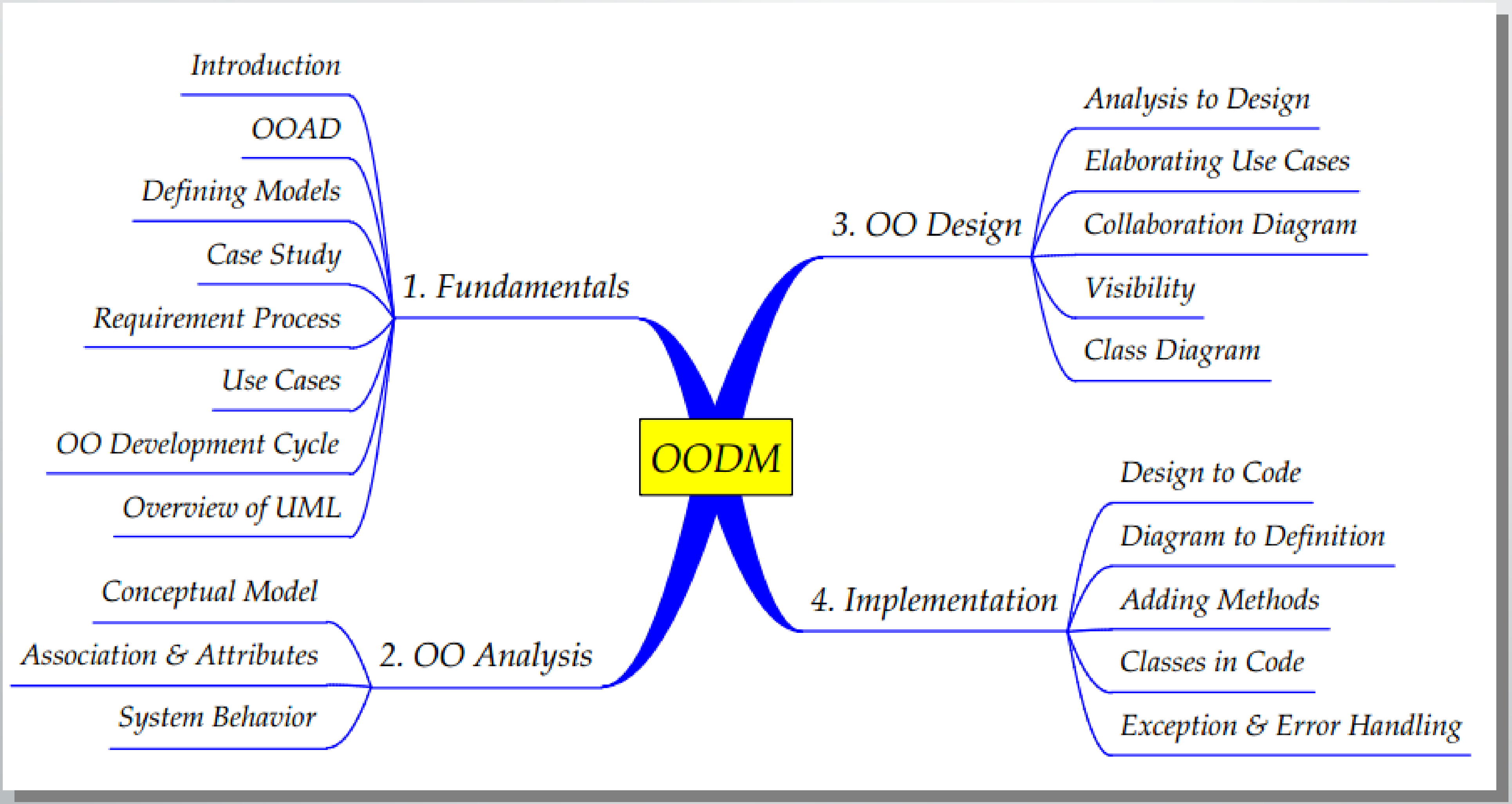
Object Oriented Design and Modeling through UML

[BE IT-6th Semester]

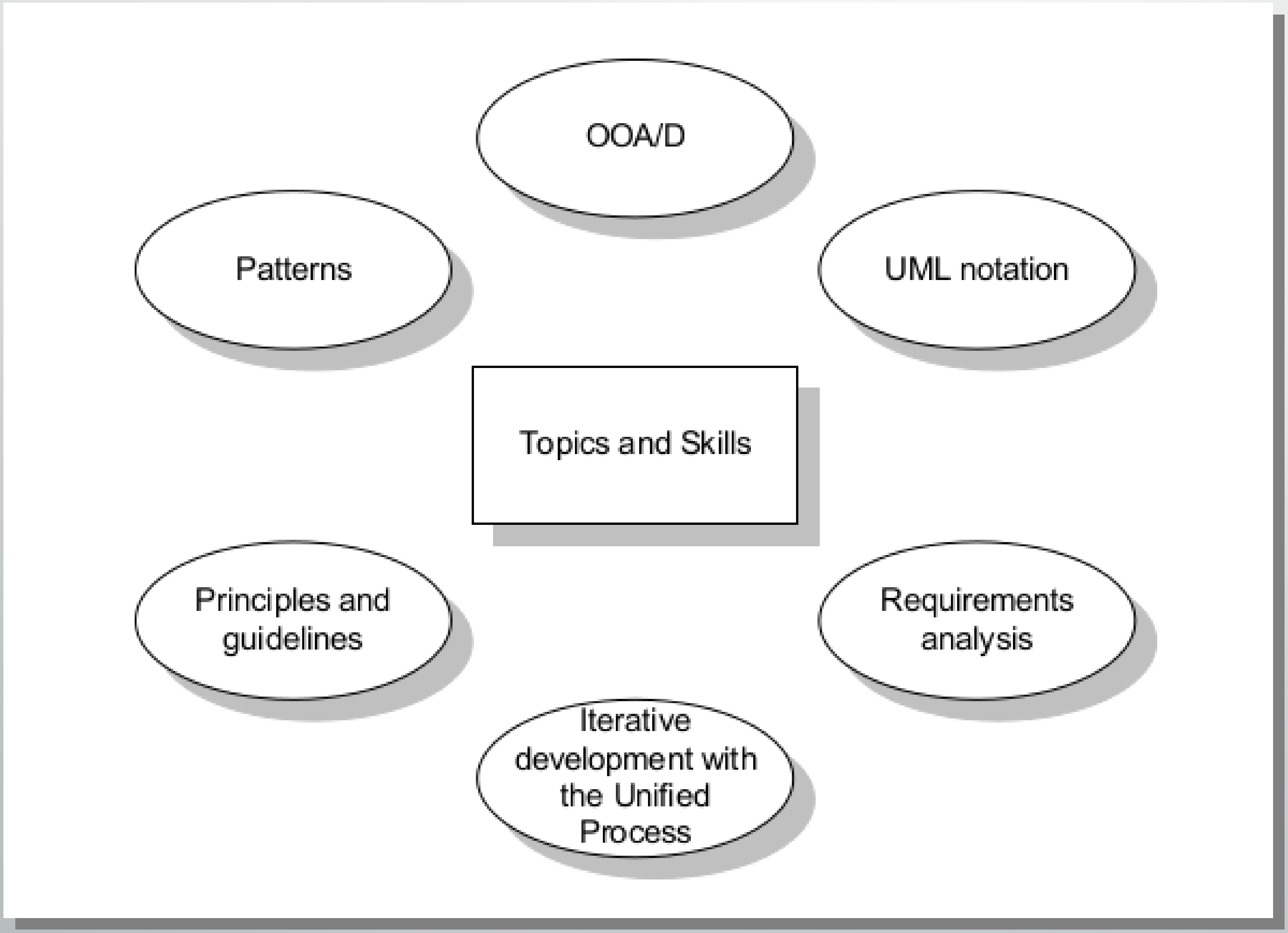
Rishi K. Marseni
{rishimarseni@gmail.com}

Course Content – Abstract



Text Book: **Applying UML and Patterns** by Craig Larman

Course objectives


OOA/D
Patterns
UML notation
Topics and Skills
Principles and guidelines
Requirements analysis
Iterative development with the Unified Process

