COURSE NAME

OBJECT ORIENTED ANALYSIS AND DESIGN

CSC 2210

(UNDERGRADUATE)

- ❑ COURSE ORIENTATION AND GENERAL DISCUSSION
- ❑ CHAPTER 1

  WHY DO WE MODEL THINGS?

VICTOR STANY ROZARIO

Assistant Professor And Special Assistant

Department Of Computer Science, AIUB

Web: https://cs.aiub.edu/profile/stany

RG Google Scholar LinkedIn

LECTURE OUTLINE

- *General Discussion and Course Orientation*
- *Course Objectives*
- *Course Requirements*
- *Text Books & References*
- *Assessments Criteria and Marks Distribution*
- *Classroom and grading policies*

CONTACT DETAILS

Victor Stany Rozario

Assistant Professor and Special Assistant

Department of Computer Science

FST, AIUB

Email – *stany@aiub.edu*

Office – DNGA02

OBJECTIVES

- *Explain the necessity of formal modelling techniques in system development*
- *Describe system analysis and design using object-oriented concepts and techniques*
- *Quote the UML building blocks along with their notations*
- *Demonstrate the use of object-oriented analysis concept with UML diagrams*
- *Solve complex engineering problems using UML concepts and tools*

COURSE REQUIREMENTS

These are the things need to be accomplished properly by the students during the term such as:

- *Attending at least 80% of the classes*
- *Attending quizzes.*
- *Submission of the assignments, project in due time, etc.*

TEXTBOOK / REFERENCES

1. The Unified Modeling Language User Guide by Grady Booch, James Rumbaugh, Ivar Jacobson

2. UML Weekend Crash Course by Thomas A Pender

3. Head first design patterns by Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates

4. Design Patterns- Elements of Reusable Object-Oriented Software by Eric, Gamma, Richard Helm, Ralph Johnson, John Vlissides

5. An Integrated Approach to Software Engineering by PankajJalote

6. Object Oriented Software Engineering-Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Overgaard

7. The Unified Modeling Language Reference Manual by Grady Booch, James Rumbaugh, Ivar Jacobson

8. Object Oriented System Analysis and Design, Second Edition by Grady Booch

MID TERM AND FINAL TERM ASSESSMENTS

Term Assessments	Term Written Exam (MCQ + Diagram Drawing)	50%
	Quizzes	40%
	Attendance & Performance	10%
TOTAL		100
Grand Total	40% of Midterm + 60% of Final Term	

CLASSROOM POLICIES

- *Must join the class in due time.*
- *Students are suggested to ask questions during or after the lecture.*
- *Do not hesitate to ask any question any number of time about the lecture*

****Additional/bonus marks may be given to any good performances during the class.*

GRADING POLICIES

All the evaluation categories & marks will be uploaded to the VUES within one week of the evaluation process except the attendance & performance, which will be uploaded along with the major (mid/final term) written exam marks.

Letter grades 'A+' through 'F' is counted as grades. Other grades 'I' and 'UW' are considered as temporary grades which are counted/calculated as 'F' grade in the CGPA. These grades must/will be converted to the actual grades, i.e. 'A+' through 'F'.

'I: INCOMPLETE' is given to students who have missed at most 30% of evaluation categories (quiz/assignment/etc.). Students must contact the course teacher for makeup, through valid application procedures immediately after grade release.

'UW: UNOFFICIAL WITHDRAW' is given when the missing evaluation categories are too high (more than 30%) to makeup. A student getting 'UW' has no option but to drop the course immediately after grade release

