

Object-Oriented Analysis and Design

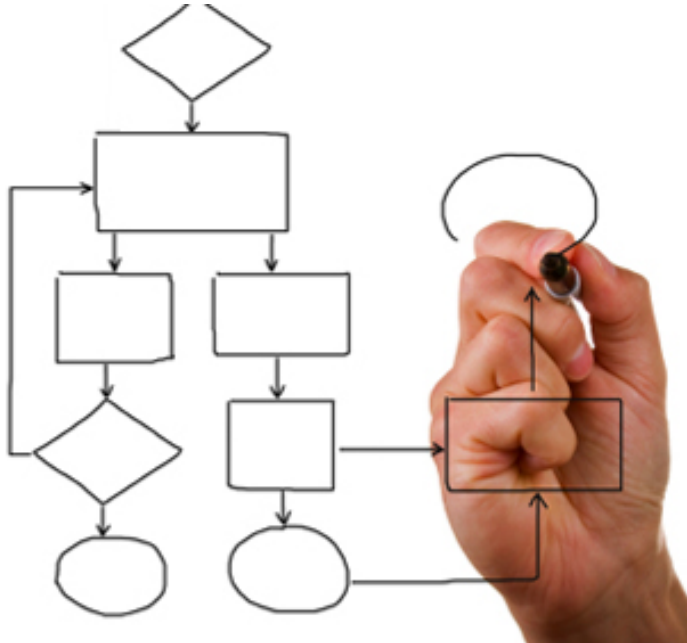
Object-Oriented Analysis and Design

- What is the Object-Oriented Technology?
- What is Object-Oriented Programming?
- What is Object-Oriented Analysis and Design?
- What is Requirement Elicitation?
- What is Object-Oriented Modeling?
- What is UML?
- What is the Software Development Process?
- What is the Unified Process?
- What is Code Reuse?
- What is Design Reuse?

Requirements



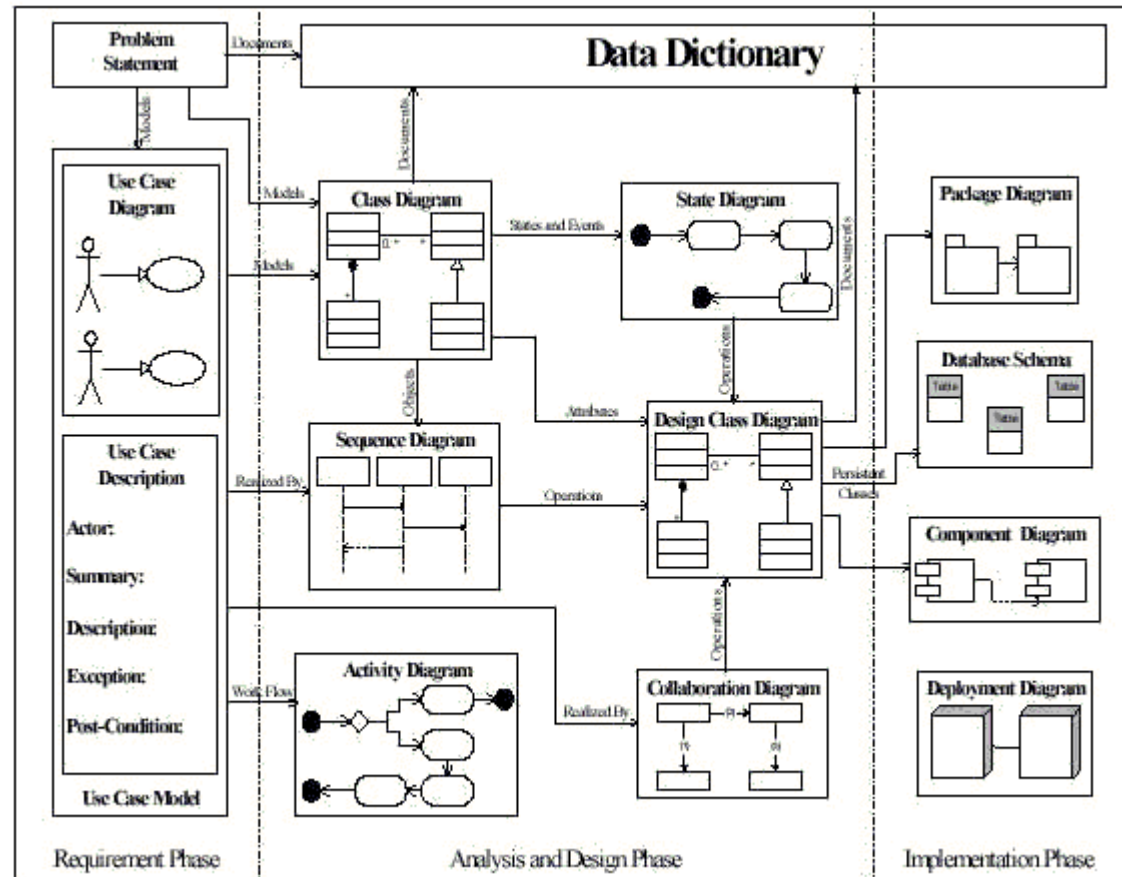
Analysis



Design



Object-Oriented Analysis and Design

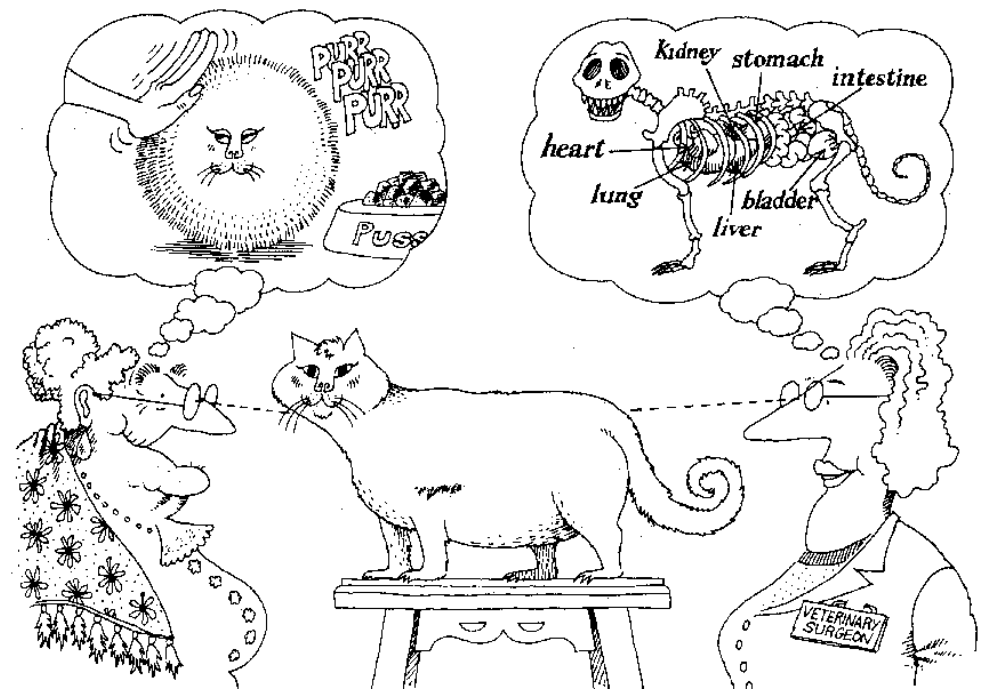


Principles of Object Orientation

- Abstraction
- Encapsulation
- Modularity
- Hierarchy

What Is Abstraction?

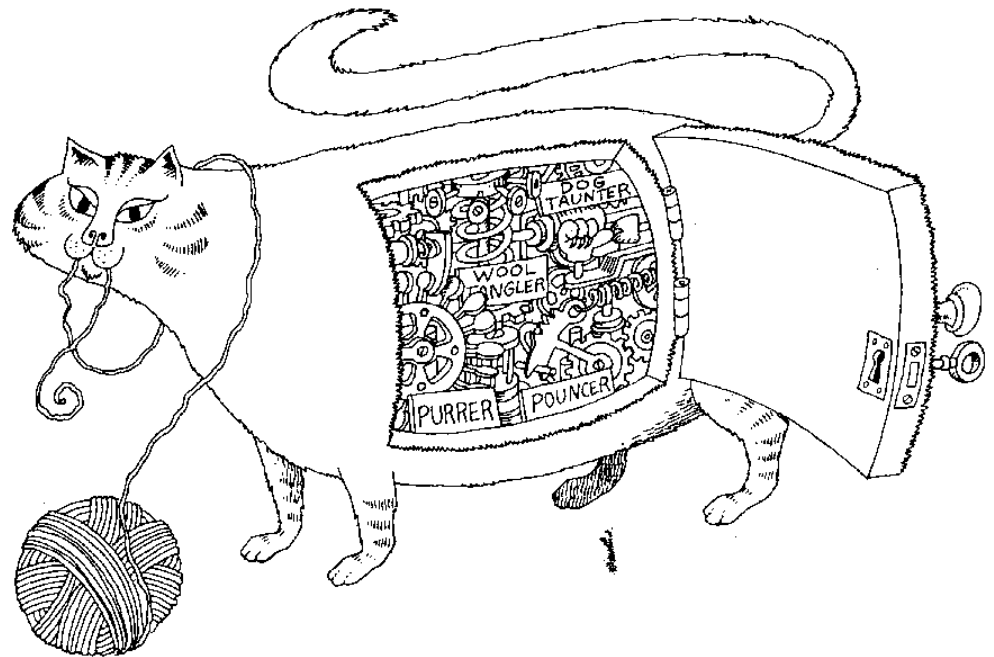
- The essential characteristics of an entity that distinguishes it from all other kinds of entities
- Depends on the perspective of the viewer
- Is not a concrete manifestation, denotes the ideal essence of something



From *Object-Oriented Analysis and Design with Applications* by Grady Booch, 1994

What Is Encapsulation?

- Hides implementation from clients
 - Clients depend on interface
 - Improves the resiliency of the system, i.e. its ability to adapt to change



From *Object-Oriented Analysis and Design with Applications* by Grady Booch , 1994

What Is Modularity?

- Breaks up something complex into manageable pieces.
- Helps people understand complex systems

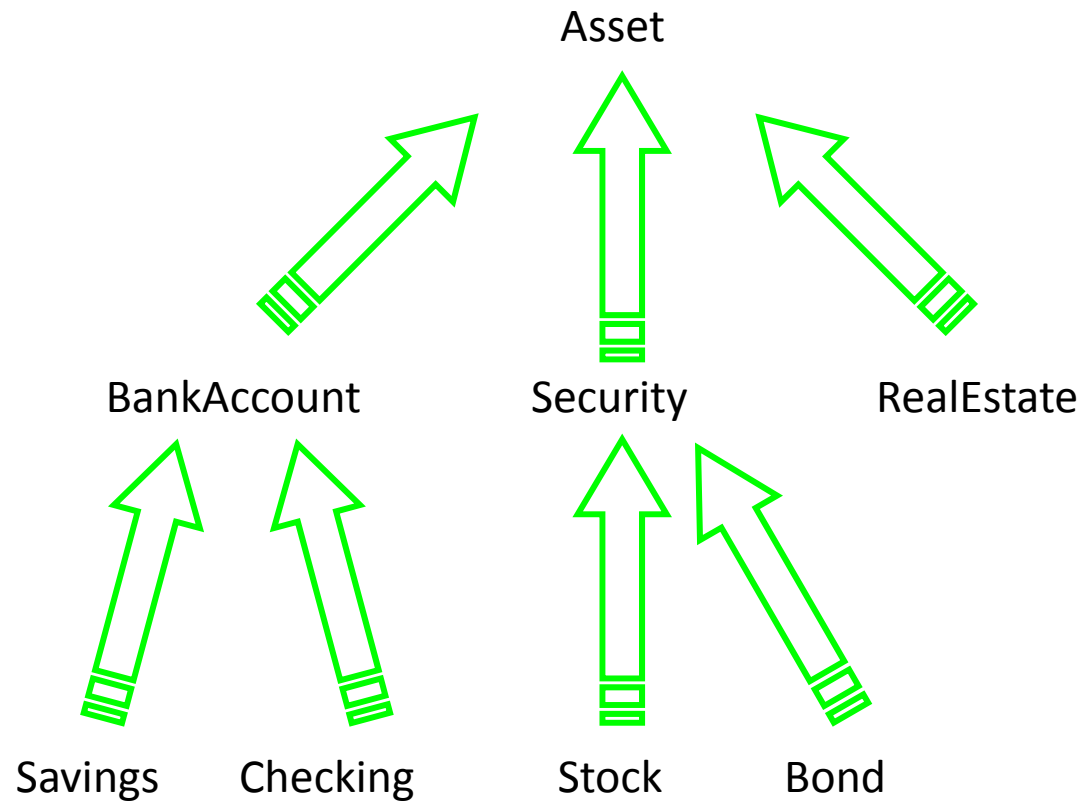


From *Object-Oriented Analysis and Design with Applications* by Grady Booch , 1994

What Is Hierarchy?

Increasing
abstraction

Decreasing
abstraction



Elements at the same level of the hierarchy should be at the same level of abstraction

The Concept of Object Orientation

Object: John Doe

Attributes

- **Name:** John Doe
- **Born:** 1972
- **Sex:** Male
- **Marital Status:** Single
- **Citizenship:** USA
- **Degree:** MBA
- **Current Position:**
VP of Purchasing
- **Years with Company:** 5
- **Starting Salary:**
\$70,000
- **Current Salary:** ...



*Attributes & Operations
As Perceived
By His Employer*

Operations

- Order
- Approve Order
- Get Bids
- Evaluate Bids
- Report to President
- Report to Board
- Notify Inventory
- Hire
- Fire
- Evaluate Employees
- Approve Vacations
- ...

Operations

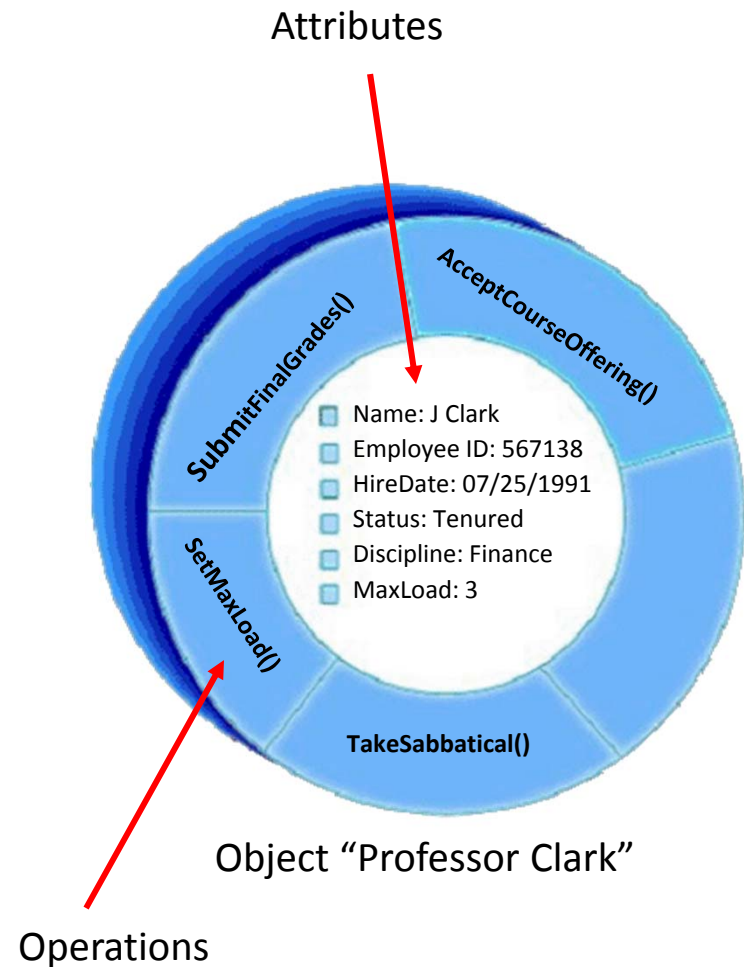
- An operation is what an object does or is capable of doing.
 - If an object is the *subject* of a sentence with an *active* voice, then the **verb** expresses an operation:
 - Dog barks
 - Ball bounces
 - Sun shines

State of an Object

- State is the condition of an object at a certain stage in its lifetime.
- An object has a set of attributes and these attributes accept a range of values. The combination of these attributes and their associated values constitute the state of an object.

A More Formal Definition

- An object is an entity with a well-defined boundary and *identity* that encapsulates *state* and *behavior*
 - State is represented by attributes and relationships
 - Behavior is represented by operations, methods, and state machines



Class

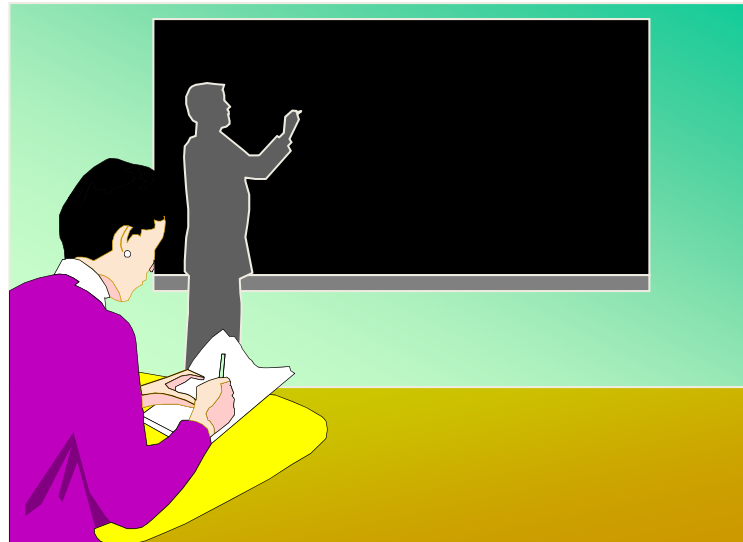
- Class is a set of objects that share the same attributes and operations.