1. OO Fundamentals

1.1. Introduction

1.2. OOAD

1.3. Defining Models

1.4. Case Study

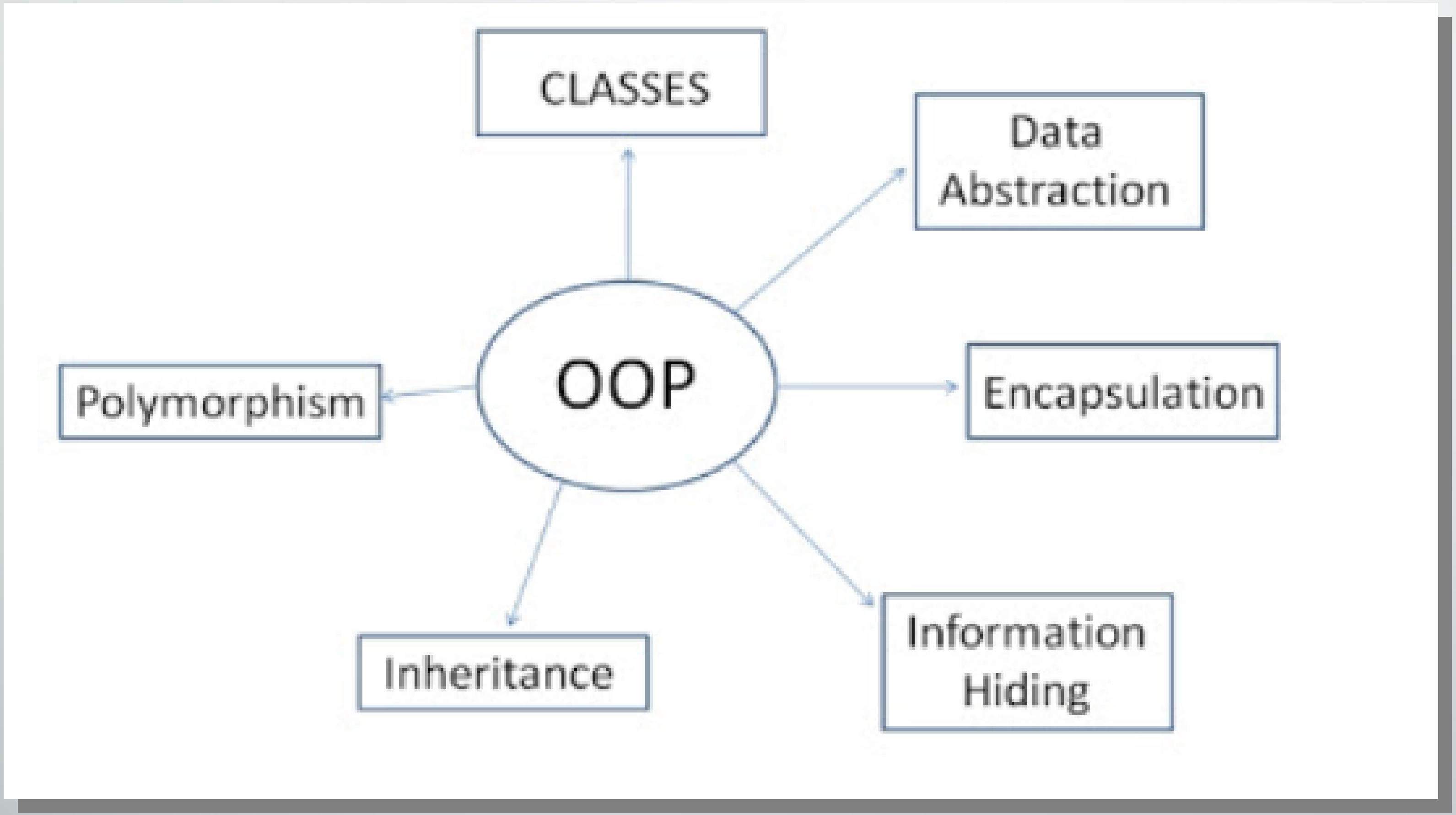
1.5. Requirement Process

1.6. Use Cases

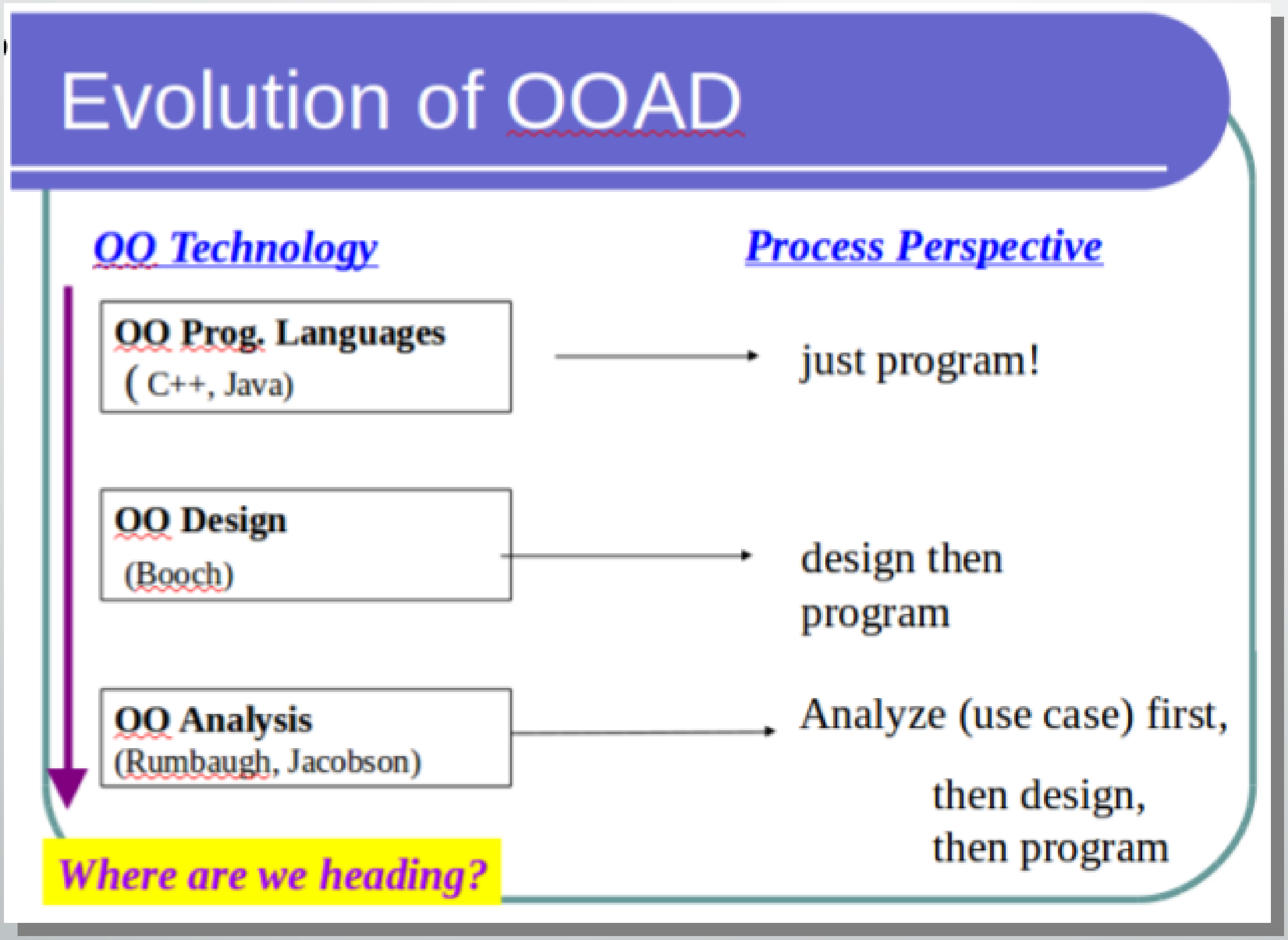
1.7. OO Development Cycle

1.8. Overview of UML

Object Orientation



Evolution of OO Technology



Evolution of OOAD

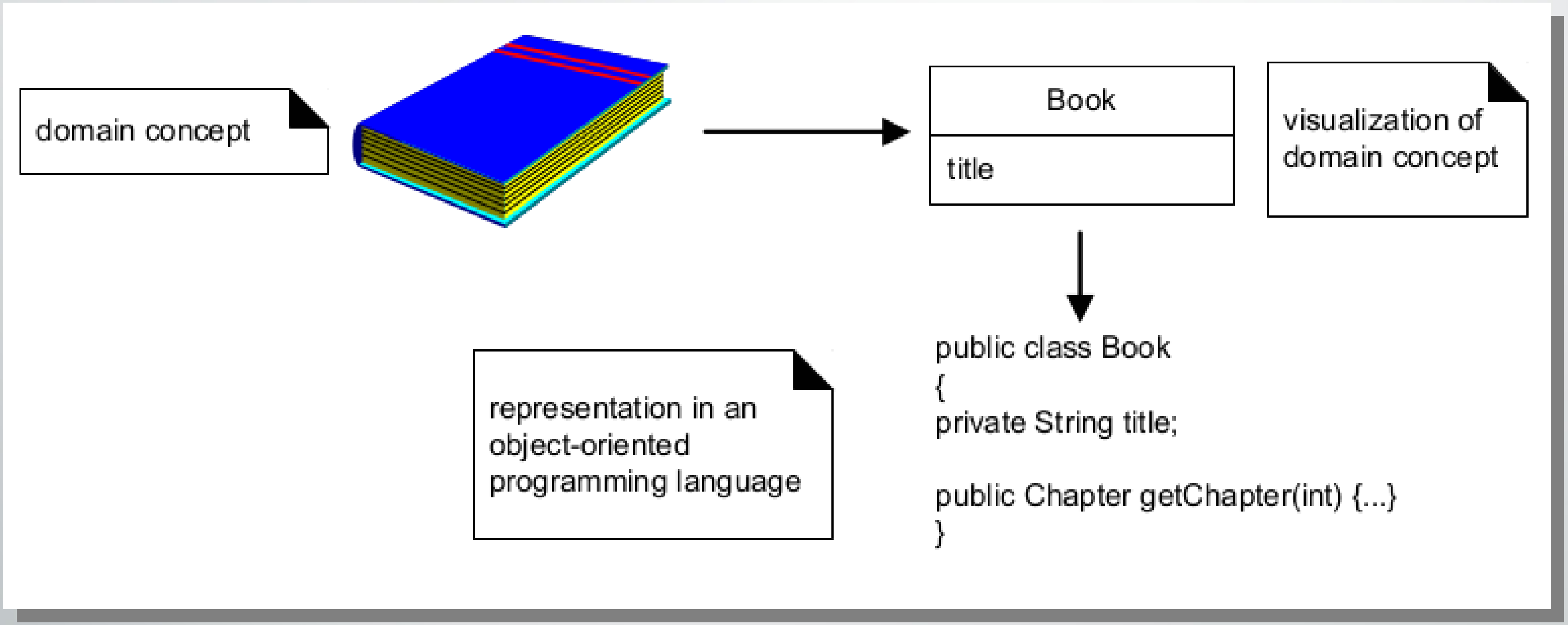
OO Technology

Process Perspective

OO Prog. Languages (C++, Java)	just program!
OO Design (Booch)	design then program
OO Analysis (Rumbaugh, Jacobson)	Analyze (use case) first, then design, then program

Where are we heading?

Overview OO Process


domain concept
Book
title
visualization of domain concept
representation in an object-oriented programming language

```
public class Book
{
private String title;

public Chapter getChapter(int) {...}
}
```

Analysis and design have been summarized in the phase **do the right thing** (analysis), and **do the thing right** (design).

OOAD

- Object Oriented Analysis and Design
- the art of developing systems based on a set of cooperating objects.
- This is not an exact science.
- a technical approach used in the analysis and design of an application or system through the application of the object-oriented paradigm and concepts including visual modeling.
- This is applied throughout the development life cycle of the application or system, fostering better product quality and even encouraging stakeholder participation and communication.

Object Oriented Analysis

- Analysis emphasizes an investigation of the problem and requirements, rather than a solution.
- "Analysis" is a broad term, best qualified, as in requirements analysis (an investigation of the requirements) or object analysis (an investigation of the domain objects).
- In the phase of OOA the typical question starts with What...? like
  “What will my program need to do?”,
  “What will the classes in my program be?” and
  “What will each class be responsible for?” .
- Hence, OOA cares about the real world and how to model this real world without getting into much detail.
- The OOA is as an investigation of the problem and requirements, rather than finding a solution to the problem.

Object Oriented Analysis

The primary tasks in object-oriented analysis (OOA) are:

- Identifying objects
- Organize the objects
- Defining the internals of the objects, or object attributes
- Defining the behavior of the objects, i.e., object actions
- Describing how the objects interact

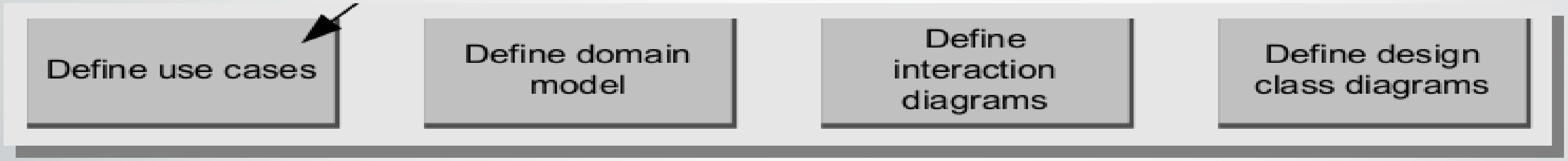
The common models used in OOA are use cases and object models.

Object Oriented Design

- Design emphasizes a conceptual solution that fulfills the requirements, rather than its implementation.
- For example, a description of a database schema and software objects. Ultimately, designs can be implemented.
- Design encompasses the disciplined approach we use to invent a solution for some problem, so design providing a path from requirements to implementation.
- In contrast, in the OOD phase, the question typically starts with How...? like “How will this class handle it’s responsibilities?”, “How to ensure that this class knows all the information it needs?” and
  “How will classes in the design communicate?” .
- The OOD phase deals with finding a conceptual solution to the problem – it is about fulfilling the requirements, but not about implementing the solution.

Defining Models

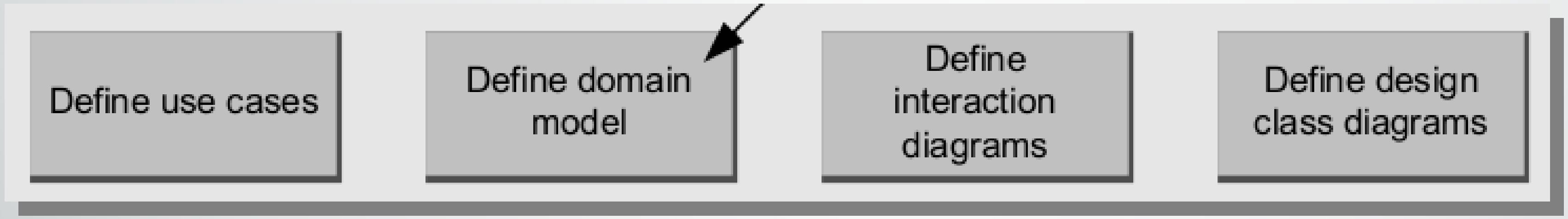
- A simple example—a "dice game" in which a player rolls two die. If the total is seven, they win; otherwise, they lose.

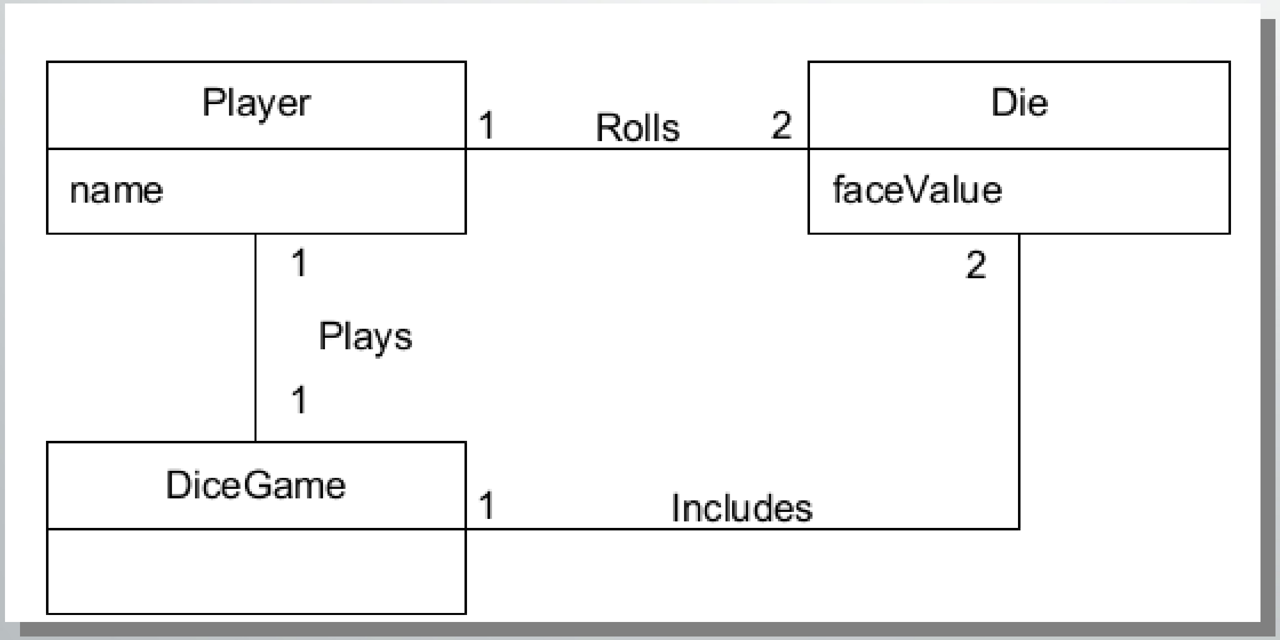


- Requirements analysis may include a description of related domain processes; these can be written as use cases.
- Play a Dice Game: A player **picks up** and **rolls the dice**. If the dice face value total seven, they win; otherwise, they lose.

Defining Models

- Object-oriented analysis is concerned with creating a description of the domain from the perspective of classification by objects.