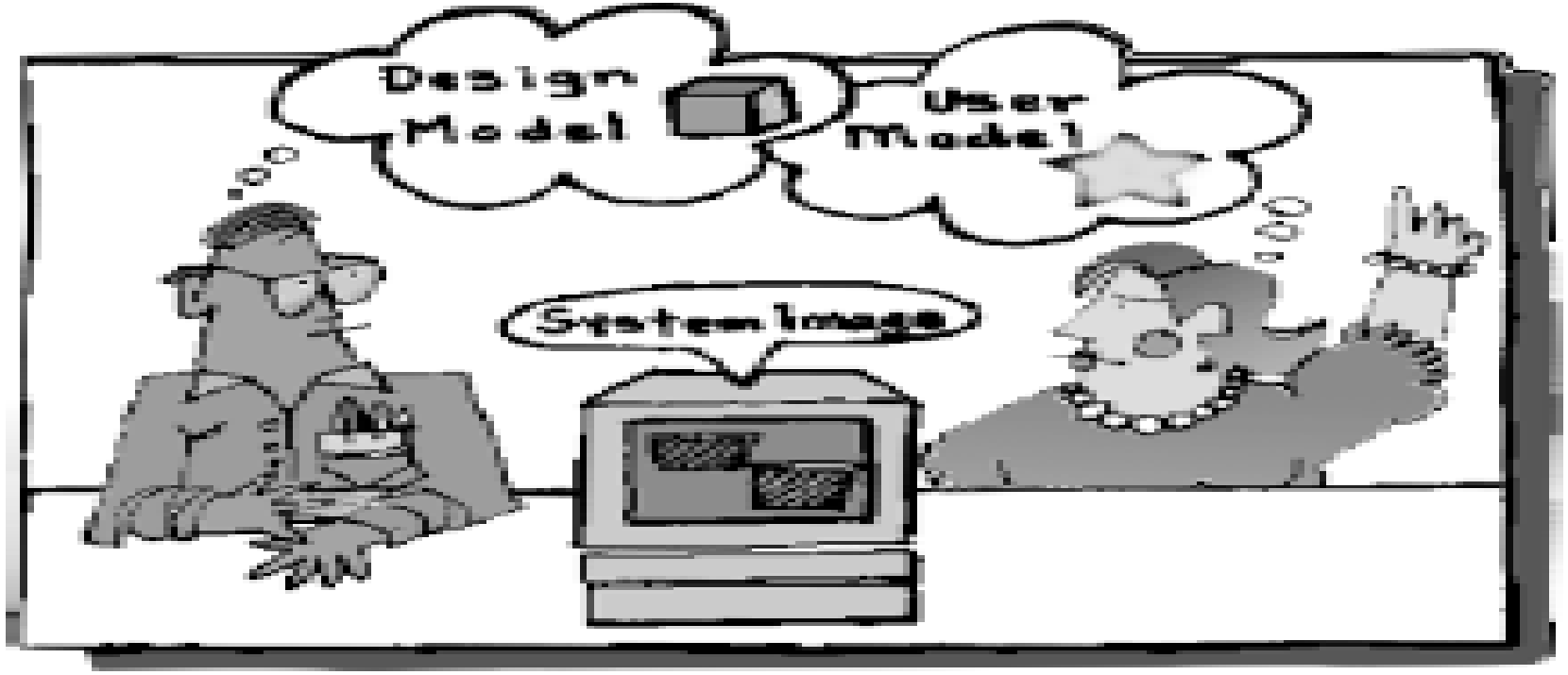
IMPORTANT!!

- *Never Come to my office or send me messages/emails with the request for an extra grade.*
- *Never miss any deadline.*
- *Marks will be **deducted**, even **worse** can happen if you copy assignment/project*

WELCOME TO OBJECT ORIENTED ANALYSIS AND DESIGN

WHY WE MODEL?

- A model is a simplification of reality
- A model provides the blueprints of a system.



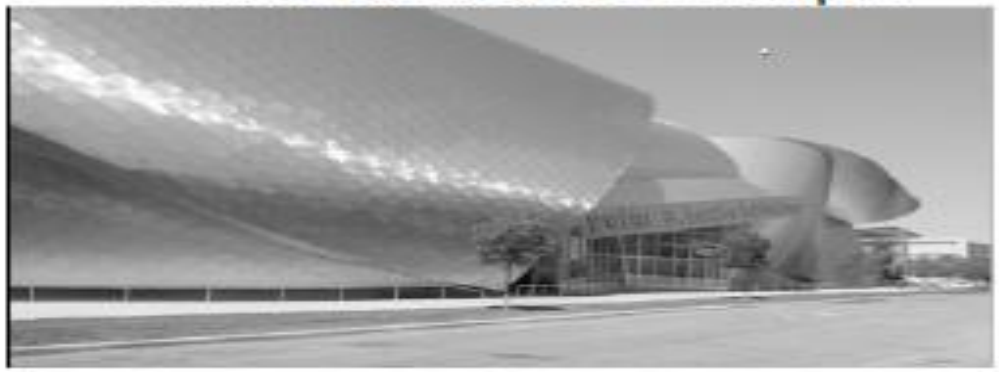
WHY WE MODEL?

- The larger and more complex the system, the more important modelling becomes, for one very simple reason:

  We build models of complex systems because we cannot fully mean of such a system in its entirety.

- There are limits to the human ability to understand complexity. Through modelling we narrow the problem we are studying by focusing on only **one aspect** at a time.

The Real World to be developed



Models to help us get there

WHY WE MODEL?

- **Modeling** is the designing of software applications before coding.
- Modeling is an Essential Part of large software projects, and helpful to medium and even small projects as well.
- **Modelling** is a proven and well-accepted engineering technique.
- Unsuccessful software projects fail in their own unique ways, but all successful projects are alike in many ways.
- There are many elements that contribute to a successful software organization; one common thread is the use of **modelling**.
- We do model so that we can better understand the system we are developing.

BENEFITS OF MODEL

- Through modelling we achieve four aims:

1. Models help us to **visualize a system** as it is or as we want it to be
2. Models permit us to specify the **structure or behaviour** of a system
3. Models give us **a template that guides us in constructing** a system
4. Models **document the decisions** we have made

LIMITATION OF HUMAN ABILITY

- The larger and more complex the system, the more important modelling becomes, for one very simple reason:
  - *We build models of complex systems because we cannot comprehend such a system in its entirety.*
- There are limits to the human ability to understand complexity. Through modelling we narrow the problem we are studying by focusing on only one aspect at a time.

