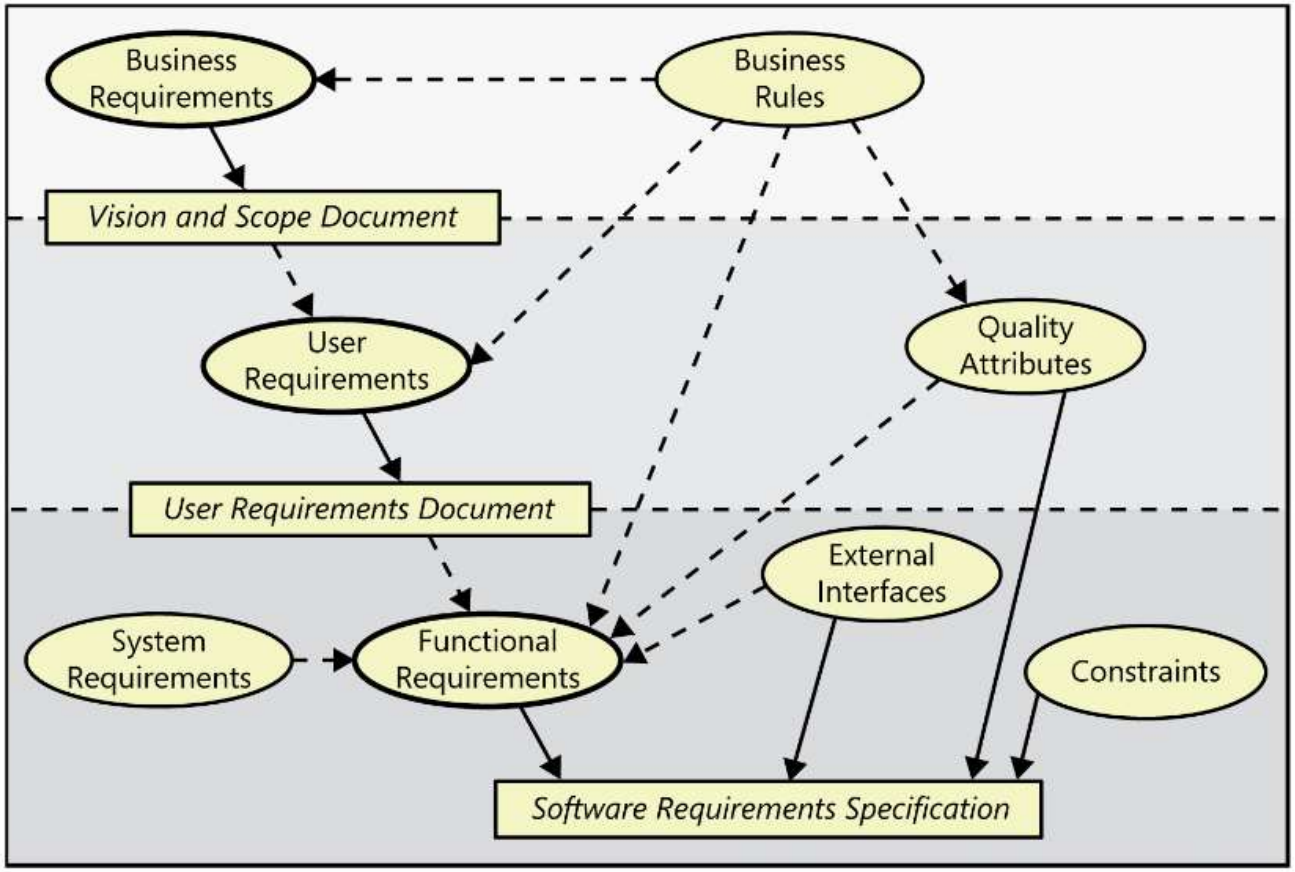
Requirements Elicitation, Analysis, and Specification

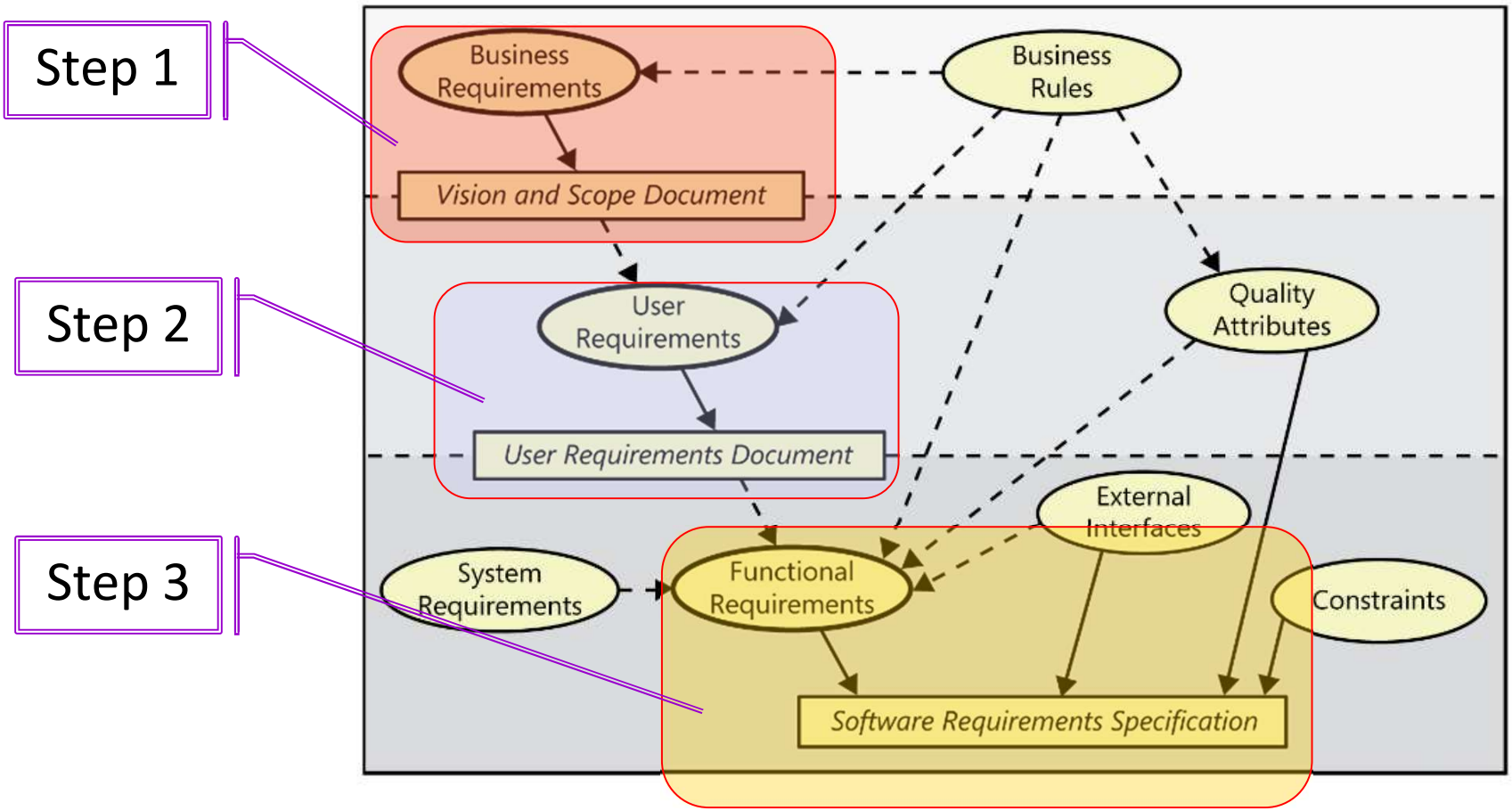
Establishing the business requirements

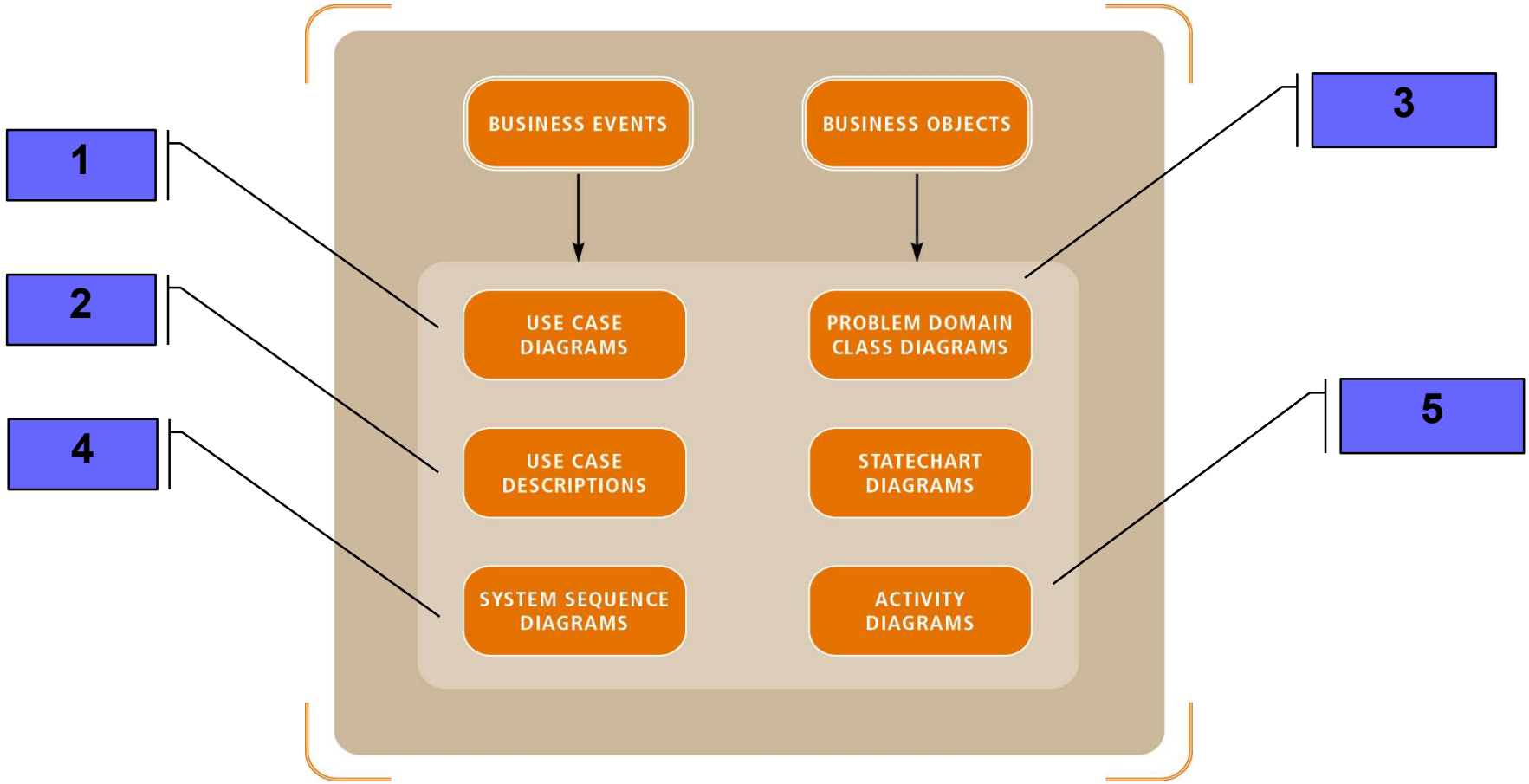
Types of Requirements



Relationships among several types of requirements information. Solid arrows mean “are stored in”; dotted arrows mean “are the origin of” or “influence.”

Types of Requirements





UML Diagrams to capture Requirements & Analysis

Defining Business Requirements

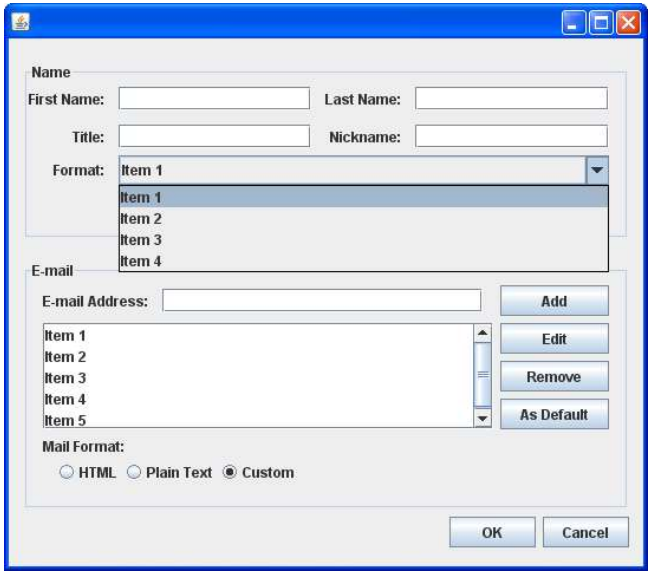
- The business requirements is a set that describes a need that leads to one or more projects to deliver a solution and the desired ultimate business outcomes.

- Business opportunities, business objectives, success metrics, and a vision statement make up the business requirements.

Defining Business Requirements

- "A customer must be provided with a form/GUI to create an order".
  - Is this a business requirements?

- "The Cancel button must be placed next to the OK button"
  - Is this a business requirements?

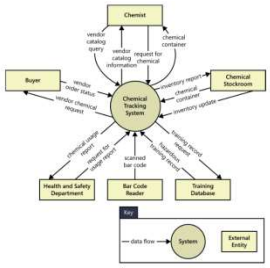


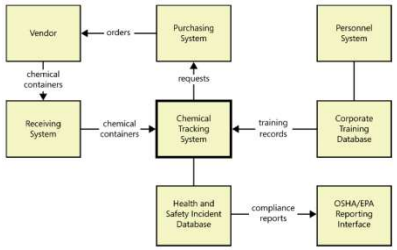
Scope representation techniques

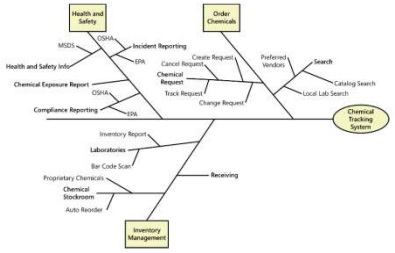
- Different models can be used to represent project scope in various ways.
- You don't need to create all of these models; consider which ones provide the most useful insight for each project.
- The models can be included in the vision and scope document or stored elsewhere and referenced as needed.

Scope representation techniques

The purpose of tools such as the context diagram, ecosystem map, feature tree, and event list is to foster clear and accurate communication among the project stakeholders.







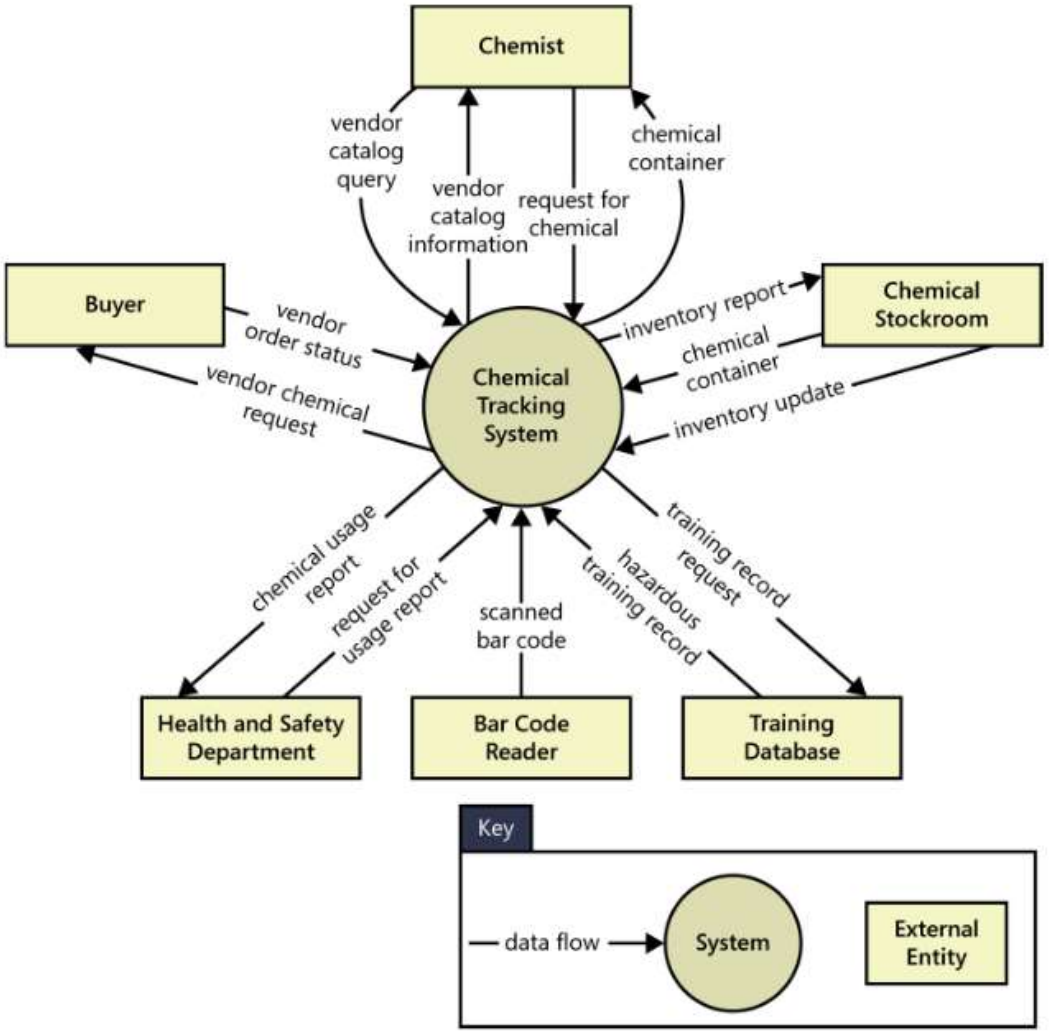
Scope representation techniques

- Context diagram
  - The scope description establishes the boundary and connections between the system you're developing and everything else in the universe.
  - The context diagram visually illustrates this boundary.
  - It identifies external entities (also called terminators) outside the system that interface to it in some way, as well as data, control, and material flows between the terminators and the system.

The context diagram is the top level in a data flow diagram developed according to the principles of structured analysis

Scope representation techniques

- Context diagram
  - The following Figure illustrates a portion of the context diagram for the Chemical Tracking System.



Scope representation techniques

- Context diagram
  - The entire system is depicted as a single circle; the context diagram deliberately provides no visibility into the system's internal objects, processes, or data.
  - The "system" inside the circle could encompass any combination of software, hardware, and human components.
  - The external entities in the rectangles can represent user classes (Chemist, Buyer), organizations (Health and Safety Department), other systems (Training Database), or hardware devices (Bar Code Reader).
  - The arrows on the diagram represent the flow of data (such as a request for a chemical) or physical items (such as a chemical container) between the system and its external entities.