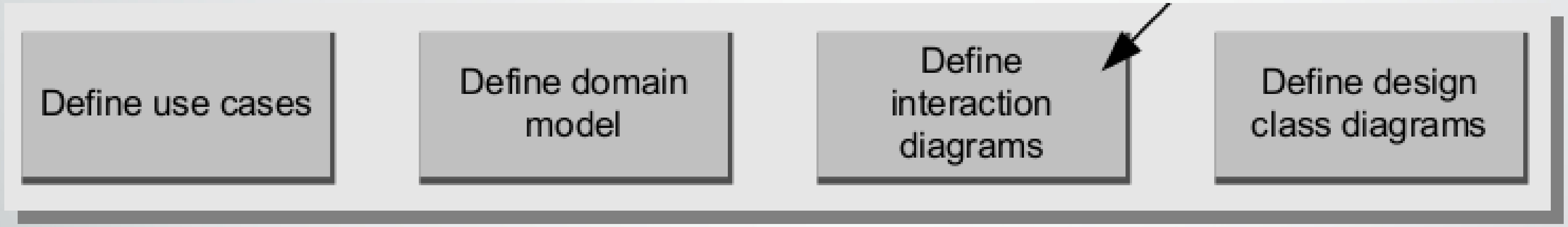


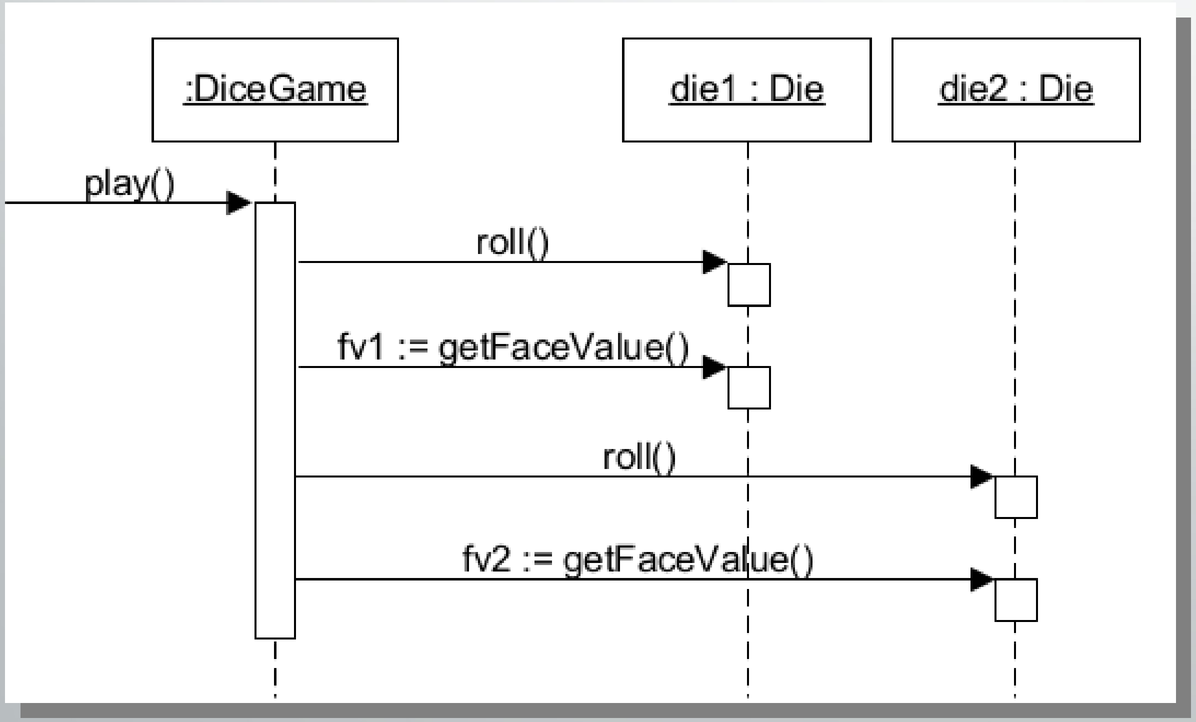


Domain Model:
> Conceptual Classes
> Association
> Attributes

Defining Models

- Object-oriented design is concerned with defining software objects and their collaborations.

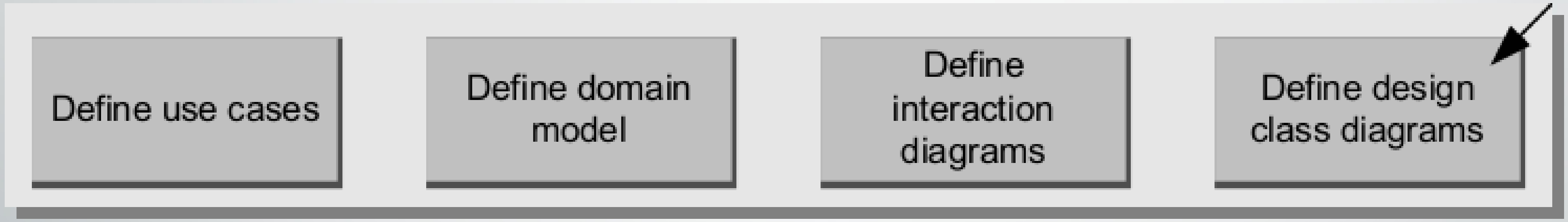


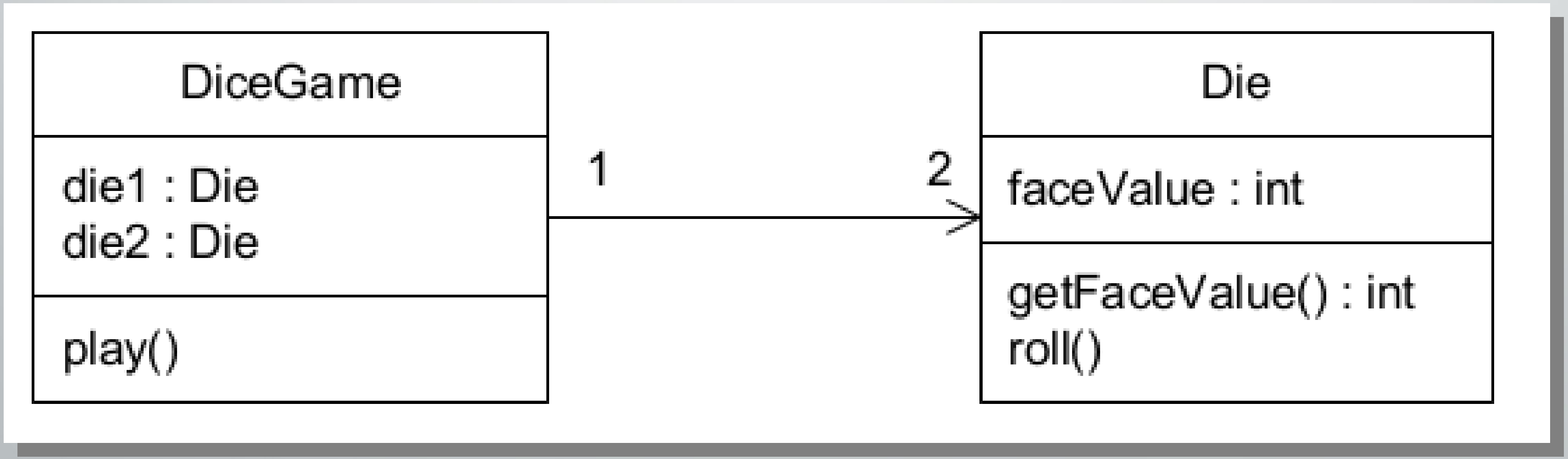


Interaction Diagrams:
> Sequence
> Communication
> Timing
> Interaction-Overview

Defining Models

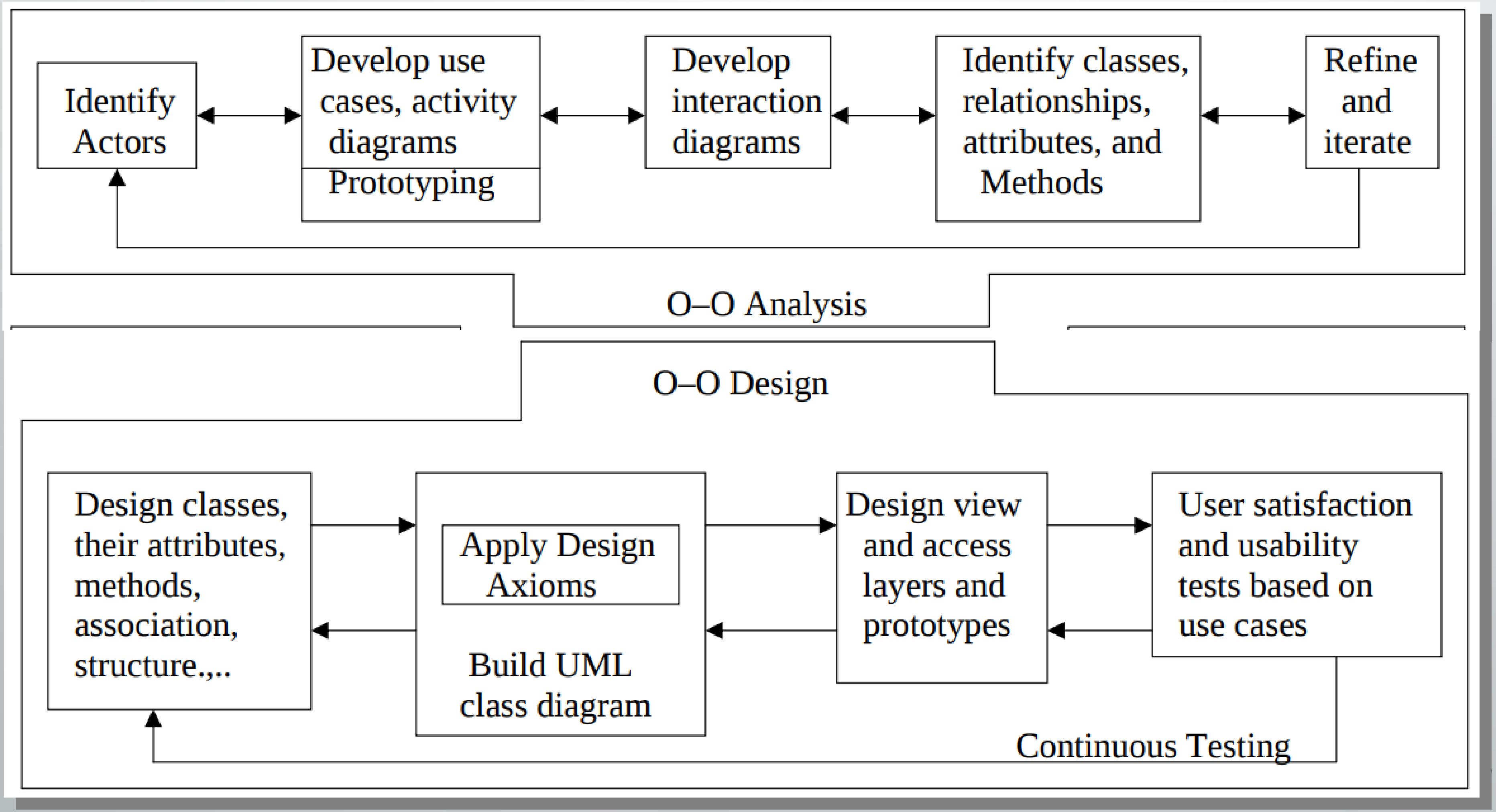
- A dynamic view of collaborating objects shown in interaction diagrams, it is useful to create a static view of the class definitions with a design class diagram.





OO Development Cycle

- The object oriented software development life cycle (SDLC) consists of three macro processes: object–oriented **analysis**, object–oriented **design** and object–oriented **implementation**.



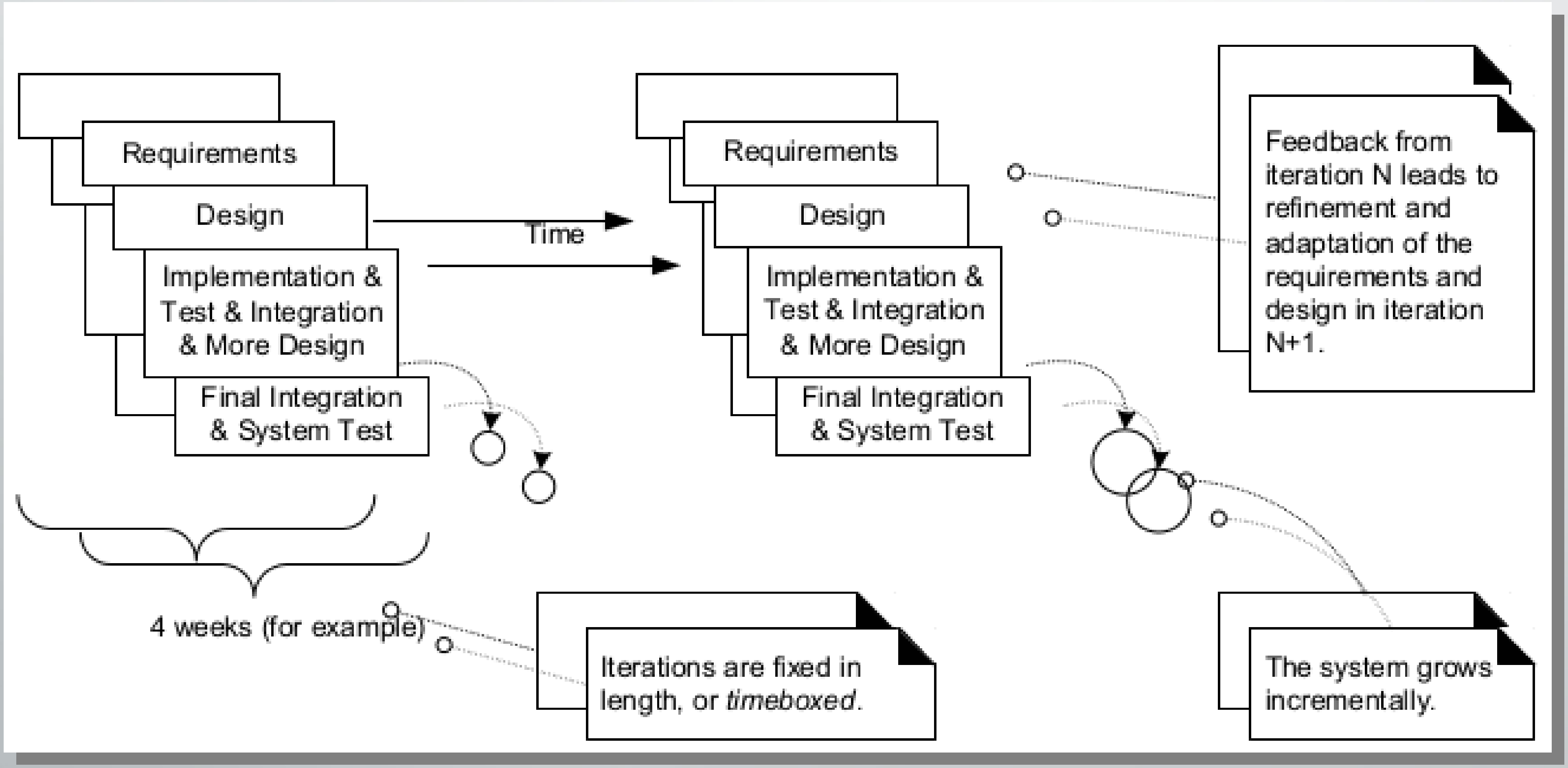
Identify Actors
Develop use cases, activity diagrams
Prototyping
Develop interaction diagrams
Identify classes, relationships, attributes, and Methods
Refine and iterate
O–O Analysis
O–O Design
Design classes, their attributes, methods, association, structure.,..
Apply Design Axioms
Build UML class diagram
Design view and access layers and prototypes
User satisfaction and usability tests based on use cases
Continuous Testing