PRINCIPLES OF MODELLING – 1ST

- *The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped.*

  -In other words, choose models intelligently.

  The right models will illuminate the wicked development problems.

PRINCIPLES OF MODELLING – 2ND

- *Every model may be expressed at different levels of precision.*

  - Sometimes a quick and simple executable model of the user interface is exactly what you need; at other times, one has to get down and dirty with the bits. In any case, the best kinds of models are those that let you choose your degree of detail, depending on who is viewing and why they need to view it.

PRINCIPLES OF MODELLING – 3RD

- *The best models are connected to reality.*

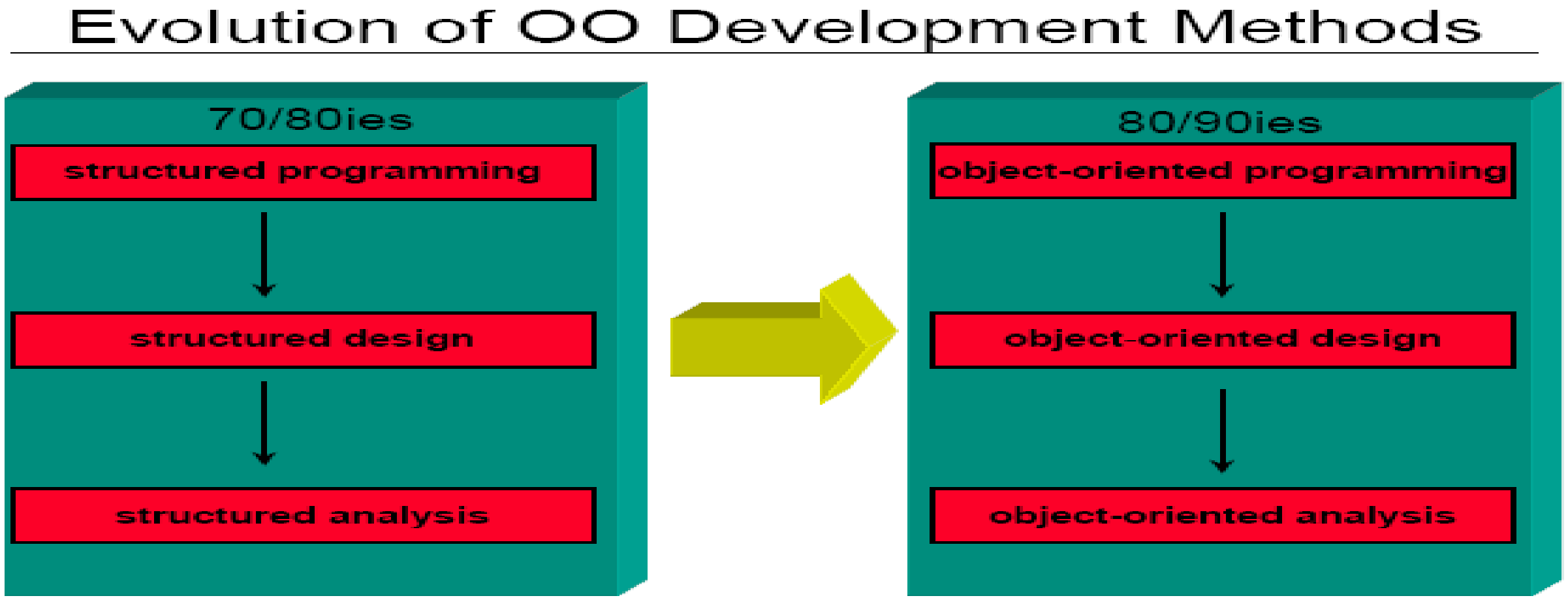
  - A Model connected to reality makes it more understandable and presentable. The more abstract a model is the more it needs to be closer to reality.

PRINCIPLES OF MODELLING – 4TH

- *No single model is sufficient.*

  - Every nontrivial system is best approached through a small set of nearly independent models.

SOFTWARE DEVELOPMENT EVOLUTION



UML

- Unified Modeling Language
- UML is a standardized general-purpose modeling language in the field of **software engineering**. The standard is managed, and was created by, the Object Management Group (OMG) (http://www.omg.org/)
- UML website: http://www.uml.org

- **Why do you use UML?**
  - Standardized graphical notation for Specifying, Visualizing, Constructing, and Documenting software systems
  - Increase understanding/communication of product to customers and developers
  - Support for UML in many software packages today (e.g., UMLet, Rational Rose, ArgoUML, etc.)

THE UML IS A LANGUAGE FOR VISUALIZING

- Some things are best modeled textually; others graphically
- The UML is more than just a bunch of graphical symbols
- One developer can write a model in the UML, and another developer, or even another tool, can interpret that model unambiguously

THE UML IS A LANGUAGE FOR SPECIFYING

- Specifying means building models that are precise, unambiguous, and complete

- In particular, the UML addresses the specification of all the important analysis, design, and implementation decisions that must be made in developing and deploying a software-intensive system.