A Sample Class

Class
Course

Attributes

Name
Location
Days offered
Credit hours
Start time
End time



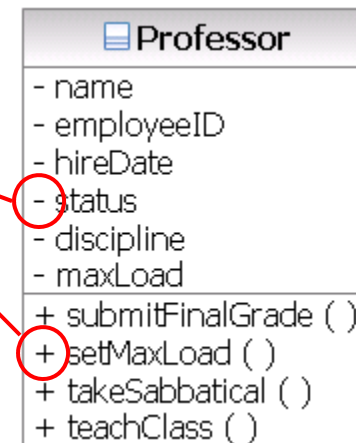
Behavior

Add a student
Delete a student
Get course roster
Determine if it is full

Representing Classes in UML

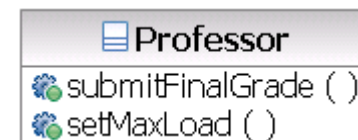
- A class is represented using a rectangle often with three compartments:
 - The class name
 - The structure (attributes)
 - The behavior (operations)

Visibility
+ = public
- = private



Representing Classes in UML (cont'd)

- Style Guidelines
 - Capitalize the first letter of class names
 - Begin attribute and operation names with a lowercase letter
- Compartments
 - Only the name compartment is mandatory
 - Additional compartments may be supplied to show other properties



Representation with the attribute compartment suppressed, operation visibility shown as “decoration”, and the operation compartment filtered to show only two operations

What Exactly Is a Class?

A *class* is a blueprint for an object.

- When you instantiate an object, you use a class as the basis for how the object is built.

What Exactly Is a Class?

An object cannot be instantiated without a class.

- Classes can be thought of as the templates, or cookie cutters, for objects as seen in the next figure.



Abstraction

- Abstraction is identifying those characteristics of an entity that distinguish it from other **kinds** of entities.
- The process of selection that separates certain attributes and operations from a concrete object is called abstraction.

Generalization

- To generalize is to conclude that characteristics of a particular entity apply to a broader range of entities.

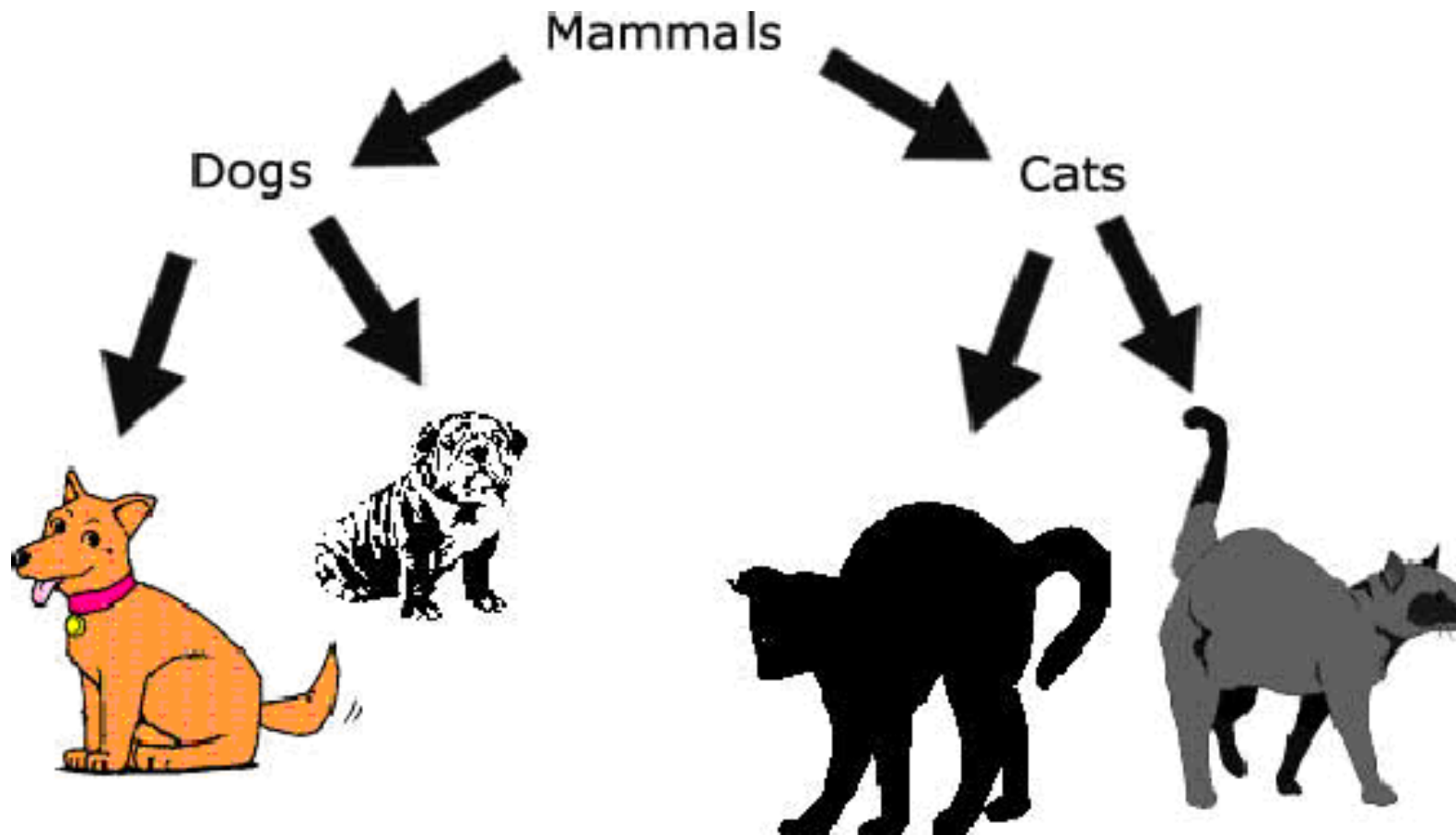
Instance

- An instance is the concrete manifestation of a class.
 - For example, **John Doe** is an “instance” of the class **Human**.

Superclass and Subclass

- A superclass results from *generalizing* a set of classes.
- A subclass results from *specializing* a superclass.
- The relationship among superclasses and subclasses is called class hierarchy.

Mammal Hierarchy



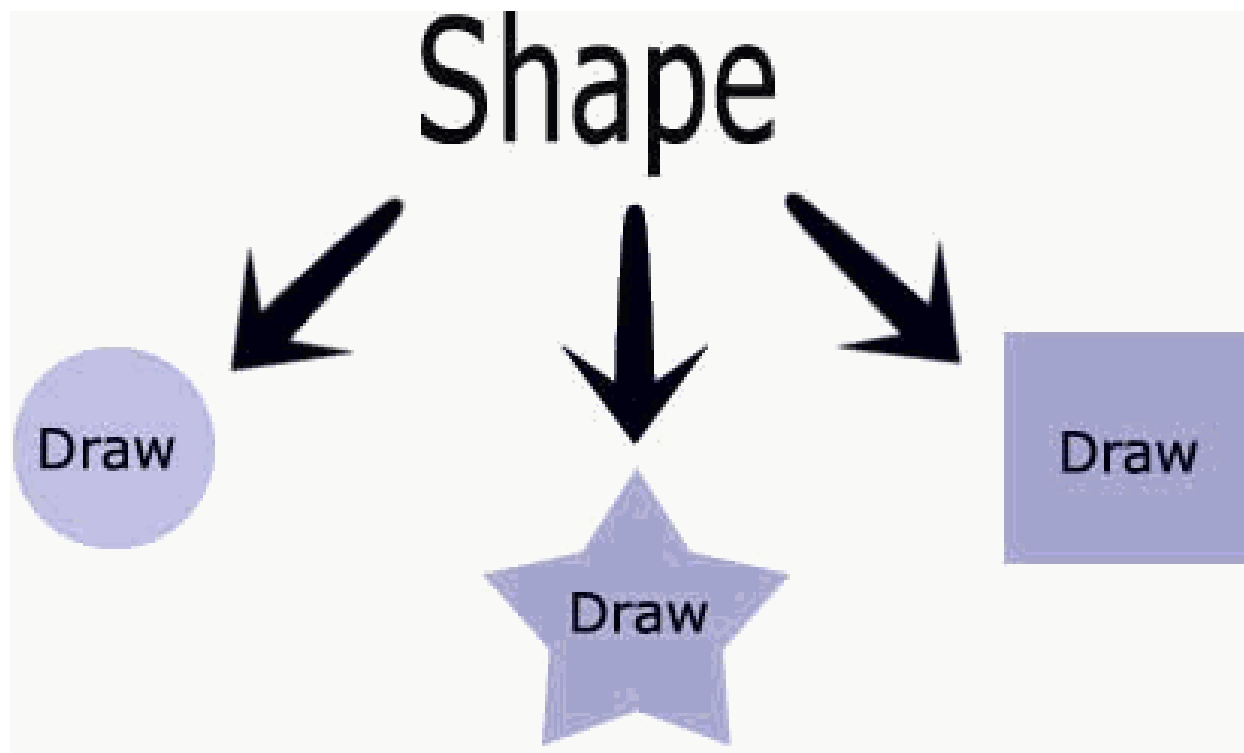
Is-a Relationships

In the Shape example, Circle, Square, and Star all inherit directly from Shape.

- This relationship is often referred to as an *is-a relationship* because a circle *is a* shape.

Polymorphism

Polymorphism literally means many shapes.



Polymorphism

The Shape object cannot draw a shape, it is too abstract (in fact, the Draw() method in Shape contains no implementation).

- You must specify a concrete shape.
- To do this, you provide the actual implementation in Circle.

Polymorphism

In short, each class is able to respond differently to the same Draw method and draw itself.

- This is what is meant by polymorphism.

Polymorphism

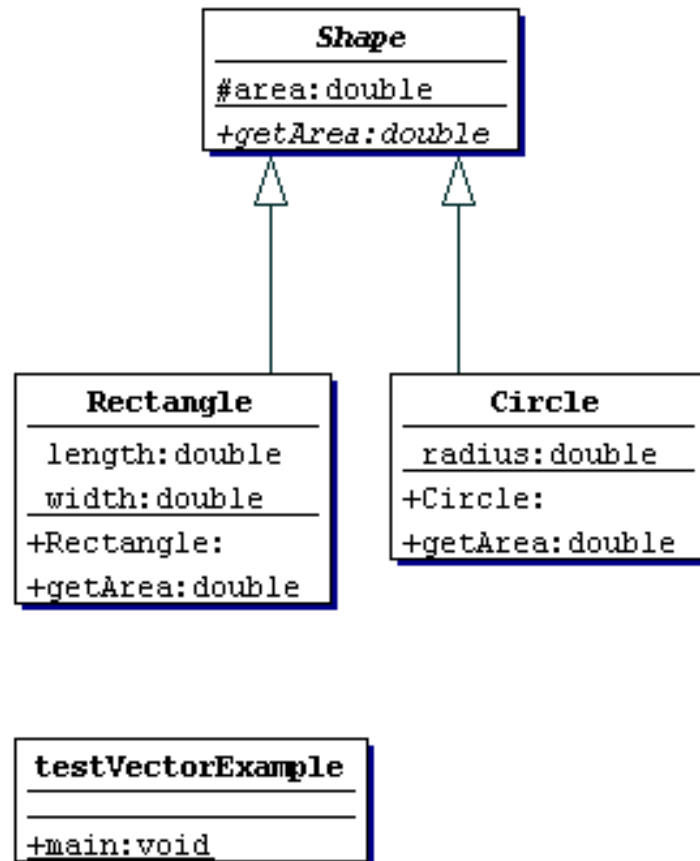
When a method is defined as abstract, a subclass must provide the implementation for this method.

- In this case, Shape is requiring subclasses to provide a getArea() implementation.

Polymorphism

- If a subclass inherits an abstract method from a superclass, it *must* provide a concrete implementation of that method, or else it will be an abstract class itself (see the following figure for a UML diagram).

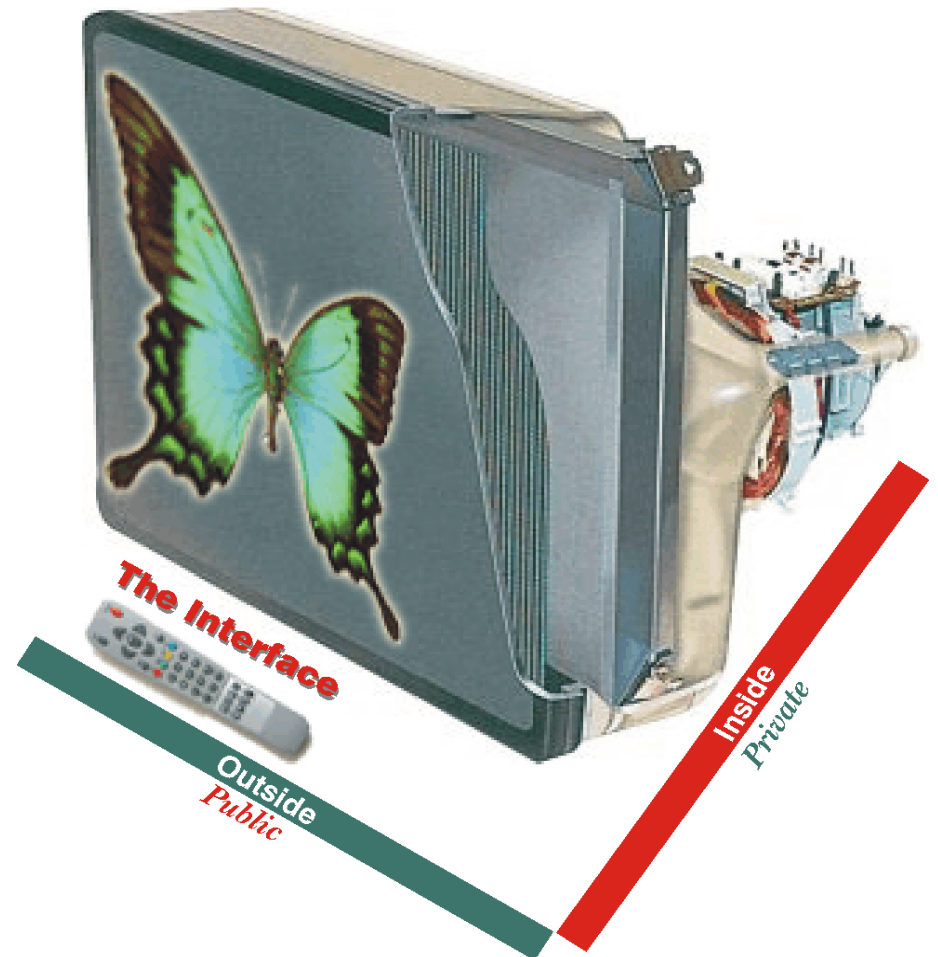
Shape UML Diagram



Encapsulation

- Encapsulation is the packaging of data and processes within one single unit.

Encapsulation
And Information Hiding
Object as “Black Box”



Information Hiding

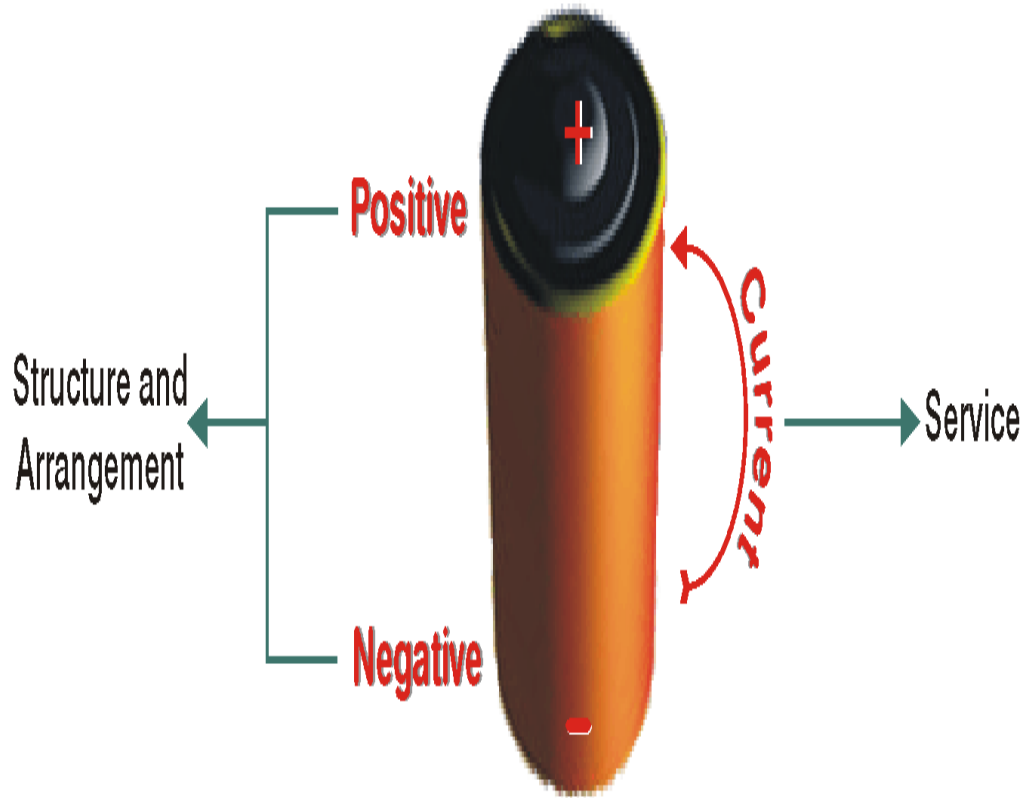
- Information hiding conceals and protects what goes on inside an object from the outside world.
- When you use an ATM, encapsulation and information hiding ensure that:
 - ❶ you are not burdened with the complexity of how the machine works,
 - ❷ cannot perform operations that you are not allowed to, and
 - ❸ cannot change the way the machine operates.