Scope representation techniques

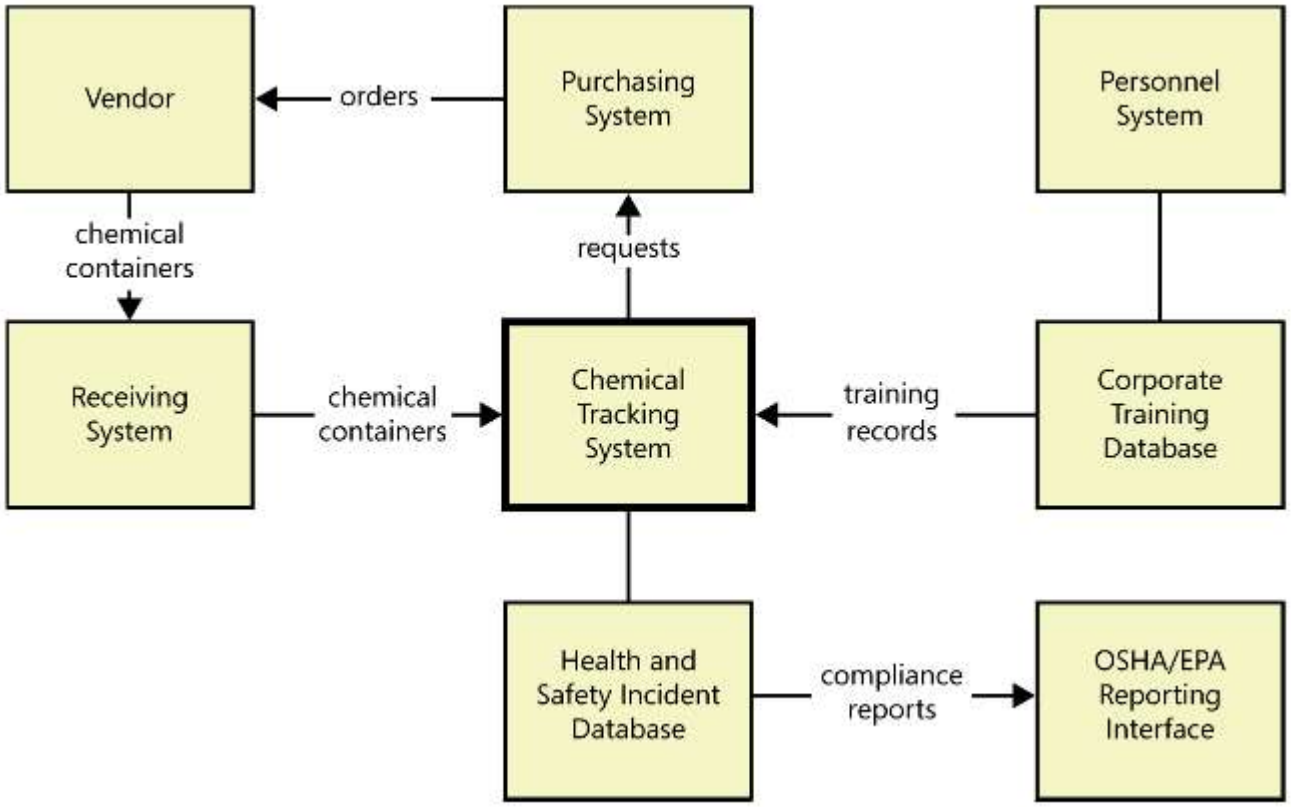
- Ecosystem map
  - An ecosystem map shows all of the systems related to the system of interest that interact with one another and the nature of those interactions
  - An ecosystem map represents scope by showing all the systems that interconnect and that therefore might need to be modified to accommodate your new system.
  - Ecosystem maps differ from context diagrams in that they show other systems that have a relationship with the system you're working on, including those without direct interfaces.

Scope representation techniques

- Ecosystem map
  - You can identify the affected systems by determining which ones consume data from your system.
  - When you reach the point that your project does not affect any additional data, you've identified the scope boundary of systems that participate in the solution.

Scope representation techniques

- Ecosystem map
  - The following Figure illustrates a portion of the Ecosystem map for the Chemical Tracking System.



Scope representation techniques

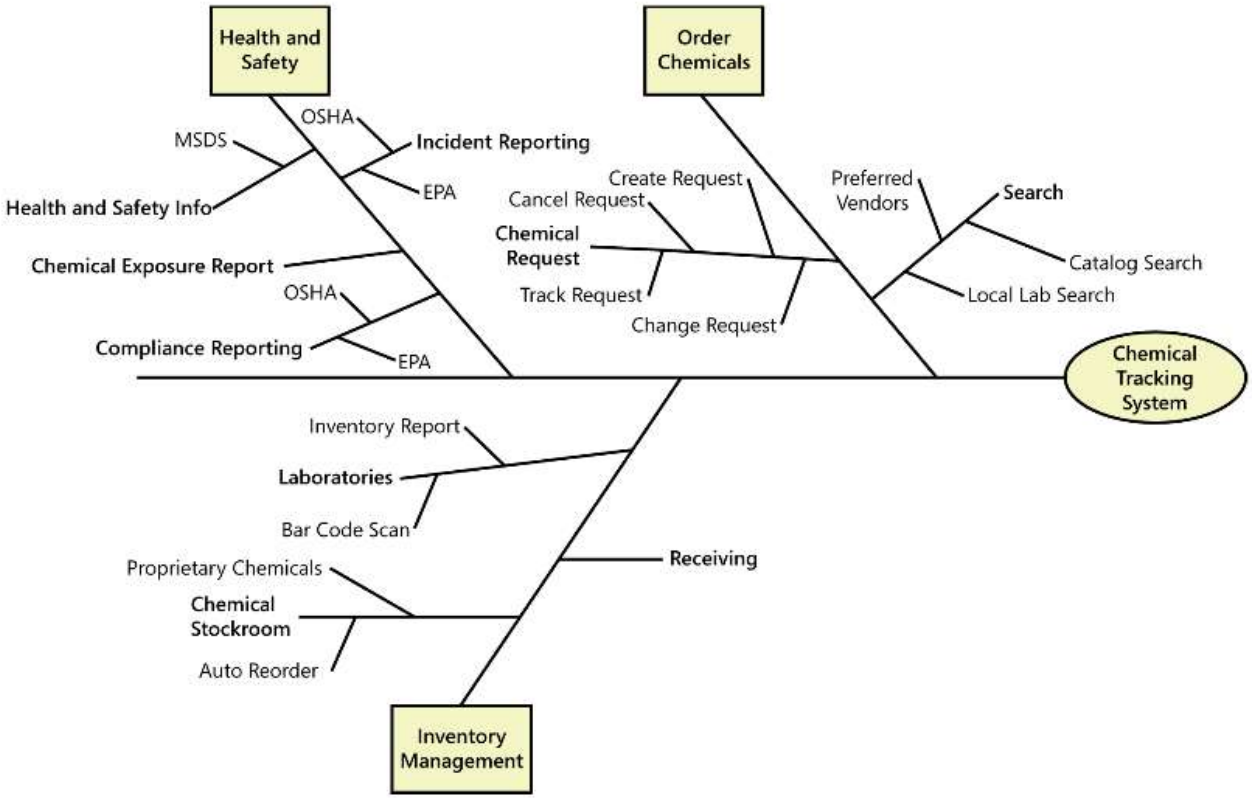
- Ecosystem map
  - The systems are all shown in boxes (such as the Purchasing System or Receiving System).
  - The primary system we are working on is shown in a bold box (Chemical Tracking System), but if all systems have equal status in your solution, you can use the same box style for all of them.
  - The lines show interfaces between systems
  - Lines with arrows and labels show that major pieces of data are flowing from one system to another
  - Some of these same flows can also appear on the context diagram.

Scope representation techniques

- Feature tree
  - A feature tree is a visual depiction of the product's features organized in logical groups, hierarchically subdividing each feature into further levels of detail
  - The feature tree provides a concise view of all of the features planned for a project, making it an ideal model to show to executives who want a quick glance at the project scope.
  - A feature tree can show up to three levels of features, commonly called level 1 (L1), level 2 (L2), and level 3 (L3).
  - L2 features are subfeatures of L1 features, and L3 features are subfeatures of L2 features.

Scope representation techniques

- Feature tree
  - The following Figure shows a partial feature tree for the Chemical Tracking System.



Scope representation techniques

- Feature tree
  - The main branch of the tree in the middle represents the product being implemented.
  - Each feature has its own line or “branch” coming off that central main branch.
  - The boxes represent the L1 features, such as Order Chemicals and Inventory Management.
  - The lines coming off an L1 branch are L2 features: Search and Chemical Request are subfeatures of Order Chemicals.
  - The branches off an L2 branch are the L3 features: Local Lab Search is a subfeature of Search.

Scope representation techniques

- Feature tree
  - When planning a release or an iteration, you can define its scope by selecting a specific set of features and subfeatures to be implemented
  - You could implement a feature in its entirety in a specific release, or you could implement only a portion of it by choosing just certain L2 and L3 subfeatures.
  - Future releases could enrich these rudimentary implementations by adding more L2 and L3 subfeatures until each feature is fully implemented in the final product.
  - So the scope of a particular release consists of a defined set of L1, L2, and/or L3 features chosen from the feature tree.

Scope representation techniques

- Feature tree
  - Tracking progress and Development
    - You can mark up a feature tree diagram to illustrate these feature allocations across releases by using colors or font variations.
    - Alternatively, you can create a feature roadmap table that lists the subfeatures planned for each release

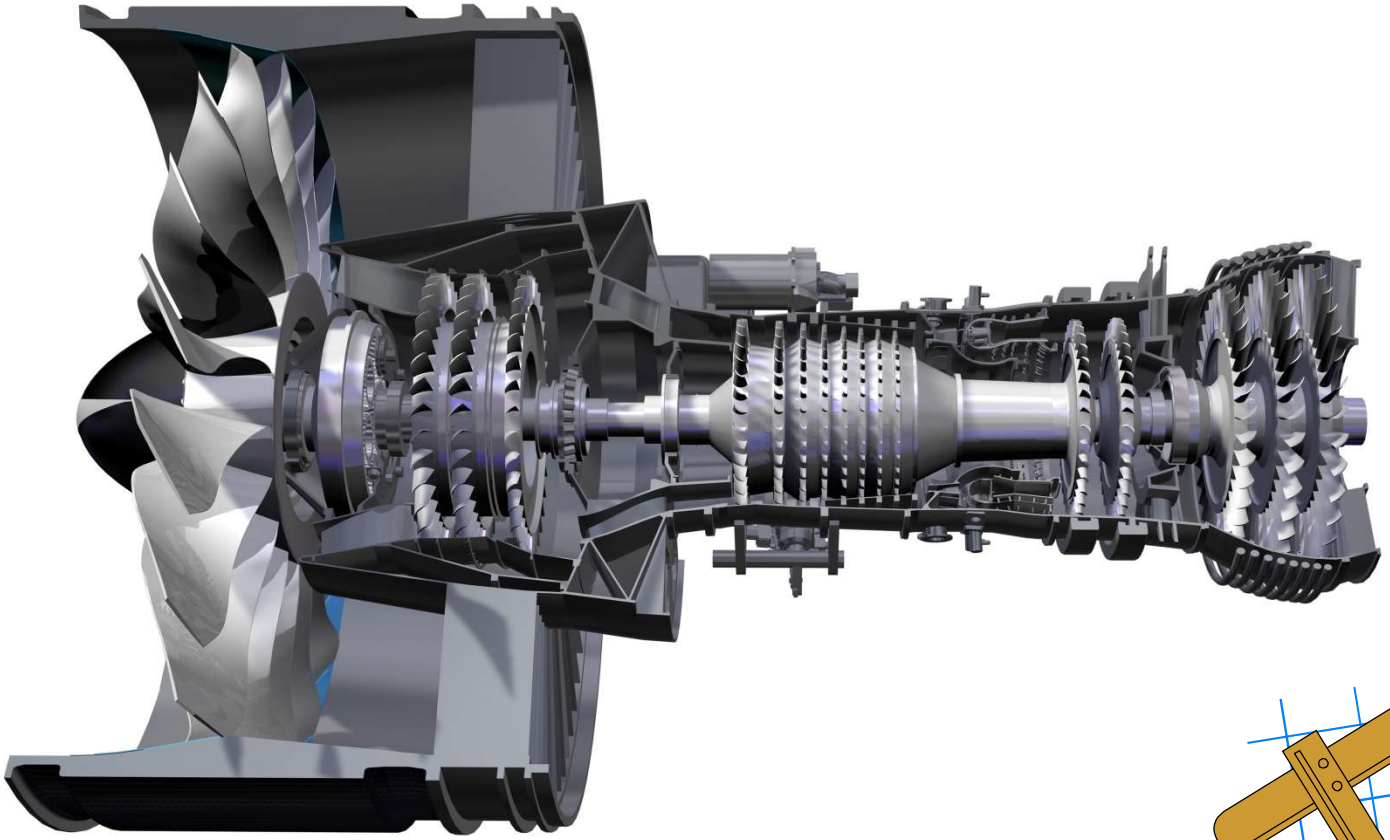
Scope representation techniques

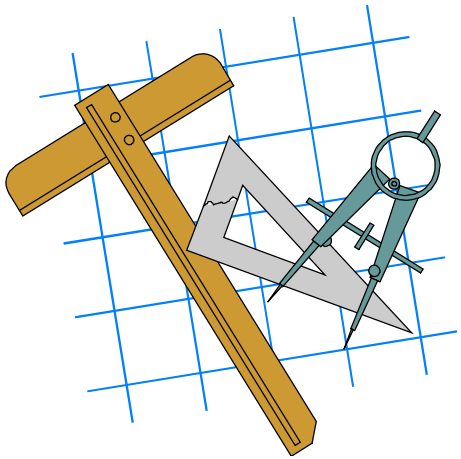
- Event list
  - An event list identifies external events that could trigger behavior in the system.
  - The event list depicts the scope boundary for the system by naming possible business events triggered by users, time-triggered (temporal) events, or signal events received from external components, such as hardware devices.
  - The event list only names the events; the functional requirements that describe how the system responds to the events would be detailed in the SRS by using event-response tables.

Scope representation techniques

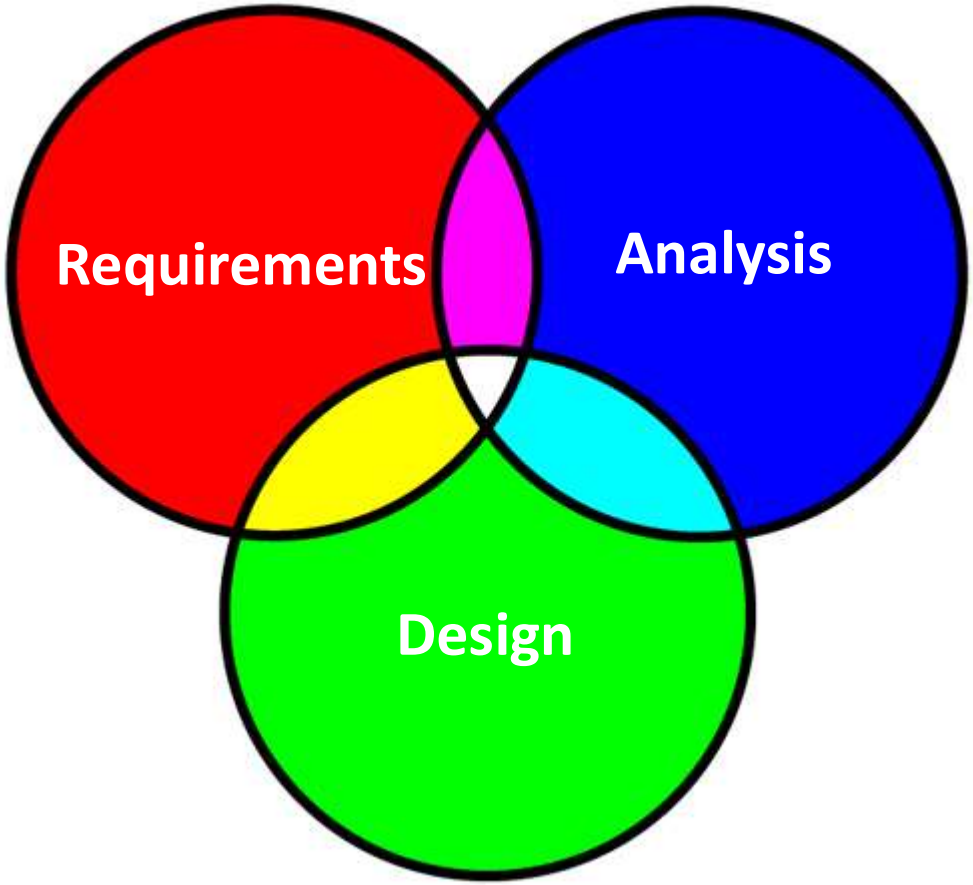
- Event list
  - The event list complements the context diagram and ecosystem map.
  - The context diagram and ecosystem map collectively describe the external actors and systems involved, whereas the event list identifies what those actors and systems might do to trigger behavior in the system being specified.

Analysis guides the Design



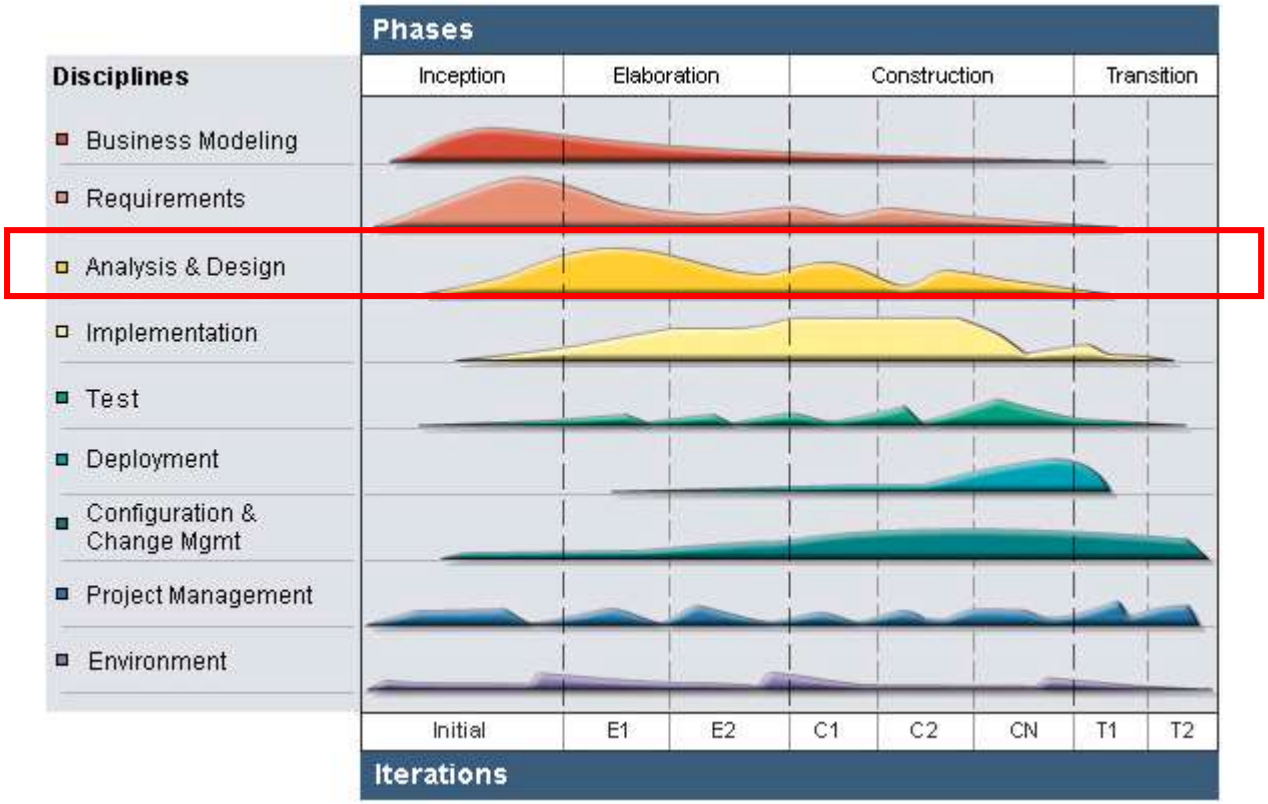


The Overlap of R/A/D means Reality

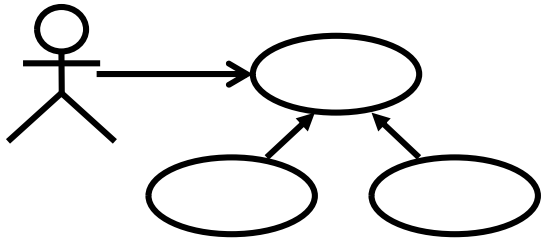


Analysis and Design

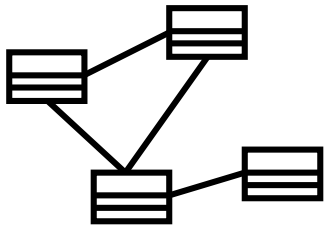
- The purposes of Analysis and Design are to:
  - Transform the requirements into a design of the system-to-be
  - Evolve a robust architecture for the system
  - Adapt the design to match the implementation environment, designing it for performance



