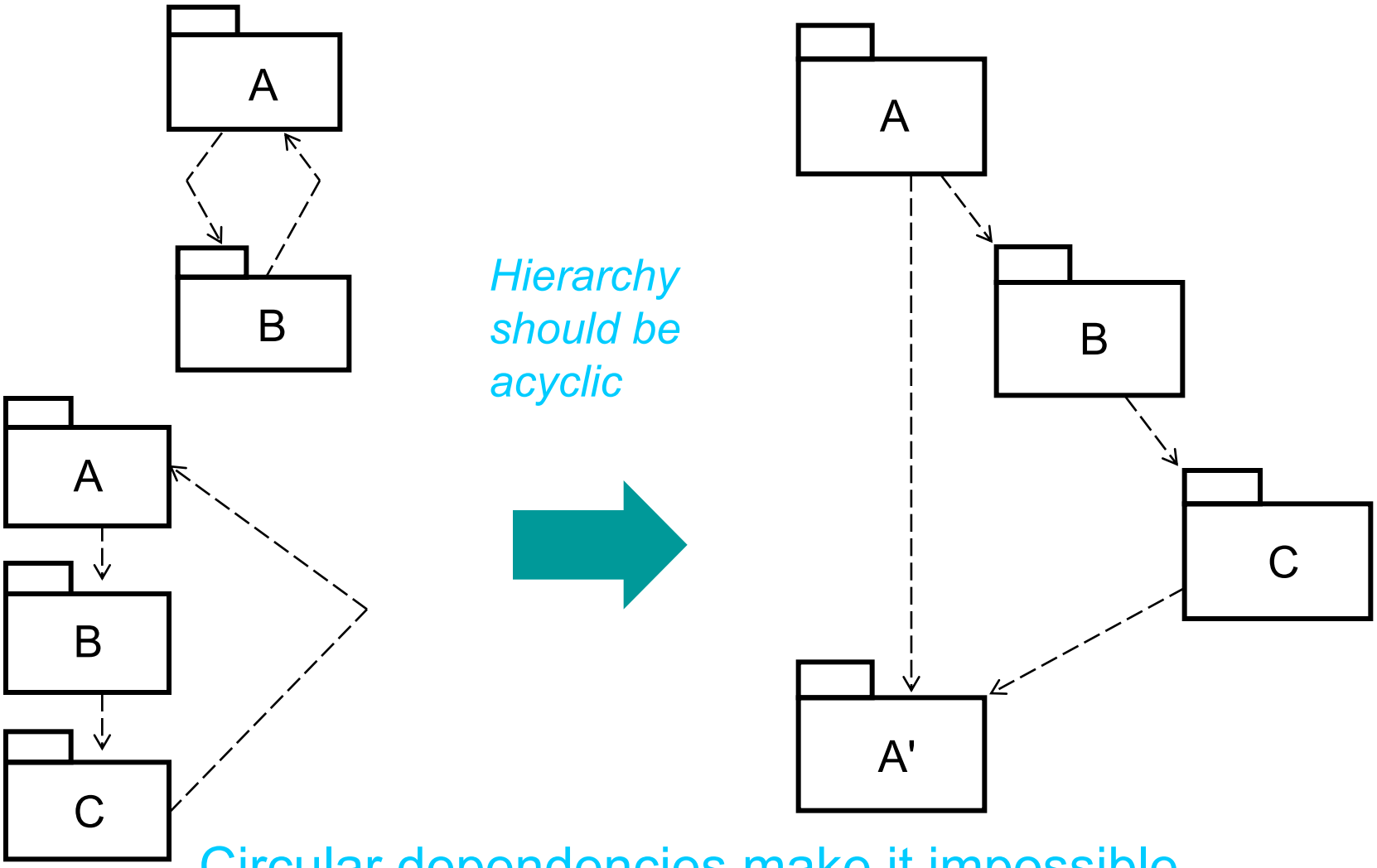
Package Relationships: Dependency

- 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



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

Architectural Analysis Steps

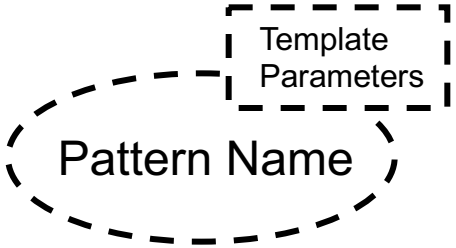
- Key Concepts
- Define the High-Level Organization of the model
- Identify Analysis mechanisms
- Identify Key Abstractions
- Create Use-Case Realizatio
- Checkpoints