Interface

Interface

Form and Substance

- An object's interface consists of operations that are available to the public.

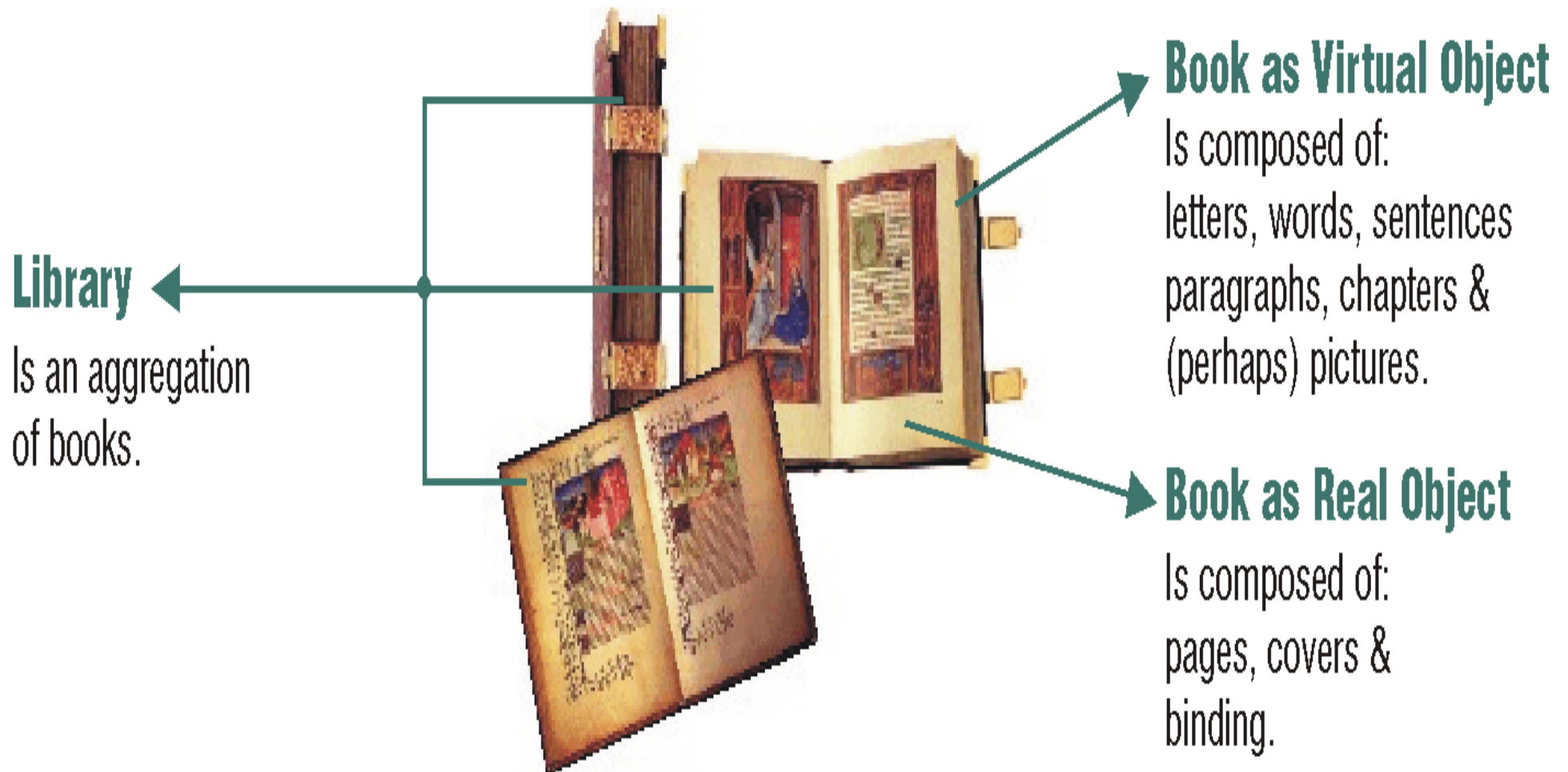


Aggregation and Composition

- In object-oriented terminology, the relationship of one object to its component objects is called **aggregation**.
- A strong form of aggregation in which the life of components relies on the life of the whole is called **composition**.

Aggregation and Composition

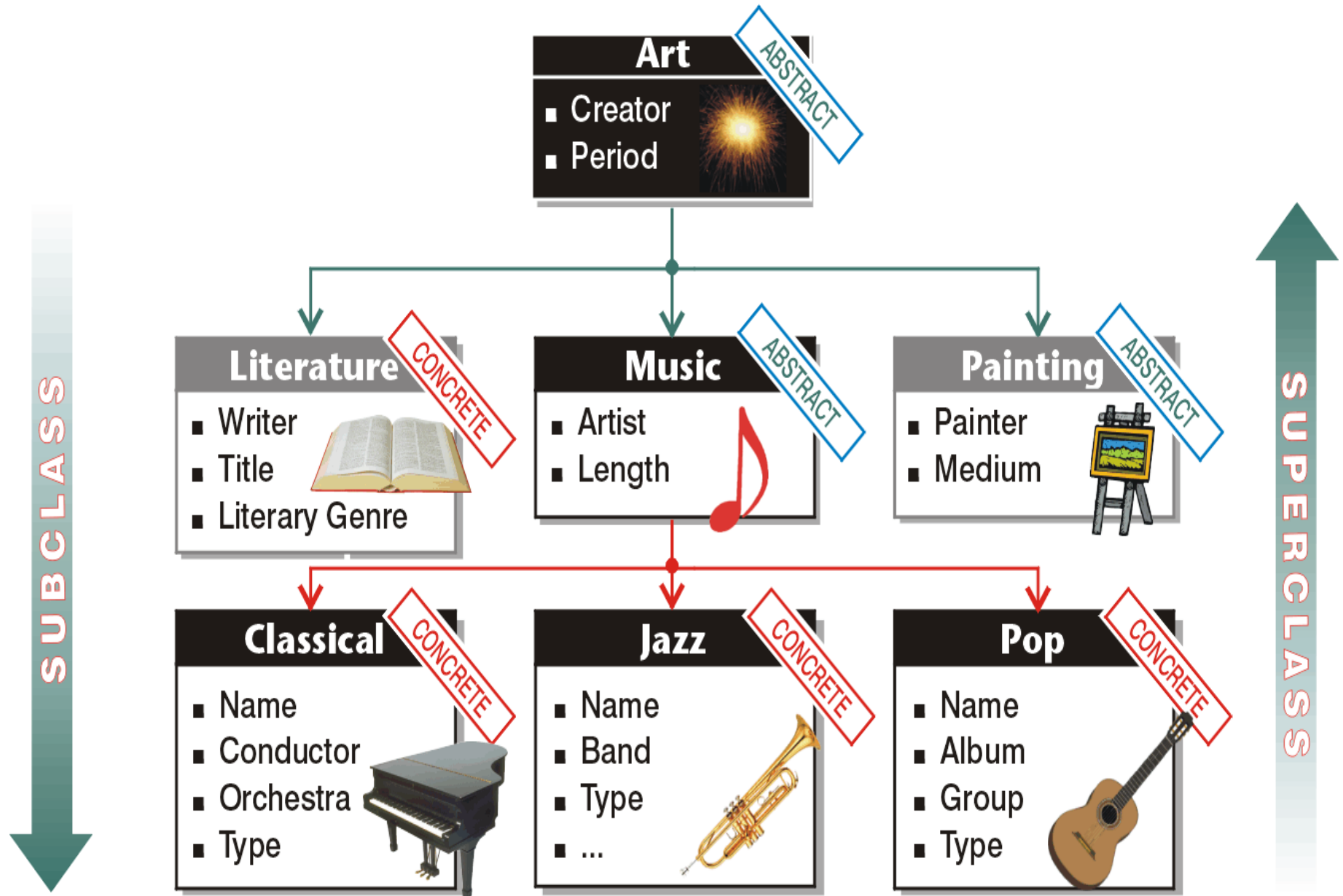
Aggregation & Composition



Abstract and Concrete Classes

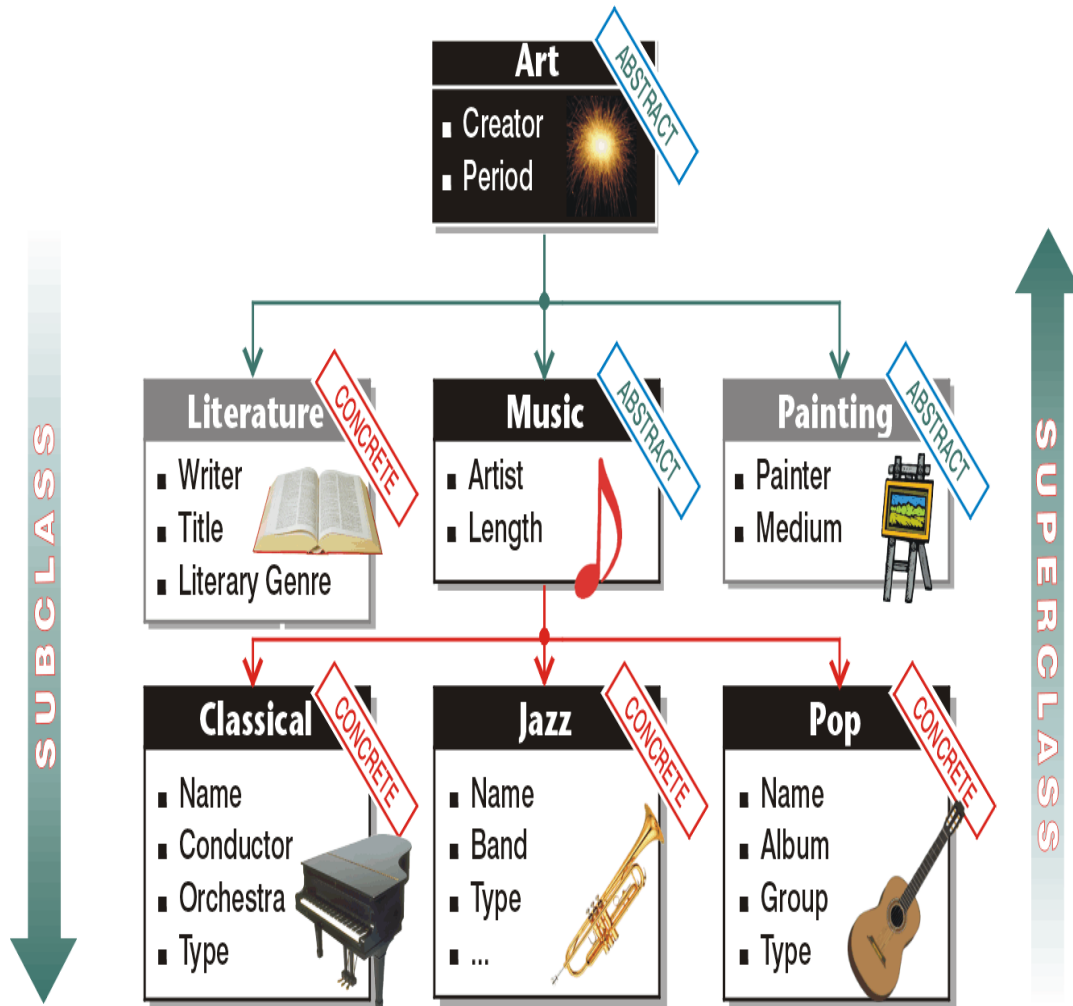
- Classes that can be instantiated into actual (real *or* virtual) objects are called **concrete** classes.
- Classes that cannot be instantiated into actual (real *or* virtual) objects are **abstract** classes.

Abstract & Concrete Classes



Inheritance

Abstract & Concrete Classes

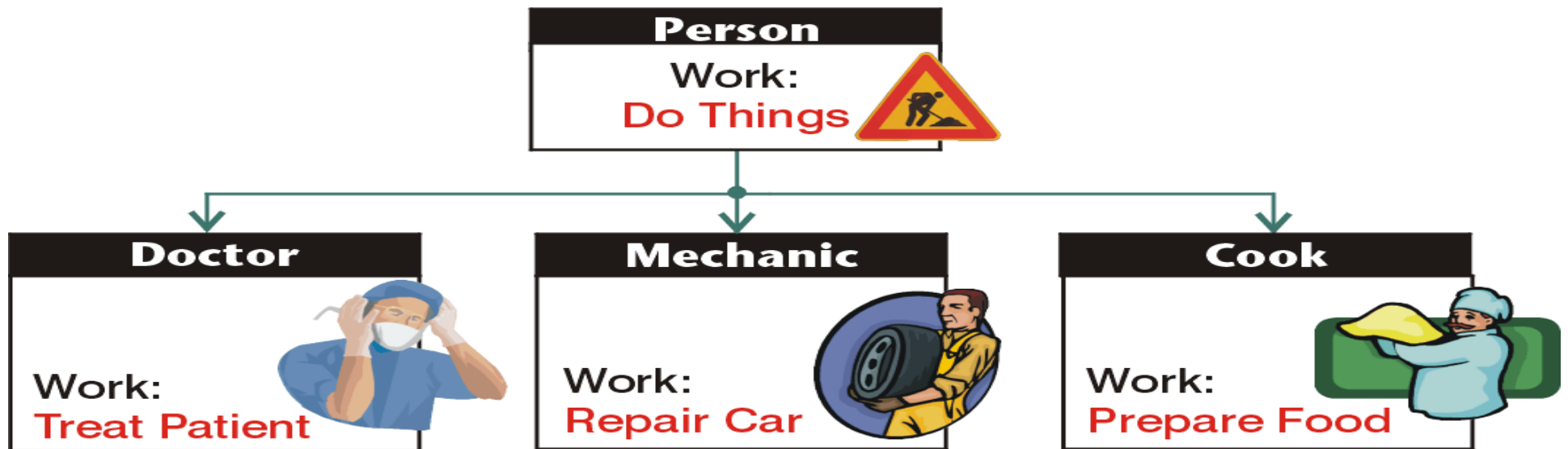


- Inheritance is the mechanism by which a subclass incorporates the behavior of a superclass.

Polymorphism

Polymorphism

Same Operation, Performed Differently



- Polymorphism is the ability of objects belonging to different classes to perform the same operation differently.

Object-Oriented Languages

- Simula
- Smalltalk
- C++
- PowerBuilder
- visual Basic
- Java
- .Net

Object-Oriented Modeling

- Object-oriented analysis and design is using an object-oriented approach to building conceptual and logical models of the system.

The Unified Modeling Language (UML)

- UML is a modeling *language* for object-oriented system analysis, design and deployment. UML is *not* a product, nor is it a process or a methodology.

The Unified Modeling Language

- The Unified Modeling Language is a language, that provides the “primitives” (or the basic elements) for building object-oriented **conceptual** (analysis) and **concrete** (design) models.

UML Supports Multiple Views of Same System

- **Owner's View**
 - what the owner (or business) wants, or the *conceptual view* of the system.
- **Architect's View**
 - how the architect conceives the solution, or the *logical view* of the system.
- **Builder's View**
 - the blueprints for building the product, or the *physical view* of the system.

UML Embodies Four Properties

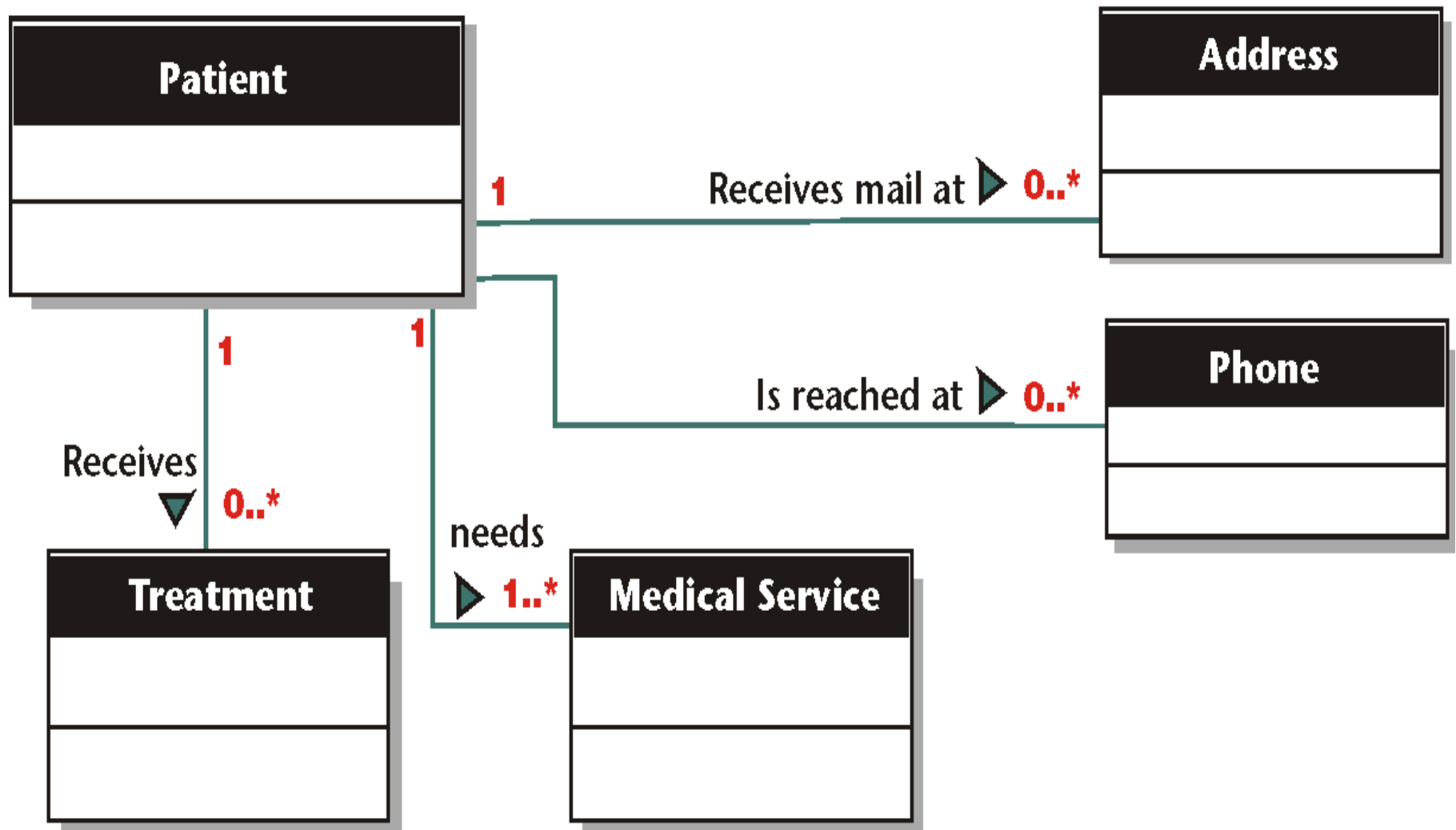
- ① Visualization
- ② Specification
- ③ Construction
- ④ Documentation

Visualization: UML Diagrams

- UML provides a set of graphical elements that are combined to form diagrams. Each diagram is a visual presentation or *view* of the system and satisfies one or *more* broad but overlapping types of modeling:
 - Behavioral
 - modeling represents the interaction of the system with the outside world.
 - Structural
 - modeling represents the components of the system and their interrelationships.
 - Dynamic
 - modeling represents how the components of the system interact with the outside world and with each other to satisfy the behavioral requirements of the system.