Analysis and Design Overview

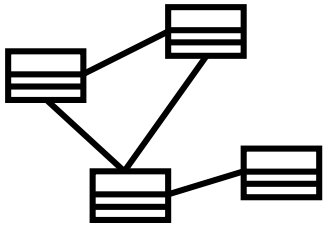


Use-Case Model

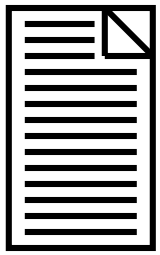


Analysis Model

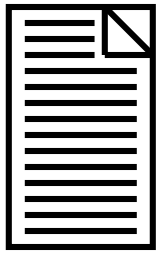
Analysis and Design



Design Model



Glossary

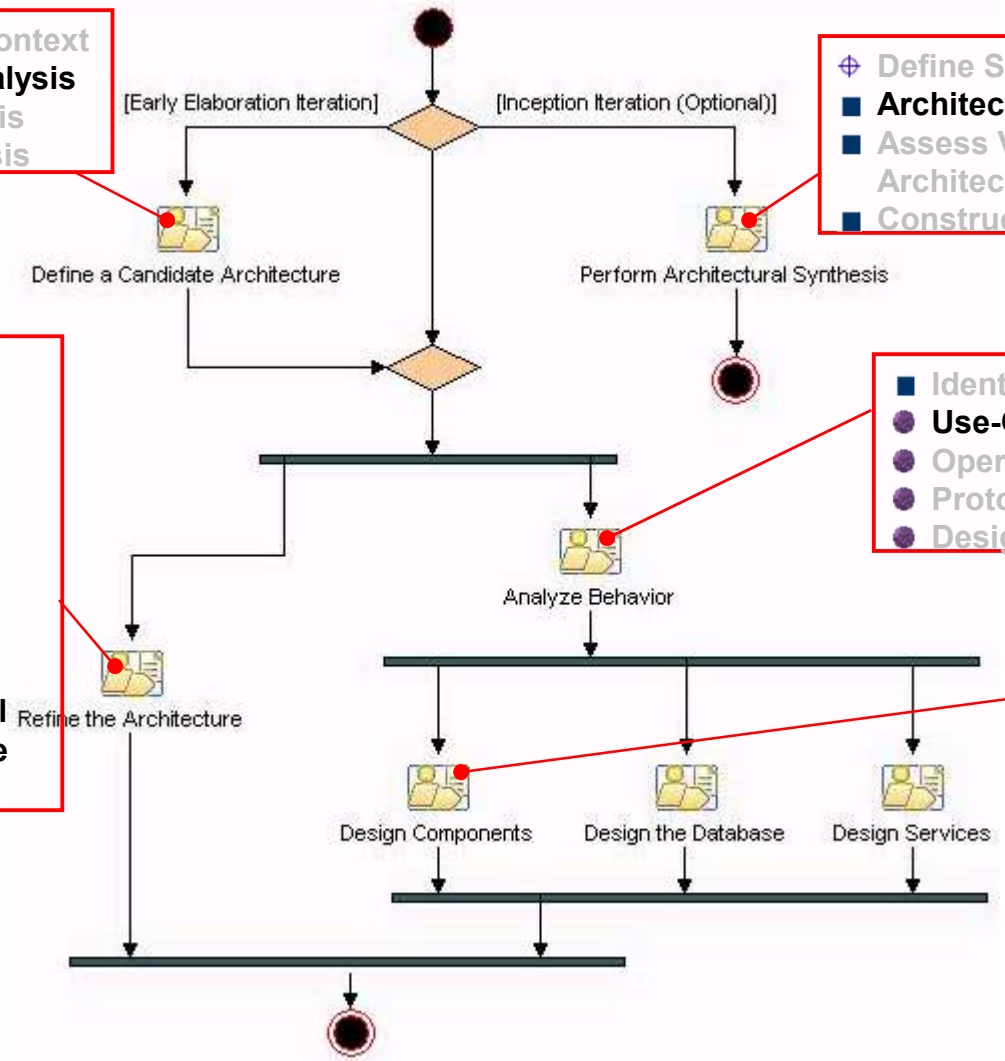


Supplementary Specification

Data Model
(optional)

Architecture Document
)

The Analysis and Design Workflow



- Define System Context
- **Architectural Analysis**
- Use-Case Analysis
- Operation Analysis

- Define System Context
- **Architectural Analysis**
- Assess Viability of Architectural PoC
- Construct Architectural PoC

- **Identify Design Mechanisms**
- **Identify Design Elements**
- Operation Analysis
- Identify Services
- Incorporate Existing Design Elements
- **Structure the Implementation Model**
- **Describe the Run-time Architecture**
- **Describe Distribution**

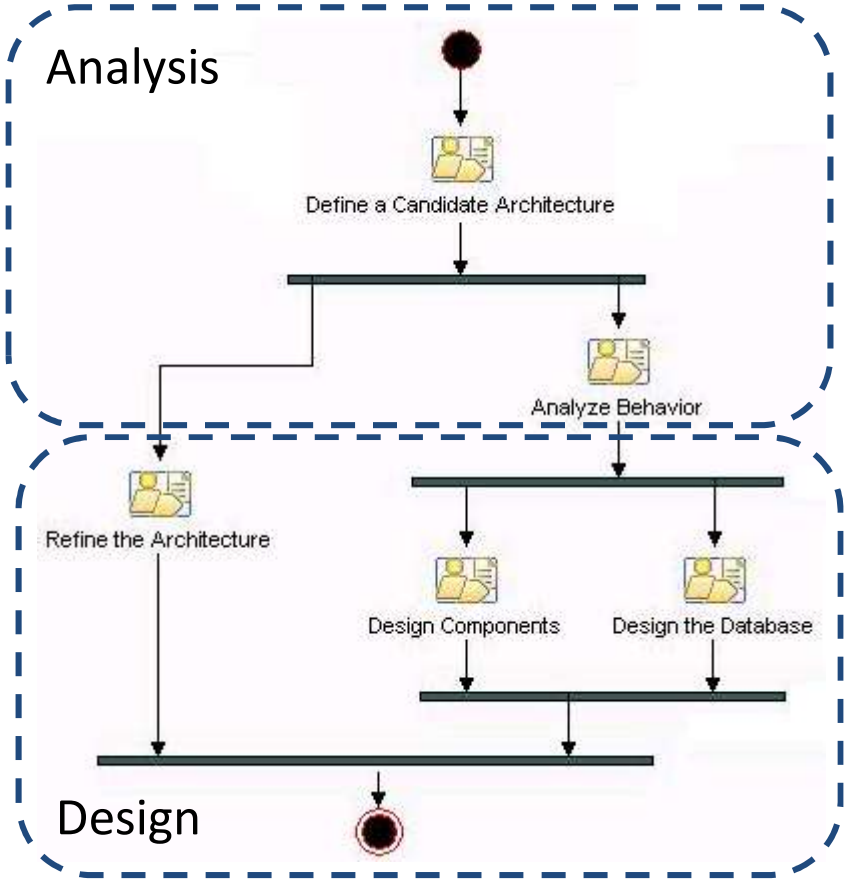
- Identify Design Elements
- **Use-Case Analysis**
- Operation Analysis
- Prototype the User-Interface
- Design the User-Interface

- **Software Architect**
- **Designer roles**
- **Other roles**

- Use-Case Design
- **Subsystem Design**
- Operation Design
- **Class Design**
- Define Testability Elements
- Design Testability Elements
- Capsule Design

OOAD - RUP

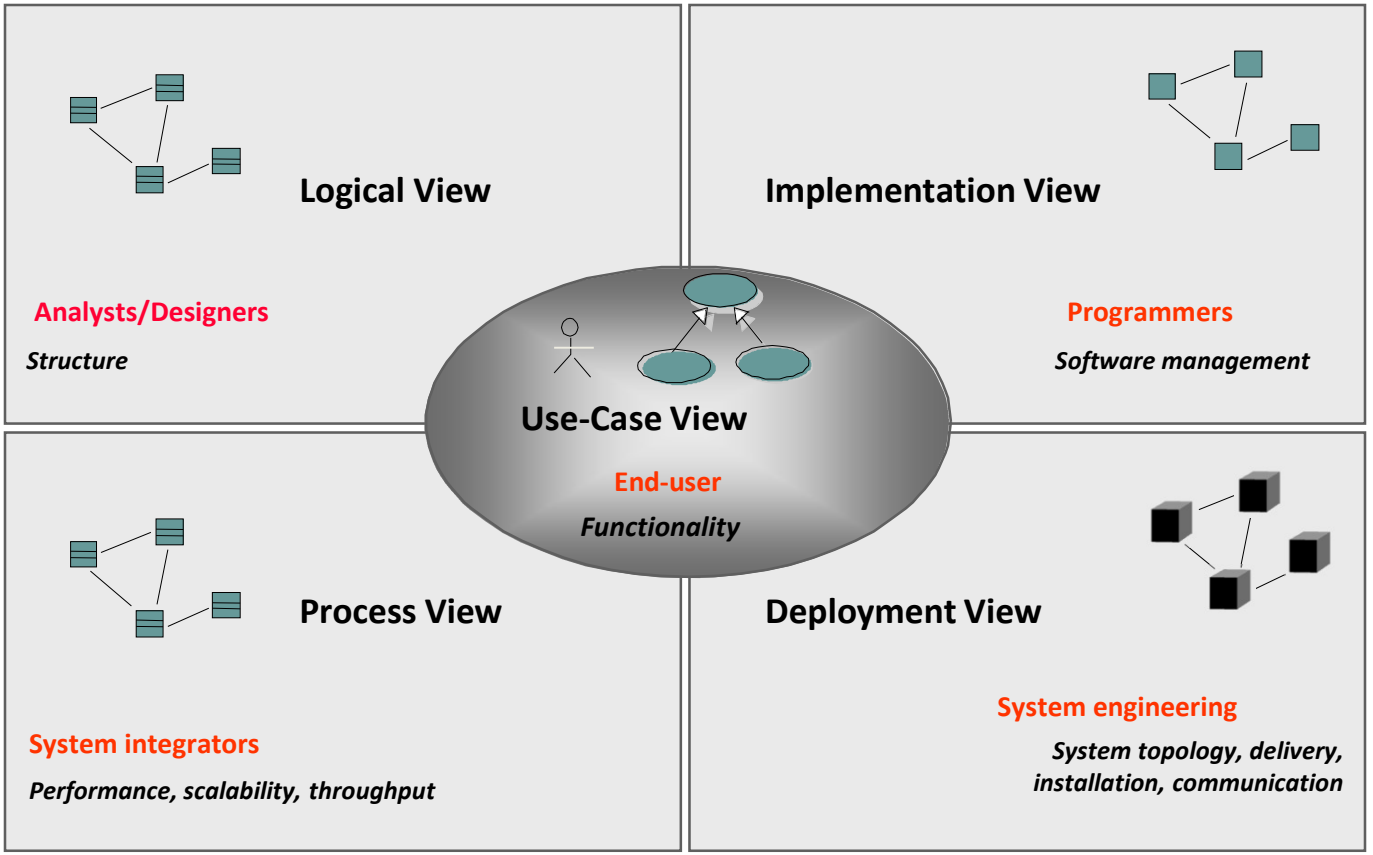
- Analysis
  - Architectural Analysis (Define a Candidate Architecture)
  - Use-Case Analysis (Analyze Behavior)
- Design
  - Identify Design Elements (Refine the Architecture)
  - Identify Design Mechanisms (Refine the Architecture)
  - Class Design (Design Components)
  - Subsystem Design (Design Components)
  - Describe the Run-time Architecture and Distribution (Refine the Architecture)
  - Design the Database



Analysis Versus Design

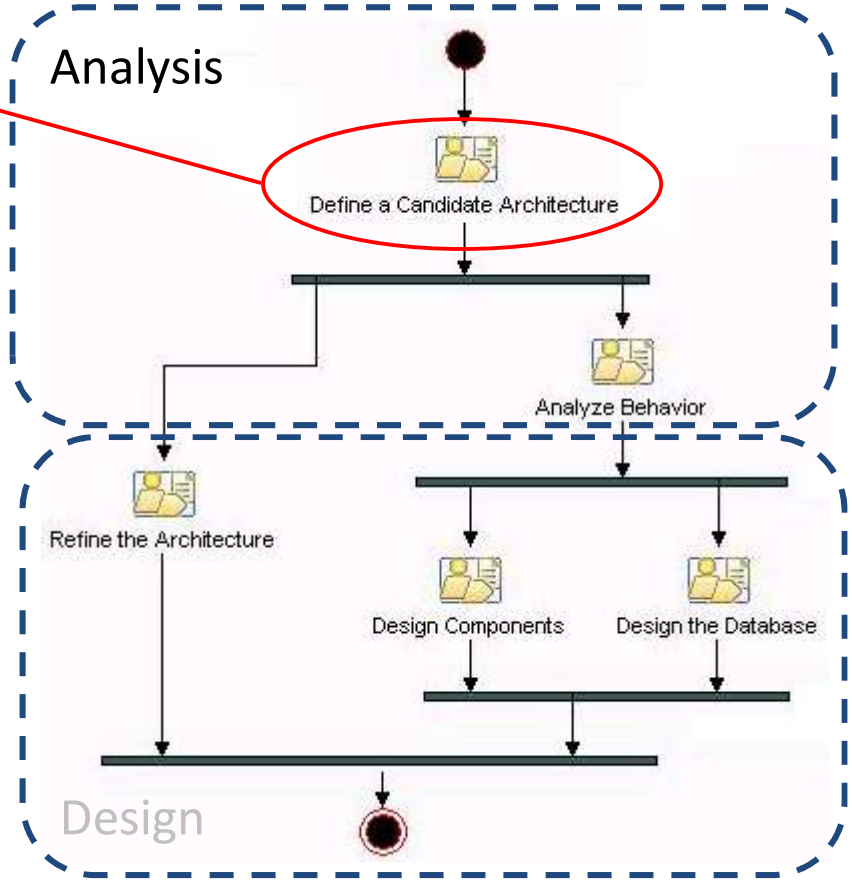
Analysis	Design
▸ Focus on understanding the problem ▸ Idealized design ▸ Behavior ▸ System structure ▸ Functional requirements ▸ A small model	▸ Focus on understanding the solution ▸ Operations and attributes ▸ Performance ▸ Close to real code ▸ Object lifecycles ▸ Nonfunctional requirements ▸ A large model

Architectural Views: The 4+1 View Model



OOAD

- Analysis
  - Architectural Analysis
    (Define a Candidate Architecture)
  - Use-Case Analysis
    (Analyze Behavior)
- Design
  - Identify Design Elements
    (Refine the Architecture)
  - Identify Design Mechanisms (Refine the Architecture)
  - Class Design (Design Components)
  - Subsystem Design
    (Design Components)
  - Describe the Run-time Architecture and Distribution (Refine the Architecture)
  - Design the Database



Architectural Analysis

- Purpose
  - To define a candidate architecture for the system based on experience gained from similar systems or in similar problem domains
  - To define the architectural patterns, key mechanisms, and modeling conventions for the system
- Role
  - Software Architect
- Major Steps
  - Define the High-Level Organization of Subsystems
  - Identify Key Abstractions
  - Develop Deployment Overview
  - Identify Analysis Mechanisms

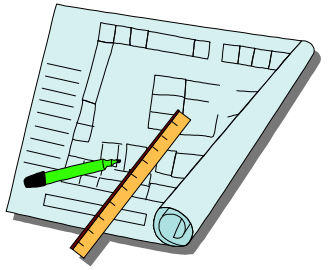
Define the High-Level Organization of Subsystems

- Purpose
  - To create an initial structure for the Design Model
- Normally the design model is organized in layers – a common **architectural pattern** for moderate to large-sized systems
- During architectural analysis, you usually focus on the two high-level layers, that is, the **application** and **business-specific** layers
  - This is what is mean by the *high-level organization of subsystems*

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

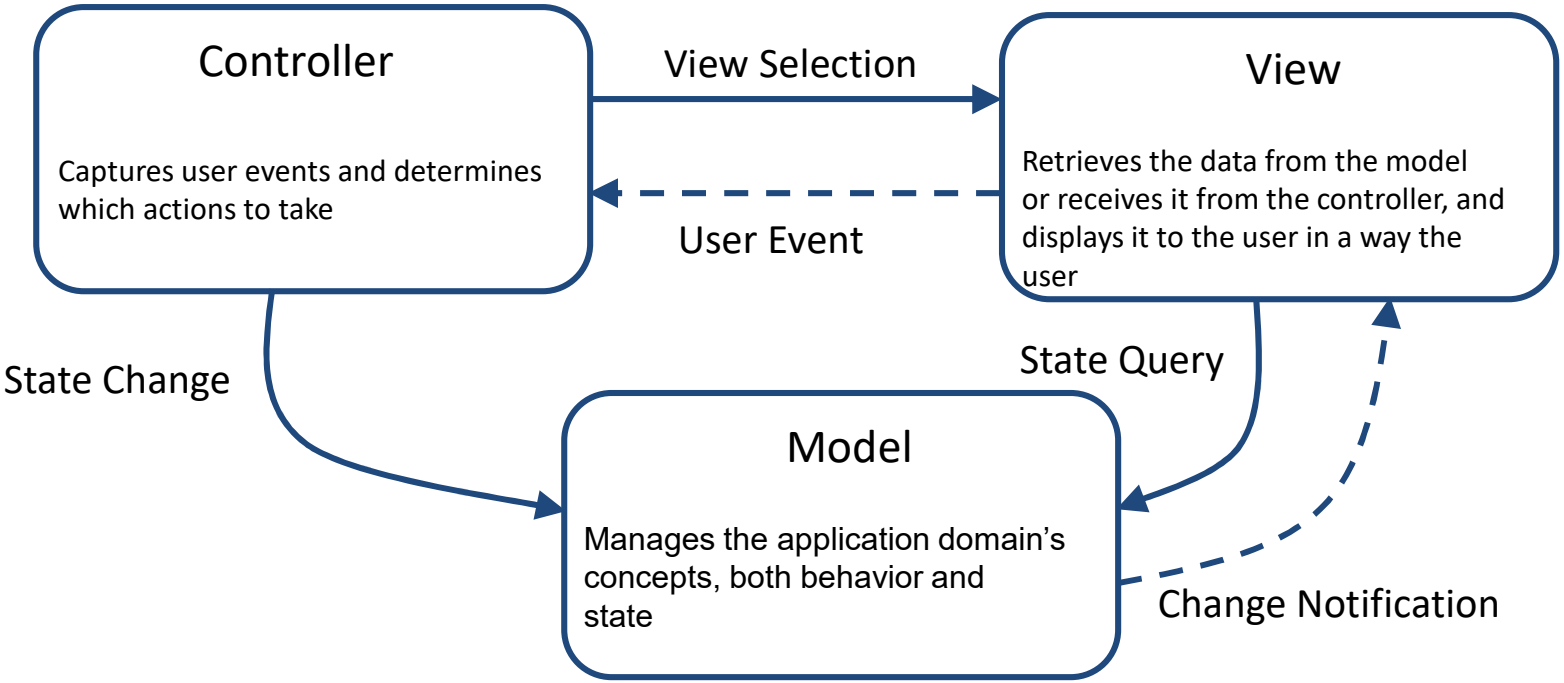


What Is an Architectural Pattern?