THE UML IS A LANGUAGE FOR CONSTRUCTING

- UML is not a visual programming language, but its models can be directly connected to a variety of programming languages

- It is possible to map from a model in the UML to a programming language such as Java, C++, or VB, or even to tables in a RDBMS

THE UML IS A LANGUAGE FOR DOCUMENTING

- The UML addresses the documentation of a system's architecture and all of its details
- The UML also provides a language for expressing requirements and for tests
- Finally, the UML provides a language for modeling the activities of project planning and release management

BUILDING BLOCKS OF UML

- The vocabulary of the UML include three kinds of building blocks:
  1. **Things** - abstractions that are primary concern in a model (John, Jesmin, Joseph, Jessy)
  2. **Relationships** - tie these things together (Father, Mother, Daughter, Son)
  3. **Diagrams** - group collections of things (Happy Family)

THINGS IN UML

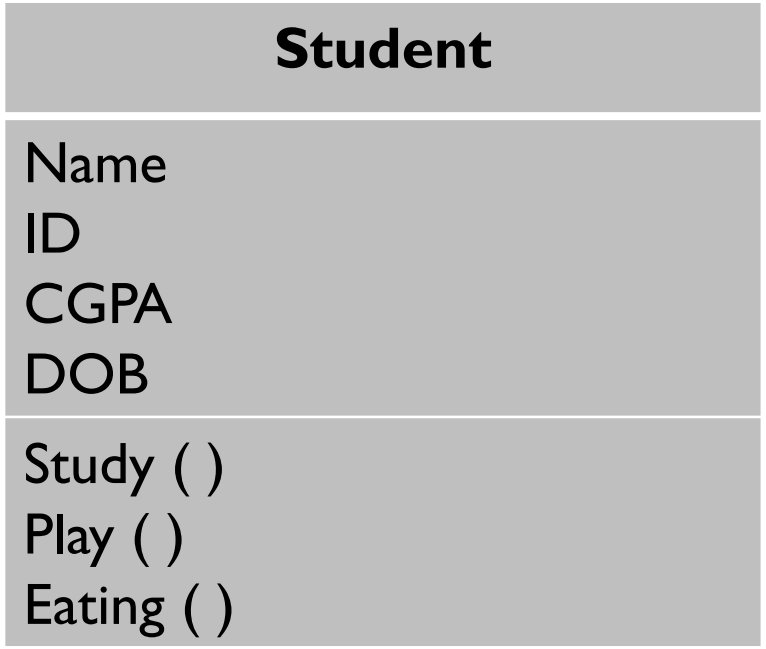
- **Structural things**
  - nouns/static parts of UML models (irrespective of time)
- **Behavioral things**
  - verbs/dynamic parts of UML models
- **Grouping things**
  - organizational parts of UML models
- **Annotation things**
  - explanatory parts of UML models

STRUCTURAL THINGS

- **Structural things** are the nouns of the UML model. These are the mostly static parts of a model, representing elements that are either conceptual or physical. There are **seven** kinds of structural things.

1. A Class

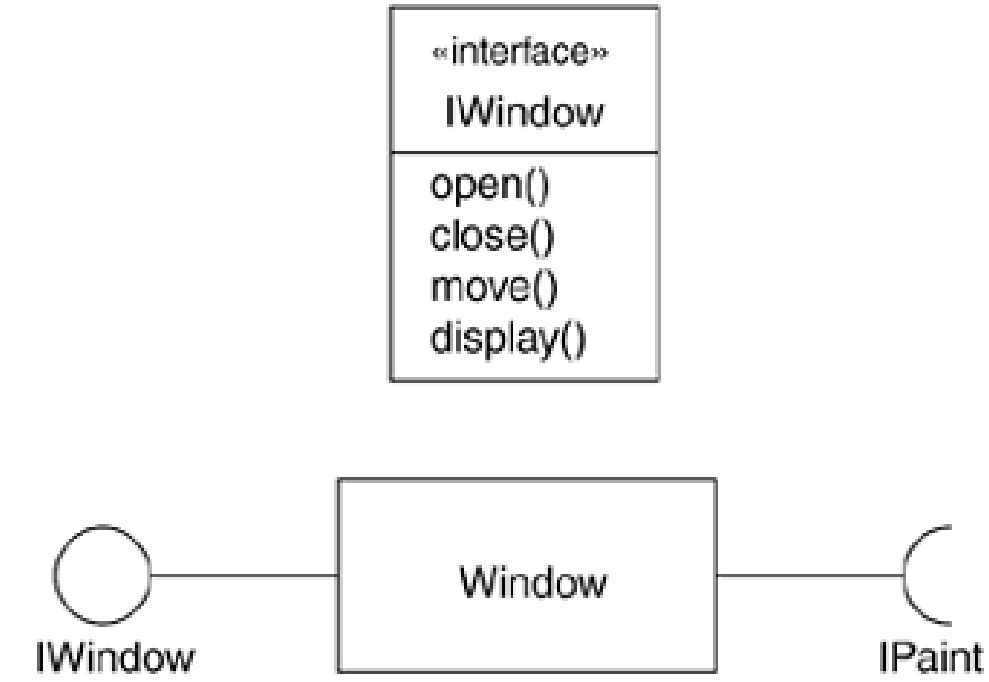
- A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.
- Description of set of objects that share the same attributes, operations, relationships and semantics



STRUCTURAL THINGS

2. An interface

- Is a collection of operations that specify a service of a class or component. An interface therefore describes the **externally** visible behavior of that element.
- You cannot instantiate an interface. Therefore, interface does not contain any constructors.
- An interface is not extended by a class; it is implemented by a class (unchanged in implementation).
- An interface can extend multiple interfaces.