A UML Diagram

Modeling Object-Oriented Concepts



Specification, Construction, and Documentation

- Specification
 - UML provides precise and complete models for the three major activities of system development: analysis, design, and implementation.
- Construction
 - UML models are compatible with object-oriented languages.
- Documentation
 - UML modeling tracks major development activities throughout the system lifecycle.

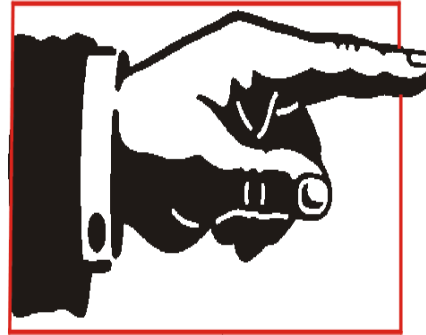
Methodology

Methodology

- Methodology is:
 - a set of methods, rules, practices, procedures, techniques and tools used to achieve a goal, or
 - the theoretical understating of the principles that determine how such methods, practices, tools, etc., are used.

Methodology

The “How” of Building Solutions



Methods

A methodology combines methods & techniques into a whole.



Tools

A methodology may require devices that help in accomplishing tasks.



Procedures

A methodology defines the path to the goal in a systematic manner.



Rules

Rules of methodology are abstractions of past experience or new ideas.



Methodology

A methodology may include other methodologies for various aspects of development.



“Philosophy”

A theoretical framework distinguishes a methodology from its components.

Methods Versus Methodology

- While “method” defines a **tactic**, “methodology” defines the **strategy**.
- Often, a tactic makes sense only in the context of strategy.

What Guides Our Actions?

- Cookbook
 - An ordered set of steps.
- Observation
 - We can learn something by observing others do it.
- Anecdotes
 - We might hear or read stories about how somebody has done something.
- Trial and Error
 - Lacking any reliable guidelines, we *imagine* some method that would work, and try it. If it does not work, we try another approach, if we can.
- Experience
 - We rely on the experience. If the results are not satisfactory, then we adjust the “experience.”
- Patterns
 - Patterns result from *collective* experience.
- Methodology
 - Methodology is both the most abstract and the most systematic guide to action. It evolves from the above but cannot be reduced to any of them.

Challenges of Methodology

- Methodology is needed not only in creating the **solution** but also in understanding the **problem**, organizing the production, assuring quality and the managing the consequences of the solution.
- As creators, we have to confront *three* elements: the **problem**, the **solution as method or methodology** (or “how”), and the **solution as answer** (or “what”).

The Problem Domain

- The problem that is to be solved, or the need that is to be satisfied exists in a **context**.
- This context is called the “**problem domain**” (or the “problem *space*”)
 - For example, a **headache** is a **symptom**, and needs to be treated within a context of the real problem.

The Solution Domain

- In building a solution, we also create a new and distinct **context** called the “solution domain” (or the “solution *space*”).
 - For example, most cars have fans to cool down the engine, even though the heat produced by the engine has no connection to transportation for which the car is built.

Software Development Methodologies

- Software development consists of a wide spectrum of activities that individual methodologies cover selectively and from different viewpoints.

Software Development Methodologies Address:

- Requirements Gathering
 - This activity determines the requirements that the product must address.
- Feasibility Study
 - Determines whether it is possible — technically, economically, legally or organizationally — to build a certain software.
- Domain Analysis
 - Discovers ❶ the meaning of requirements within the context, ❷ concepts within the domain that are related to the problem and can affect the solution, and possibly ❸ the consequences of the solution on the problem domain.
- Analysis
 - Analyzing the requirements to build a **conceptual** model of the solution (the product).
- Design
 - Transforms the “what” into “how.” Design itself consists of several distinct activities; **logical design**, **physical design**, and **architectural design**.

Software Development Methodologies Address:

- Implementation
 - Turns the blueprints of design into an actual product. Programming is usually the most important component of this activity, but it is not the only one.
- Testing and Quality Control
 - Verifies that the product functions according to specifications.
- Deployment and Training
 - This activity consists of ensuring the correct installation on the target platform, user training, creating help files and user manuals, setting up of Web sites to guide users, packaging, et cetera.
- Maintenance
 - Solving problems that may emerge after the deployment of the software, or changes in the environment.

The *Ad Hoc* Approach

- The **ad hoc** approach is development without an overall theoretical framework
 - To succeed, the ad hoc approach must rely overwhelmingly on:
 - the ingenuity of participants to improvise solutions for unforeseen problems
 - the ability of the participants to coordinate and communicate with each other, and
 - what can be conveniently described as “luck”, meaning that the right people hit the right targets under the right circumstances.
- As should be expected, **ad hoc** is a high-risk approach.

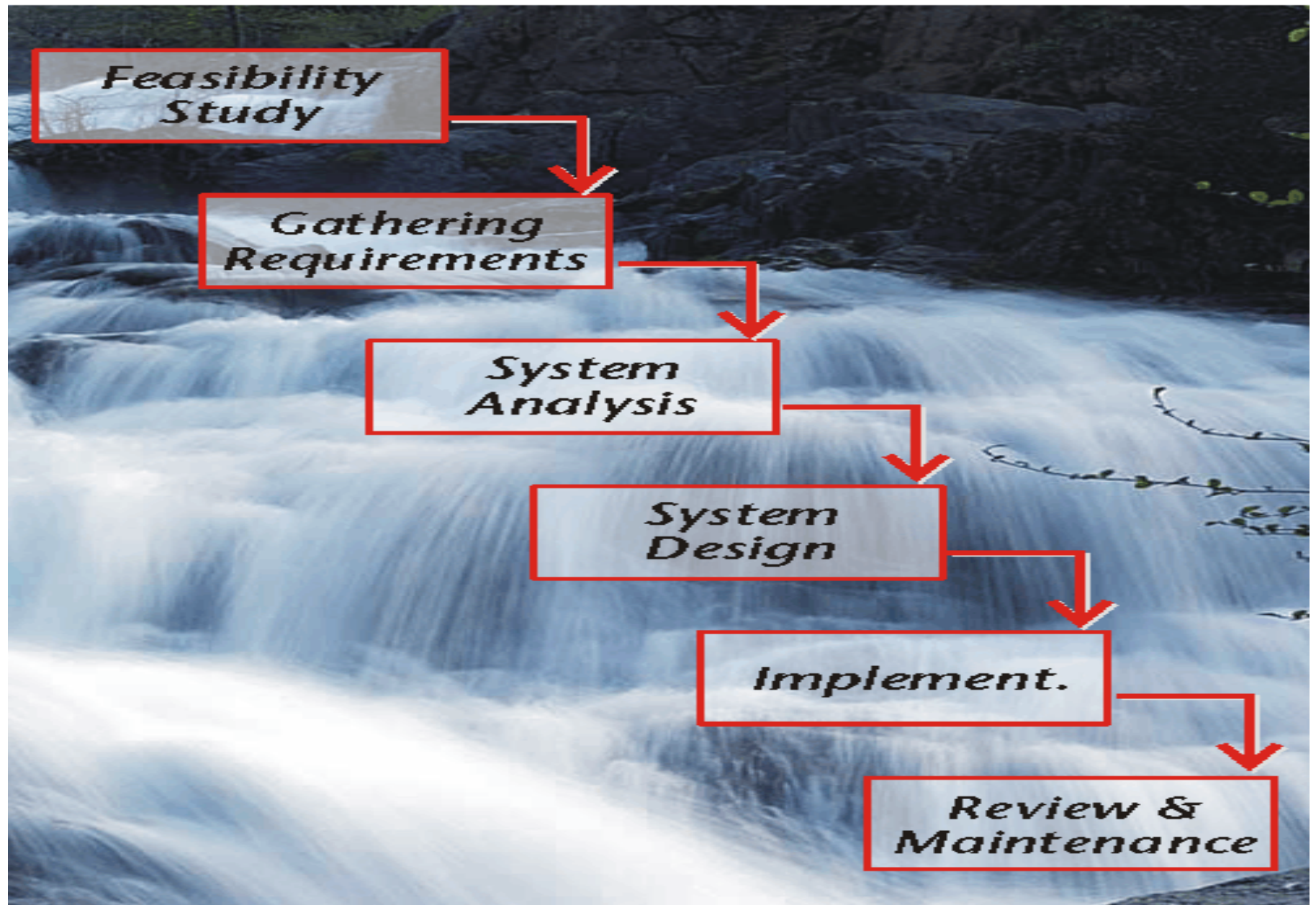
System Development Life Cycle (SDLC)

- SDLC methodologies view software development as primarily a project management process rather than a technical one.
- Each phase is a “**milestone**” and the resulting documents are the “deliverables.”

The Waterfall Model

- The waterfall model specifies a set of sequential phases for software development
 - Each step cannot begin until the previous step has been completed *and* documented.
 - It is document-driven.
 - Design is conceived as **processes and data**, not as objects.

The Waterfall Model



The Limits of Waterfall Model

- Inflexibility
 - Cannot easily swim upstream
- Over-Reliance on Documentation
 - Relies too much on documentation
- Detachment from Technology
 - “One size” methodology cannot fit all technologies.
- Detachment from Marketplace
 - Slow-paced methodology.
- Detachment from the Profession
 - Programming is not the same as assembly of cars or baking breads, nor programmers work the same way as manufacturing workers or bakers.

Prototyping

- Prototyping is the creation of a working model of the essential features of the final product for testing and verification of requirements.
- There are two types of prototyping: incremental or evolutionary and throwaway.

Incremental and Iterative Approach

- In incremental development, the product is built through successive versions that are refined and expanded with each iteration
 - Three concepts underlie the incremental approach:
 - The Initialization Step
 - The Control List
 - The Iteration Step

Throwaway Approach

- In the incremental approach,
 - the initial prototype is revised and refined repeatedly until it becomes the final product.
- In the throwaway approach,
 - the prototype is discarded after the stakeholders in the development are confident that they have arrived at the correct specifications and the development on the “real” product can start.

Problems with Prototyping

- Unbalanced Architecture
 - Since the main thrust of prototyping is towards the user interface, the developers tend to include more and more functionality in the outer layers of the information system, creating a distorted architecture.
- The Illusion of Completeness
 - Since the clients might not understand why the developers insist that a lot more is to be done, the prototype may successfully present the user interface for a complex functionality the feasibility of which is far from certain.
- Diminishing Changeability
 - Since prototyping can leave little trace of how the development evolved, modifying the application can resemble an archeological undertaking to piece together a lost civilization.
- Prototyping can result in too little documentation or, more importantly, too little modeling.

The Spiral Model

- The spiral model is a risk-oriented lifecycle model that breaks a software project up into mini-projects.

Rapid Application Development (RAD)

- Rapid Application Development is selecting techniques, methods, practices and procedures that would result faster development and shorter schedules.

Characteristics of RAD

- Requirements Planning
 - Aims at eliciting information and requirements from the senior people and verifying the goals.
- Design
 - The design phase begins once the *top-level* requirements of the system are identified. To discover more detailed requirements, RAD relies on Joint Application Development (JAD) workshops.
- Implementation
 - Once the users approve the preliminary design, a detailed design of the system is created and code is generated. Implementation phase is heavily dependent upon CASE tools.
- Enhancements and Maintenance
 - In the framework of RAD, a software is never completed until it is retired.
 - there is no significant difference between development and maintenance. (This position is in opposition to most SDLC methodologies.)

Agile Methodologies

- Agile methodologies aim at being adaptive rather than predictive.

Extreme Programming (XP)

- A better-known example of agile methods, and one of the earliest ones, is Extreme Programming (XP).
- XP has a set of clear-cut practices that can be grouped in four categories:
 - Planning
 - Designing
 - Coding
 - Testing

Extreme Programming (XP)

- Planning
 - user stories
 - release plan
 - pair programming
- Designing
 - CRC cards
 - Spike solutions
 - refactoring
- Coding
 - collective ownership
 - sequential integration
 - no overtime
- Testing
 - Unit testing
 - Acceptance testing
 - Integration testing

Modeling

- Modeling, as a methodology, is the systematic representation of the relevant features of a product or a system from particular perspectives.

Modeling for Software Development

- Software modeling is shaped by four interweaved factors:
 - ① how the real world is seen,
 - ② how software is defined,
 - ③ the process of development, and
 - ④ the modeling language.

Object-Oriented Development

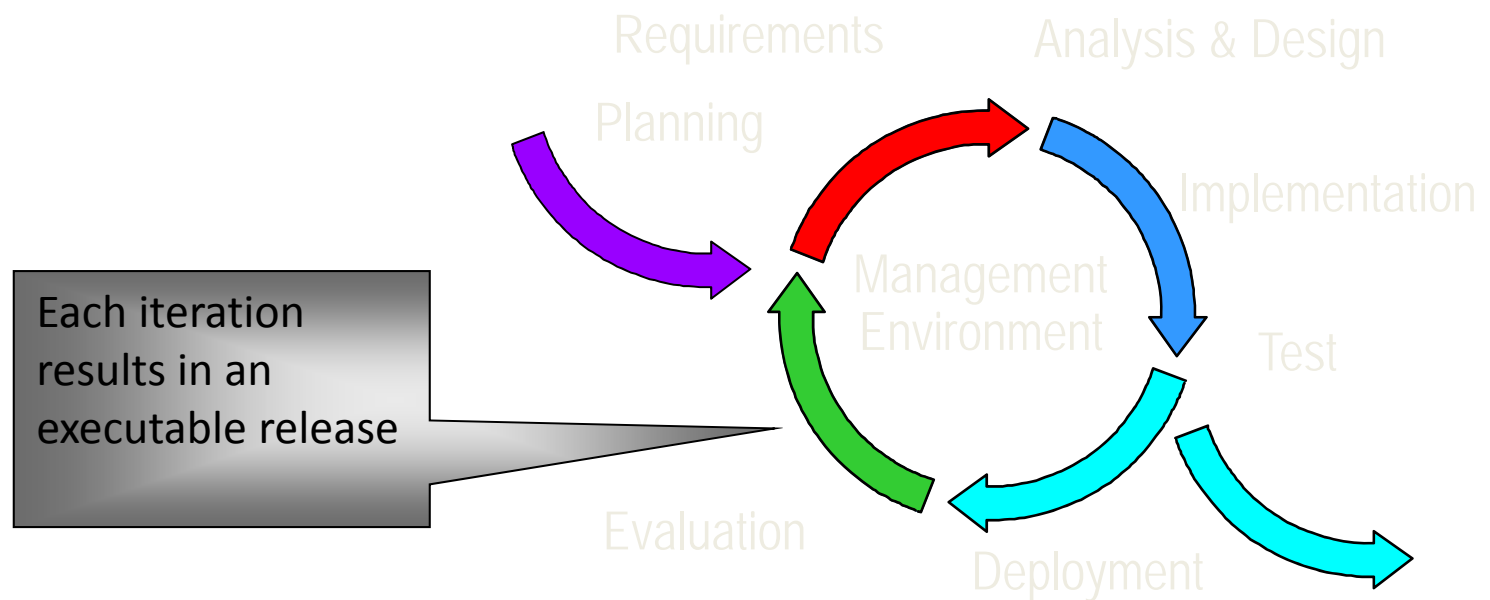
- A full object-oriented development requires three things:
 - ❶ an object-oriented technology,
 - ❷ an object-oriented analysis and design, and
 - ❸ a project plan adapted to an object-oriented approach

Object-Oriented Development

- Object-oriented *technology* is mature and widely used in the software industry.
- Object-oriented software development is **iterative**.
- Unified Process and Rational Unified Process are common OO iterative development processes