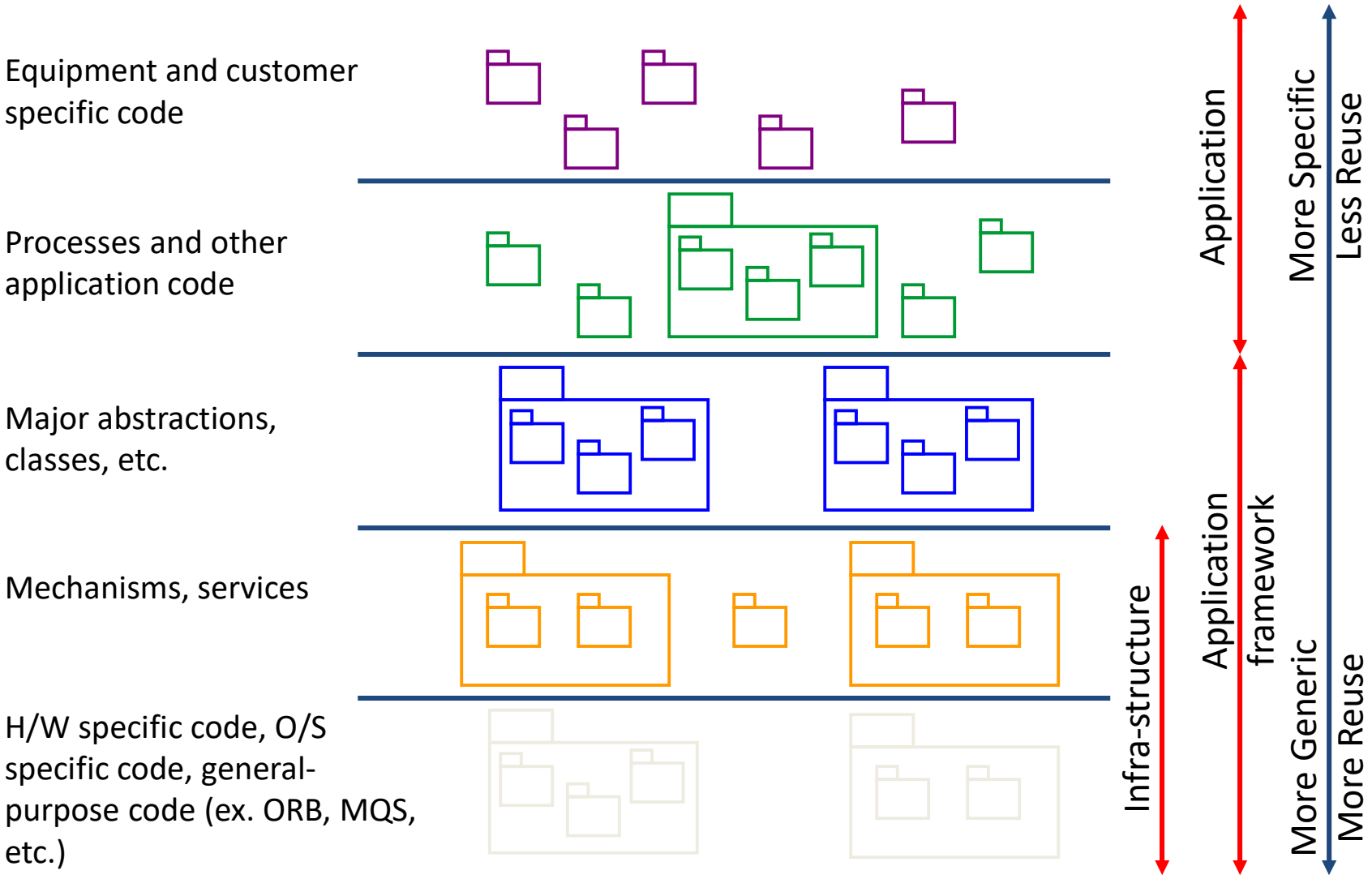
- An architectural pattern expresses a fundamental structural organization schema for software systems
  - It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them – *Buschman et al, "Pattern-Oriented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

The Model-View-Controller (MVC) Architecture

- Conceived in the mid-1980's
- Extensively applied in most object-oriented user interfaces
- Adapted to respond to specific platform requirements, such as J2EE



Layered Architectures

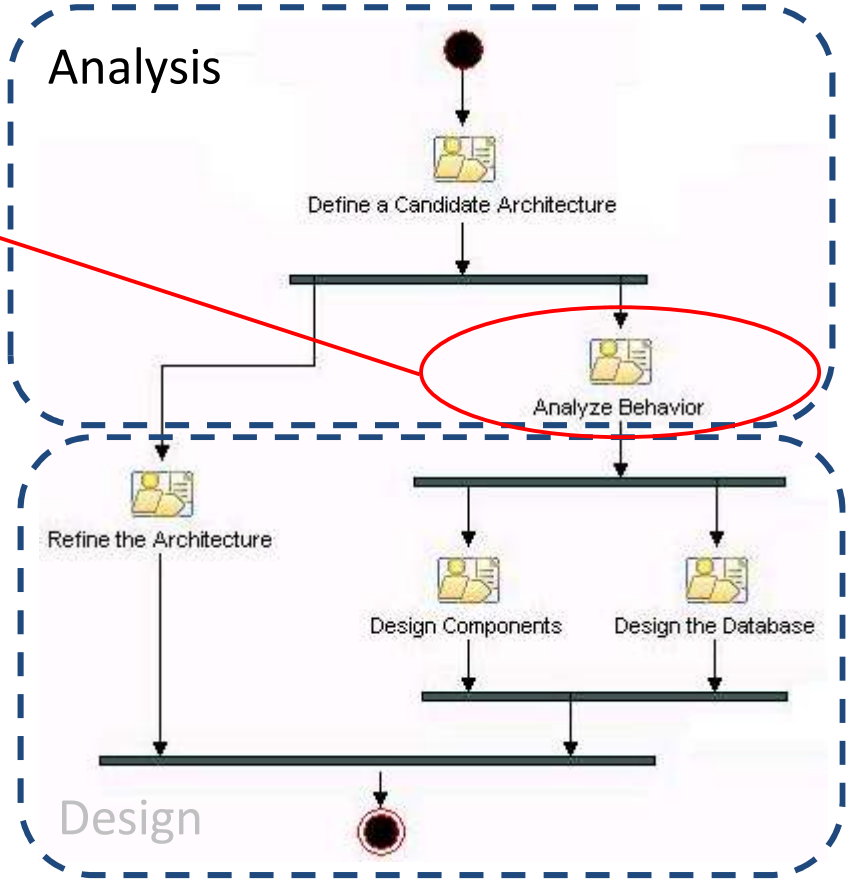


Layering Considerations

- 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

OOAD

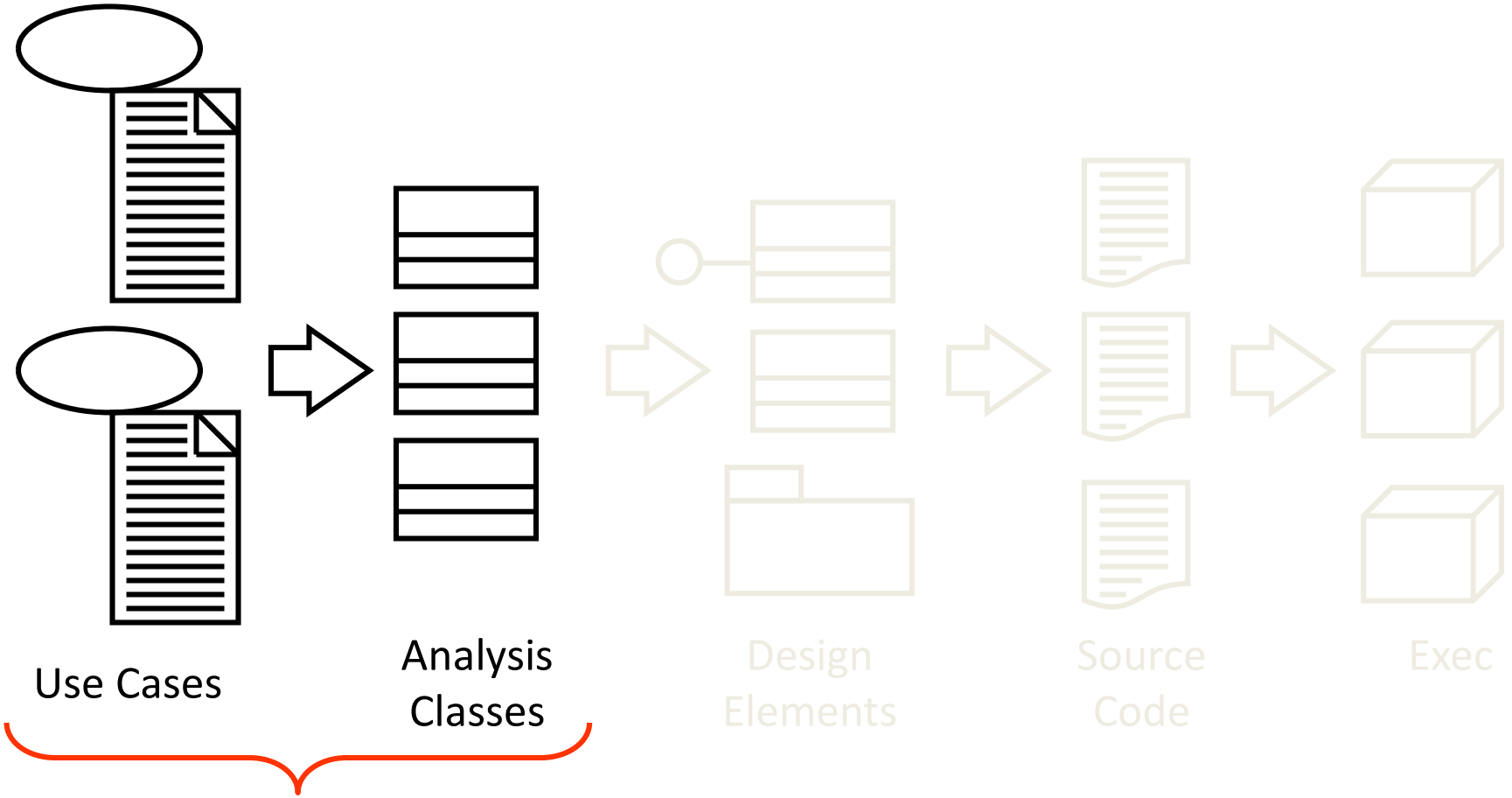
- Analysis
  - Architectural Analysis (Define a Candidate Architecture)
  - Use-Case Analysis (Analyze Behavior)
- Design
  - Identify Design Elements (Refine the Architecture)
  - Identify Design Mechanisms (Refine the Architecture)
  - Class Design (Design Components)
  - Subsystem Design (Design Components)
  - Describe the Run-time Architecture and Distribution (Refine the Architecture)
  - Design the Database



Use-Case Analysis

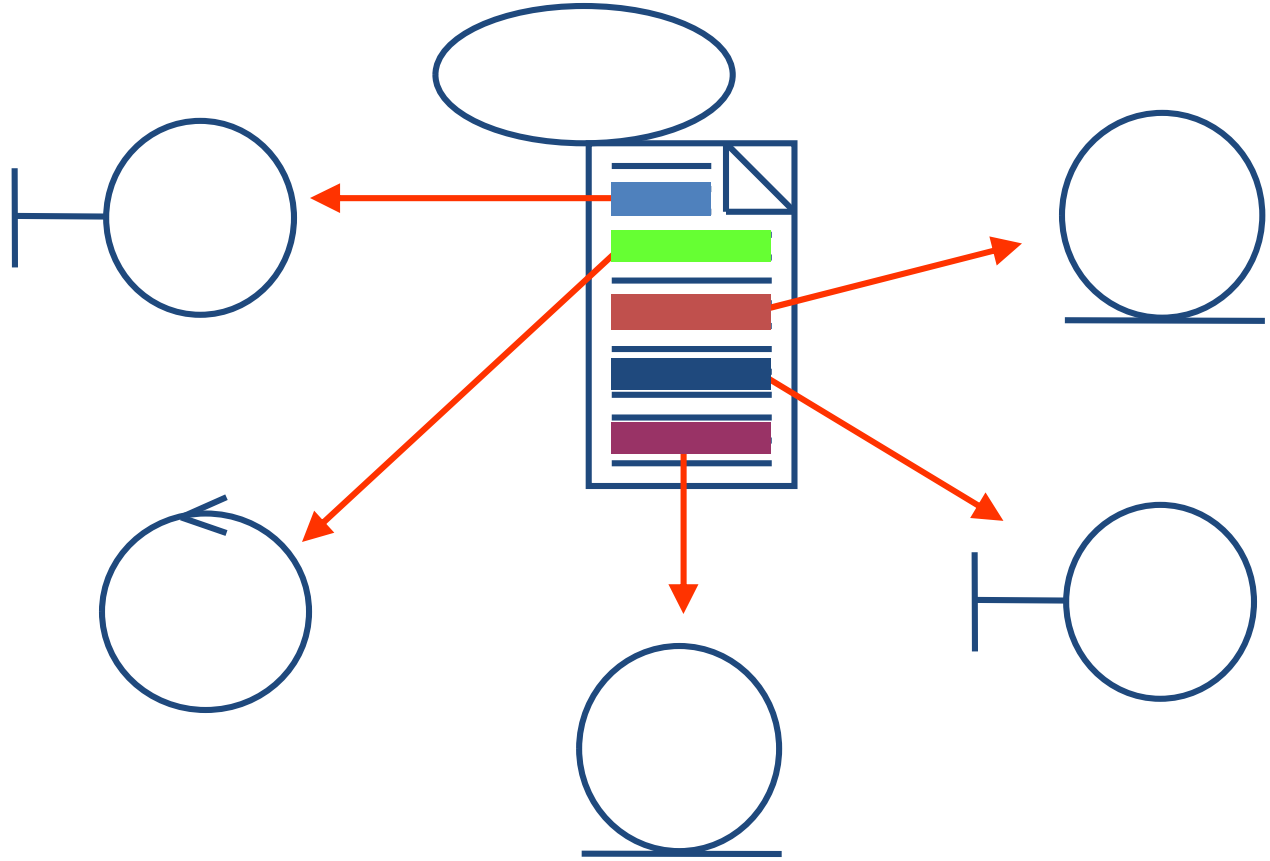
- Purpose
  - To identify the analysis classes of our system, including:
    - their “responsibilities”, attributes and associations to other classes, and
    - usage of analysis mechanisms
- Role
  - Designer
- Major Steps
  - Create Analysis Use-Case Realization
  - Supplement the Use-Case Description
  - Model Use-Case Scenarios with Interaction Diagrams
  - Model Participating Classes in Class Diagrams
  - Reconcile the Analysis Use-Case Realizations
  - Qualify Analysis Mechanisms

Analysis Classes: A First Step Toward Executables

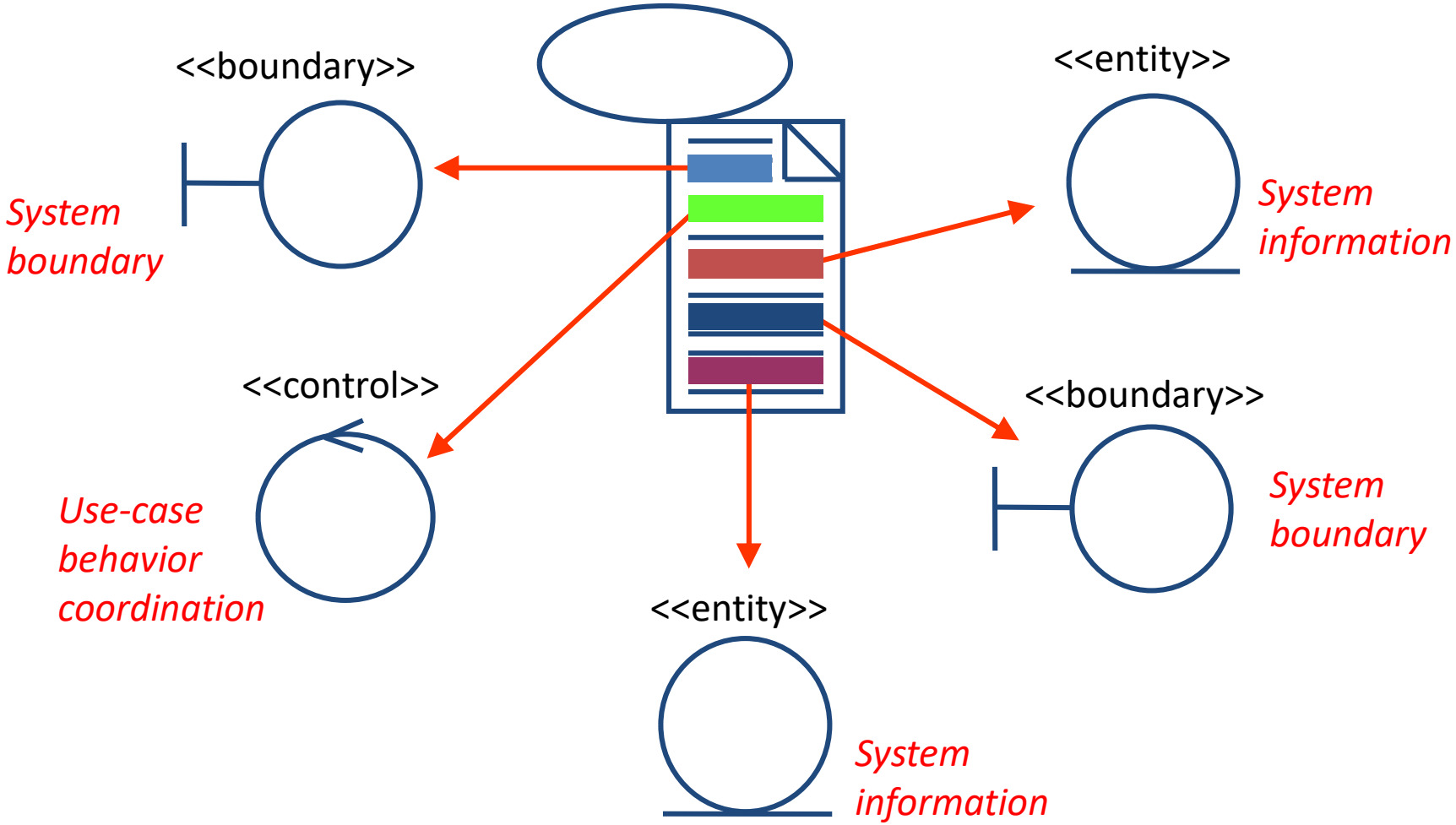


Find Classes from Use-Case Behavior

- The complete behavior of a use case has to be distributed to analysis classes



What Is an Analysis Class?



Example: Course Registration System

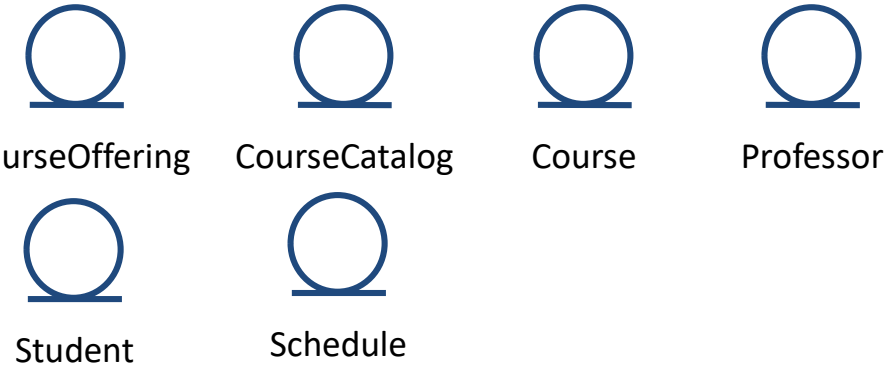
- Basic flow of events for the Submit Grades use case:

This use case starts when a Professor wishes to submit student grades for one or more classes completed in the previous semester:

1. The system displays a list of course offerings the Professor taught in the previous semester.
2. The Professor selects a course offering.
3. The system retrieves a list of all students who were registered for the course offering. The system displays each student and any grade that was previously assigned for the offering.
4. For each student on the list, the Professor enters a grade: A, B, C, D, F, or I. The system records the student's grade for the course offering. If the Professor wishes to skip a particular student, the grade information can be left blank and filled in at a later time. The Professor may also change the grade for a student by entering a new grade.