OO Development Cycle

- object–oriented **analysis**
- object–oriented **design**
- **Iterative** Development
- **Continuous** Testing
- **UML** Modeling
- **Layered** Architecture:
  - → User Interface Layer
  - → Business Layer
  - → Access Layer

OO Development Cycle

Iterative and Incremental development

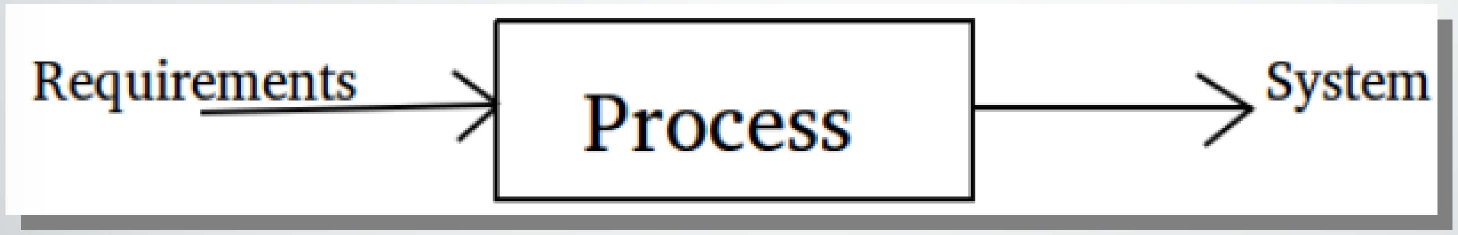


Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Time
Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.
4 weeks (for example)
Iterations are fixed in length, or *timeboxed*.
The system grows incrementally.

Unified Process

Process:

Process in OOAD stands for Software Engineering Process(SEP). Process defines who, what, when and how of developing software.



“People are more important than any process. Good people with a good process will outperform good people with no process every time.” —Grady Booch

Unified Process

- The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework.
- **Unified Process is an architecture-centric, use- case driven, iterative and incremental development process that leverages unified modeling language and is compliant with the system process engineering meta-model**.
- The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP).
- Other examples are OpenUP and Agile Unified Process.
- A Unified process (UP) is a popular iterative modern process model (framework) derived from the work on the UML and associated process.

Unified Process

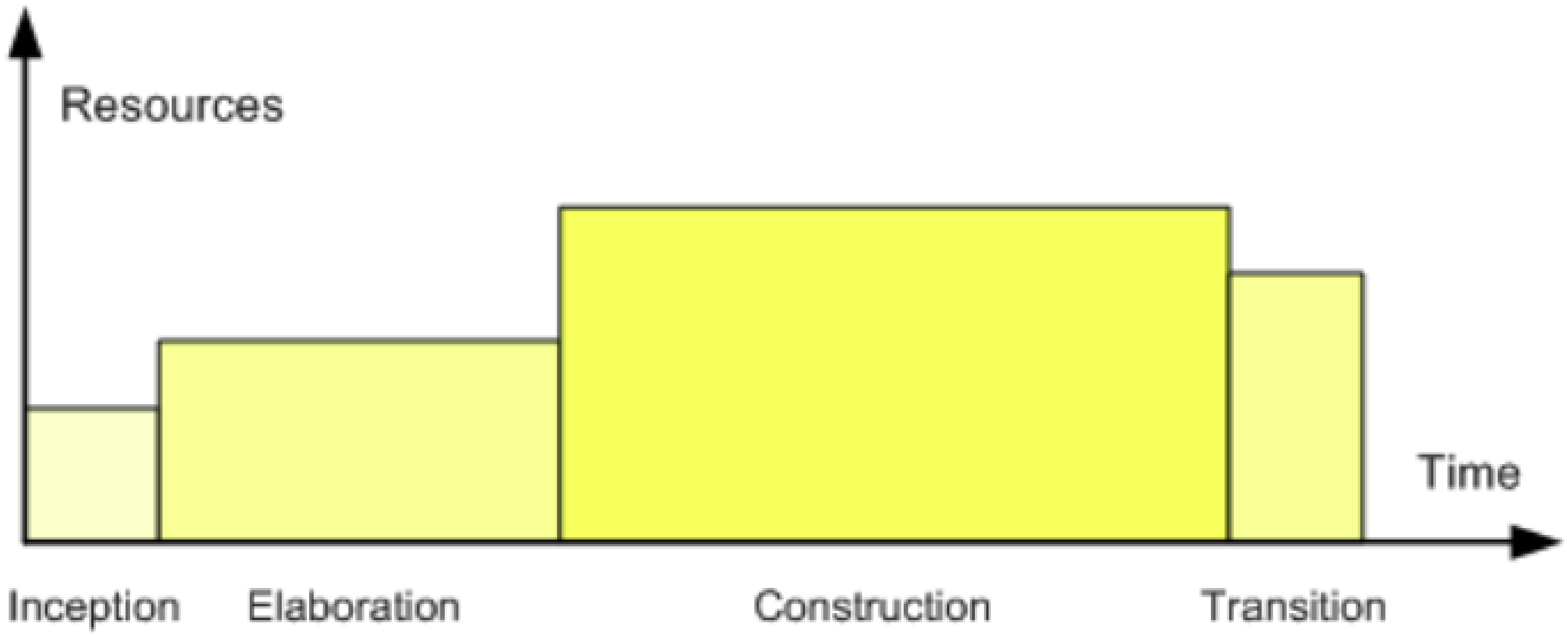


figure: UP phases – Resource-Time diagram

Unified Process

- The Unified Process is a modeling technique.
- The leading object-oriented methodology for the development of large-scale software
- Maps out when and how to use the various UML techniques
- The Unified Process is an adaptable methodology.
- Iterative development is a skillful approach to software development, and lies at the heart of how OOA/D is presented.
- The Unified Process is an example iterative process for projects using OOA/D.
- The Unified Process (UP) combines commonly accepted best practices, such as an iterative lifecycle and risk-driven development, into a cohesive and well-documented description.

Unified Process

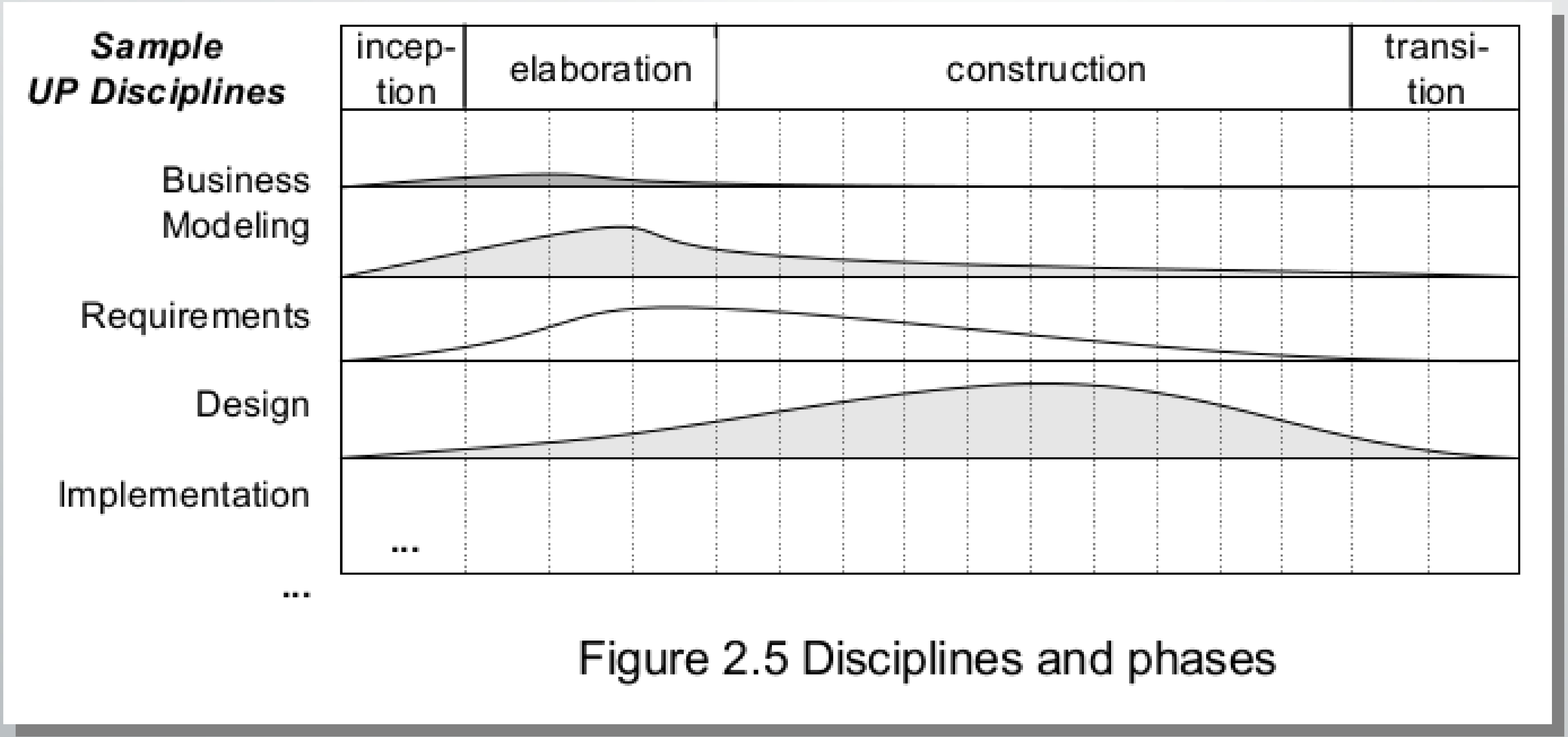


Figure 2.5 Disciplines and phases

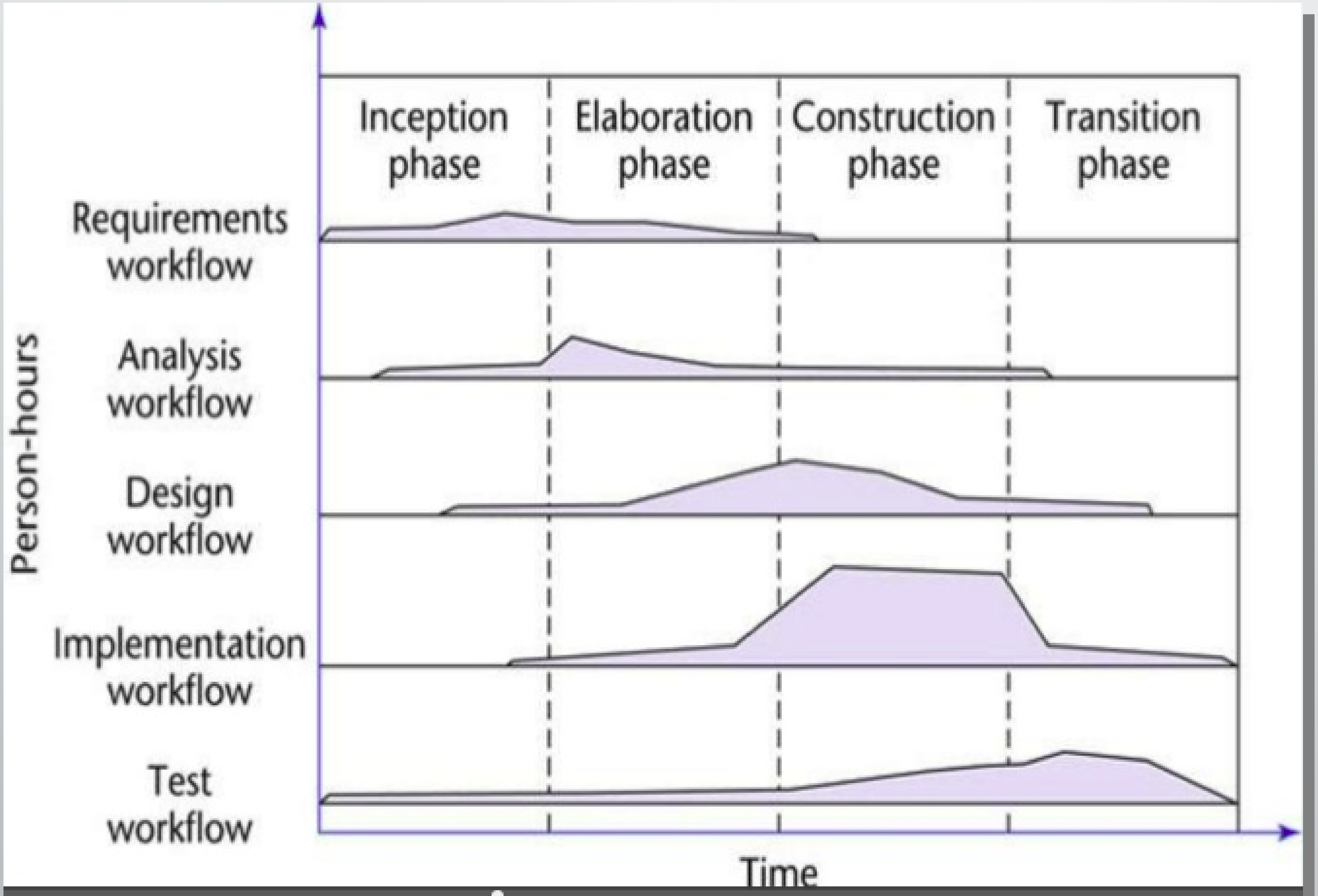
Unified Process

Phase vs Workflow

Phase: is business context of a step

Workflow: is technical context of a step

Unified Process



UP Phases

1. Inception
[approximate vision, business case, scope, vague estimates.]

- In this phase, we develop an approximate vision of the system, make the business case,
- define the scope, and produce rough estimates for cost and schedule.