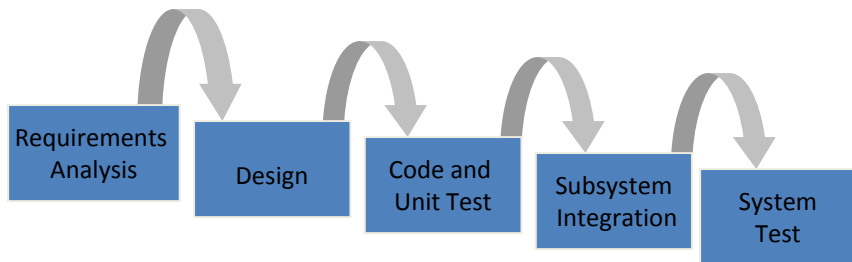
Definition of Iterative Development

- Iterative development = steering a project by using periodic objective assessments, and re-planning based on those assessments
- Good iterative development means:
 - Addressing risks early
 - Using an architecture-driven approach
 - Measuring objectively



Contrasting Traditional and Iterative Processes

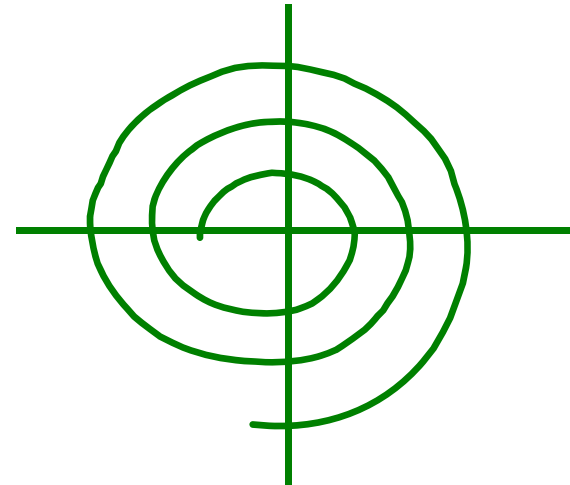
Waterfall Process



- Requirements-driven and mostly custom development
- Late risk resolution
- Diseconomy of scale



Iterative Process



- Architecture-driven and component-based
- Early risk resolution
- Economy of scale



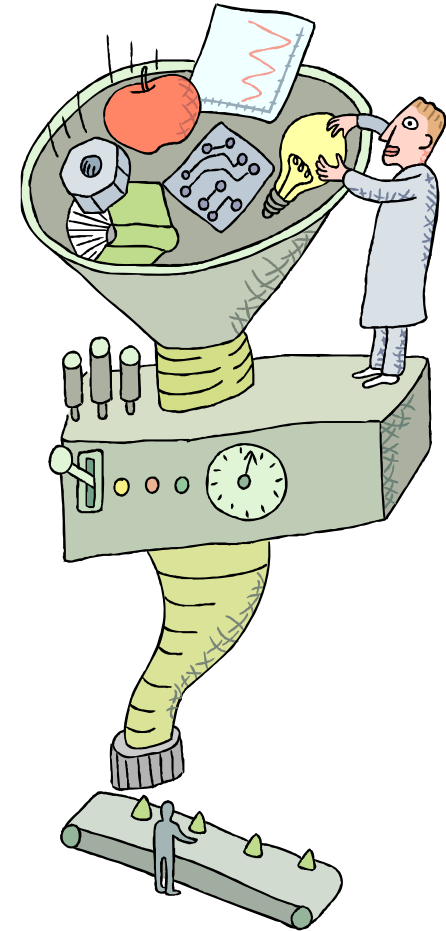
Iterations and Phases

Inception	Elaboration		Construction			Transition	
Preliminary Iteration	Architecture Iteration	Architecture Iteration	Development Iteration	Development Iteration	Development Iteration	Transition Iteration	Transition Iteration

- **Inception:** To achieve concurrence among all stakeholders on the lifecycle objectives for the project
- **Elaboration:** To baseline architecture providing a stable basis for the design and implementation efforts in Construction
- **Construction:** To complete the development of the product
- **Transition:** To ensure the product is available for its end users

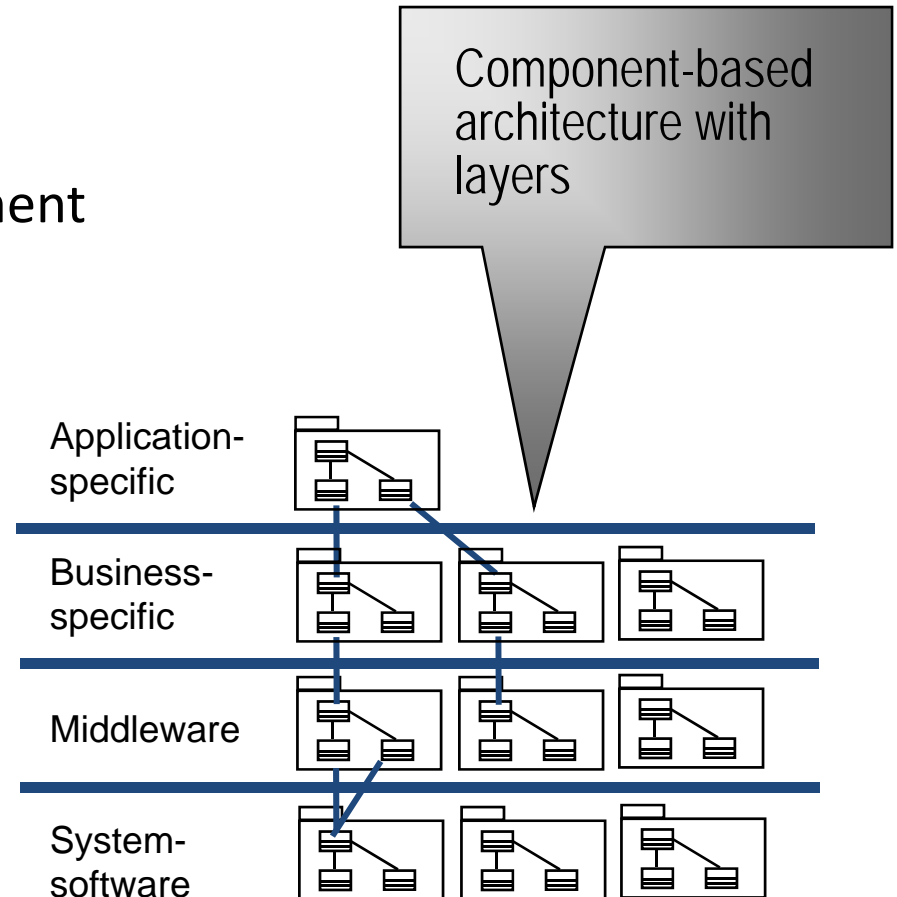
Managing Requirements

- Ensures that you:
 - Solve the right problem
 - Build the right system
- By taking a systematic approach to
 - Eliciting
 - Organizing
 - Documenting
 - Managing
- The changing requirements of a software application



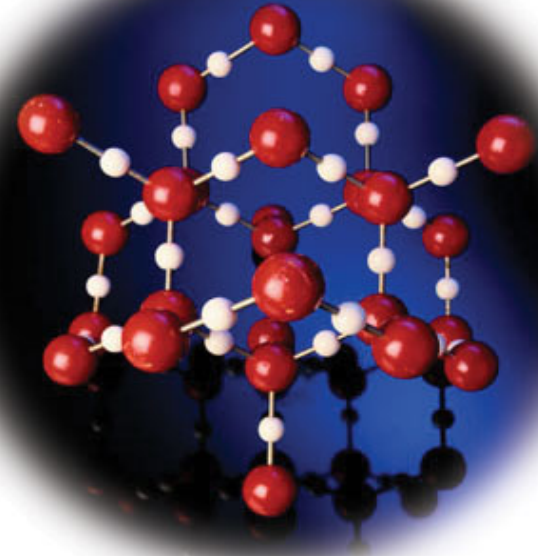
Use Component-Based Architectures

- Basis for reuse
 - Component reuse
 - Architecture reuse
- Basis for project management
 - Planning
 - Staffing
 - Delivery
- Intellectual control
 - Manage complexity
 - Maintain integrity



Model Visually (UML)

- Captures structure and behavior
- Shows how system elements fit together
- Keeps design and implementation consistent
- Hides or exposes details as appropriate
- Promotes unambiguous communication
 - The UML provides one language for all practitioners



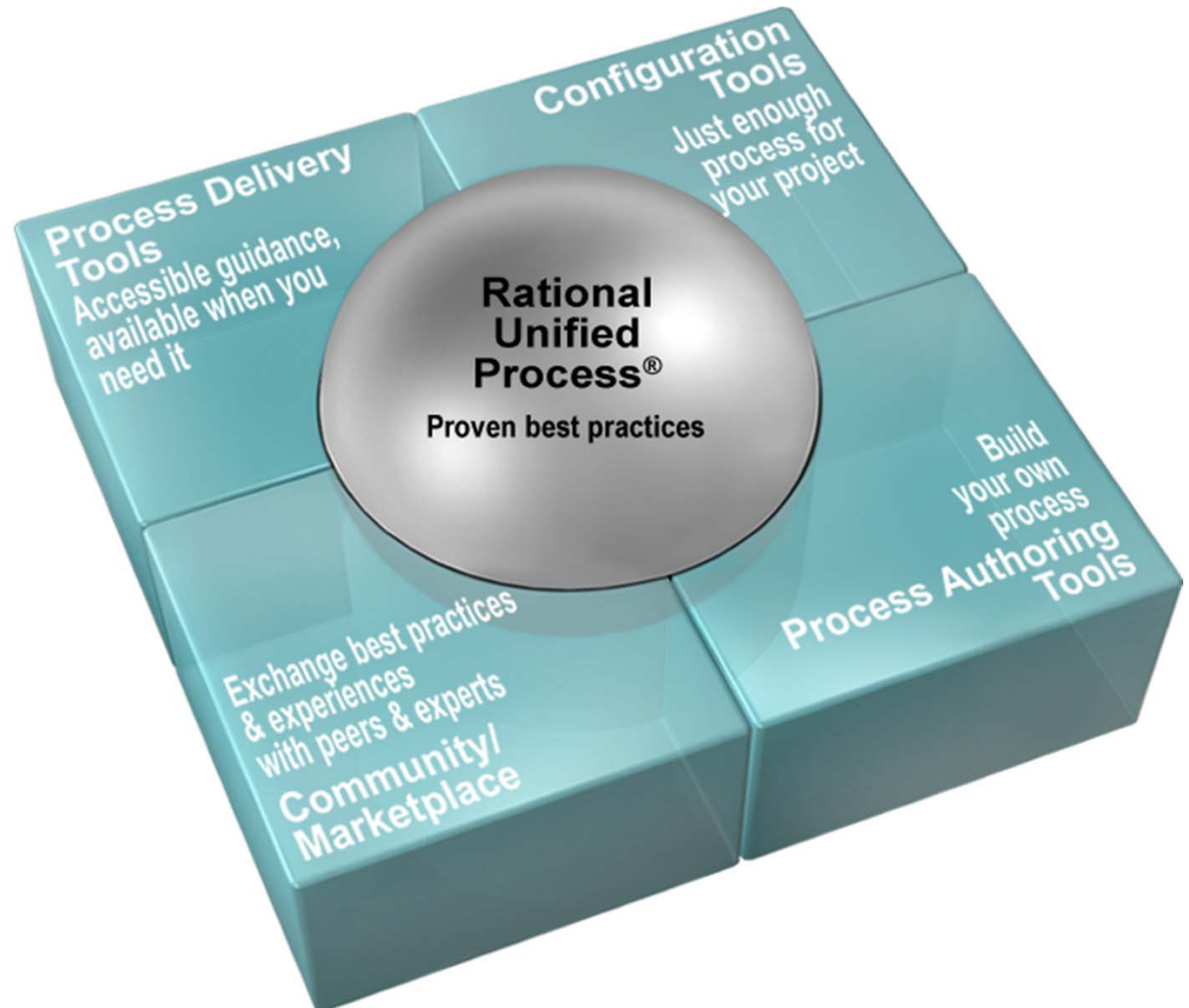
Continuously Verify Quality

Software problems are
100 to 1000 times more costly
to find and repair after deployment

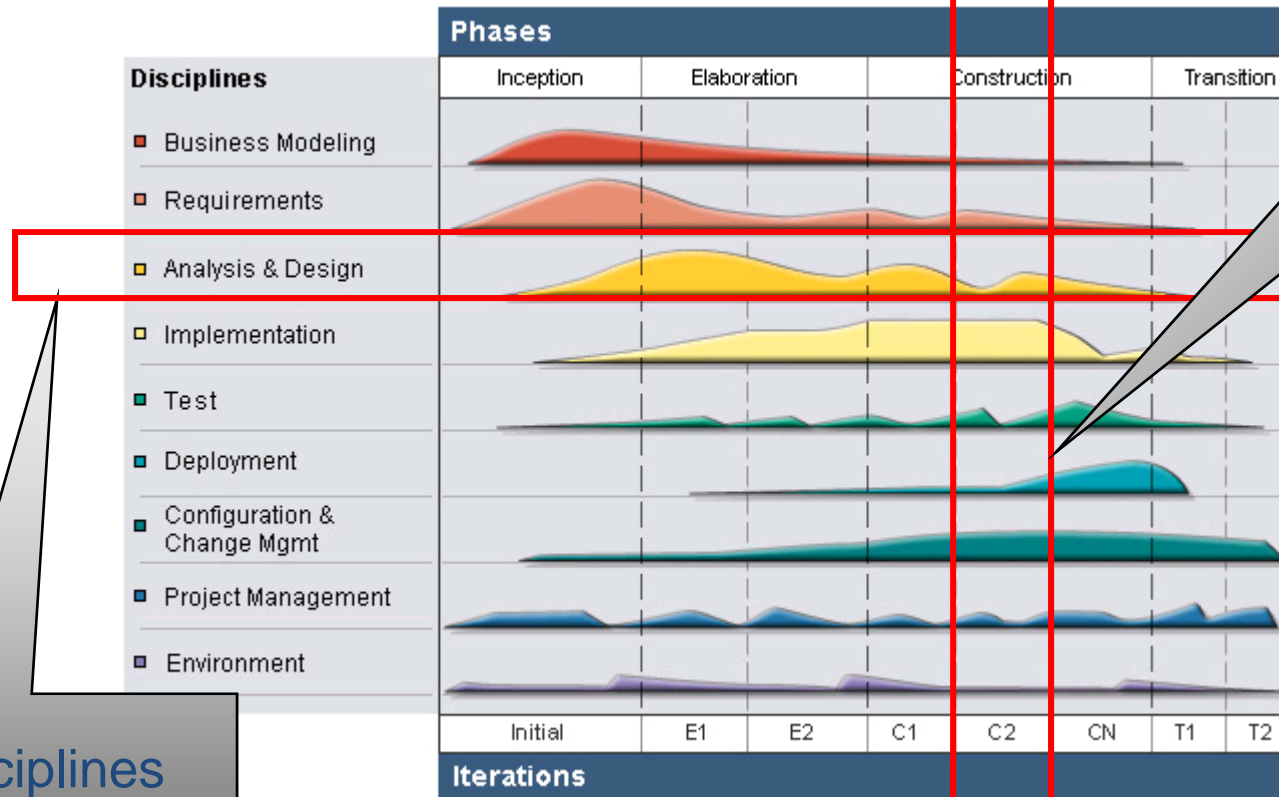


Rational Unified Process Implements Best Practices

- Iterative approach
- Guidance for activities and artifacts
- Process focus on architecture
- Use cases that drive design and implementation
- Models that abstract the system



Bringing It All Together



In an **iteration**, you walk through all disciplines

Disciplines group activities logically

What Is the UML?