Patterns and Frameworks

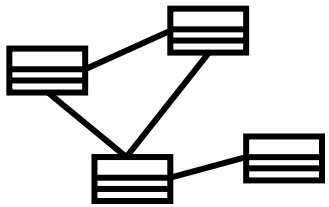
- Pattern
  - Provides a common solution to a common problem in a context
- Analysis/Design pattern
  - Provides a solution to a narrowly-scoped technical problem
  - Provides a fragment of a solution, or a piece of the puzzle
- Framework
  - Defines the general approach to solving the problem
  - Provides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

What Is a Design Pattern?

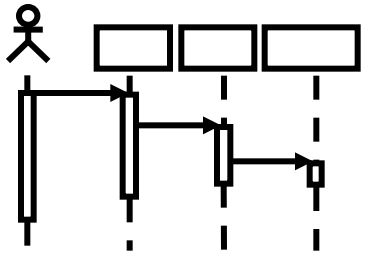
- A design pattern is a solution to a common design problem.
  - Describes a common design problem
  - Describes the solution to the problem
  - Discusses the results and trade-offs of applying the pattern
- Design patterns provide the capability to reuse successful designs.



Parameterized Collaboration



Structural Aspect

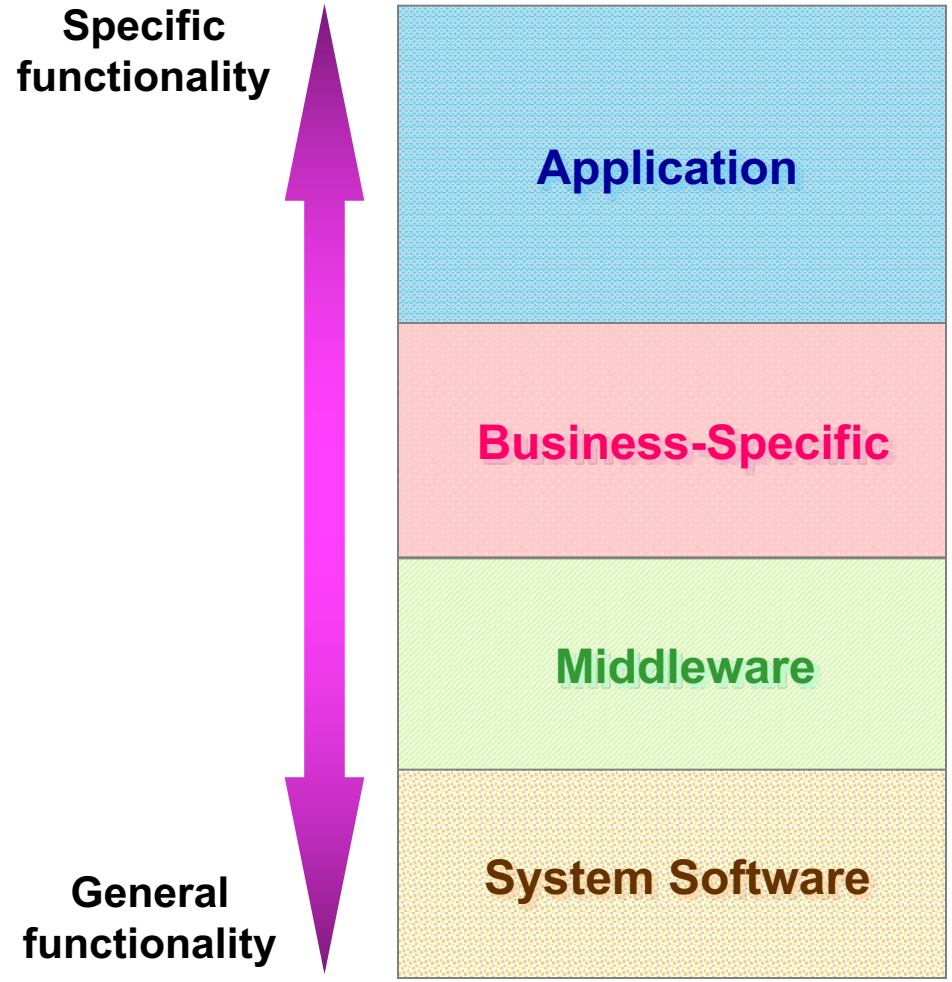


Behavioral Aspect

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.