2. Elaboration
[refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope,more realistic estimates.]

- In this phase, we refine the vision, identify and describe all requirements, finalize the scope, design and implement the core architecture and functions, resolve high risks, and produce realistic estimate for cost and schedule.

UP Phases

3. Construction
[iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.]

- In this phase, implementation of lower-risk, predictable, and easier elements and also the preparation for the development is done.

4. Transition
[beta tests, deployment.]

- In this phase, completion of the beta test and deployment is done so users have a working system and are ready to benefit as expected.

UP Phases

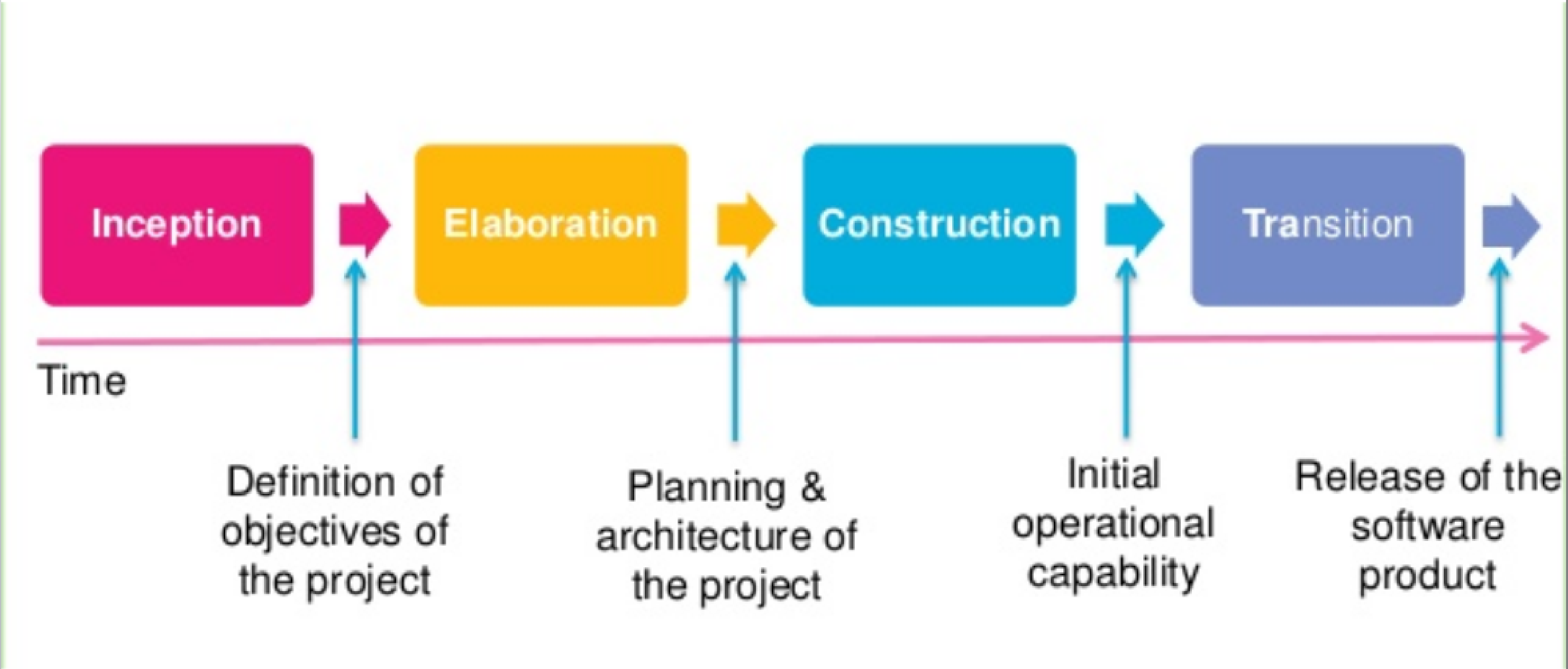

Inception
Elaboration
Construction
Transition
Time
Definition of objectives of the project
Planning & architecture of the project
Initial operational capability
Release of the software product

Figure: UP-Phases outcomes

UNIFIED
MODELING
LANGUAGE

UML™

Unified Modeling Language

- The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.
- UML was originally motivated by the desire to standardize the disparate notational systems and approaches to software design developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in 1994–1995, with further development led by them throug 1996.
- In 1997 UML was adopted as a standard by the Object Management Group(OMG), and has been managed by this organization ever since. In 2005 UML was also published by the International Organization for Standardization (ISO) as an approved ISO standard. [2] Since then it has been periodically revised to cover the latest revision of UML.

UML

- ✓ a . What is UML?
- ✓ b . Why UML?
- ✓ c . UML history
- ✓ d . UML versions

a. What is UML?

The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

The Unified Modeling Language, or the UML, is a graphical modeling language that provides us with a syntax for describing the major elements (called artifacts in the UML) of software systems. The UML is a graphical language for capturing the artifacts of software developments. The language is very rich, and carries with it many aspects of Software Engineering best practice.

b. Why UML?

Other industries have languages and notations, which are understood by every member of that particular field.

$$\int_{0}^{\infty} {}^{1}\!/_{x^{2}}\,dx$$

Figure 1 - A Mathematical Integral

Although the picture above is a fairly simple drawing (a stylized "S" figure), mathematicians the world over recognize instantly that I am representing an integral. Although this notation is simple, it masks a very deep and complicated topic (though perhaps not as deep as the concept represented by the figure of eight on its side!) So the notation is simple, but the payoff is that mathematicians all around the world can clearly and unambiguously communicate their ideas using this, and a small collection of other symbols. Mathematicians have a common language. So do musicians, electronic engineers, and many other disciplines and professions.

b. Why UML?(cont.)

To date, Software Engineering has lacked such a notation. Between 1989 and 1994, a period referred to as the “method wars”, more than 50 software modeling languages were in common use – each of them carrying their own notations! Each language contained syntax peculiar to itself, whilst at the same time, each language had elements which bore striking similarities to the other languages. To add to the confusion, no one language was complete, in the sense that very few software practitioners found complete satisfaction from a single language!

c. UML History

In the mid 1990’s, three methods emerged as the strongest. These three methods had begun to converge, with each containing elements of the other two. Each method had its own particular strengths:

Booch was excellent for design and implementation. Grady Booch had worked extensively with the Ada language, and had been a major player in the development of Object Oriented techniques for the language. Although the Booch method was strong, the notation was less well received (lots of cloud shapes dominated his models - not very pretty!)

OMT (Object Modelling Technique) was best for analysis and data-intensive information systems.

OOSE (Object Oriented Software Engineering) featured a model known as Use Cases. Use Cases are a powerful technique for understanding the behaviour of an entire system (an area where OO has traditionally been weak).

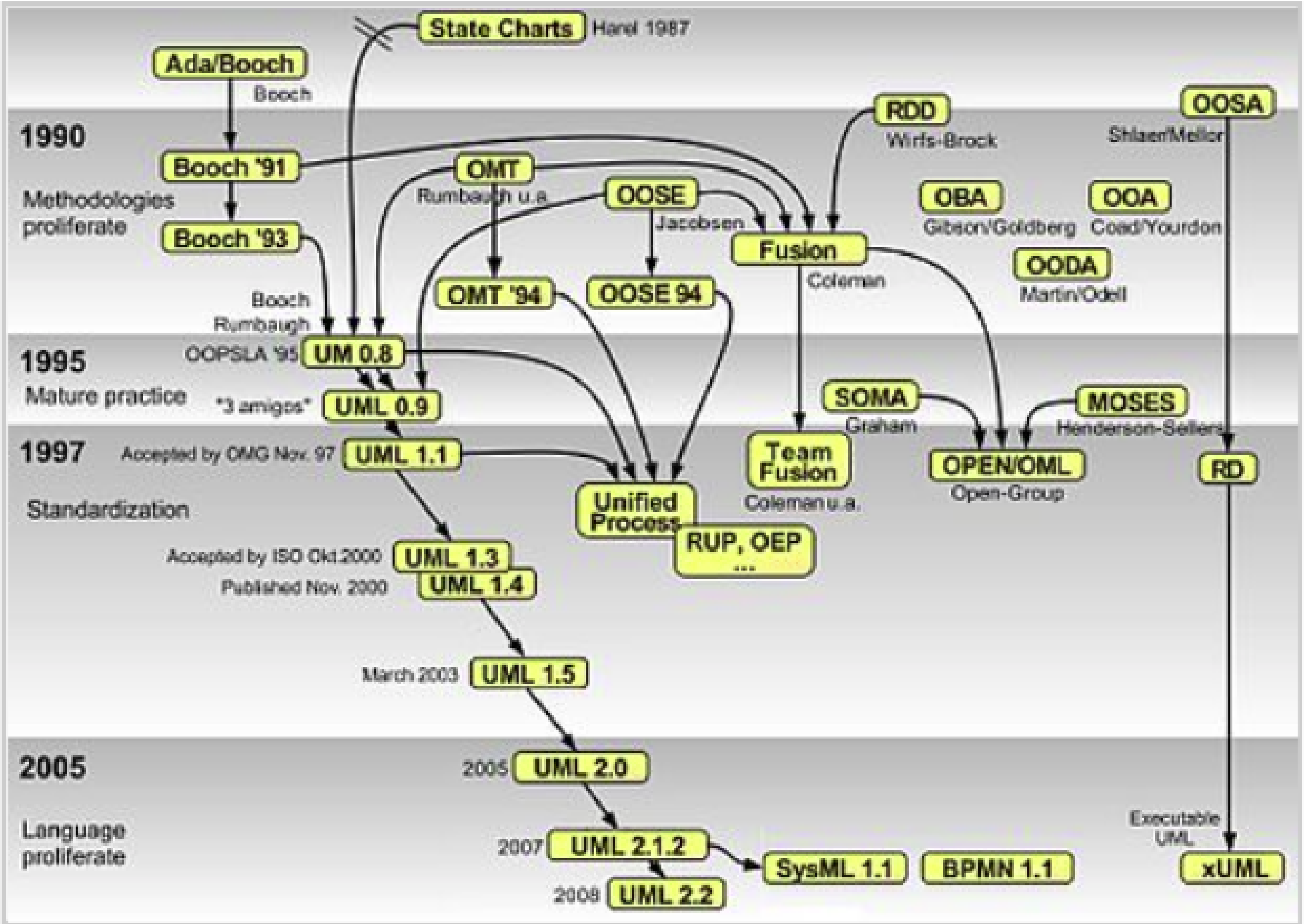
c. UML History(cont.)

In 1994, Jim Rumbaugh, the creator of OMT, stunned the software world when he left General Electric and joined Grady Booch at Rational Corp. The aim of the partnership was to merge their ideas into a single, unified method (the working title for the method was indeed the "Unified Method").