- The UML is a language for
 - Visualizing
 - Specifying
 - Constructing
 - Documentingthe artifacts of a software-intensive system



FileEditViewHistoryBookmarksToolsHelp

OMG Unified Modeling Languag... xOMG Formal Specifications x

www.uml.org/#UML2.0

Secure Search

CORBA

MOF

CWM

UML

OMG

Object Management Group

Celebrating 25 Years

BPMN

DDS

UML

MARTE

UML

Model

About Us

Press Room

Calendar

Documents

Members Only

Technology

Industries

Programs

Follow us:

Google Custom Search

Search

Unified Modeling Language™ (UML®)

Resource Page

UNIFIED
MODELING
LANGUAGE

U

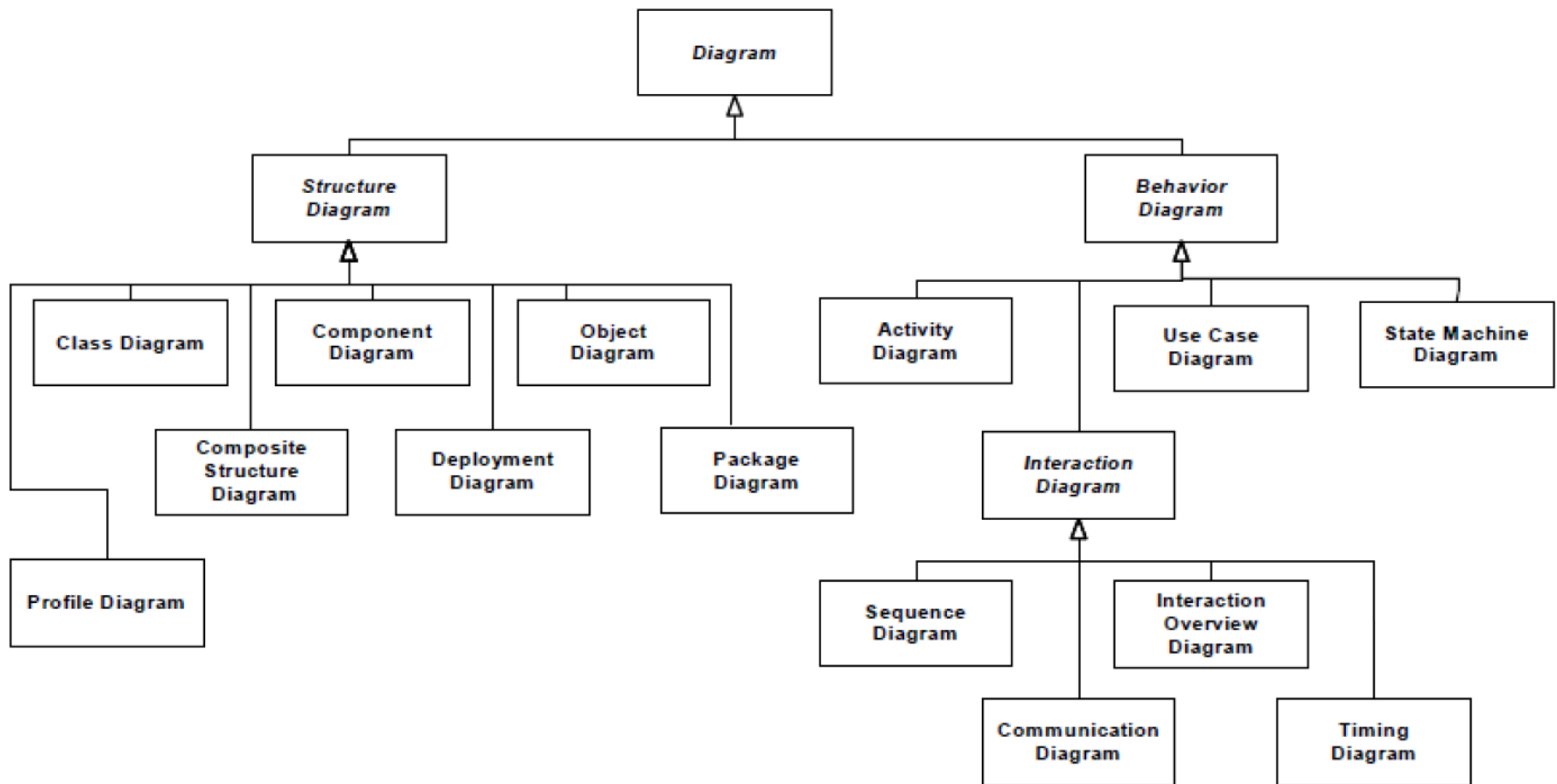
ML

| [Introduction to UML](#) | [UML Success Stories](#) | [UML Certification Program](#) | [Vendor Directory](#) |

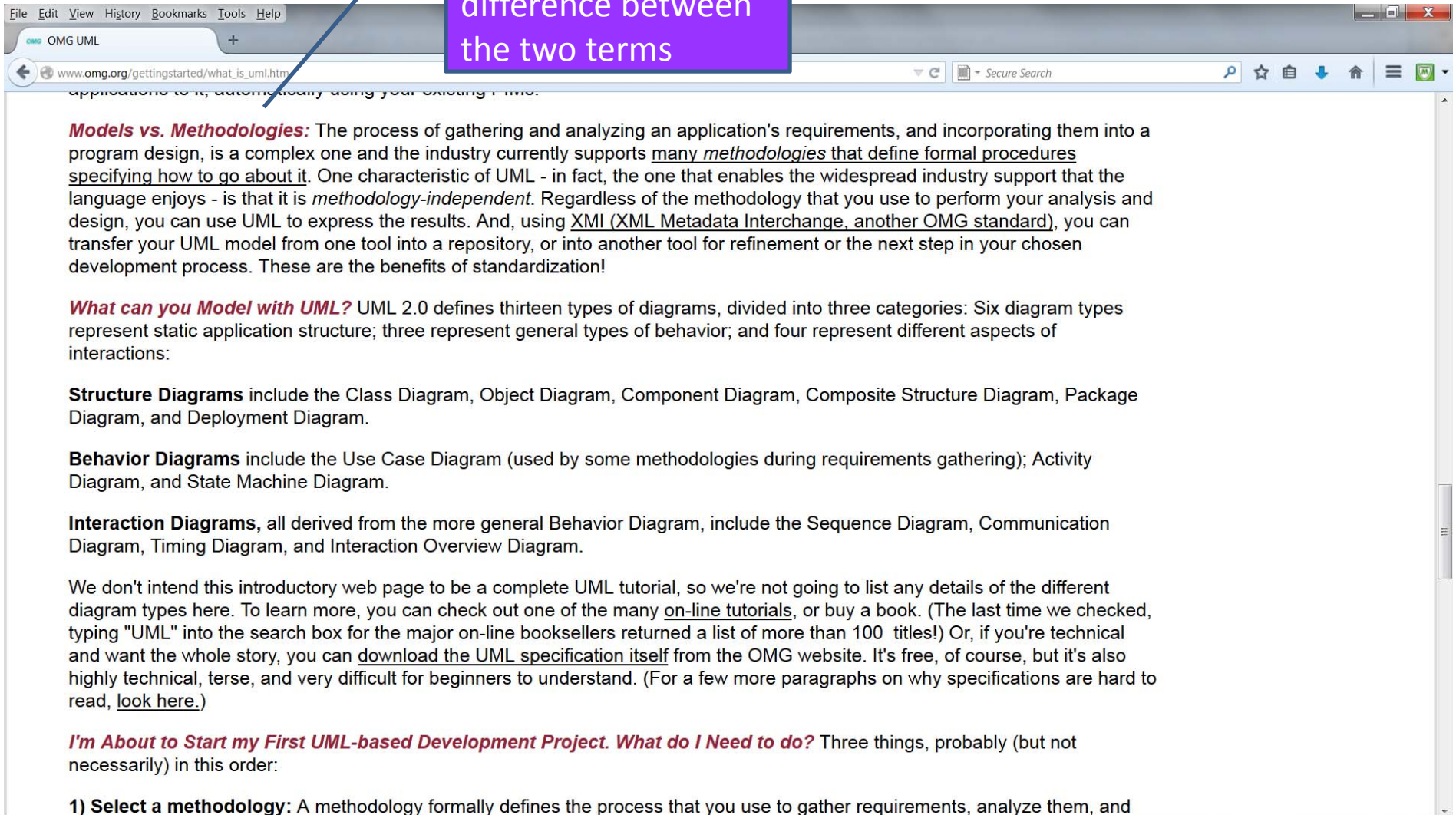
Getting Started With UML:

The Unified Modeling Language™ - UML - is [OMG's](#) most-used specification, and the way the world models not only application structure, behavior, and architecture, but also business process and data structure.

UML, along with the [Meta Object Facility \(MOF™\)](#), also provides a key foundation for OMG's [Model-Driven Architecture®](#), which unifies every step of development and integration from business modeling, through architectural and application modeling, to development, deployment, maintenance, and evolution.



Make a note for the difference between the two terms

A screenshot of a web browser displaying the OMG UML website. A purple callout box with white text points to the page content, stating "Make a note for the difference between the two terms". The browser's address bar shows "www.omg.org/gettingstarted/what_is_uml.htm". The page content includes several paragraphs explaining UML, its methodologies, and diagram types. The text is formatted with bold headings and underlined links.

File Edit View History Bookmarks Tools Help

OMG UML

www.omg.org/gettingstarted/what_is_uml.htm

Secure Search

Models vs. Methodologies: The process of gathering and analyzing an application's requirements, and incorporating them into a program design, is a complex one and the industry currently supports many methodologies that define formal procedures specifying how to go about it. One characteristic of UML - in fact, the one that enables the widespread industry support that the language enjoys - is that it is *methodology-independent*. Regardless of the methodology that you use to perform your analysis and design, you can use UML to express the results. And, using XMI (XML Metadata Interchange, another OMG standard), you can transfer your UML model from one tool into a repository, or into another tool for refinement or the next step in your chosen development process. These are the benefits of standardization!

What can you Model with UML? UML 2.0 defines thirteen types of diagrams, divided into three categories: Six diagram types represent static application structure; three represent general types of behavior; and four represent different aspects of interactions:

Structure Diagrams include the Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram.

Behavior Diagrams include the Use Case Diagram (used by some methodologies during requirements gathering); Activity Diagram, and State Machine Diagram.

Interaction Diagrams, all derived from the more general Behavior Diagram, include the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

We don't intend this introductory web page to be a complete UML tutorial, so we're not going to list any details of the different diagram types here. To learn more, you can check out one of the many on-line tutorials, or buy a book. (The last time we checked, typing "UML" into the search box for the major on-line booksellers returned a list of more than 100 titles!) Or, if you're technical and want the whole story, you can download the UML specification itself from the OMG website. It's free, of course, but it's also highly technical, terse, and very difficult for beginners to understand. (For a few more paragraphs on why specifications are hard to read, look here.)

I'm About to Start my First UML-based Development Project. What do I Need to do? Three things, probably (but not necessarily) in this order:

1) Select a methodology: A methodology formally defines the process that you use to gather requirements, analyze them, and

File Edit View History Bookmarks Tools Help

OMG UML

www.omg.org/gettingstarted/what_is_uml.htm

Secure Search

Applications to it, automatically using your existing files.

Models vs. Methodologies: The process of gathering a program design, is a complex one and the industry currently specifies how to go about it. One characteristic of UML language enjoys - is that it is *methodology-independent*. In design, you can use UML to express the results. And, using transfer your UML model from one tool into a repository, development process. These are the benefits of standardization!

What can you Model with UML? UML 2.0 defines thirteen types of diagrams, divided into three categories: Six diagram types represent static application structure; three represent general types of behavior; and four represent different aspects of interactions:

Structure Diagrams include the Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram.

Behavior Diagrams include the Use Case Diagram (used by some methodologies during requirements gathering); Activity Diagram, and State Machine Diagram.

Interaction Diagrams, all derived from the more general Behavior Diagram, include the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

We don't intend this introductory web page to be a complete UML tutorial, so we're not going to list any details of the different diagram types here. To learn more, you can check out one of the many [on-line tutorials](#), or buy a book. (The last time we checked, typing "UML" into the search box for the major on-line booksellers returned a list of more than 100 titles!) Or, if you're technical and want the whole story, you can [download the UML specification itself](#) from the OMG website. It's free, of course, but it's also highly technical, terse, and very difficult for beginners to understand. (For a few more paragraphs on why specifications are hard to read, [look here](#).)

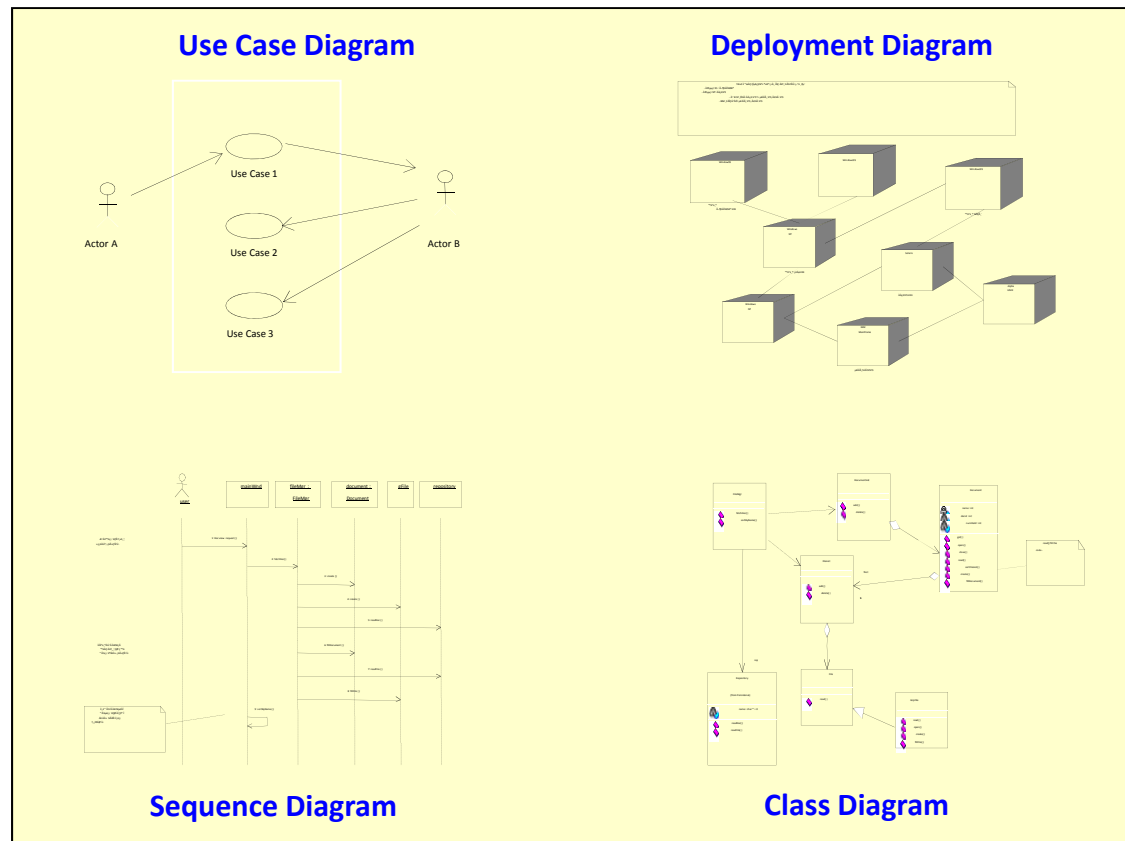
I'm About to Start my First UML-based Development Project. What do I Need to do? Three things, probably (but not necessarily) in this order:

1) Select a methodology: A methodology formally defines the process that you use to gather requirements, analyze them, and

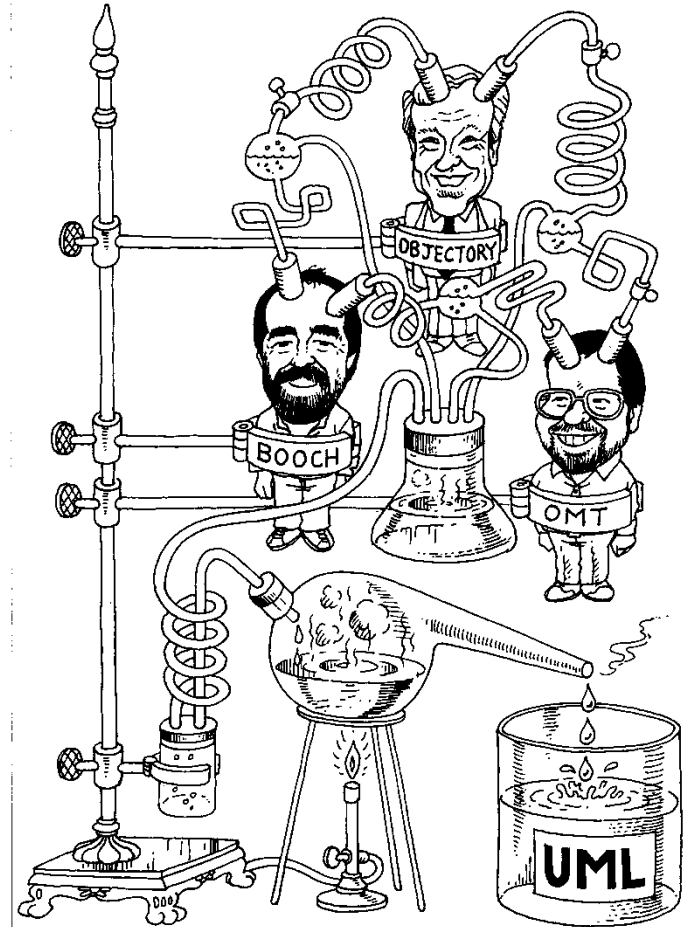
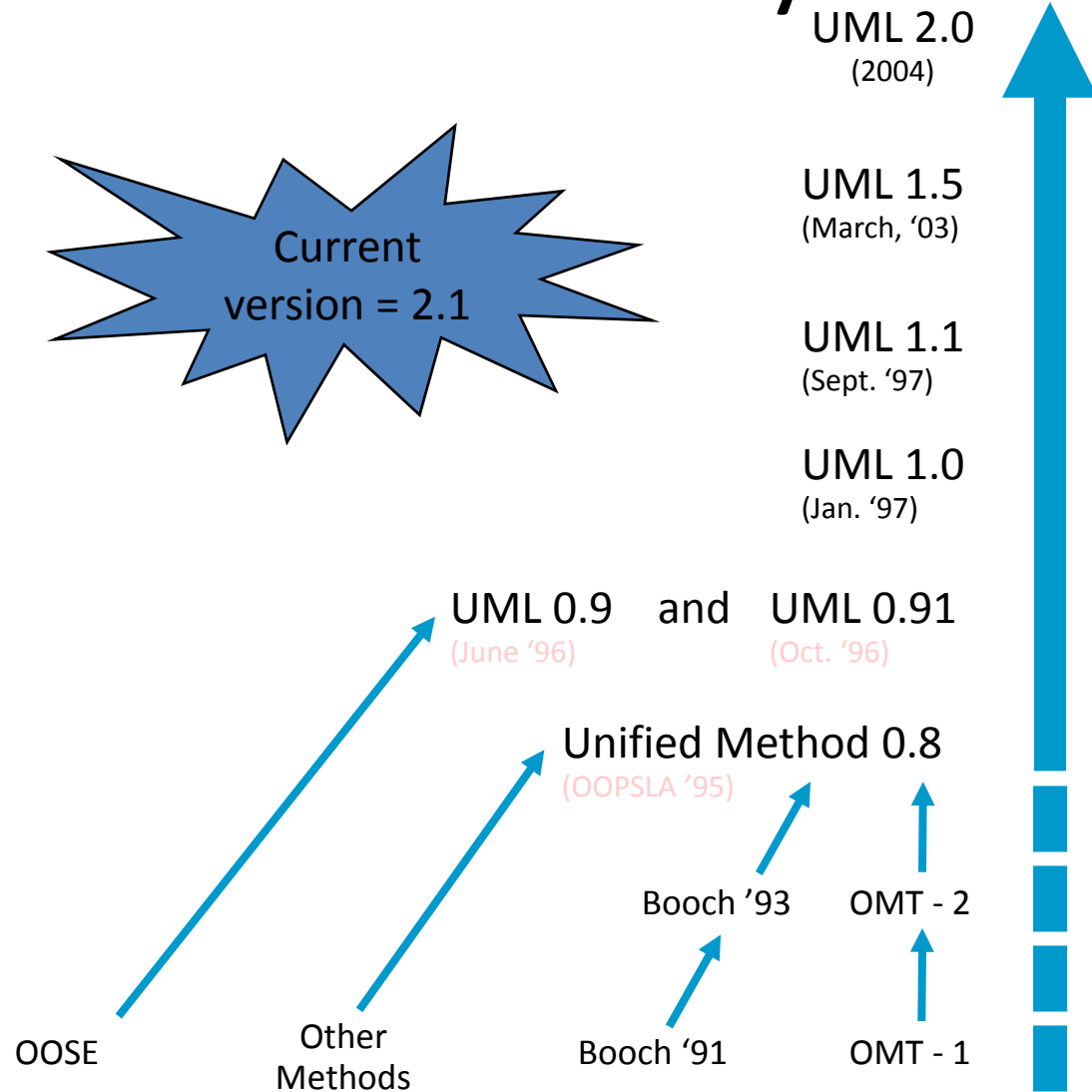
13 diagrams
3 categories