Example: Layers

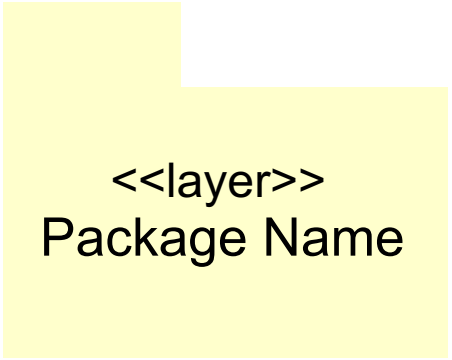
Layer 7	Provides miscellaneous protocols for common activities
Layer 6	Structure information and attaches semantics
Layer 5	Provides dialog control and synchronization facilities
Layer 4	Breaks messages into packets and guarantees delivery
Layer 3	Selects a route from send to receiver
Layer 2	Detects and corrects errors in bit sequences
Layer 1	Transmits bits: velocity, bit-code, connection, etc.

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

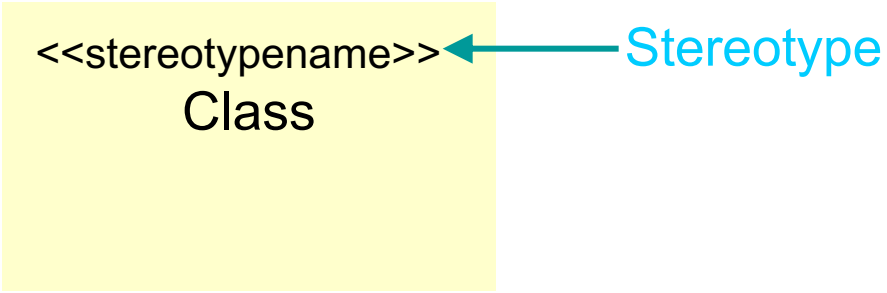
Modeling Architectural Layers

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

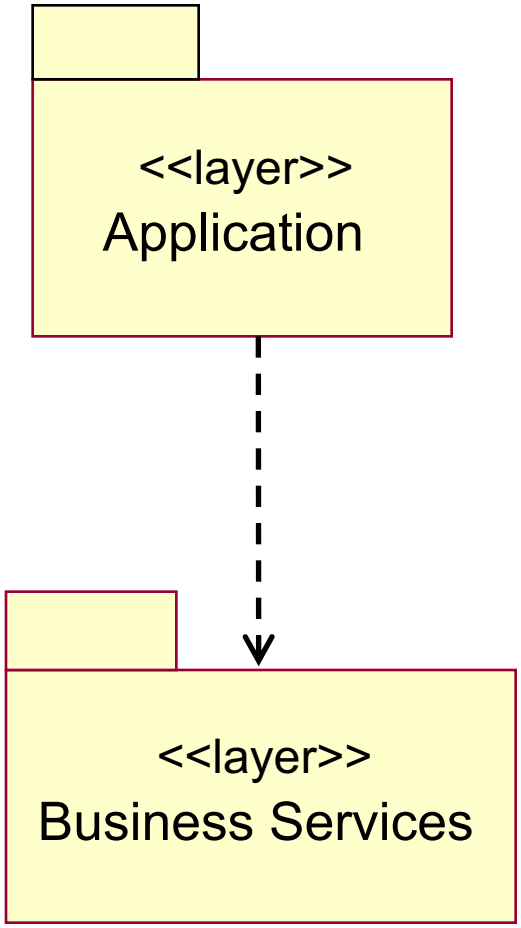


What Are Stereotypes?

- Stereotypes define a new model element in terms of another model element.
- Sometimes you need to introduce new things that speak the language of your domain and look like primitive building blocks.



Example: High-Level Organization of the Model



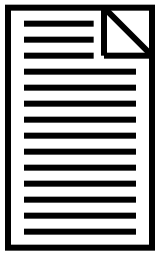
Architectural Analysis Steps

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

What Are Architectural Mechanisms?