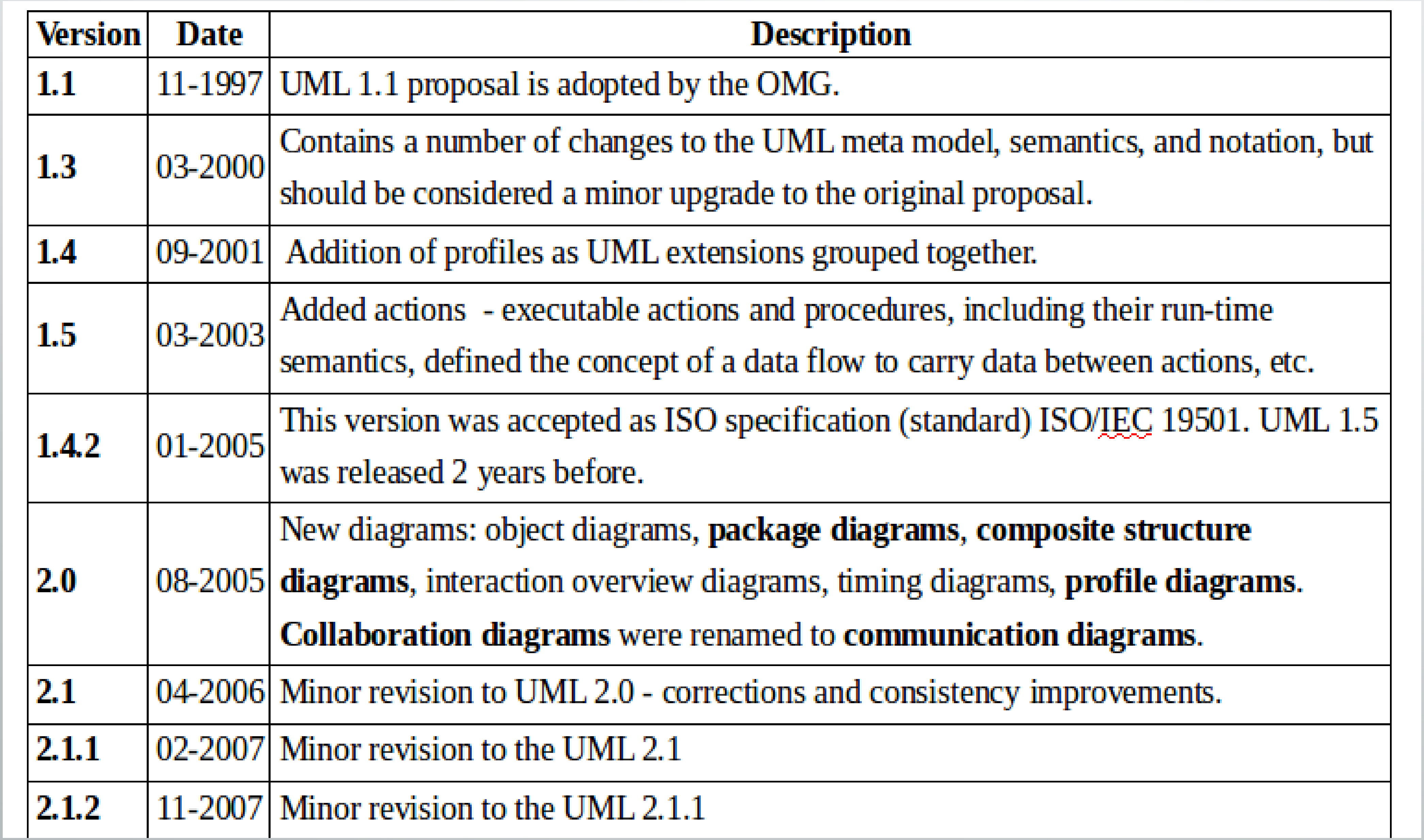
By 1995, the creator of OOSE, Ivar Jacobson, had also joined Rational, and his ideas (particularly the concept of "Use Cases") were fed into the new Unified Method – now called the Unified Modelling Language. The team of Rumbaugh, Booch and Jacobson are affectionately known as the "Three Amigos".

Despite some initial wars and arguments, the new method began to find favour amongst the software industry, and a UML consortium was formed. Heavyweight corporations were part of the consortium, including Hewlett-Packard, Microsoft and Oracle.

The UML was adopted by the OMG in 1997, and since then the OMG have owned and maintained the language. Therefore, the UML is effectively a public, non-proprietary language.

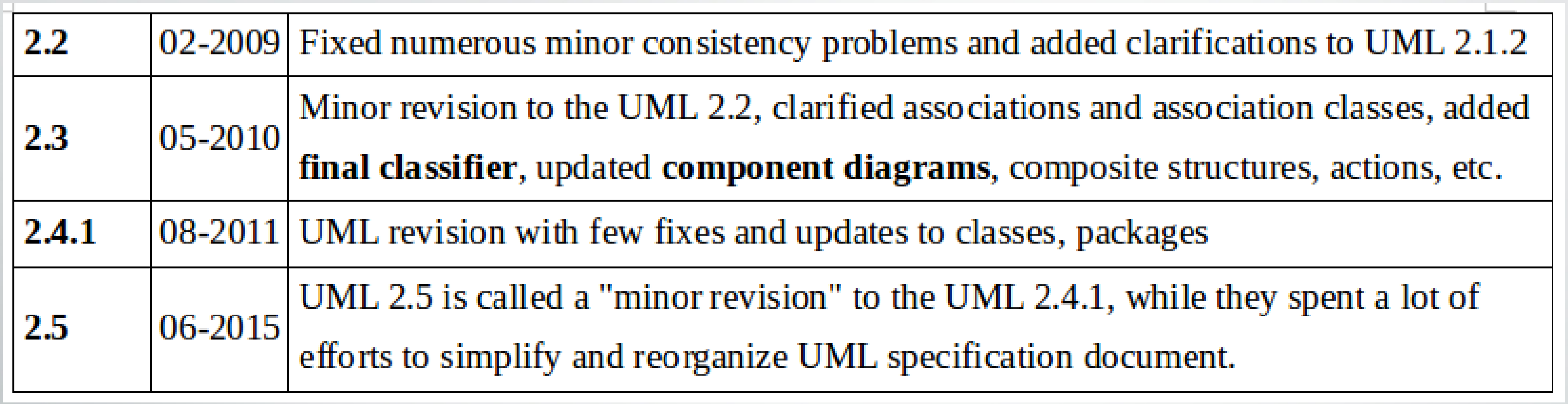

State Charts
Harel 1987
Ada/Booch
Booch
1990
Booch '91
Methodologies proliferate
Booch '93
OMT
Rumbaugh u.a.
OOSE
Jacobsen
RDD
Wirfs-Brock
OOSA
Shlaer/Mellor
OBA
Gibson/Goldberg
OOA
Coad/Yourdon
Fusion
Coleman
OODA
Martin/Odell
OMT '94
OOSE 94
Booch
Rumbaugh
1995
OOPSLA '95
UM 0.8
Mature practice
"3 amigos"
UML 0.9
SOMA
Graham
MOSES
Henderson-Sellers
1997
Accepted by OMG Nov. 97
UML 1.1
Team Fusion
Coleman u.a.
OPEN/OML
Open-Group
RD
Standardization
Unified Process
RUP, OEP ...
Accepted by ISO Okt.2000
UML 1.3
Published Nov. 2000
UML 1.4
March 2003
UML 1.5
2005
2005
UML 2.0
Language proliferate
Executable UML
2007
UML 2.1.2
SysML 1.1
BPMN 1.1
xUML
2008
UML 2.2

d. UML Versions



Version	**Date**	**Description**
1.1	11-1997	UML 1.1 proposal is adopted by the OMG.
1.3	03-2000	Contains a number of changes to the UML meta model, semantics, and notation, but should be considered a minor upgrade to the original proposal.
1.4	09-2001	Addition of profiles as UML extensions grouped together.
1.5	03-2003	Added actions - executable actions and procedures, including their run-time semantics, defined the concept of a data flow to carry data between actions, etc.
1.4.2	01-2005	This version was accepted as ISO specification (standard) ISO/IEC 19501. UML 1.5 was released 2 years before.
2.0	08-2005	New diagrams: object diagrams, **package diagrams**, **composite structure diagrams**, interaction overview diagrams, timing diagrams, **profile diagrams**. **Collaboration diagrams** were renamed to **communication diagrams**.
2.1	04-2006	Minor revision to UML 2.0 - corrections and consistency improvements.
2.1.1	02-2007	Minor revision to the UML 2.1
2.1.2	11-2007	Minor revision to the UML 2.1.1

d. UML Versions(cont.)



2.2	02-2009	Fixed numerous minor consistency problems and added clarifications to UML 2.1.2
2.3	05-2010	Minor revision to the UML 2.2, clarified associations and association classes, added **final classifier**, updated **component diagrams**, composite structures, actions, etc.
2.4.1	08-2011	UML revision with few fixes and updates to classes, packages
2.5	06-2015	UML 2.5 is called a "minor revision" to the UML 2.4.1, while they spent a lot of efforts to simplify and reorganize UML specification document.

Notation and basic building blocks of UML

The vocabulary of the UML encompasses three kinds of building blocks:

- ✓ a. Things/ Elements
- ✓ b. Relationships
- ✓ c. Diagrams

Generalization, Intension and Extension

a. Elements of UML

There are four kind of things in the UML…

- **Structural Things**
- **Behavioral Things**
- **Grouping Things**
- **Annotational Things**

These things are the basic object-oriented building blocks of the UML. You use them to write well-formed models.

a. Elements of UML(cont.)
[Structural Things]

These are nouns of UML models. These are the mostly static parts of a model, representing elements that are either conceptual or physical.

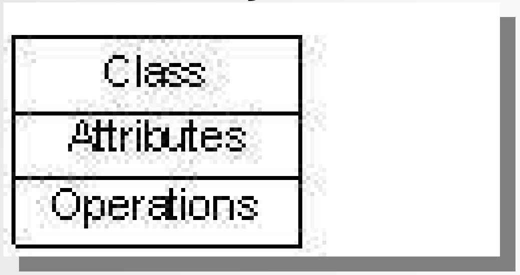
There are seven kind of structural things:

1. Class
2. Interface
3. Collaboration
4. Use Case
5. Active Class
6. Component
7. Node

a. Elements of UML(cont.) [Structural Things]

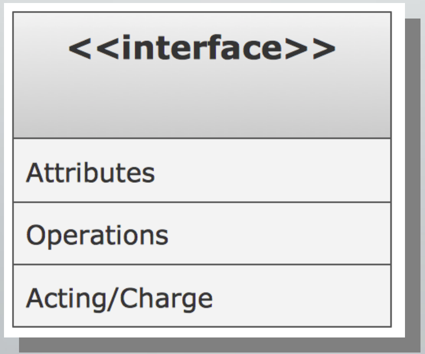
1. Class:

Class represents set of objects having similar responsibilities.



2. Interface:

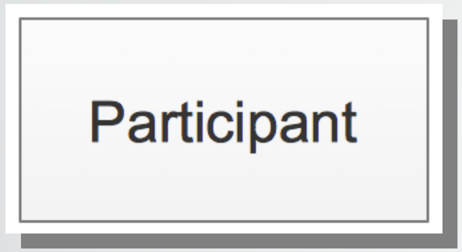
Interface is a collection of operations that specify a service of a class or components. An interface therefore describes the externally visible behavior of that elements. It represents a complete behavior of a class or component or only a part of that behavior.

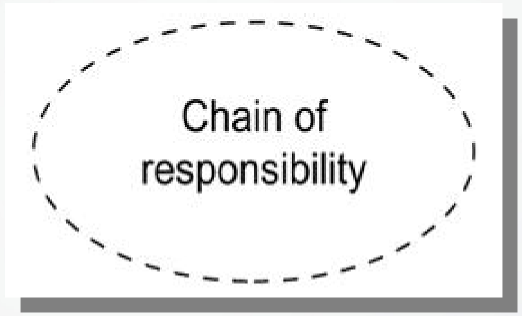


a. Elements of UML(cont.) [Structural Things]

3. Collaboration:

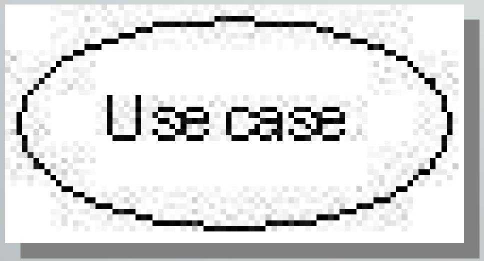
Collaboration defines interaction between elements.

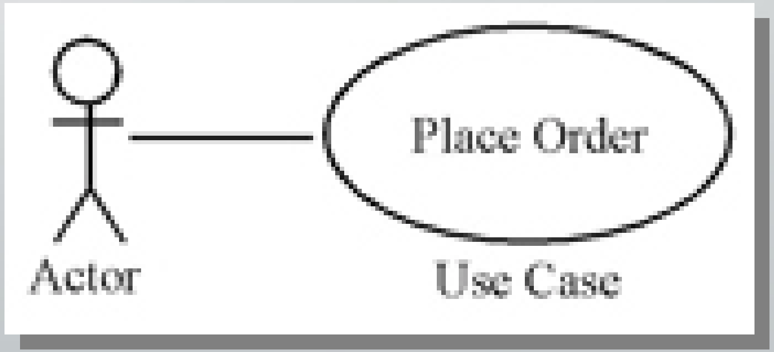




4. Use case:

Use case represents a set of actions performed by a system for a specific goal. A sequence of actions that a system performs that yields an observable result.

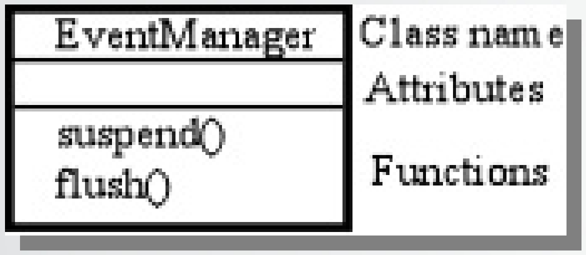


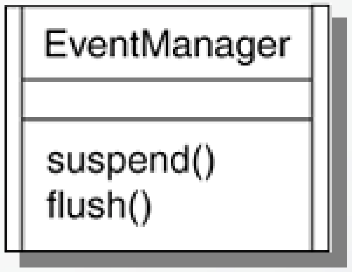


a. Elements of UML(cont.) [Structural Things]

5. Active Class:

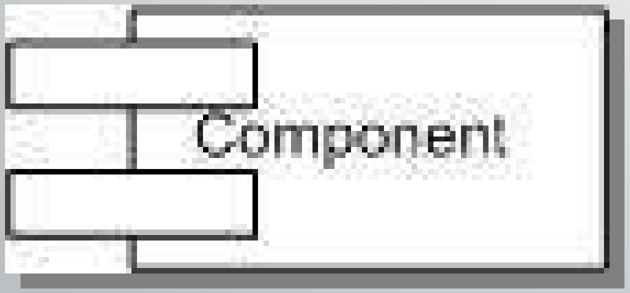
Like a class but its represents behavior that runs concurrent with other behaviors, i.e. threading.





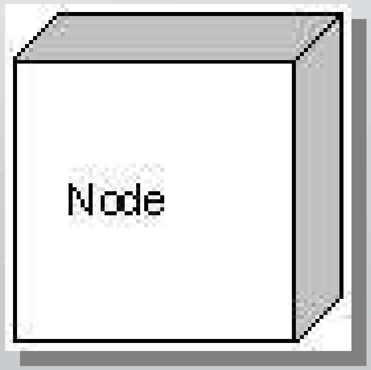
6. Component:

Component describes physical part of a system.