UML Covers All Aspects of Software Development

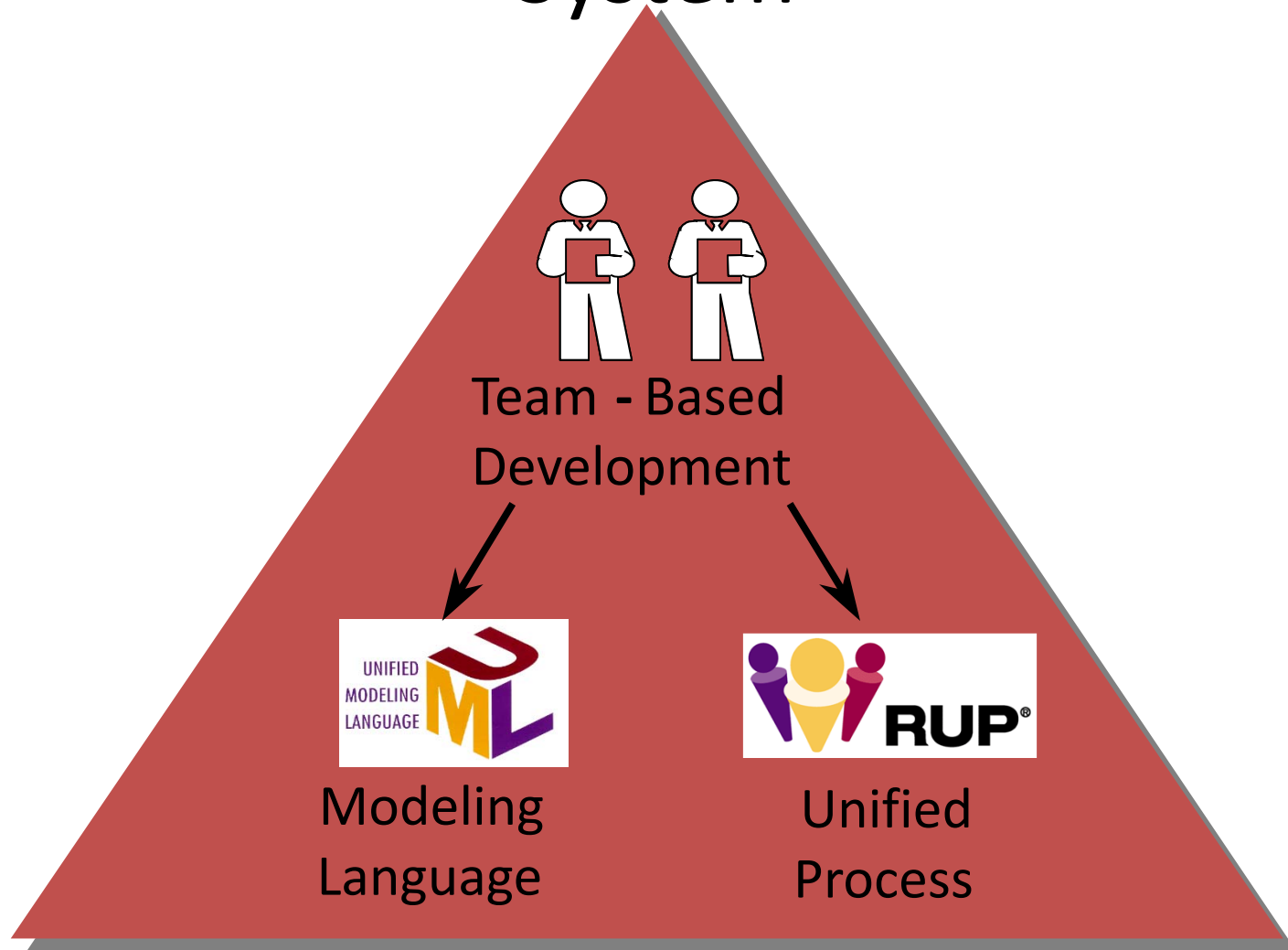
- The UML addresses system architecture, requirements, tests, project planning, and release management



History of the UML



A Language Is Not Enough to Build a System



What is a Model?

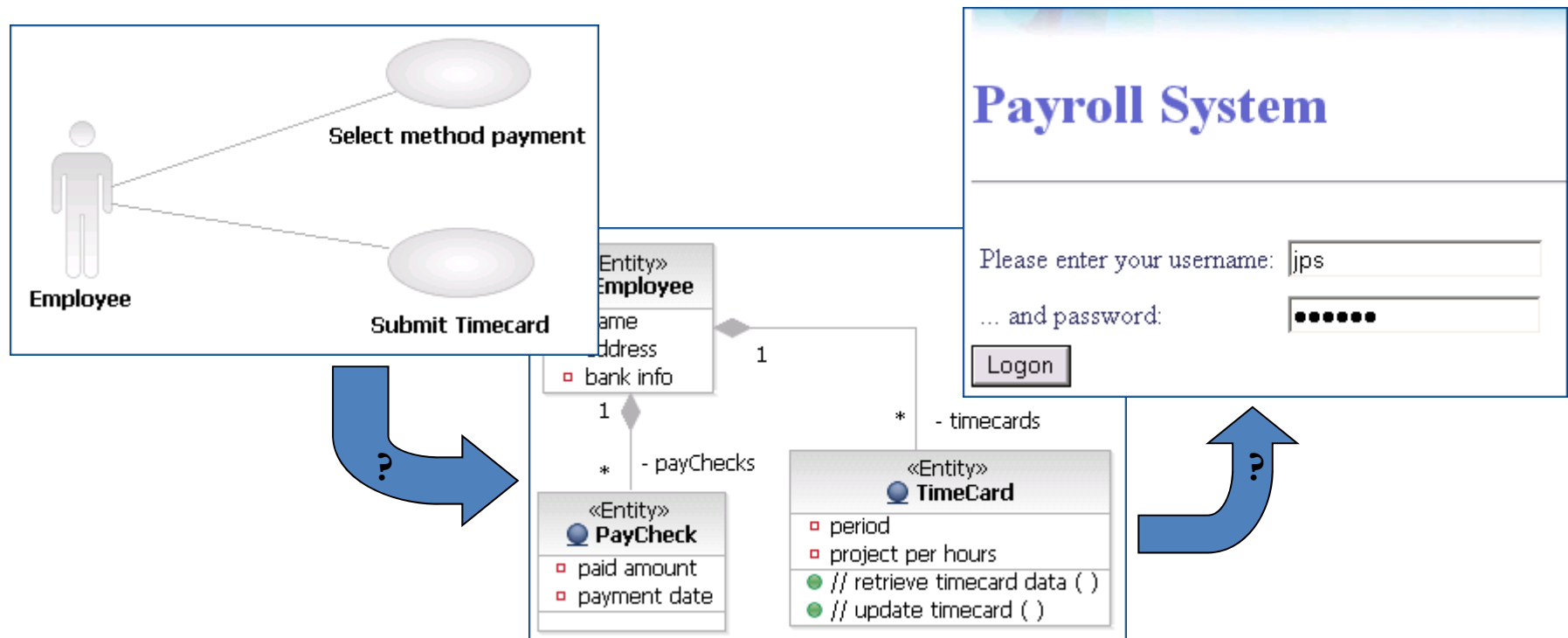
- A model is a simplification of reality
- Modeling achieves four aims:
 - Helps you to visualize a system as you want it to be
 - Permits you to specify the structure or behavior of a system
 - Gives you a template that guides you in constructing a system
 - Documents the decisions you have made
- You build models of complex systems because you cannot comprehend such a system in its entirety
- You build models to better understand the system you are developing

Principles of Visual Modeling

- The model you create influences how the problem is attacked
- Every model may be expressed at different levels of precision
- The best models are connected to reality
- No single model is sufficient

Model-Driven Development

- A natural evolution of object-oriented technologies
- The encapsulation of business logic in (UML) models
- The use of these models to *automate* the development of applications, code generation, testing and maintenance

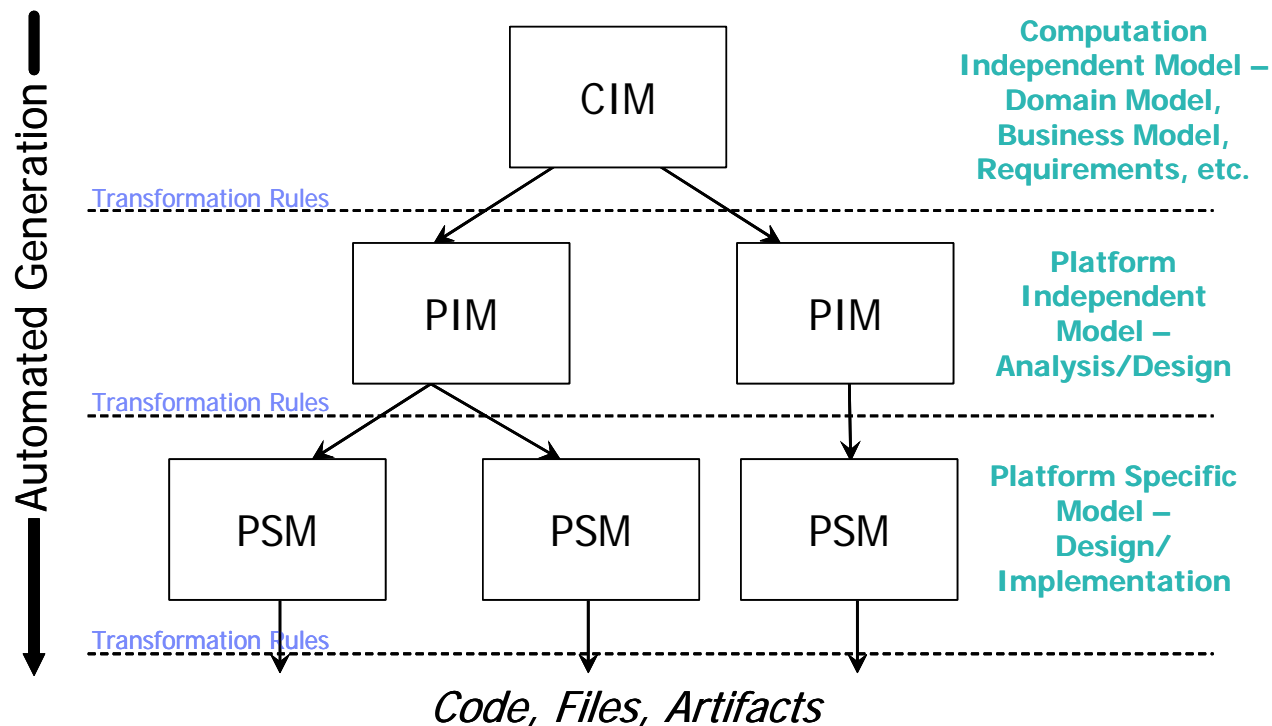


Model-Driven Architecture (MDA)



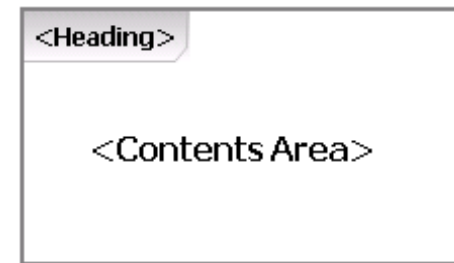
- An architectural style
- An OMG initiative

<http://www.omg.com/>



UML Diagrams

- UML diagrams contain graphical elements (nodes connected by paths) that represent elements in the UML model
 - Each diagram has a *contents area*
 - As an option, it may have a *frame* and a *heading* as shown on the right
- Two main types
 - Structure Diagrams show the static structure of the objects in a system
 - Behavior Diagrams show the dynamic behavior of the objects
 - A complete taxonomy of UML diagrams is provided in chapter 4



What is an Interaction Diagram?

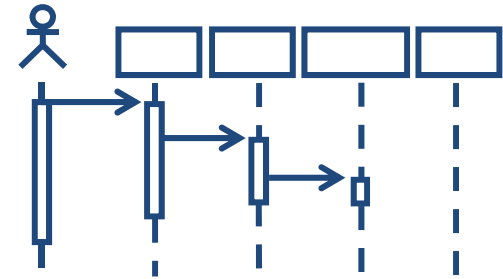
- Generic term that applies to several behavior-type diagrams that emphasize object interactions
 - Sequence Diagram (the most common variant)
 - Communication Diagram
 - Specialized Variants
 - Timing Diagram
 - Interaction Overview Diagram

Objects Need to Collaborate

- Objects are useless unless they can collaborate to solve a problem
 - Each object is responsible for its own behavior and status
 - No one object can carry out every responsibility on its own
- How do objects interact with each other?
 - They interact through messages
 - A client object sends a message to a supplier object to perform some activity

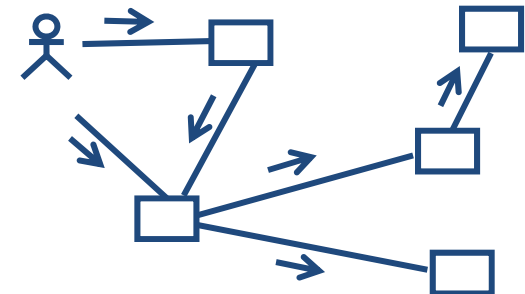
Interaction Diagrams

- Sequence Diagram
 - Time oriented view of object interaction



Sequence Diagrams

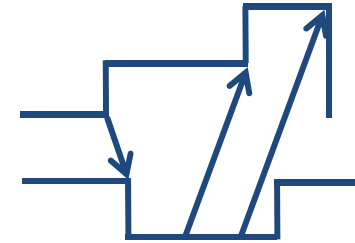
- Communication Diagram
 - Structural view of messaging objects



Communication Diagrams

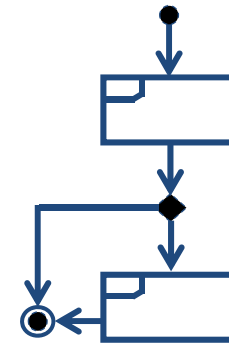
Interaction Diagrams

- Timing Diagram
 - Time constraint view of messages involved in an interaction



Timing Diagrams

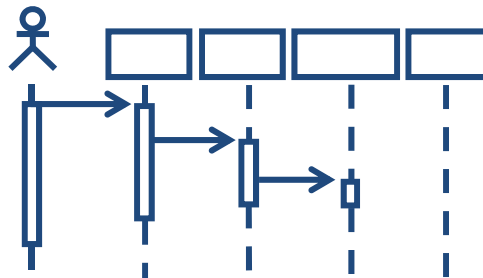
- Interaction Overview Diagram
 - High level view of interaction sets combined into logic sequence



Interaction Overview
Diagrams

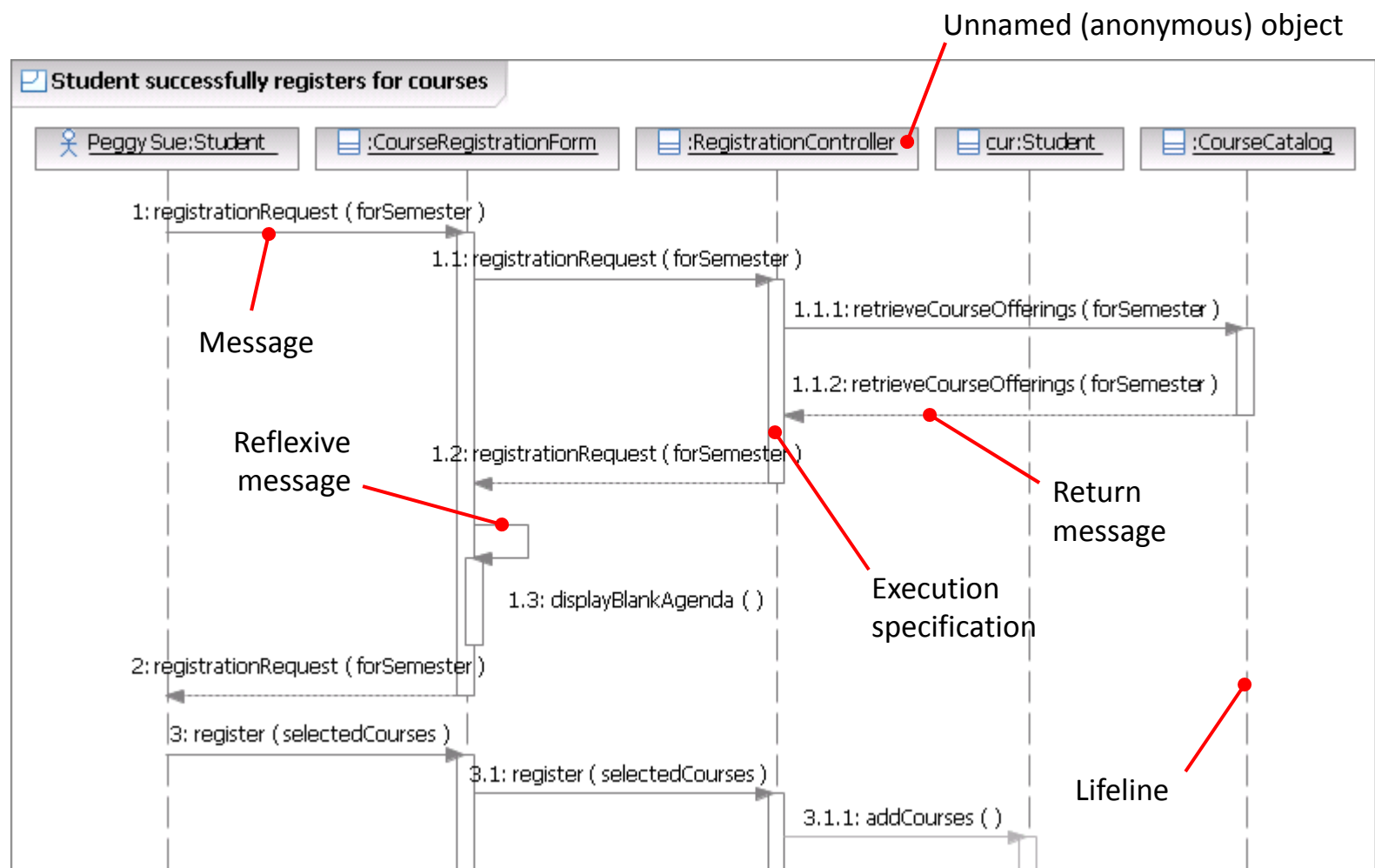
What Is a Sequence Diagram?