Required Functionality

Implementation Environment



Supplementary Specification

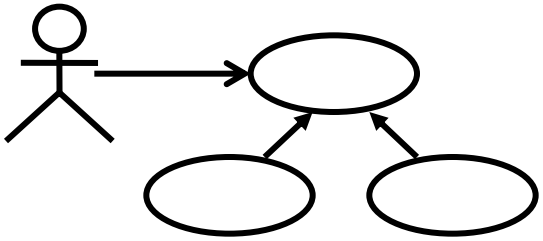
"realized by client classes using"

"constrained by"

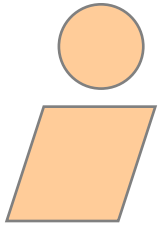
Mechanisms

COTS Products
Databases
IPC Technology, etc.

"responsible for"



Use-Case Model



Architect

Architectural Mechanisms: Three Categories

- Architectural Mechanism Categories
  - Analysis mechanisms (conceptual)
  - Design mechanisms (concrete)
  - Implementation mechanisms (actual)

Why Use Analysis Mechanisms?

Analysis mechanisms are used during analysis to reduce the complexity of analysis and to improve its consistency by providing designers with a shorthand representation for complex behavior.

Oh no! I found a group of classes that has persistent data. How am I supposed to design these things if I don't even know what database we are going to be using?

That is why we have a persistence analysis mechanism. We don't know enough yet, so we can bookmark it and come back to it later.

Sample Analysis Mechanisms

- Persistency
- Communication (IPC and RPC)
- Message routing
- Distribution
- Transaction management
- Process control and synchronization (resource contention)
- Information exchange, format conversion
- Security
- Error detection / handling / reporting
- Redundancy
- Legacy Interface

Examples of Analysis Mechanism Characteristics

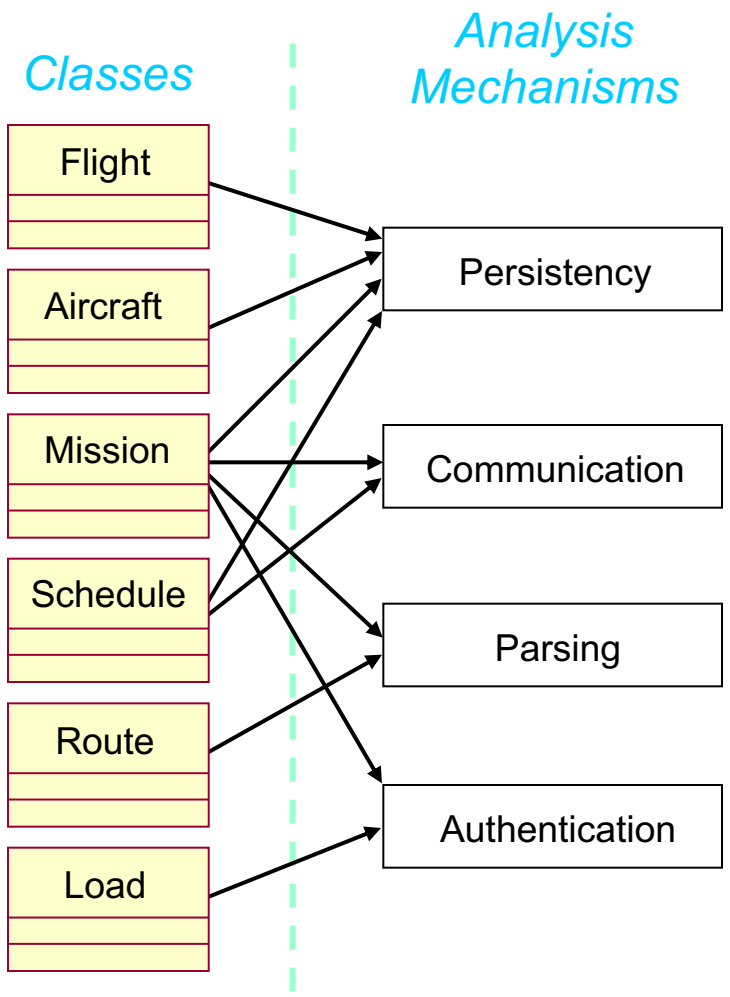
- Persistency mechanism
  - Granularity
  - Volume
  - Duration
  - Access mechanism
  - Access frequency (creation/deletion, update, read)
  - Reliability
- Inter-process Communication mechanism
  - Latency
  - Synchronicity
  - Message size
  - Protocol