Example: Course Registration System

- Key abstractions (previously identified)



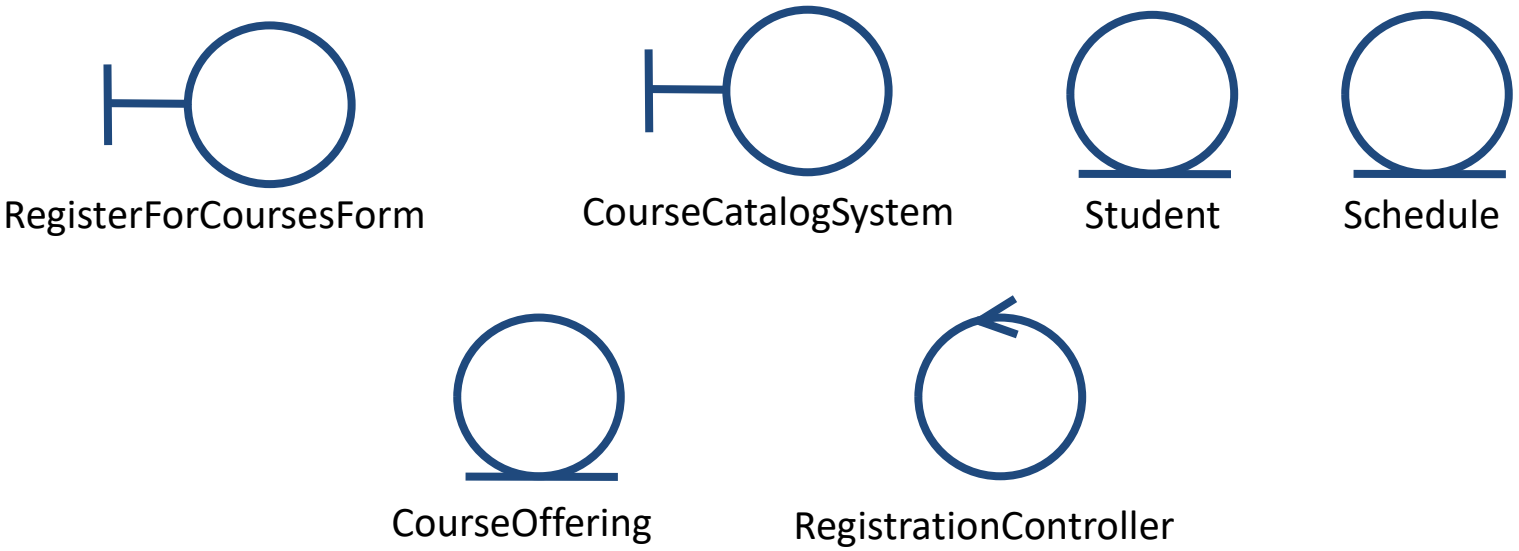
- Newly identified classes

- How would you characterize the new class?

Summary: Course Registration System Example

Use-Case Model

Design Model

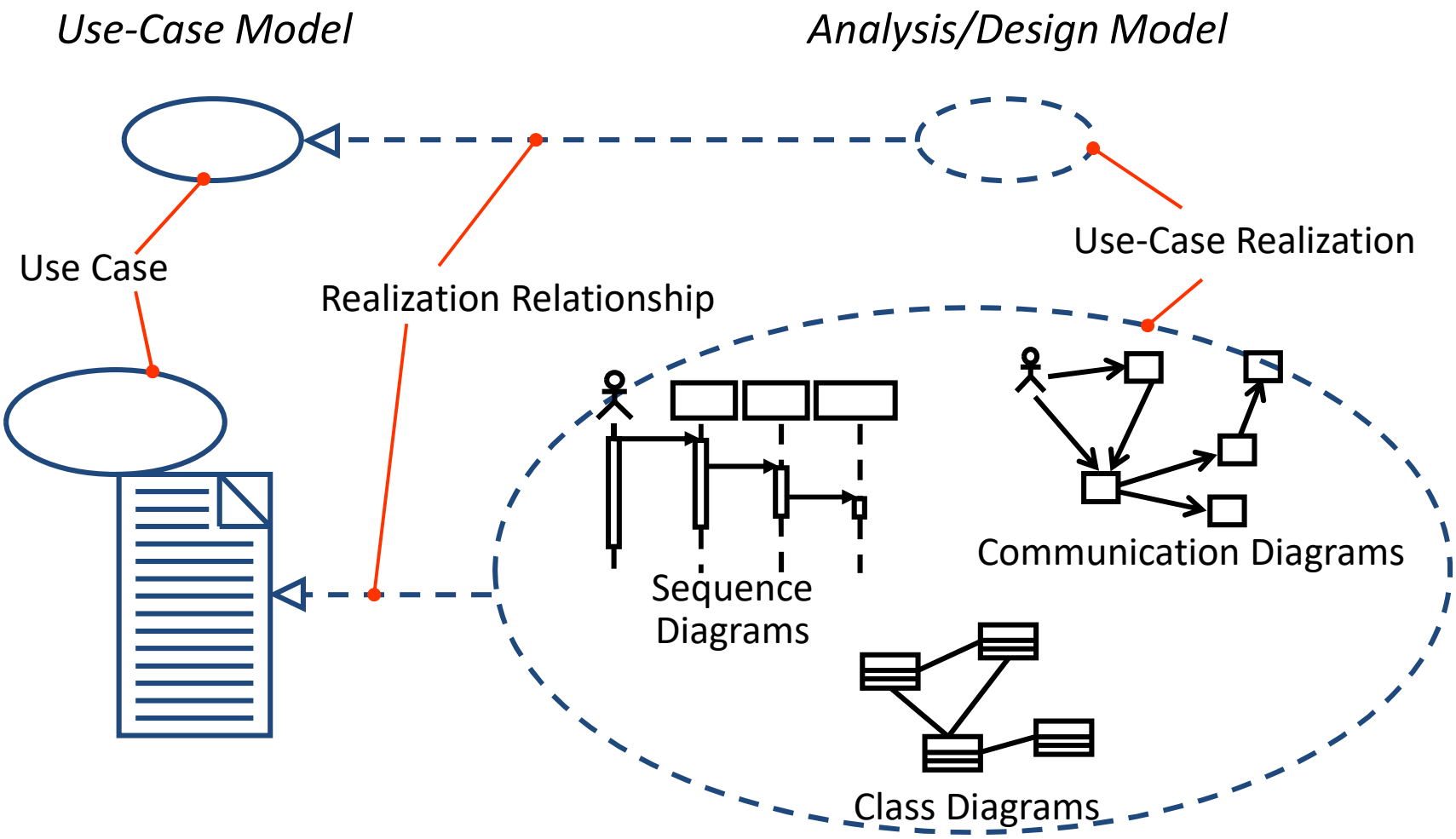


What Is a Use-Case Realization?

- The bridge between Requirements-centric tasks and Analysis/Design-centric tasks
- It provides:
  - A way to trace behavior in the Analysis and Design Models back to the Use-Case Model
  - A construct in the Analysis and Design Models, which organizes work products related to the use case but which belong to the design model
    - Typically consist of sequence and class diagrams
- Shown as a collaboration* stereotyped <<use-case realization>>

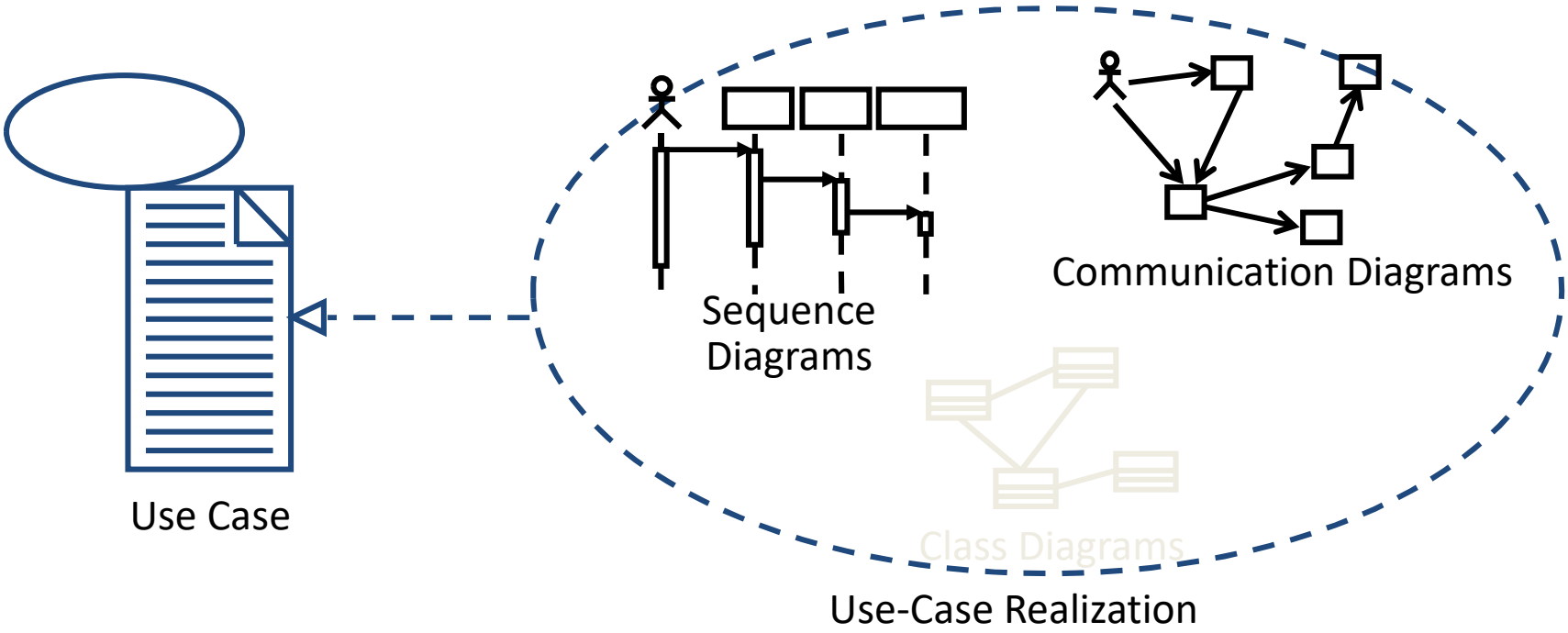
* UML Collaboration = structure of collaborating elements (roles), each performing a specialized function, which collectively accomplish some desired functionality

What Is a Use-Case Realization?



Distribute Use-Case Behavior

- For each use-case flow of events:
  - Create one or more interaction diagrams (sequence diagrams recommended)
  - Identify analysis objects responsible for the required use-case behavior
  - Allocate use-case responsibilities to analysis classes



Guidelines: Interaction Diagrams and Use Cases

- Each initial interaction diagram describes one use-case scenario
  - Diagrams should be named after the use-case scenarios
  - The interaction should begin with an actor, since an actor always invokes the use case

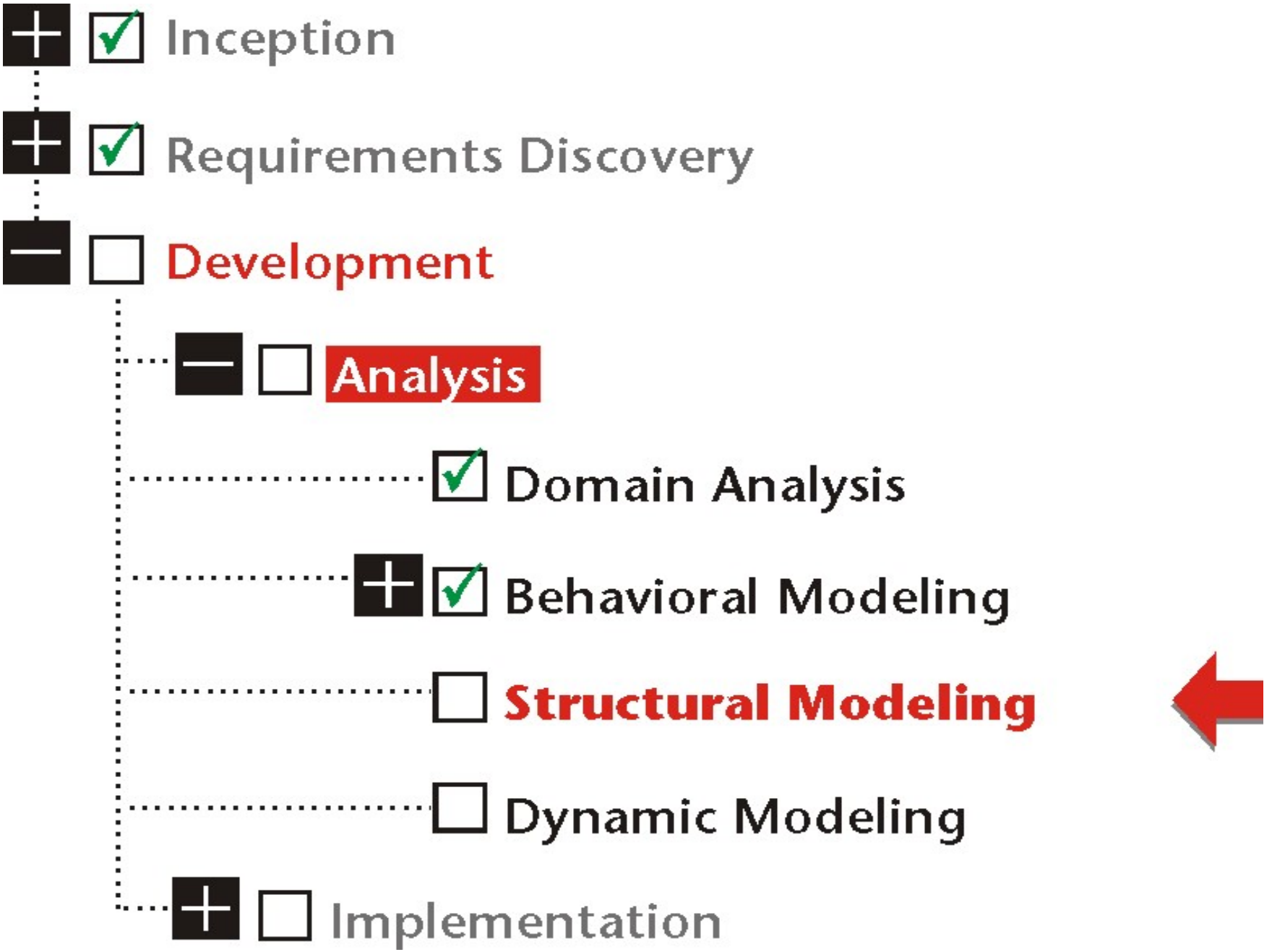
- One diagram is not enough
  - At least one diagram for the main flow of events
  - Plus at least one diagram for each non-trivial alternative or exceptional flow
  - Separate diagrams for complex flows

- *We will discuss System Sequence Diagram, Activity diagrams , and Interaction Sequence/Communication Diagrams later.*

Structural Modeling

Topics

- The essentials of structural modeling.
- Building blocks of structural modeling.
- Basic object-oriented concepts in the context of structural modeling.
- Discovering class candidates.
- Elaborating and defining classes.
- Relationships among classes.
- Class diagrams.

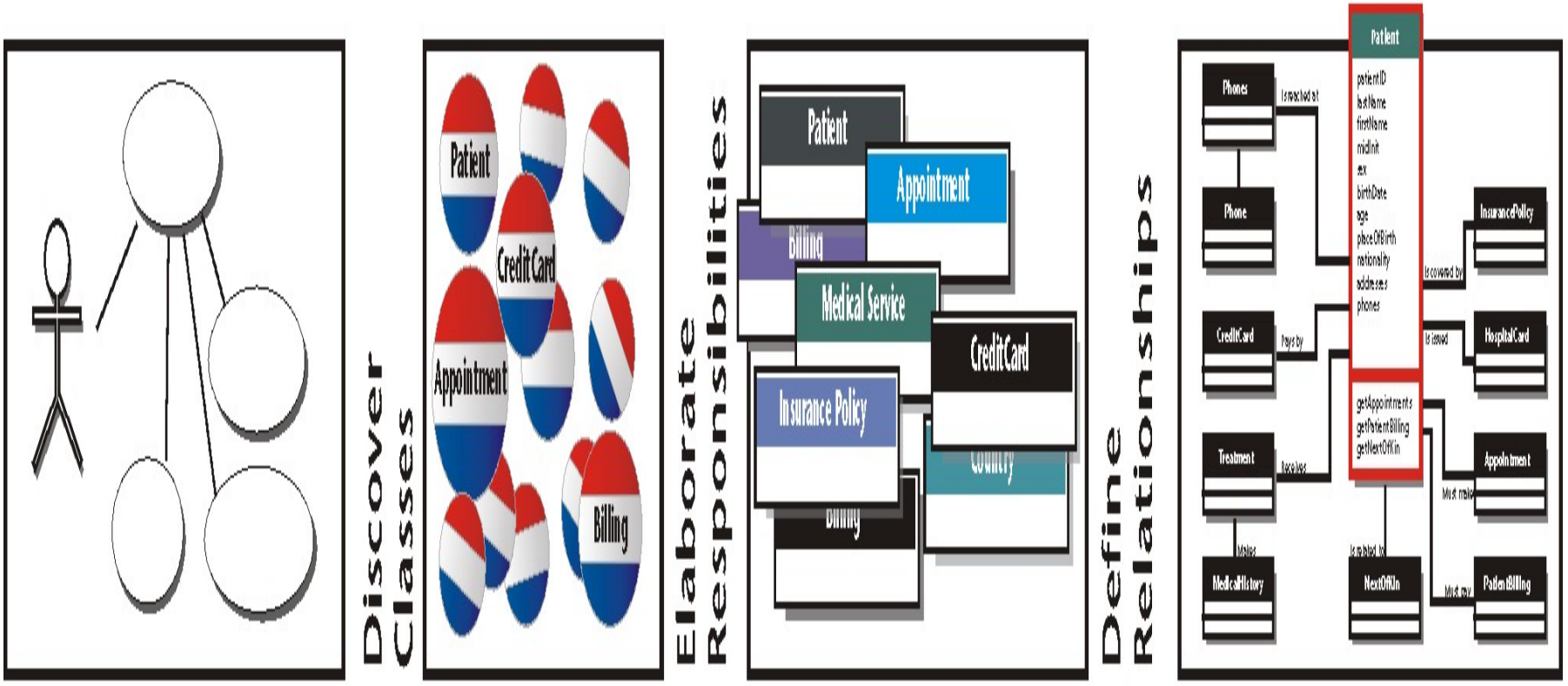

Inception
Requirements Discovery
Development
Analysis
Domain Analysis
Behavioral Modeling
Structural Modeling
Dynamic Modeling
Implementation

Building Blocks of Information Systems

- An information system must have a structure that supports the system's behavior.
- The structure of an information system cannot be monolithic.
- A flexible and reliable structure, therefore, needs building blocks that satisfy the specific requirements of the structure.

Building the Conceptual Structure

From Use Cases to Structural Modeling



Behavioral Model ▶ Class Candidates ▶ Conceptual Classes ▶ Structural Model(s)

Structural Modeling

- Structural modeling represents a view of the building blocks of a system or an entity and their interrelationships within a given scope.

Structural Modeling

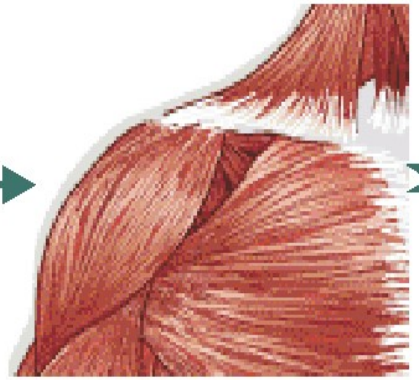
- Common Traits of Structural Modeling
  - **View** may be conceptual, logical or physical.
  - **Scope** can be any relevant or selected range.
  - A **building block** may be fine or coarse.