7. Node:

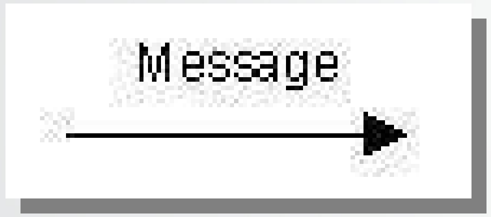
A node can be defined as a physical element that exists at run time.

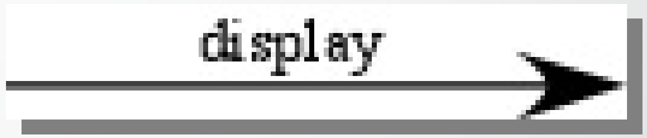


a. Elements of UML(cont.) [Behavioral Things]

1. Interaction -

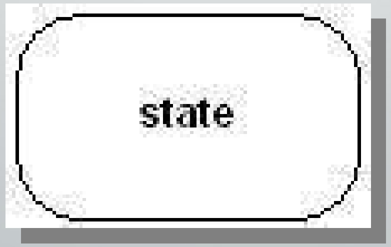
A behavior made up of a set of messages exchanged among a set of objects in a particular context to accomplish a specific purpose.

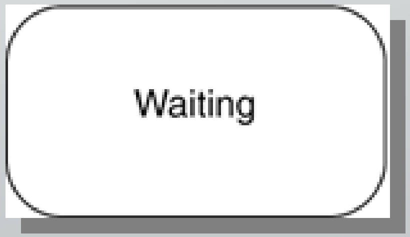




2. State machine:

State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change.

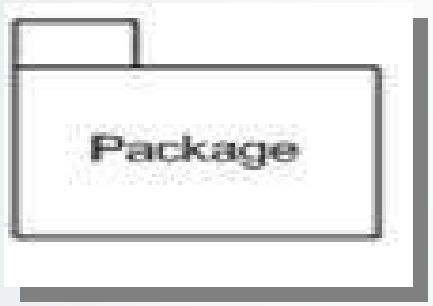


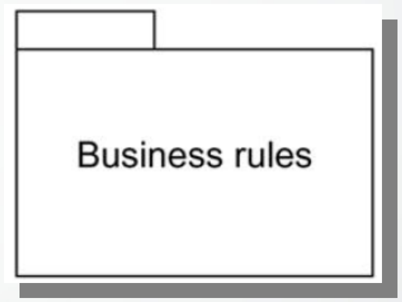


a. Elements of UML(cont.) [Grouping Things]

Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available:

Package:



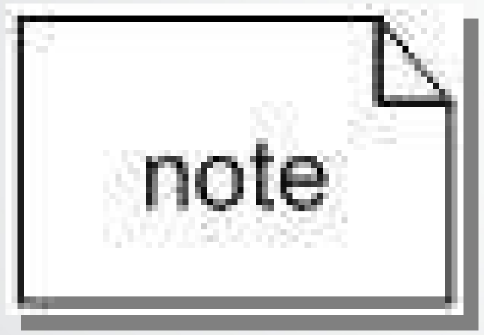


Package is the only one grouping thing available for gathering structural and behavioral things. Package is a general purpose machine for organizing elements into groups. Structural things, behavioral things, and even other grouping things may be placed in a package.

a. Elements of UML(cont.) [Annotational Things]

Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. Note is the only one Annotational thing available.

Note:



A note is used to render comments, constraints etc. of an UML element.

b. Relationship

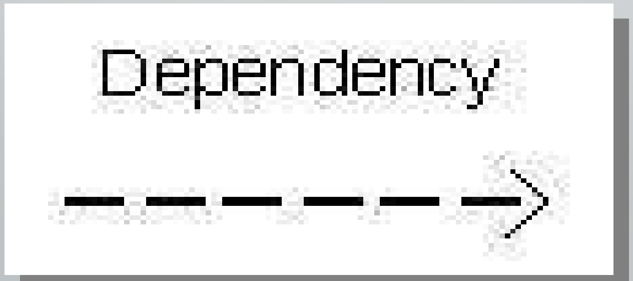
Relationship is another most important building block of UML. It shows how elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships available. They are as follows:

- Dependency
- Association
- Generalization
- Realization

- **Dependency**

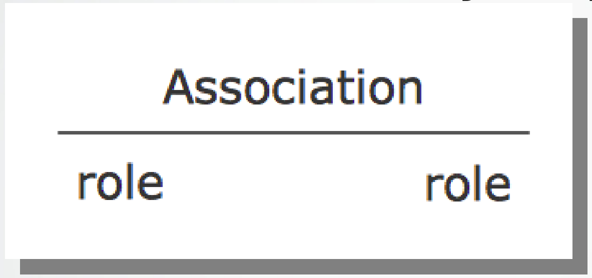
  Dependency is a relationship between two things in which change in one element also affects the other one.



b. Relationship(cont.)

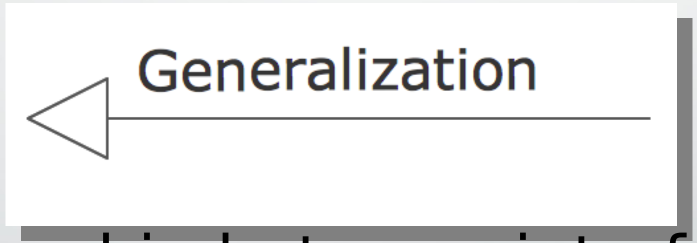
- **Association**

  Association is basically a set of links that connects elements of an UML model. It also describes how many objects are taking part in that relationship.



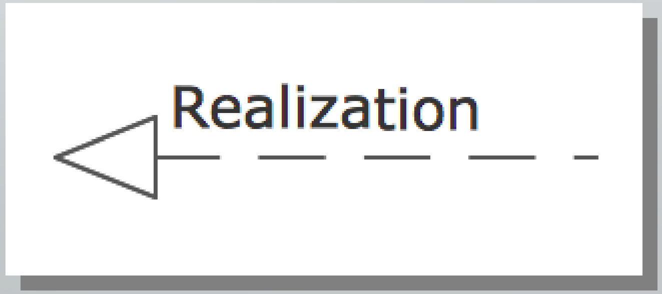
- **Generalization**

  Generalization is an association between the more general classifier and the more special classifier.



- **Realization**

  Realization is a relationship between interfaces and classes or components that realize them.



c. UML Diagram

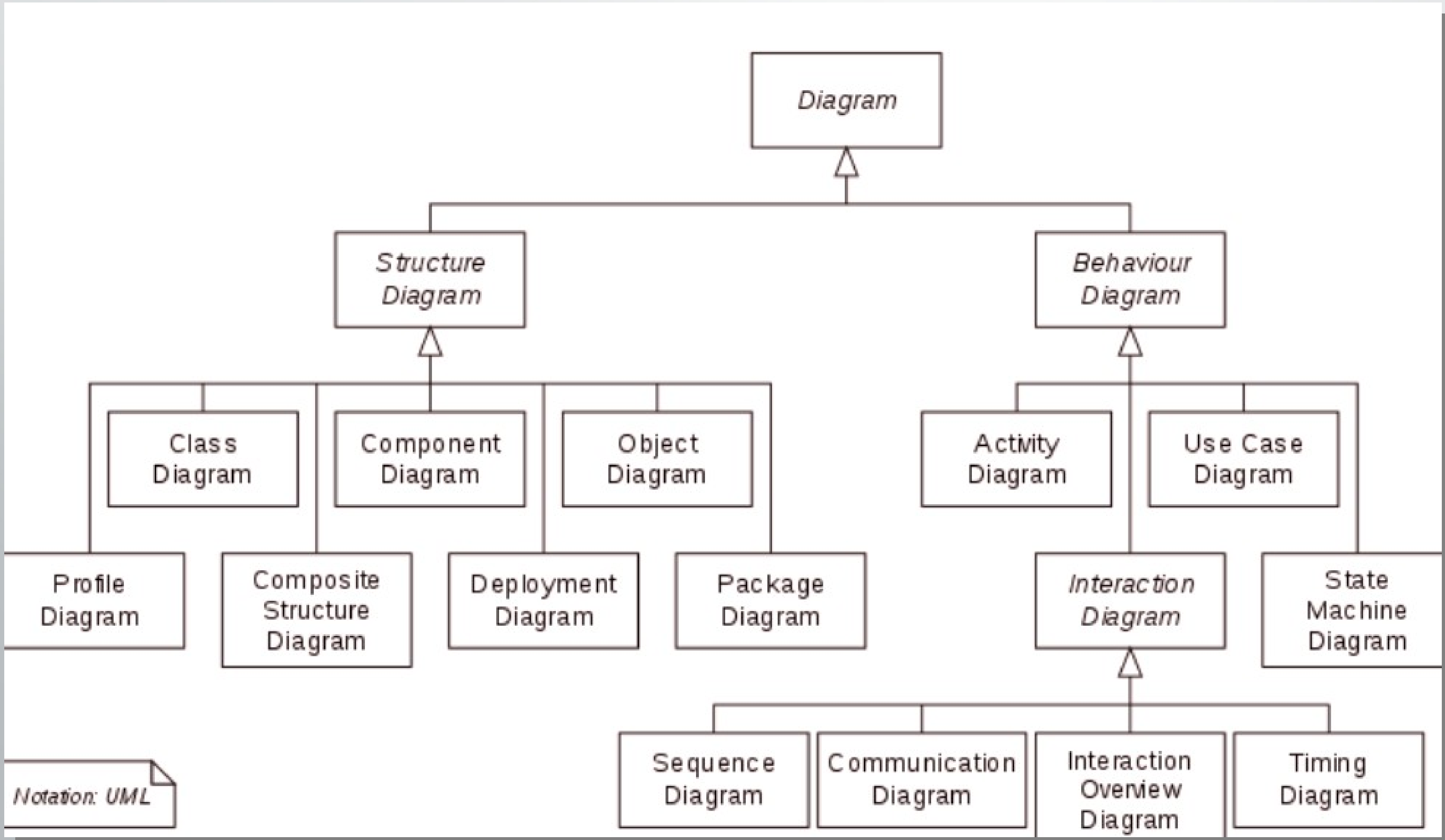
The UML models depict how the classes and objects in a system interact with one another. "UML models are diagrams of three primary types: structural, behavioral, and interaction. Generally, structural diagrams define the underlying software system (the code), behavioral diagrams describe what happens in the system under certain conditions, and interaction diagrams explain control flow." UML diagrams are used to provide a graphical representation of the system being modeled. UML 2.0 defines thirteen diagrams that are broadly classified into three categories with each category containing one or more diagrams that fall under that category.

These categories are:

- **The Structural Diagrams**
- **The Behavioral Diagrams**
- **The Interaction Diagrams**

The next section discusses these categories and shows the UML diagrams that comprise each of these categories.

UML Diagrams

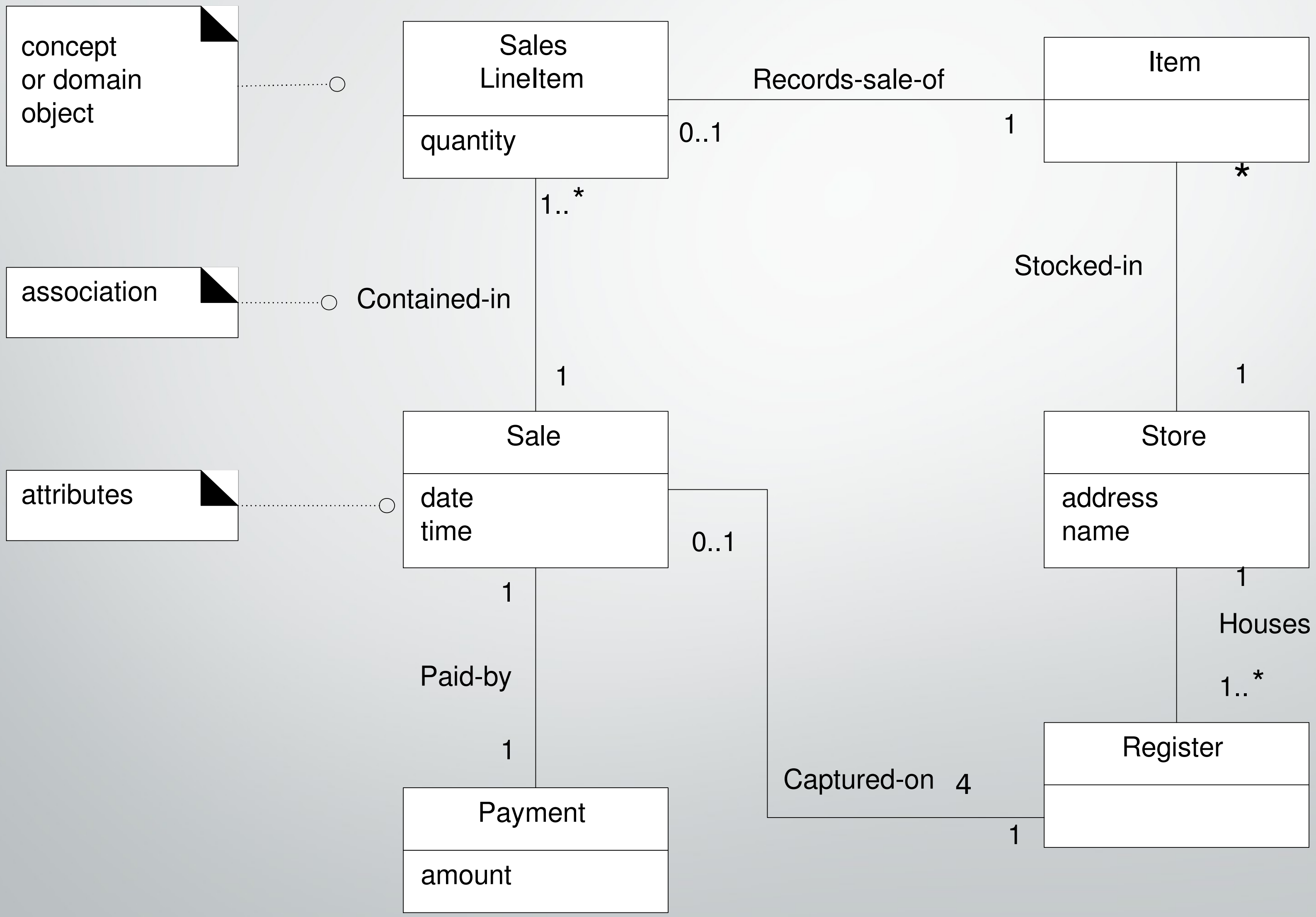


c. UML Diagram [The Structural Diagrams]

These relate to the static structure of a system, i.e., they represent elements that are static in nature. These diagrams are fundamental to the UML modeling of a system and portray the static structure of the system as a whole. The Structural Diagrams are comprised of the following.