Interface

STRUCTURAL THINGS

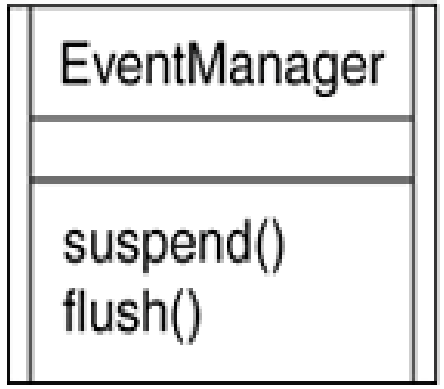
3. **A collaboration:** A collection of UML building blocks (classes, interfaces, relationships) that work together to provide some functionality within the system. (e.g., Interoperability, block chain technology to conned external system in providing one-stop e-services

4. **A Use case** is a description of set of sequence of actions that a system performs that yields an observable result of value to a particular **actor**.

STRUCTURAL THINGS

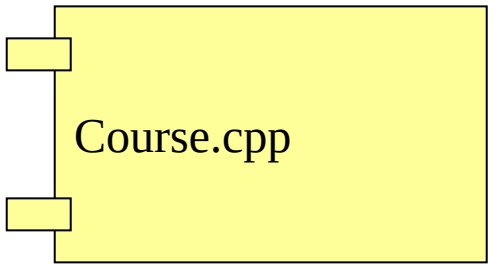
5. An active class

- is a class whose objects (known active object) own one or more processes or threads and therefore can initiate control activity (*modify variables, change program behavior, and so on*).
- Active object initiate and control the flow of activity, while passive object wait for another object to call them generally for store data and serve other classes.
- Graphically, an active class is rendered as a class with double lines on the left and right

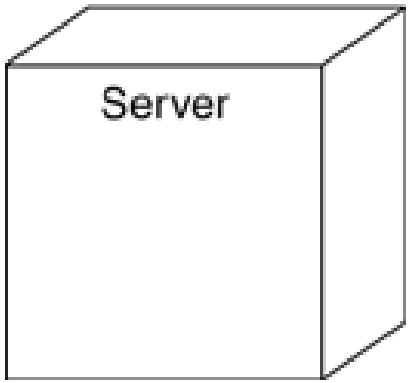


STRUCTURAL THINGS

6. **A Component** is a packaging of classes, interfaces, and collaboration. Total program is divided into various parts or module.



7. **A Node:** Computational resources that exists at run time, typically h/w resources
   e.g., memory, processor, etc.

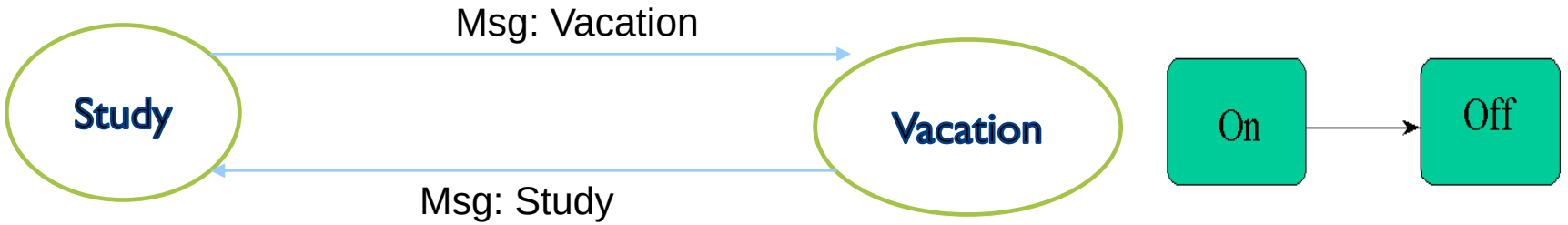


BEHAVIORAL THINGS

- These are the verbs of a model.
- Dynamic parts of UML models.
- Representing behavior over time and space.
- There are **two** kinds of behavioral things