Defining the Building Block

Selecting the Right Viewpoint In Modeling the Structure

Organ: Body Unit

Anatomy

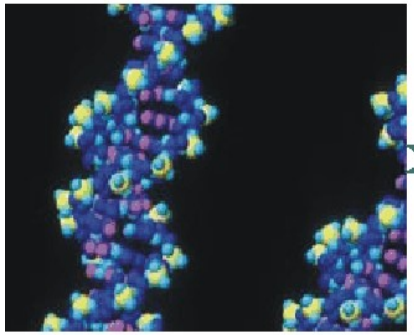


Tissue: Organ Unit

Histology

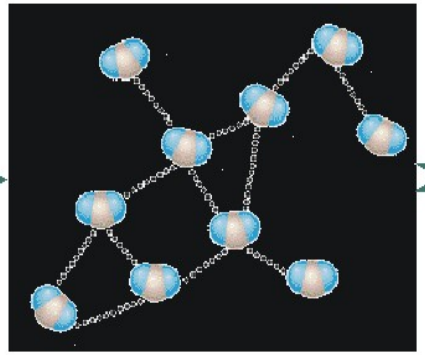
Cell: Tissue Unit

Cytology



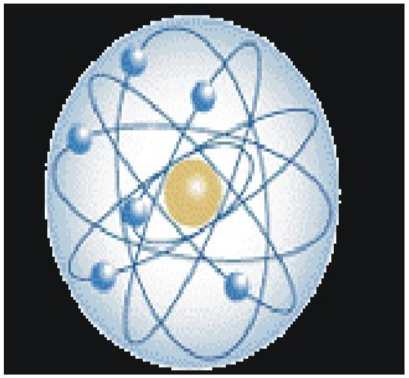
DNA: Cell Unit

Biochemistry



Molecule: DNA Unit

Physics



Atom: Molecule Unit

Classes As Object Templates

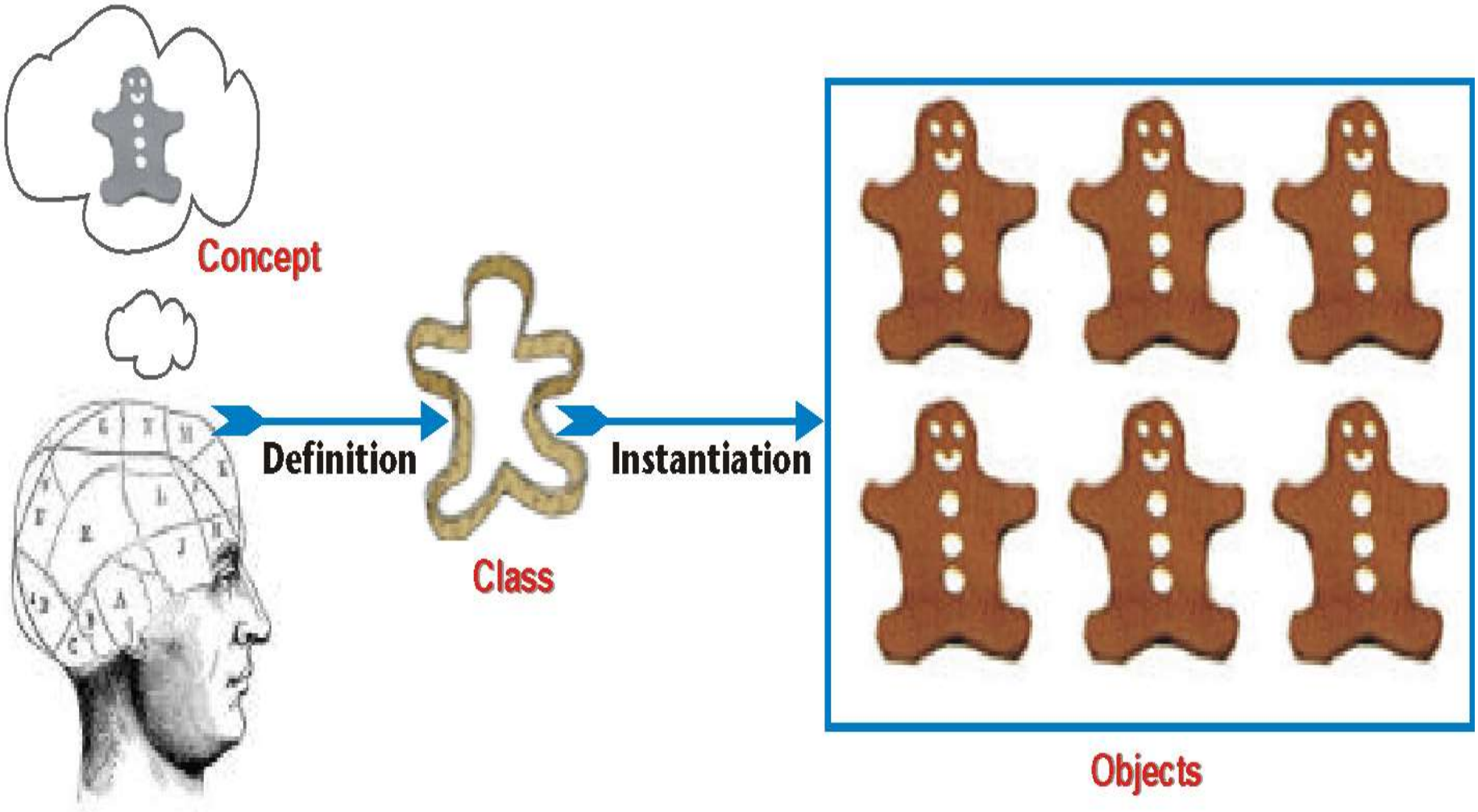
- In the virtual world of software, a class is:

  ❶ An **abstraction** of objects.

  ❷ A **template** for creating objects.

Class as Cookie Cutter

Virtual objects come from classes, not vice versa



Classes As Building Blocks

- Classes are the building blocks of structural modeling (modeling is an abstraction of reality).

- Objects are the structural units of the actual information system (information system emulates reality).

Objects As Black Boxes

- An information system object is a dynamic black box; it interacts with outside entities to provide services but conceals its inner workings.
  - The internal structure of an object is known only to the object itself.

Encapsulation

- Encapsulation is enclosing data and processes within one single unit.
- Encapsulation enables the object to enforce business rules with authority.
- Encapsulation results in two spaces:
  - Private. Data and processes that are inside the object are labeled as "private."
  - Public. Whatever the object exposes — that is, makes visible to the outside world — is "public."

Encapsulation

Public vs. Private

Private

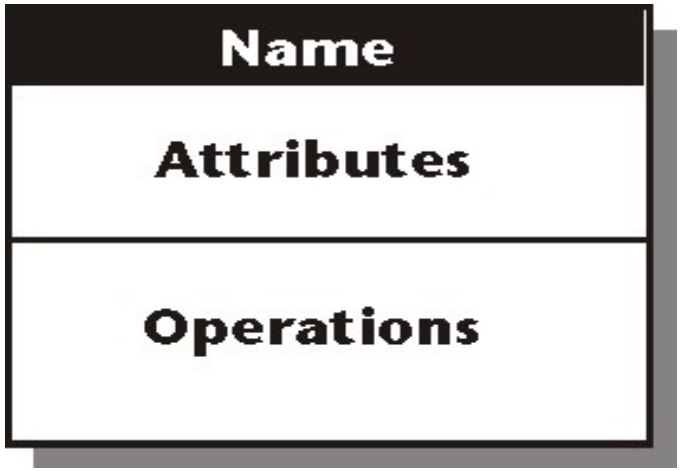
Implementation

Public

Interface

Information Hiding

- Information hiding conceals the inner entities and the workings of the object from outside entities.
- The box metaphor for classes and objects emphasizes: encapsulation and information hiding.



Interface As A Contract

- An object's interface is a contract between the object and the entities that use it.
- the object promises to perform services for an outside entity that behaves according to rules that the object expects.
- The interface of an object, once formalized and made available for use, cannot be changed unless all parties to the contract agree to the change.

Interface

- The interface of an object is both the services that it offers to the outside world and how these services are structured and arranged (what you see is what you get).

- The outside view of a class, object, or module, which emphasizes its abstraction while hiding its structure and the secrets of its behavior." [Booch 1994, 515.]

Structuring the Interface

- The interface of an object — its services — must itself be structured in a predictable manner. It has:
  - ❶ Name
  - ❷ Attributes
  - ❸ Operations

Name

- Rules and conventions that apply to naming classes:
  - Class name must be a noun or a noun phrase.
  - Class name is usually singular (except for collection class).
  - Class name is always capitalized.
    - Student, ApprovalNotice
  - Definite or indefinite articles must be avoided — never APatient

Attribute

- Attribute is what an object knows.
- Class attributes are placeholders: it is the objects that fill the placeholders — or variables — with values.
- Attribute names begin with a *lower-case* letter; firstName.
  - The lower-case start is a convention to distinguish attributes and operations from classes.

Operation

- Operation defines what an object **does** or what can be **done to** it
  - A class merely *defines* what an object is expected to do.
  - It is the object that carries out the actual operation: a Plane class does not fly; a plane object does
- Rules for naming operations are the same as for naming attributes.
  - move (verb)
  - getStarted (verb)

Visibility

- The visibility of an object's attributes or operations defines their availability to other objects.

Symbols for Visibility

Symbol	Visibility	Description
+	**Public**	The attribute or operation is visible to all entities.
-	**Private**	The attribute or operation is private and cannot be (directly) accessed by outside entities.
#	**Protected**	The attribute or operation is available only to the object or its descendants. (*See chapter 15*, Components & Reuse.)
~	**Package (Friend)**	Only other objects in the package (or component) can use the attribute.

Composite and Collection Objects

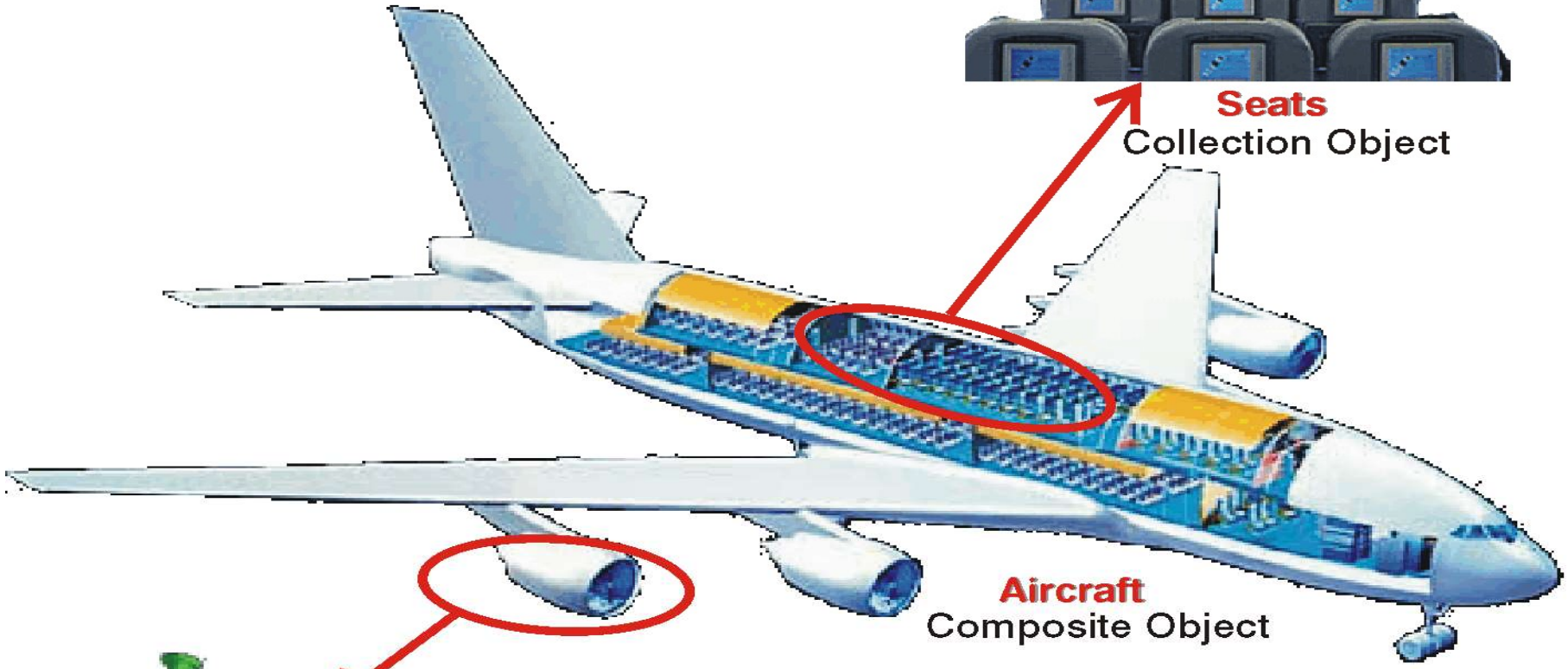
- A composite object is one that is composed of other objects.

- A collection object is a composite object that manages a set of objects instantiated from the *same* class.

Composite & Collection Objects

When an object is made up of other objects

Seats
Collection Object



Aircraft
Composite Object

Engine Assembly
Composite Object

Finding Classes

- To discover business objects, we must start by mining the flow of use cases.

- The messages exchanged between the actors and the system refer to objects that are affected by the interaction between the two:
  - by parsing the messages that the steps in a use case scenario specify, we can start the discovery of classes.

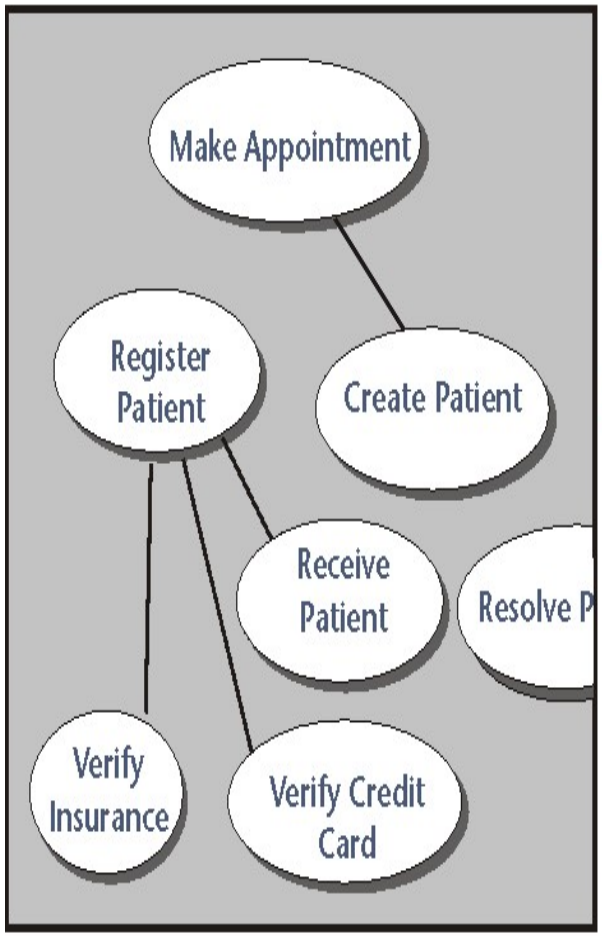
Responsibilities

- An object's responsibilities consist of what it does and what it knows; in other words, its operations and its attributes.

- Before we can define classes in detail, we must discover class candidates and outline their tentative responsibilities.

Finding Class Candidates

Parsing the Behavior for Structure



✓ Identify grammatical objects

✓ Find objects hidden in verbs

✓ Outline class responsibilities

✗ Exclude actors as subjects

✗ Discard obvious misfits

WALDEN MEDICAL CENTER

Patient Management

Preliminary Class Candidates

(Partial Listing)

Class	Responsibilities	Use Cases
Appointment	▪ Make appointment for patient. ▪ Know appointment. ▪ Cancel appointment ▪ Know referral source. ▪ Track appointment.	▪ 100 Refer Patient ▪ 120 Make Appointment ▪ 190 Resolve Problem
Credit Card	▪ Knows patient credit card ▪ Verifies patient credit card	▪ 142 Verify Credit Card ▪ 190 Resolve Patient Billing Issues
Health Insurance Plan	▪ Knows patient's health insurance plan. ▪ Verifies patient's health insurance plan	▪ 140 Register Patient ▪ 144 Verify Insurance Plan
Hospital ID Card	▪ Issues hospital card.	▪ 130 Receive Patient ▪ 140 Register Patient ▪ 144 Issue Hospital Card
Medical Service	▪ Knows medical service	▪ 111 Refer Patient ...
Patient	▪ Knows patient's personal data. ▪ Knows patient's contact data. ▪ Knows patient's appointments. ▪ Knows patient insurance data. ▪ Knows patient's financial data ▪ Knows patient's referral source. ▪ Knows patient medical history ▪ Issues hospital card.	▪ 100 Refer Patient ▪ 120 Make Appointment ...
Patient Statement (Bill)	▪ Knows patient open (unpaid) billing items. ▪ Knows past items in bill	▪ 160 Print Patient Statement ▪ 190 Resolve Problem

Use Cases

Candidates List

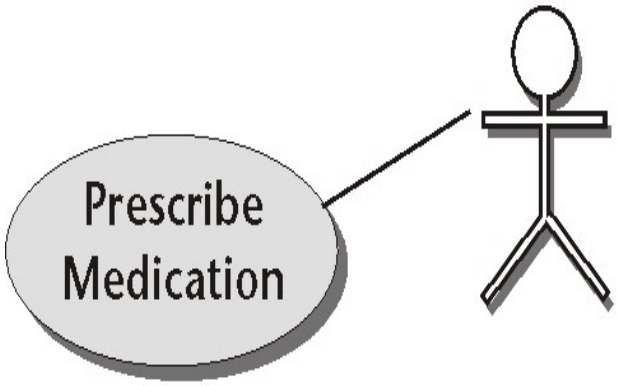
Finding Class Candidates

Make Appointment	Normal Flow:	
		1. Appointment clerk verifies that the needed medical service is provided by the hospital. **2.** Appointment clerk records patient's personal and contact data. **3.** Appointment clerk records information about the referral source. **4.** Appointment clerk consults hospital's schedule to find a free slot for the required medical service. **5.** Appointment clerk verifies that the patient is available for the appointment. ➔ Loop 1: **Repeat** steps 4-5 until hospital's schedule matches patient's availability. **6.** Appointment clerk makes the appointment. ➔ Loop 2: **Repeat** steps 4-6 for each appointment.

Actor vs. Class

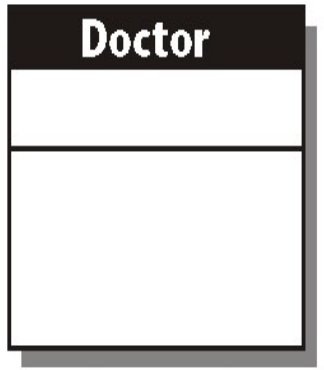
An entity can be both
But the two functions are not the same

Actor



As actor, doctor
is an entity outside
the system.

Class



As employee, doctor is
an entity inside
the system.

Patient Management: Preliminary Class Candidates

Class	Responsibilities	Use Cases
Appointment	▪ Make appointment for patient. ▪ Know appointment. ▪ Cancel appointment. ▪ Know referral source. ▪ Track appointment.	▪ 100: Refer Patient ▪ 120: Make Appointment ▪ 130: Receive Patient
Credit Card	▪ Knows patient credit card. ▪ Verifies patient credit card.	▪ 142: Verify Credit Card ▪ 190: Resolve Patient Billing Issue
Health Insurance Plan	▪ Knows patient's health insurance plan. ▪ Verifies patient's health insurance plan.	▪ 140: Register Patient ▪ 145: Verify Insurance Plan
Hospital ID Card	▪ Issues hospital card.	▪ 130: Receive Patient ▪ 140: Register Patient ▪ 144: Issue Hospital Card
Medical Service	▪ Knows medical service.	▪ 100: Refer Patient …
Patient	▪ Knows patient's personal data. ▪ Knows patient's contact data. ▪ Knows patient's appointments. ▪ Knows patient's insurance data. ▪ Knows patient's financial data. ▪ Knows patient's referral source. ▪ Knows patient next of kin. ▪ Issues hospital card.	▪ 100: Refer Patient ▪ 120: Make Appointment …
Patient Statement (Bill)	▪ Knows patient open (unpaid) billing items. ▪ Knows paid items from the last statement. ▪ Prints billing statement.	▪ 160: Print Patient Statement ▪ 190: Resolve Patient Billing Issue
Payment	▪ Knows the payment credited to patient's account.	▪ 190: Resolve Patient Billing Issue
Referral Source	▪ Knows the referral source.	▪ 120: Make Appointment
Hospital Schedule	▪ Knows appointments. ▪ Knows openings for medical services.	▪ 120: Make Appointment

Elaborating Classes

- To fully define a class or an object is to define responsibilities in detail.
- Patient class has a number of obligations:
  - Knows patient's personal data.
  - Knows patient's contact data.
  - Knows patient's appointments.
  - Knows patient's insurance data.
  - Knows patient's financial data.
  - Knows patient's referral source.
  - Knows patient's medical history.
  - Knows patient's next of kin.
  - Issues hospital card.

Delegating Responsibilities

- An object can delegate responsibilities to objects that collaborate with it.