- The Class diagram
- The Component diagram
- The Composite Structure diagram
- The Deployment diagram
- The Object diagram
- The Package diagram
- The Profile diagram

Example: Domain Model

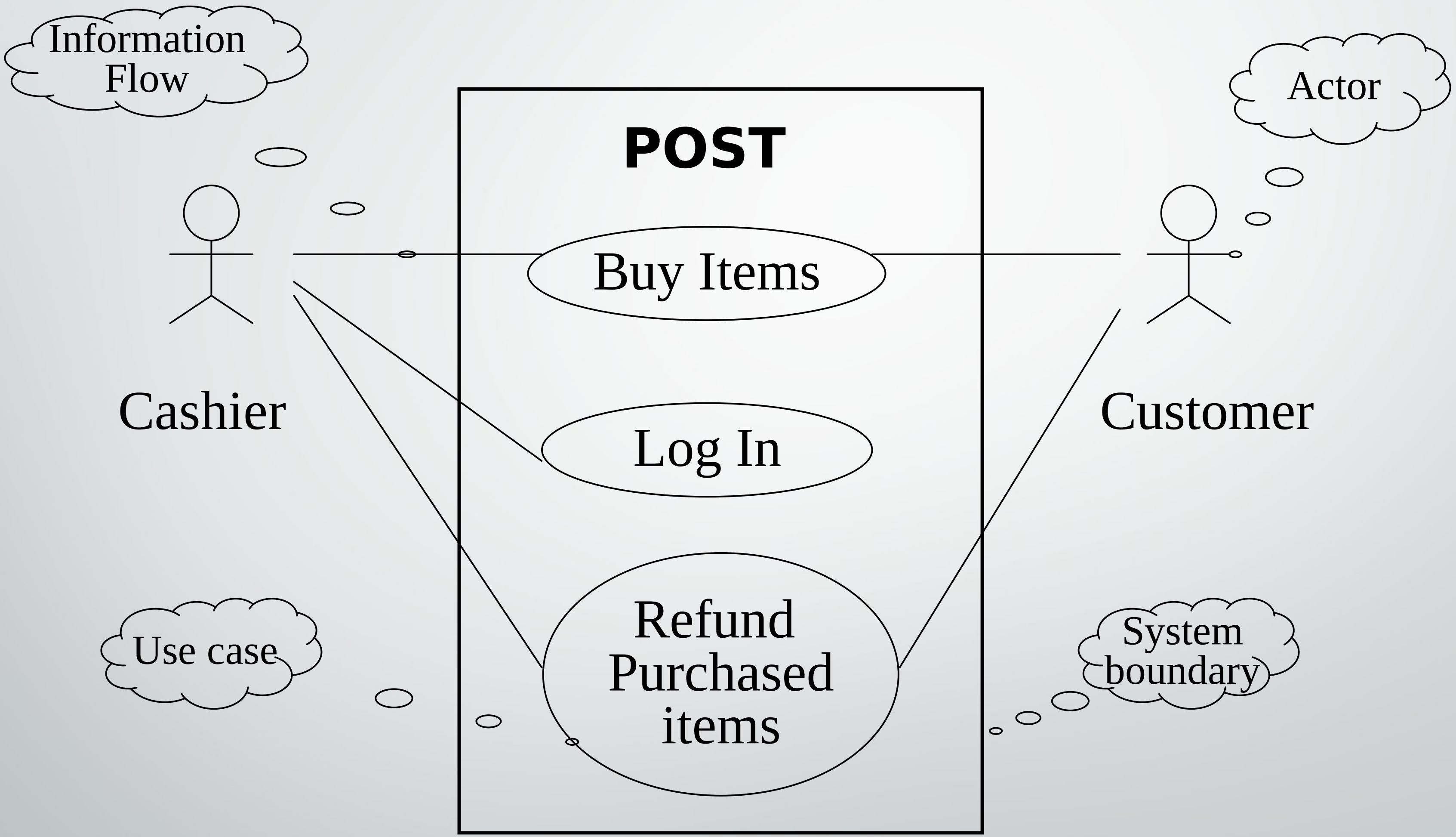


c. UML Diagram
[The Behavioral Diagrams]

The Behavioral Diagrams model how the system functions. The following are examples of Behavioral Diagrams.

- Use Case Diagram
- Activity Diagram
- State Machine Diagram

Example: Use Case Diagram



c. UML Diagram
[The Interaction Diagrams]

These diagrams represent how flow of data and control takes place in the system that is being modeled. They are a subset of the Behavioral Diagrams. The examples in this category are:

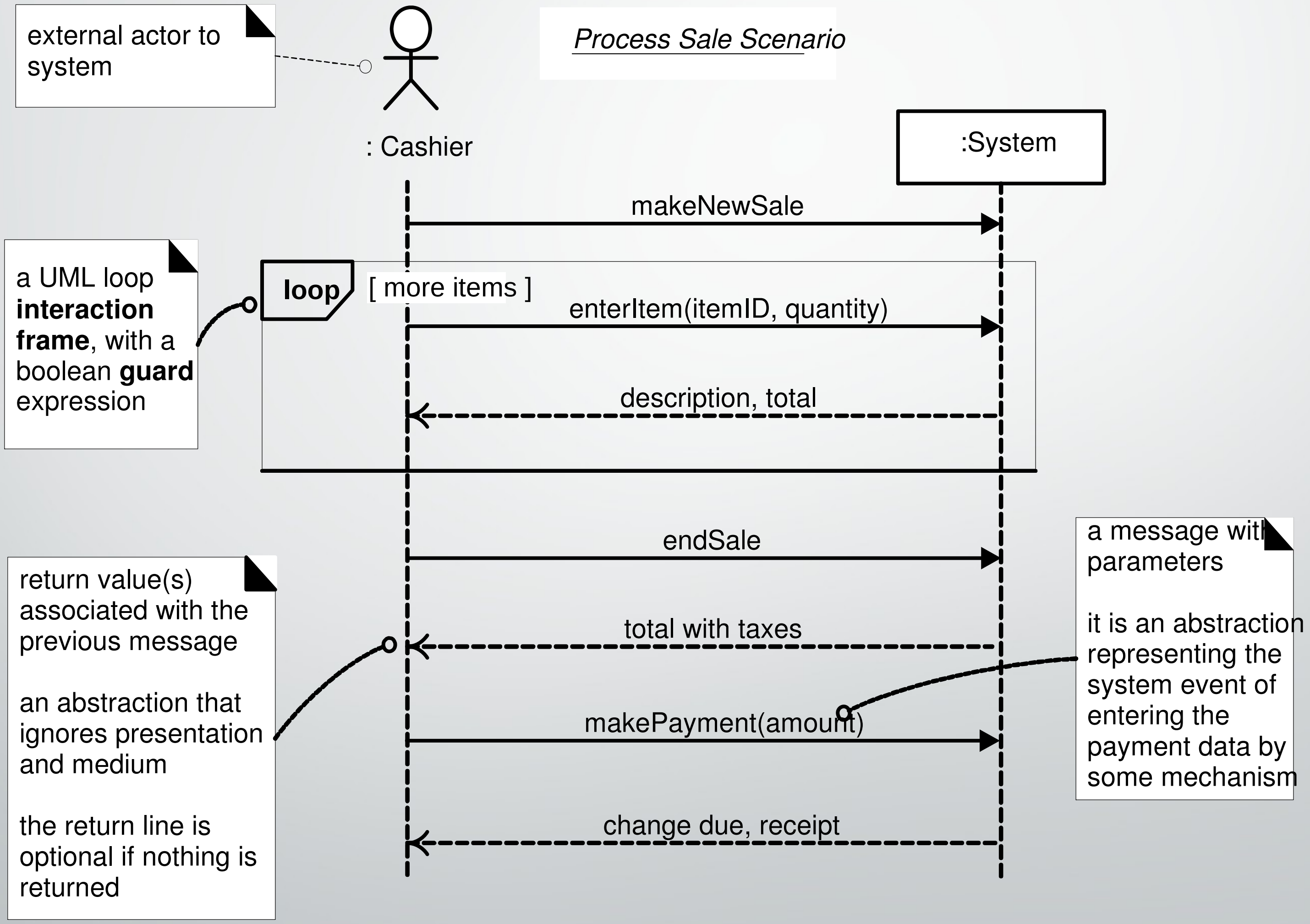
· Communication Diagram

· Sequence Diagram

· UML Timing Diagram

· Interaction Overview Diagram

Example: Sequence Diagram



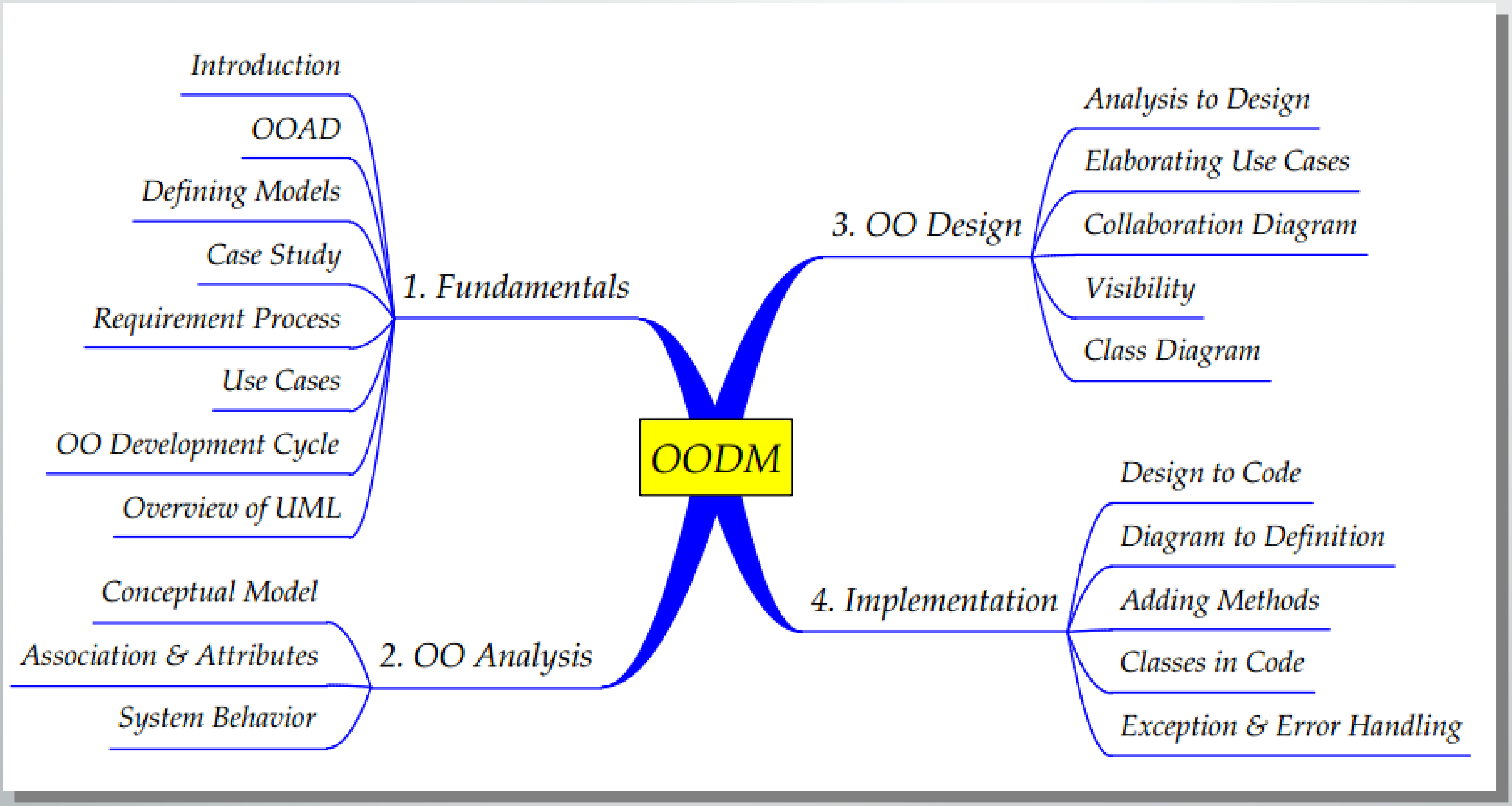
external actor to system
Process Sale Scenario
: Cashier
:System
makeNewSale
a UML loop **interaction frame**, with a boolean **guard** expression
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale
a message with parameters

it is an abstraction representing the system event of entering the payment data by some mechanism
return value(s) associated with the previous message

an abstraction that ignores presentation and medium

the return line is optional if nothing is returned
total with taxes
makePayment(amount)
change due, receipt

Course Content – Abstract



Text Book: **Applying UML and Patterns** by Craig Larman