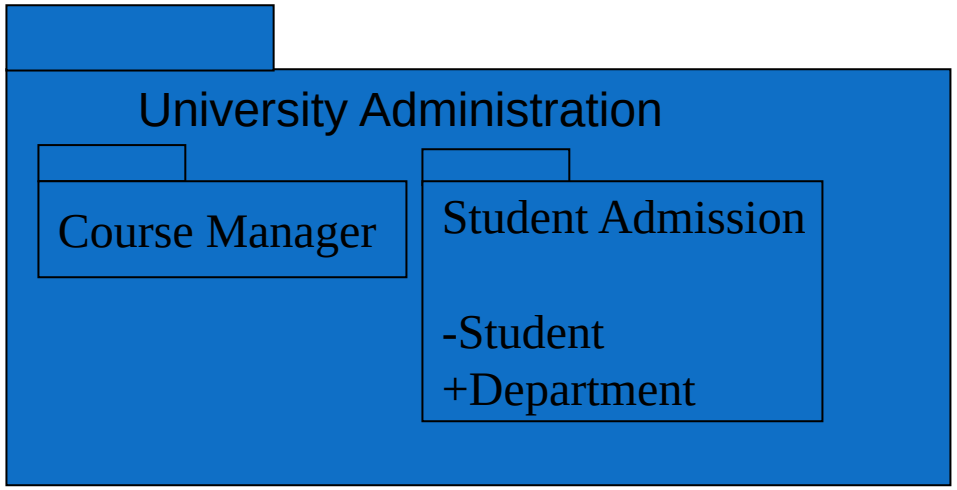
1. **Interaction:** A set of object exchanging messages to accomplish a specific purpose

2. **A state machine:** specifies the sequence of states that an object or an interaction goes through during its lifetime in response to events.



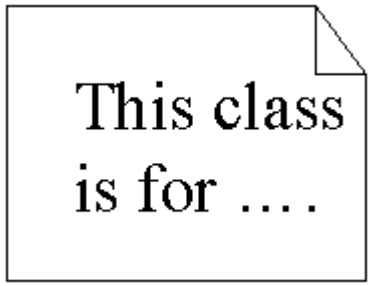
GROUPING THINGS

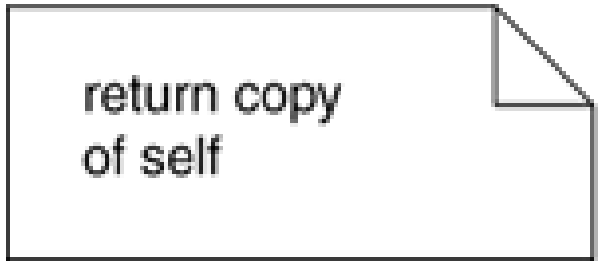
- The organizational parts of UML models. These are the boxes into which a model can be decomposed. There is one primary kind of grouping thing, namely, **packages**.
- Organizing elements (structural/behavioral) into groups.
- Purely conceptual. only exists at development time. And can be nested.
- Variations of packages are: Frameworks, models, & subsystems.



ANNOTATIONAL THINGS

- Annotational things are the **explanatory** parts of UML models.
- These are the comments you may apply to describe, illuminate, and remark about any element in a model.
- There is one primary kind of Annotational thing, called a **note**.