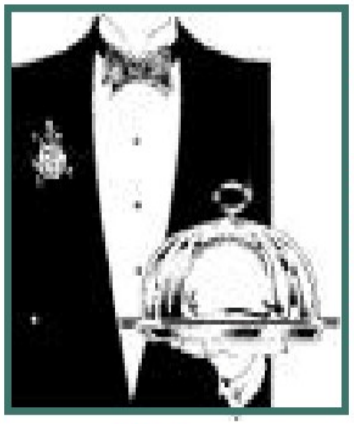
- “If the type [of an attribute] has a complex data structure, we often have to make a new class of the attribute type … ”

Delegation

Assigning the Actual Work to Another Object

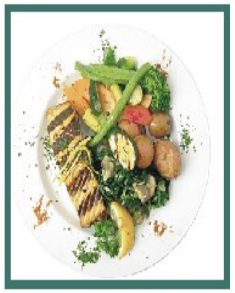
Assistant Waiter

EXPOSED



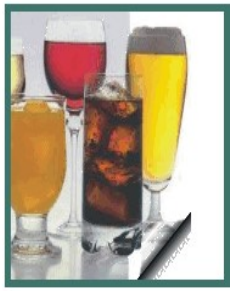
HIDDEN

Waiter



Cook

Pastry Chef



Barman

Collaboration

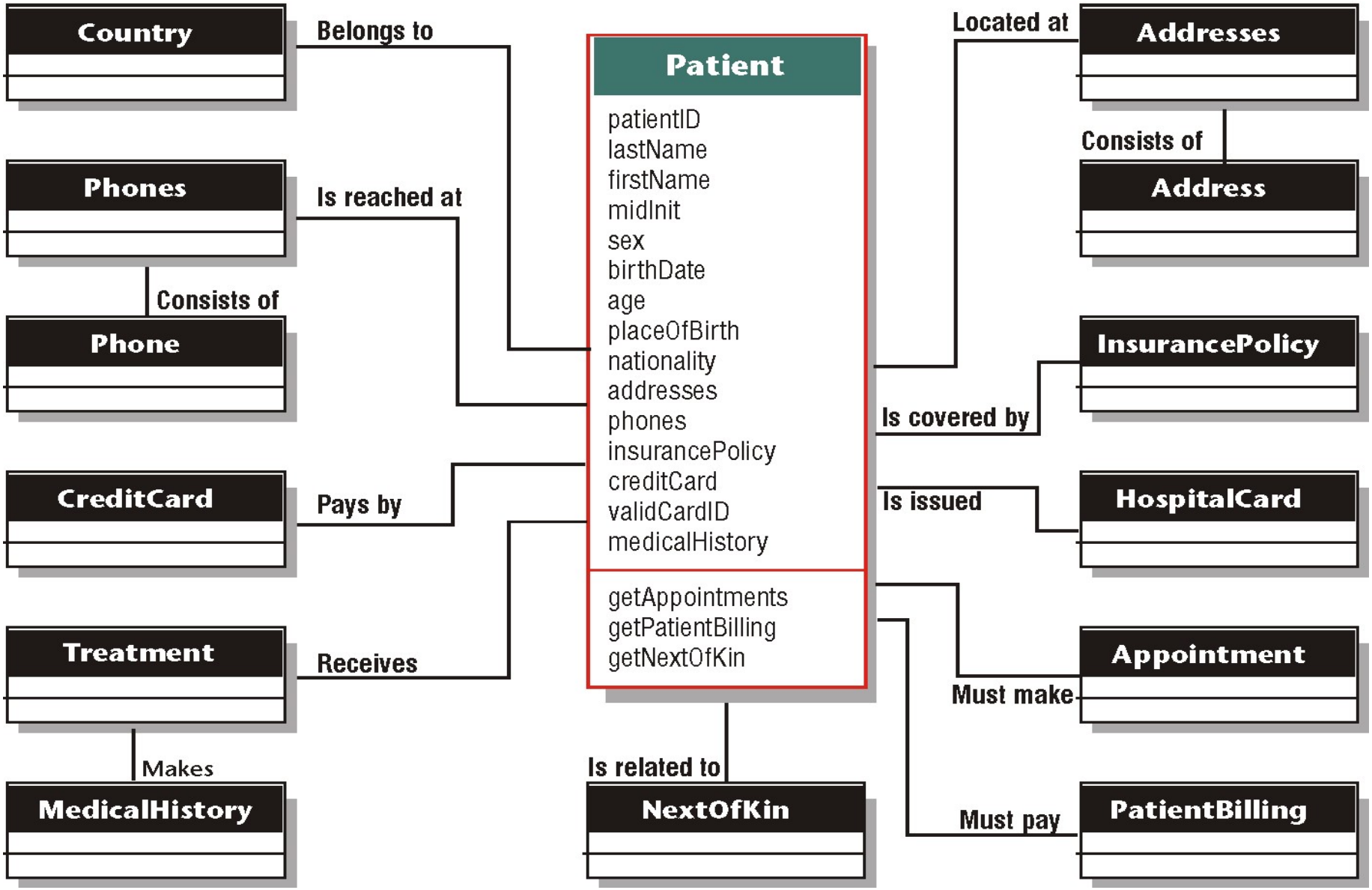
Attribute	Collaborators
`patientID`	
`lastName`	
`firstName`	
`midInit`	
`sex`	
`birthDate`	
`age`	
`placeOfBirth`	
`nationality`	▪ Country
`addresses`	▪ Address ▪ Addresses
`phones`	▪ Phone ▪ Phones
`insurancePolicy`	▪ InsurancePolicy
`creditCard`	▪ CreditCard
`validCardID`	▪ HospitalCard
`medicalHistory`	▪ Treatment ▪ MedicalHistory

Operations

Operation	Collaborators
`getAppointments`	▪ Appointment
`getPatientBilling`	▪ PatientBilling
`getNextOfKin`	▪ NextOfKin

The Conceptual Patient

Analyze Responsibilities to Discover Collaborators



Relationships

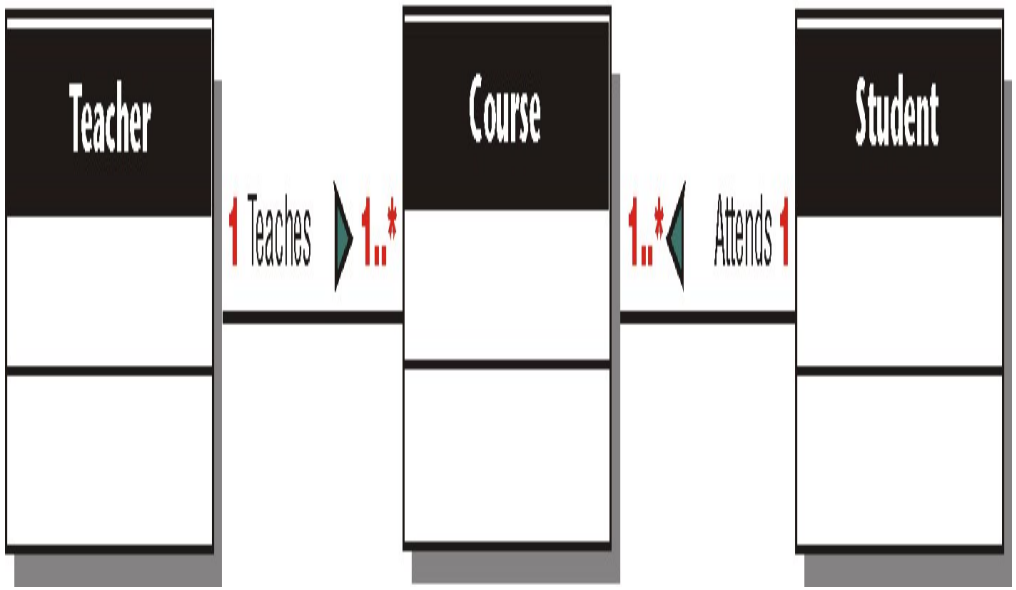
- Association
- Aggregation
- Composition

Association

- Association is a structural relationship that defines the link between objects of one class with the objects of another class

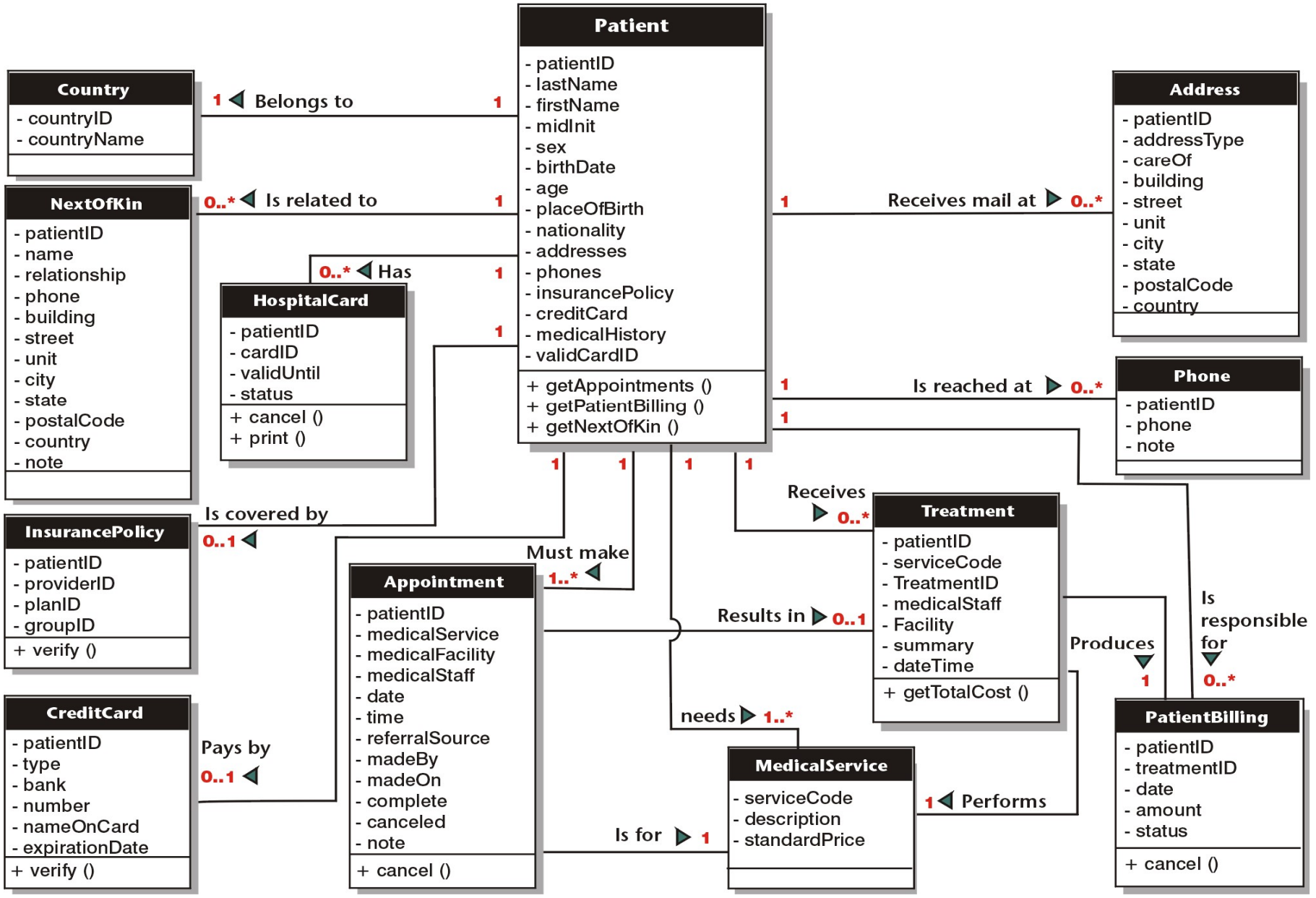
Class Diagram

- Class diagram shows a set of classes and their interrelationships.



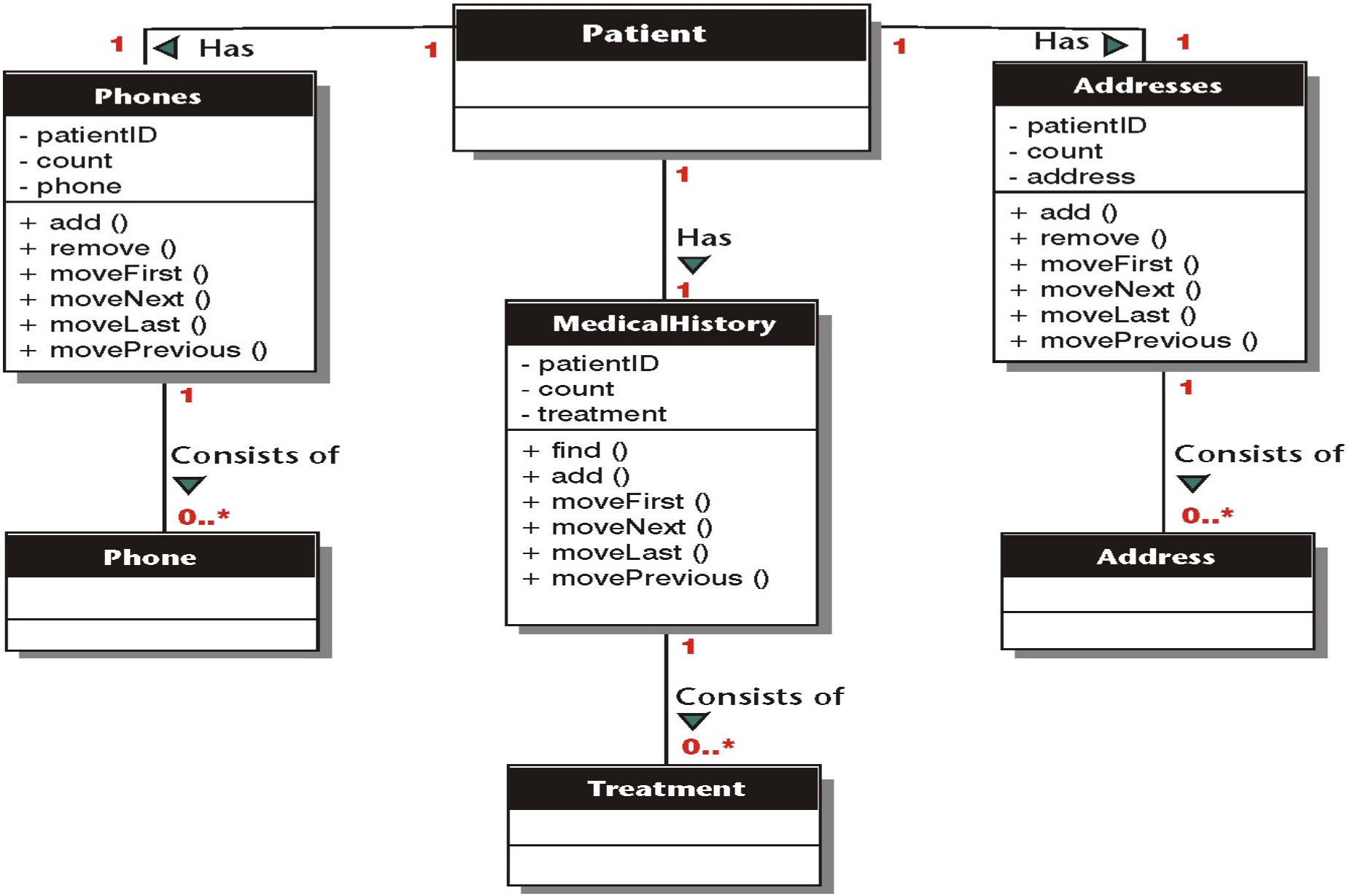
Patient & Its Associations

A Diagram Must Have a Point



Patient & Its Collection Attributes

Different Message, Different Model



Multiplicity

- Multiplicity specifies how many instances of one class can associate with instances of another class.

Multiplicity

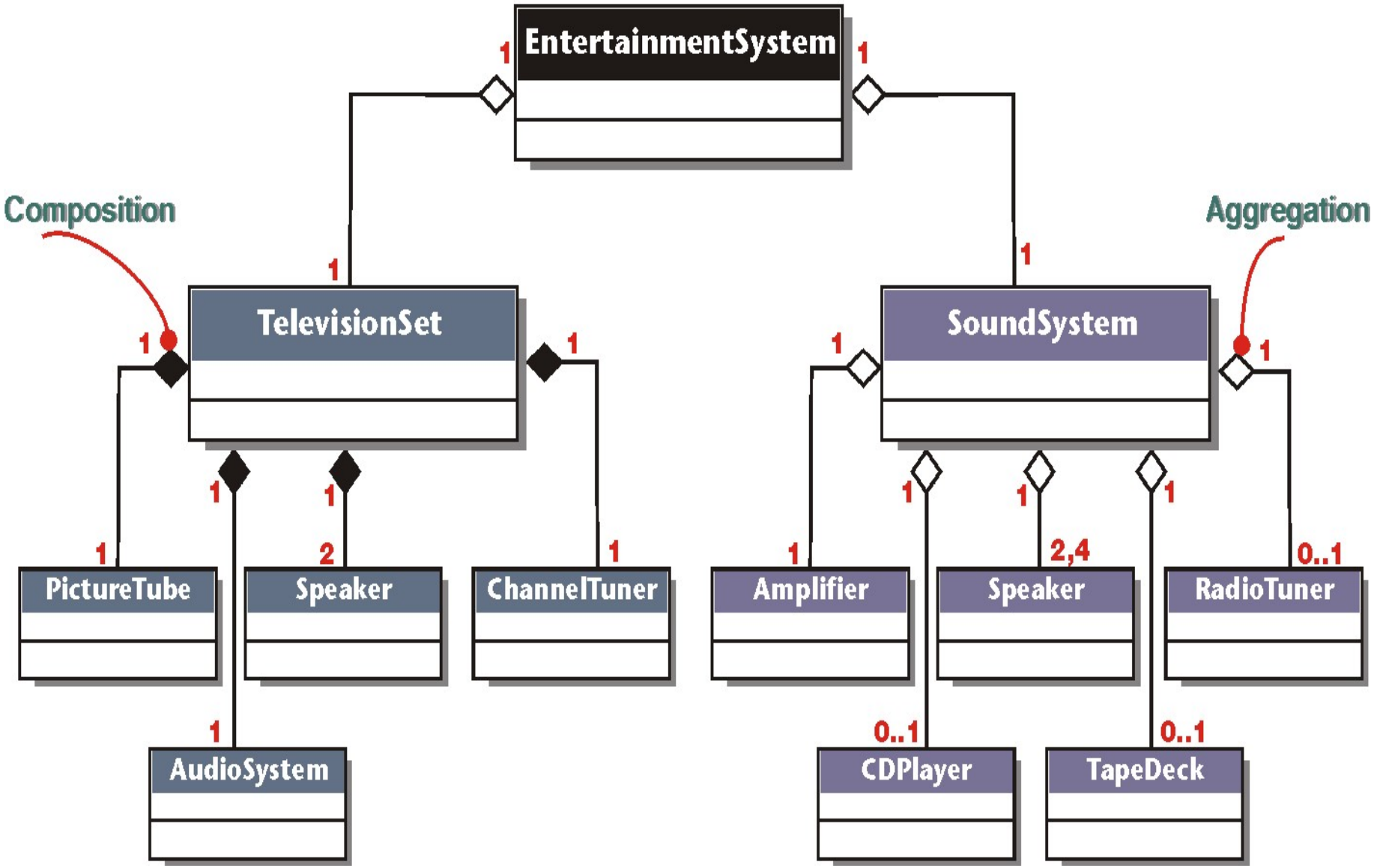
Multiplicity	Meaning	Example
1	Exactly one	A patient must have one, and only one, nationality.
0..1	Zero or one	A patient can have no insurance plan or can have one.
1..*	One or more	A patient must have at least one appointment to receive medical service, but can have as many as necessary.
0..*	Zero or more	A patient can have no billing activity or many.
20..40	A defined range	A part-time worker must work at least twenty hours a week, but no more than forty.
2,4, 6, 8	A non-continuant range	Tables are set for 2, 4, 6, or 8 people.

Aggregation and Composition

- Aggregation represents the relationship of a whole to a part. Composition is a form of aggregation in which the part is exclusively owned by the whole and its lifecycle is dependent on the lifecycle of the whole.