Example: Analysis Mechanism Characteristics (continued)

- Legacy interface mechanism
  - Latency
  - Duration
  - Access mechanism
  - Access frequency
- Security mechanism
  - Data granularity
  - User granularity
  - Security rules
  - Privilege types
- Others

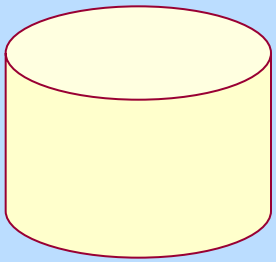
Describing Analysis Mechanisms

- Collect all analysis mechanisms in a list
- Draw a map of classes to analysis mechanisms
- Identify characteristics of analysis mechanisms
- Model using collaborations

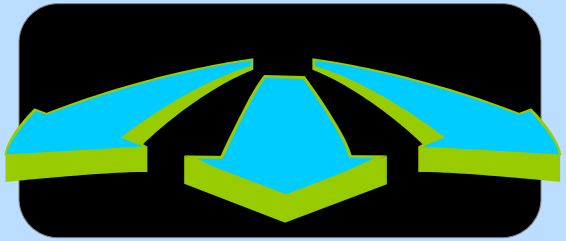


Example: Course Registration Analysis Mechanisms

Persistence

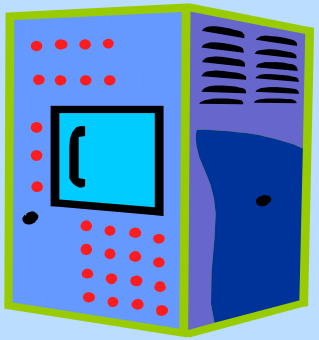


Distribution



Security

Legacy Interface



Architectural Analysis Steps

- Key Concepts
- Define the High-Level Organization of the model
- Identify Analysis mechanisms
- 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 classes
- Model analysis classes and relationships on class diagrams
  - Include a brief description of an analysis class
- Map analysis classes to necessary analysis mechanisms