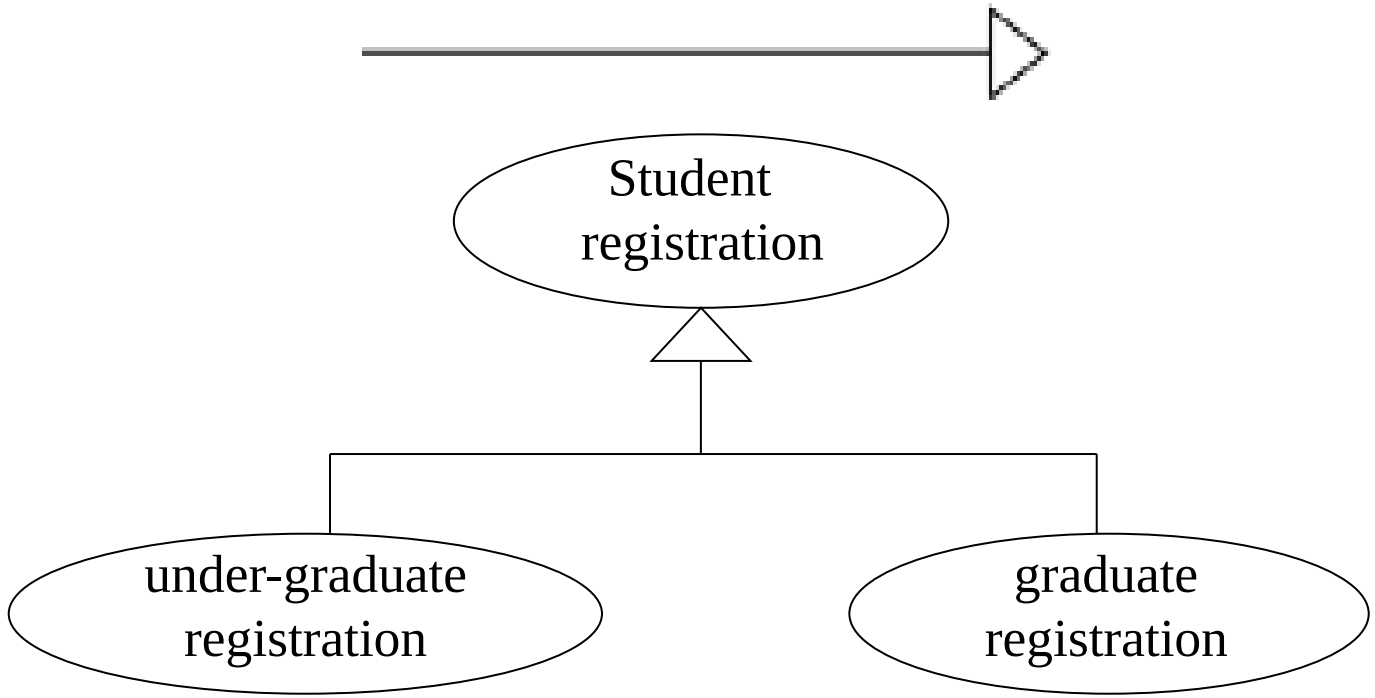
RELATIONSHIPS IN UML

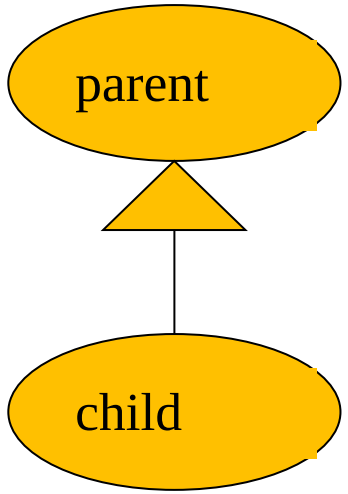
- There are four kinds of relationships in the UML
  - Association
  - Generalization
  - Realization
  - Dependency

Association: is a structural relationship that describes a set of links. A link is a connection between objects.

RELATIONSHIPS IN UML

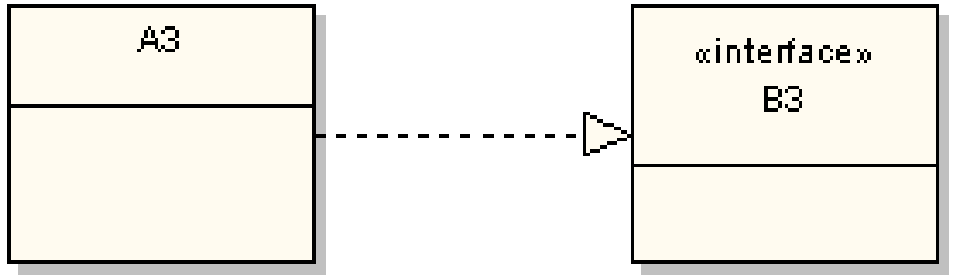
Generalization: can be defined as a relationship which connects a specialized element with a generalized element. It basically describes inheritance (IS-A) relationship in the world of objects.





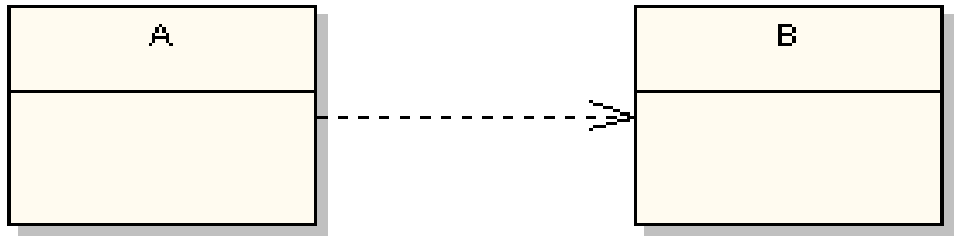
RELATIONSHIPS IN UML

Realization: denotes the implementation (interface) of the functionality defined in one class by another class.




```
class A3 implements B3
```

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




```
class A depends on class B
```

DIAGRAMS IN UML

A **diagram** is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships). The UML includes nine such diagrams.

Structural Diagrams

Represent the static aspects of a system.

- Class Diagram
- Object Diagram
- Component Diagram
- Deployment Diagram

Behavioral Diagrams

Represent the dynamic aspects of a system.

- Use case Diagram
- Sequence Diagram (Interaction)
- Collaboration Diagram
- Statechart Diagram
- Activity Diagram

DIAGRAMS IN UML

- **Class diagram** shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams address the static design view of a system. Class diagram that includes active classes address the **static** process view of a system.
- **Object diagram** shows a set of objects and their relationships. Object diagrams represent static snapshots on instances of the things found in class diagrams. These designs address the **static** design or process view of a system from the perspective of real or prototypical cases.
- **Component diagram** shows an encapsulated class and its interfaces, ports, and internal structure consisting of nested components and connectors. Component diagrams address the static design implementation view of a system.
- **Use case diagram** shows a set of use cases and actors and their relationships. Use case diagrams address the static use case view of a system.

DIAGRAMS IN UML

- Both **sequence diagrams** and **communication diagrams** are kinds of interaction diagrams. An **interaction diagram** shows an interaction, consisting of a set of objects or roles, including the messages that may be dispatched among them. Interaction diagrams address the dynamic view of a system.