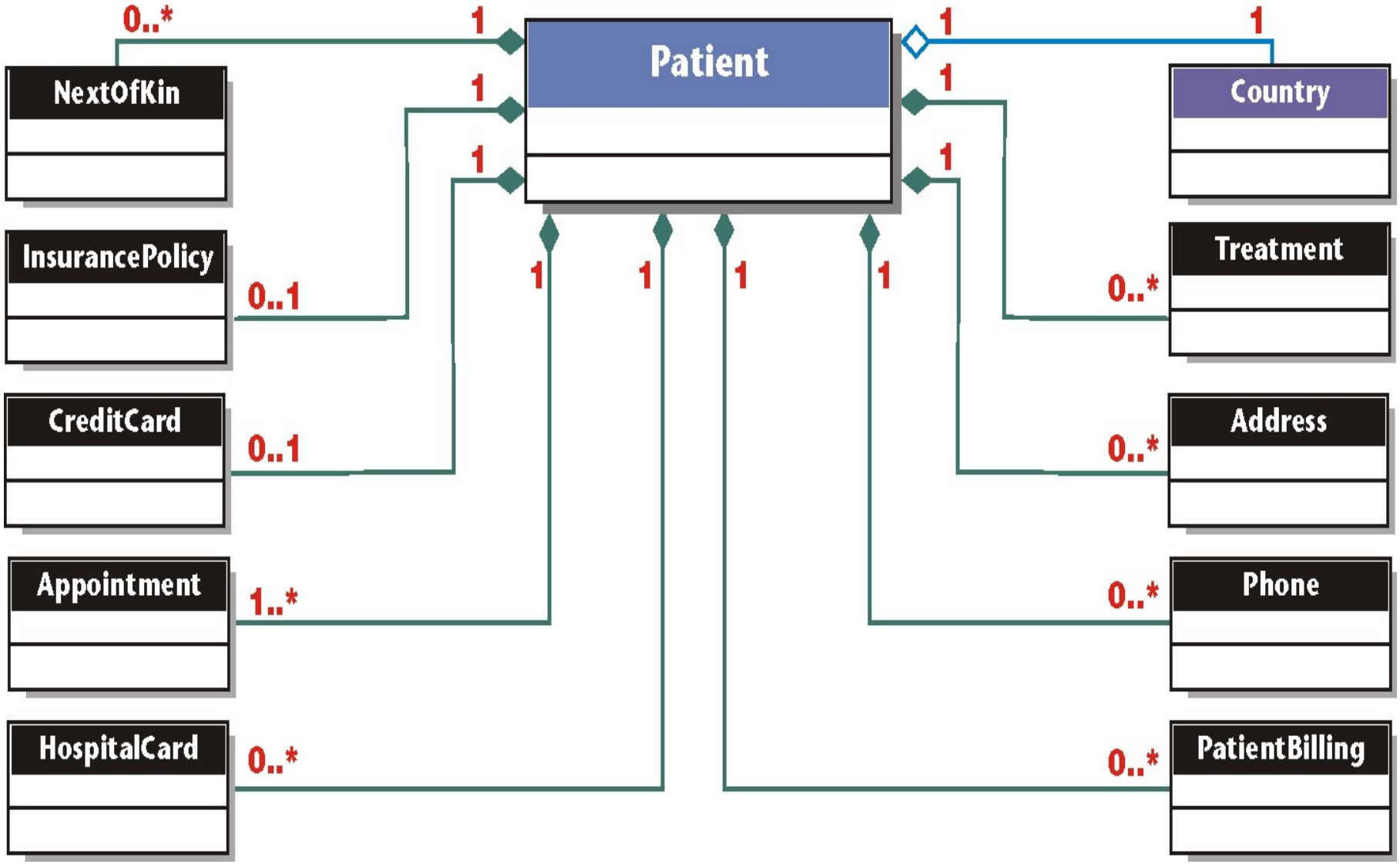
Aggregation & Composition

How Parts Relate to the Whole



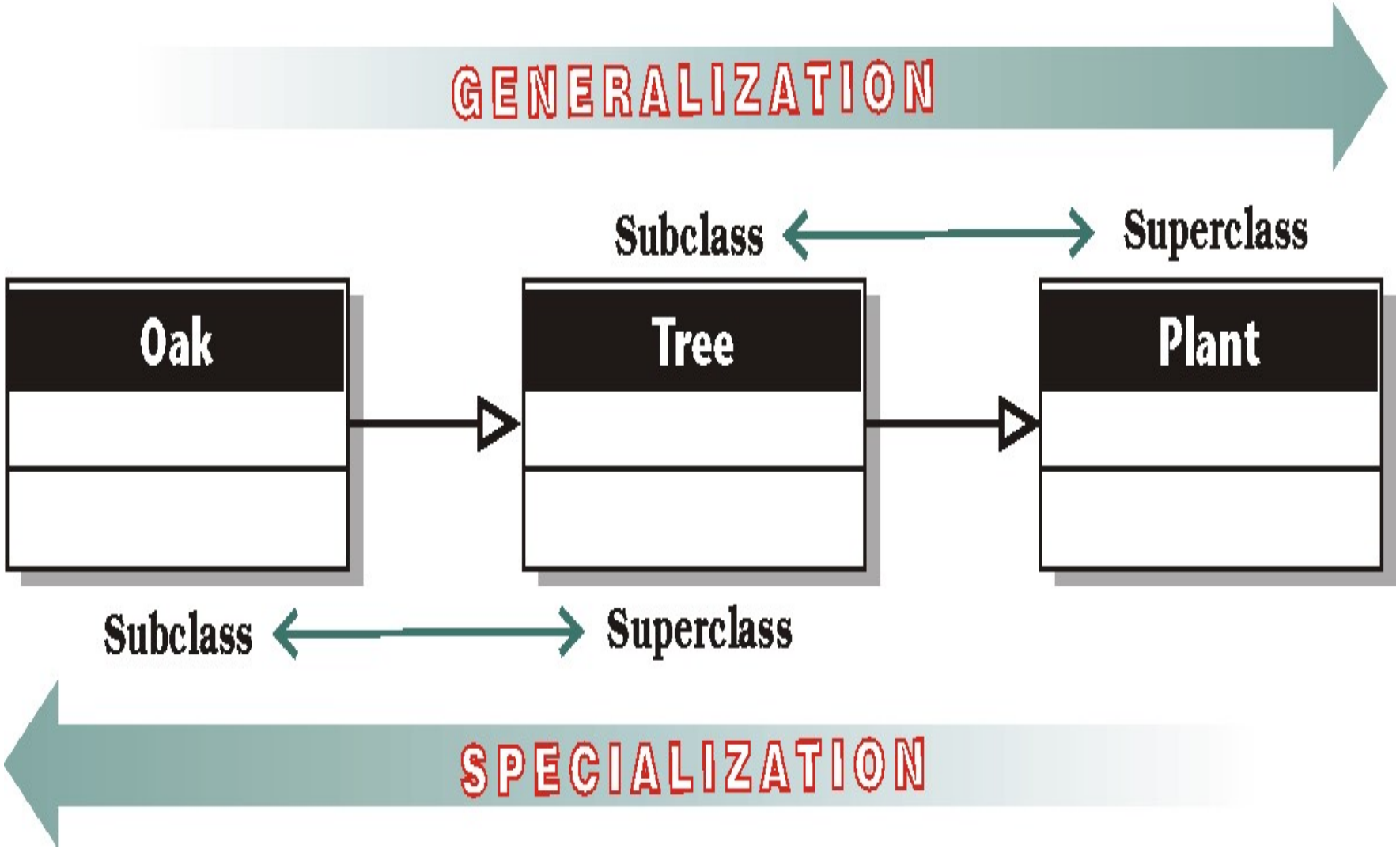
Patient as an Aggregation

Streamlining Complexity



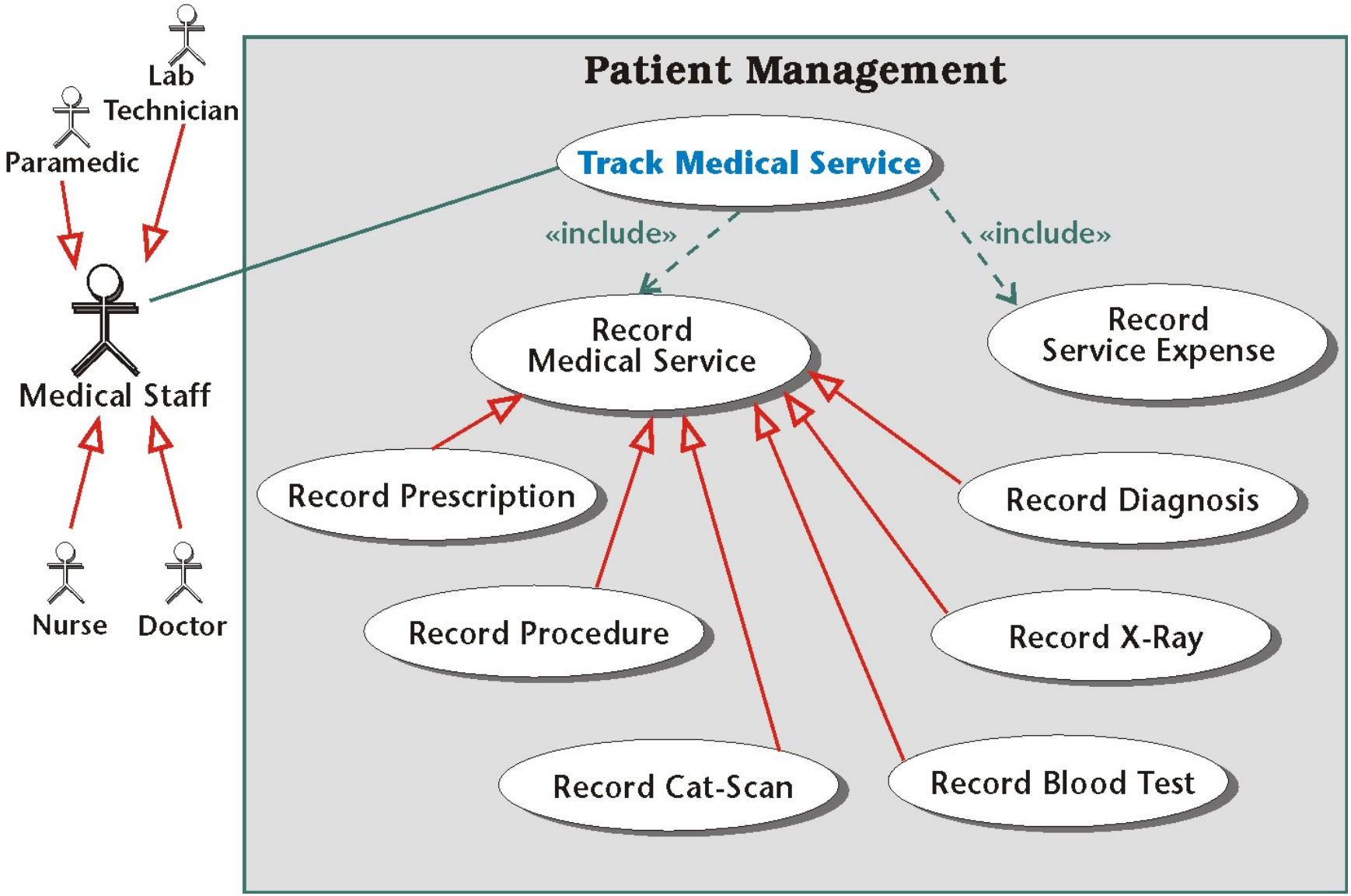
Generalization & Specialization

More Abstract vs. More Concrete



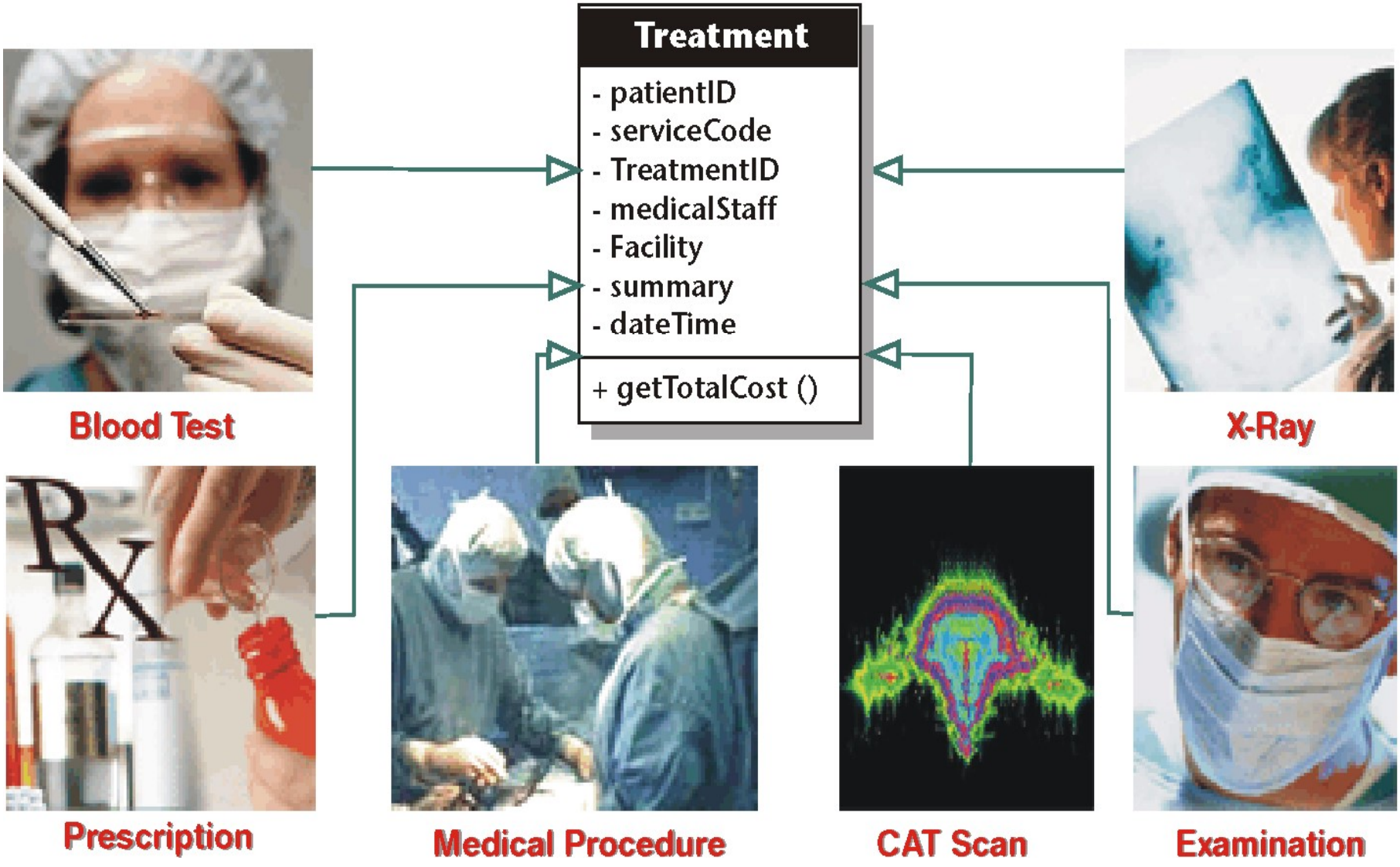
Track Medical Service

A Source for Class Generalization



Generalization of Treatment

Without Losing Specialization



OutdoorPowerEquipmentDepot – Case Study

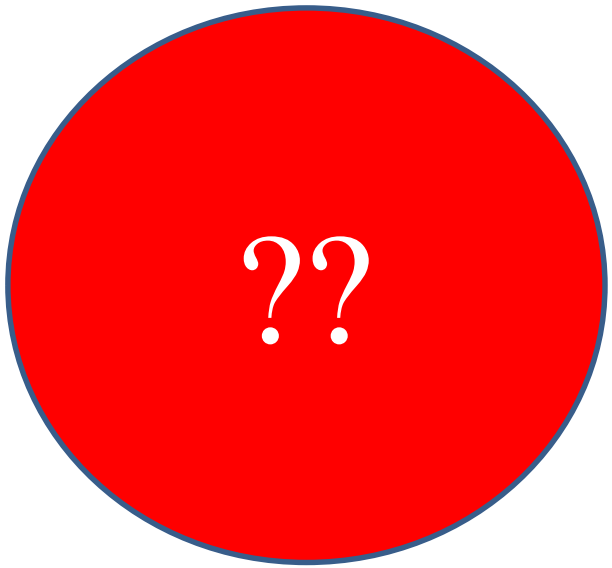
- **After we review the requirements for the OutdoorPowerEquipmentDepot – Case Study we need to:**

  1. **List the classes that are needed to create the domain/analysis class diagram**
  2. **List the attributes for every class identified to be used in the domain/analysis class diagram**
  3. **List the relationships between the classes that are needed to create the domain/analysis class diagram**
  4. **Use Visio to create the domain/analysis class diagram**

- **Lets start with the first one …**

OutdoorPowerEquipmentDepot – Case Study

- **List the classes that are needed to create the domain/analysis class diagram**



OutdoorPowerEquipmentDepot – Case Study

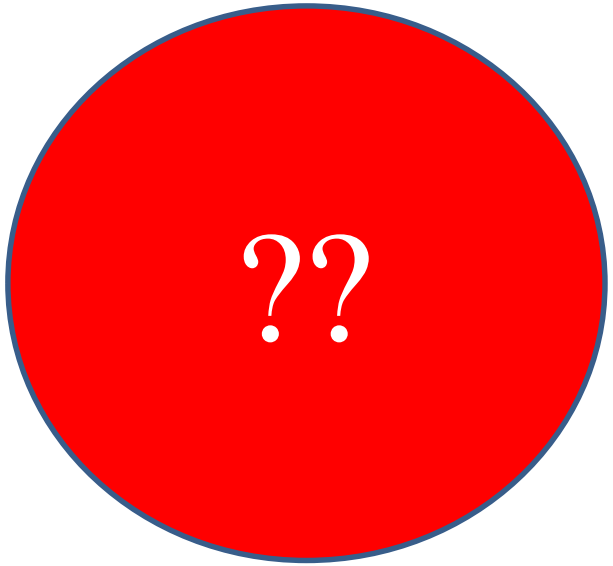
- **List of classes that are needed to the domain/analysis class diagram**

- Customer
  - PowerMemebership
- Salesman (Account Specialist)
- StoreManager
- Technician
- Delivery Driver
- Order
  - Product
  - Service
- Products
  - Buy
    - No ReplacementPlan
    - Year ReplacementPlan
    - Liftime ReplacementPlan
  - Rental
    - Daily
    - Mondtly
    - Yearly
  - Trade-in

- Service
  - Delivery
  - Repair
- Payment
- Warranty
- Discounts
  - Special Discount
  - Storage Space limitation promotional offer
  - Restocking
- Manufacturer Rebate
- What else??

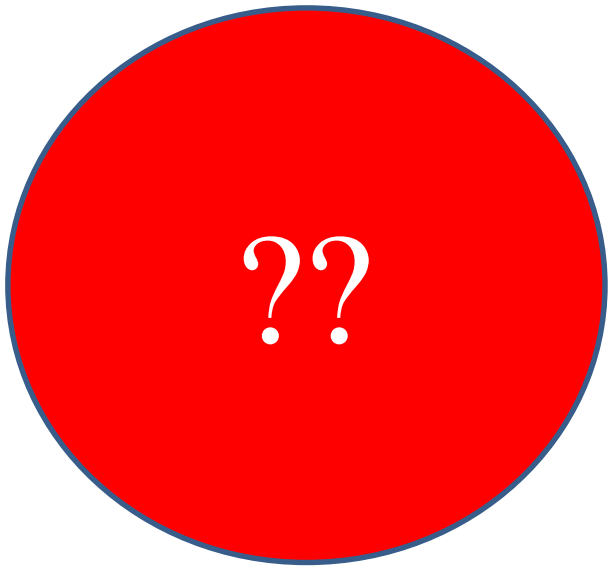
OutdoorPowerEquipmentDepot – Case Study

- **List the attributes for every class identified to be used in the domain/analysis class diagram**



OutdoorPowerEquipmentDepot – Case Study

- **List the relationships between the classes that are needed to create the domain/analysis class diagram**



OutdoorPowerEquipmentDepot – Case Study

- **Create the domain/analysis class diagram**