Example: Key Abstractions

Professor

Student

Schedule

CourseCatalog

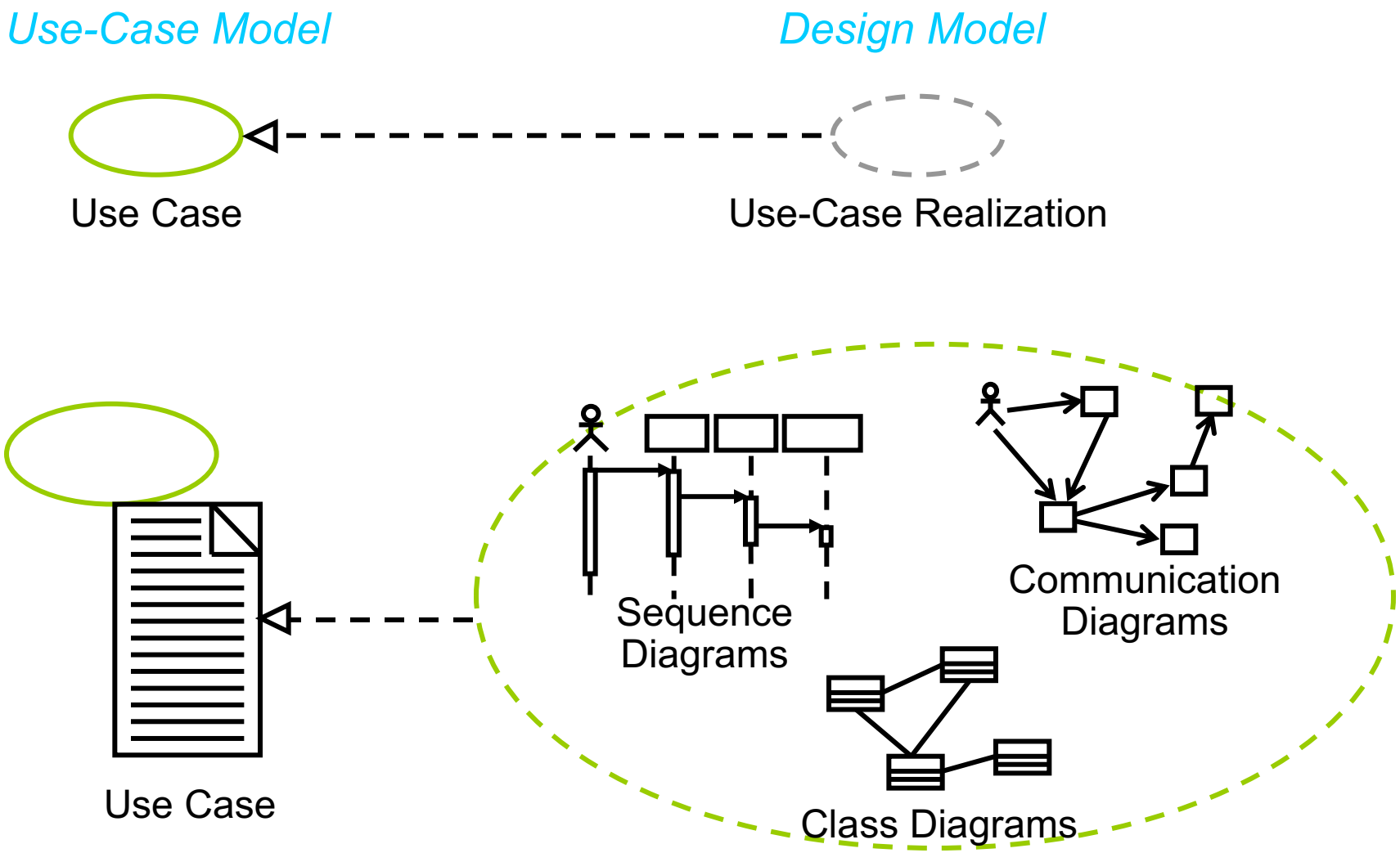
CourseOffering

Course

Architectural Analysis Steps

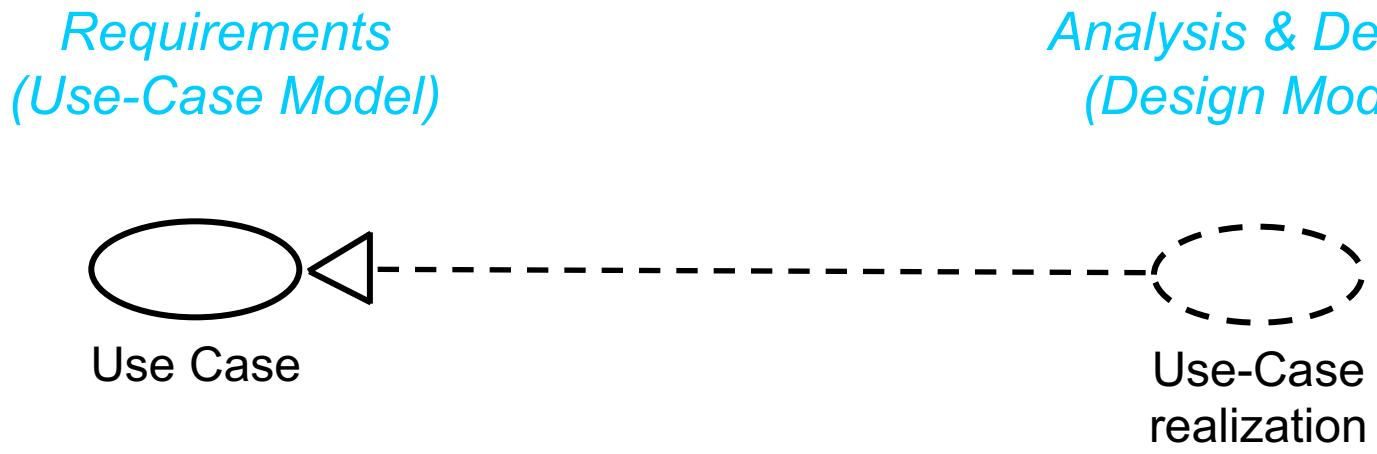
- Key Concepts
- Define the High-Level Organization of the model
- Identify Analysis mechanisms
- Identify Key Abstractions
- Create Use-Case Realizations
- Checkpoints

Review: What is a Use-Case Realization?



The Value of Use-Case Realizations

- Provides traceability from Analysis and Design back to Requirements



Architectural Analysis Steps

- Key Concepts
- Define the High-Level Organization of the model
- Identify Analysis mechanisms
- 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?

Checkpoints (continued)

- 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.?