- A sequence diagram is an interaction diagram that emphasizes the time ordering of messages
- The diagram shows:
 - The objects participating in the interaction
 - The sequence of messages exchanged



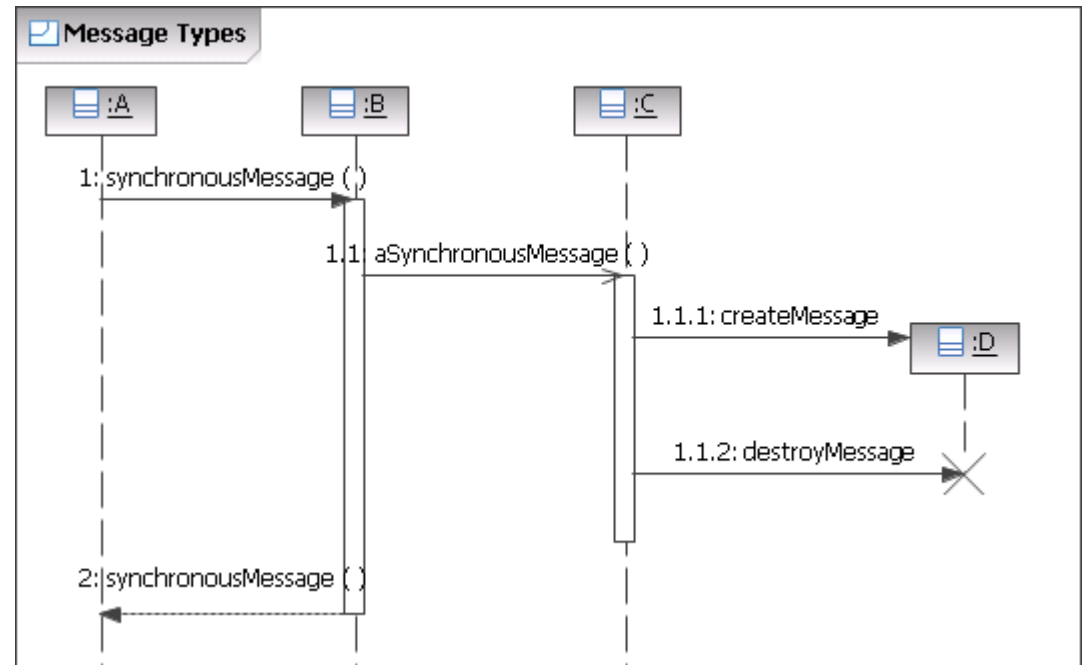
Sequence Diagrams

Example

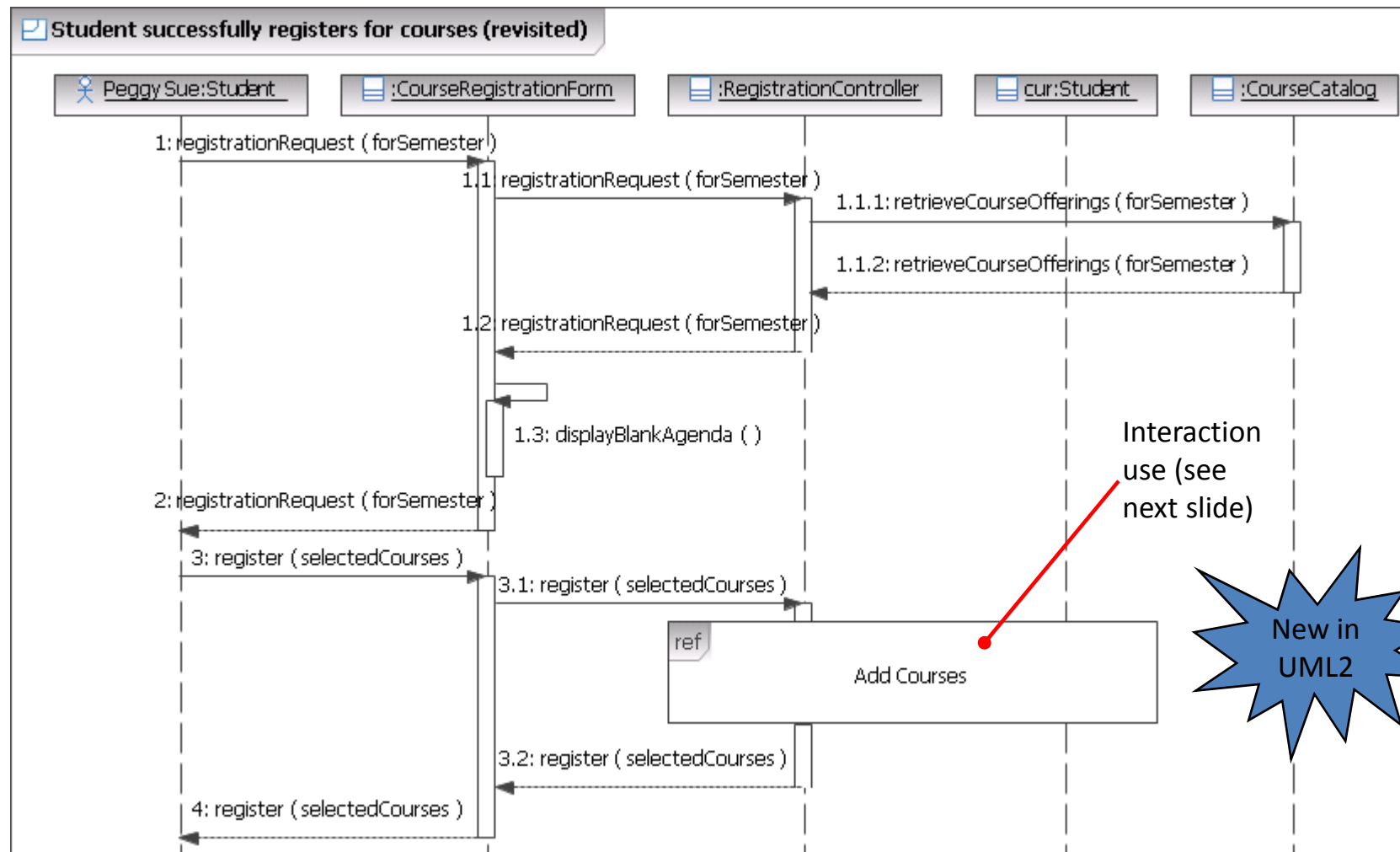


Message Types

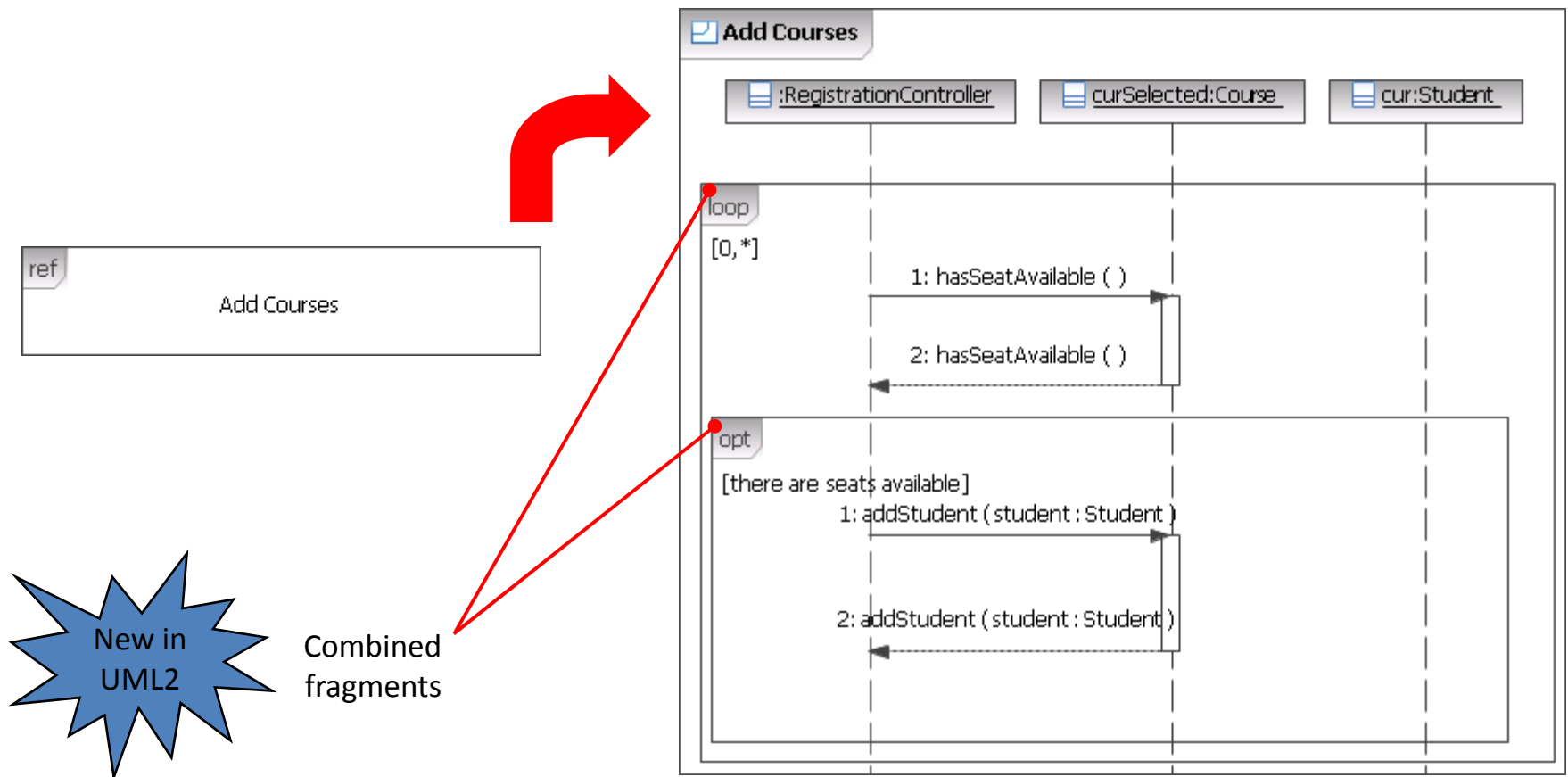
- Synchronous message
 - Call to an operation
- Asynchronous message
 - Asynchronous call to an operation
 - Asynchronous send action of a signal
- Create Message
- Delete Message
- Reply message to an operation call



Interaction Use



Interaction Use and Combined Fragments

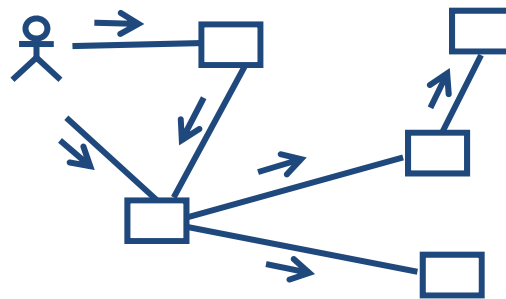


What Is a Communication Diagram?

- A communication diagram emphasizes the organization of the objects that participate in an interaction

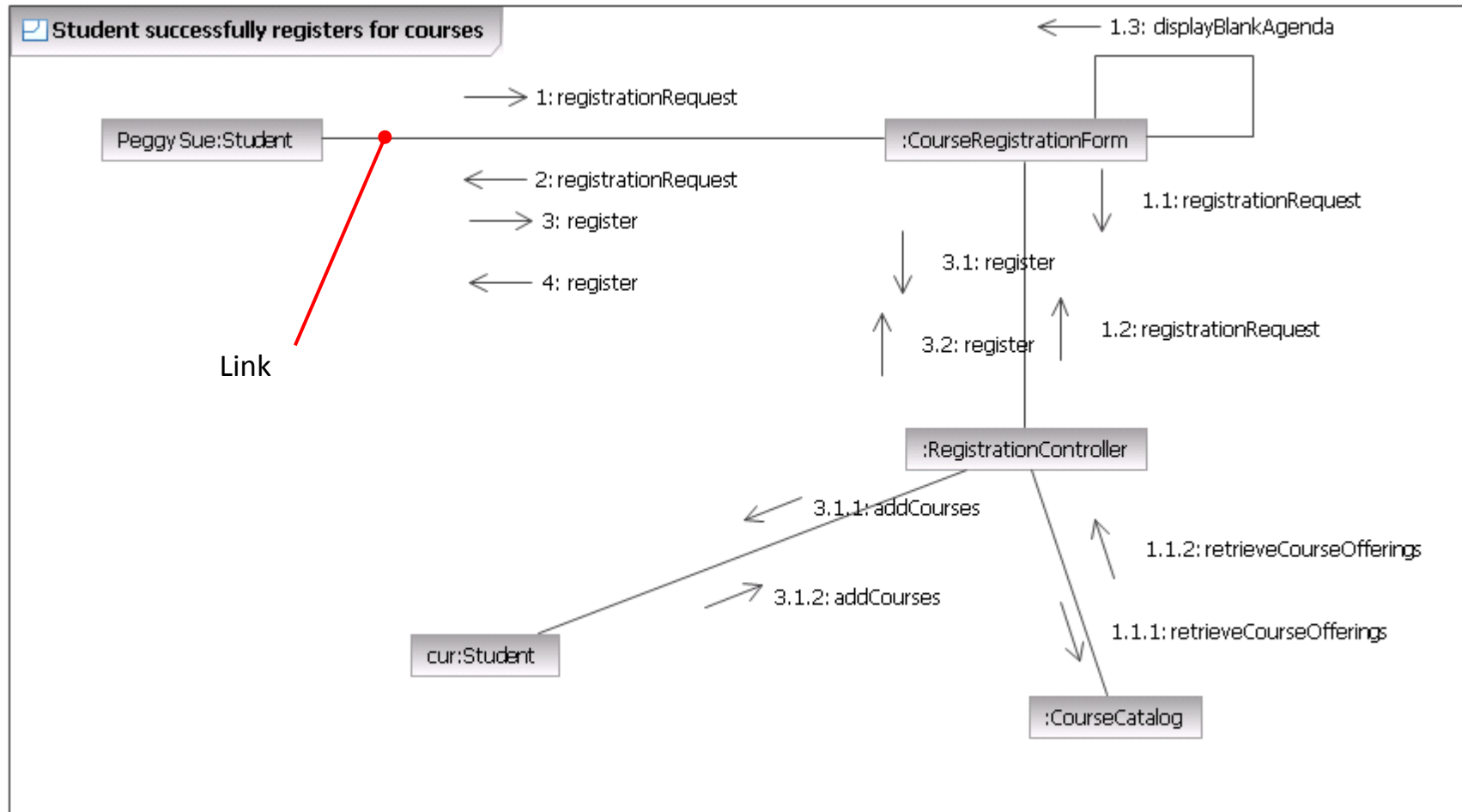
The communication diagram shows:

- The objects participating in the interaction
- Links between the objects
- Messages passed between the objects



Communication Diagrams

Example



Discussion: Communication vs. Sequence Diagrams

- In UML 1.x, collaboration diagrams (as they were called) and sequence diagrams were completely equivalent
 - The major difference, which is retained in UML 2.x, is the ability to explicitly show links in collaboration/communication diagrams
- In UML 2.x, communication diagrams are much less expressive and precise than sequence diagrams
 - Communication diagrams have lost some expressiveness while sequence diagrams were significantly improved
 - UML 2.1 define communication diagrams as “simple Sequence Diagrams that use none of the structuring mechanisms such as InteractionUses and CombinedFragments”
 - It is possible that communication diagrams will eventually fall out of use

Finally Remember the following Relationships between Classes ...

1. Is-a
2. Has-a
3. Uses

Relationships between Classes

- Is-a Vehicle



Relationships between Classes

- Has-a



Relationships between Classes

- Uses