  A sequence diagram is an interaction diagram that emphasizes the time-ordering of messages; **a communication diagram** is an interaction diagram that emphasizes the structural organization of the objects or roles that send and receive messages.

- **State diagram** shows a state machine, consisting of states, transitions, events, and activities. A state diagrams shows the dynamic view of an object.

DIAGRAMS IN UML

- **Activity diagram** shows the structure of a process or other computation as the flow of control and data from step to step within the computation. Activity diagrams address the dynamic view of a system. They are especially important in modeling the function of a system and emphasize the flow of control among objects.

- **Deployment diagram** shows the configuration of run-time processing nodes and the components that live on them. Deployment diagrams address the static deployment view of an architecture (networking). A node typically hosts one or more artifacts.

- **Artifact diagram** shows the physical constituents of a system on the computer. Artifacts include files, databases, and similar physical collections of bits. Artifacts are often used in conjunction with deployment diagrams.

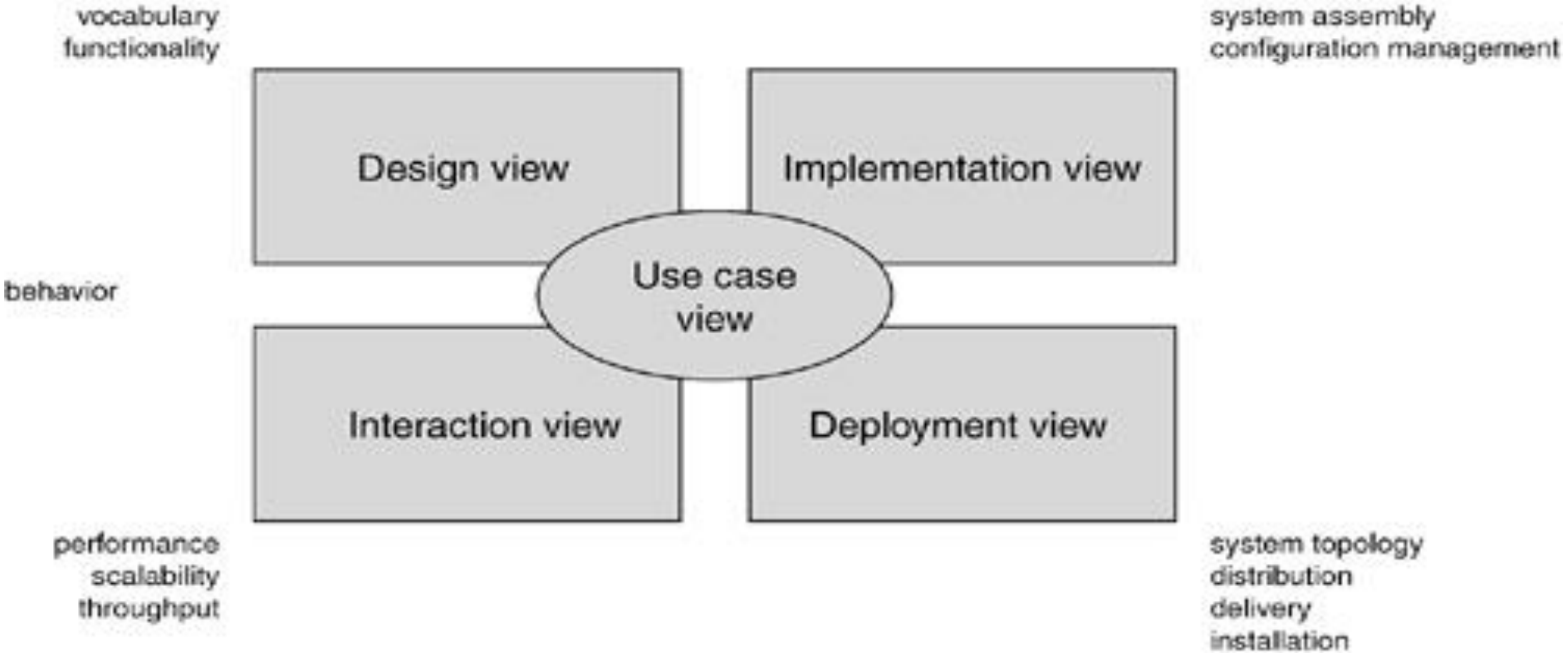
DIAGRAMS IN UML

- **Package diagram** shows the decomposition of the model itself into organization units and their dependencies (package contains a set of diagrams and their dependency grouping).

- **Timing diagram** is an interaction diagram that shows actual times across different objects or roles, as opposed to just relative sequences of messages.

- **Interaction overview diagram** is a hybrid of an activity diagram and a sequence diagram.

SYSTEM ARCHITECTURE MODELING



UML ARCHITECTURE

- Visualizing, specifying, constructing and documenting a software intensive system demands that the system be viewed from a number of perspectives (e.g., ATM system development design in several diagram drawing perspective).
- Different stakeholders- end users, analysts, developers, system integrators, testers, technical writers and project managers- each bring different agenda to a project, and each looks at that system in different ways at different times over the project's life.

1. The **use case view** of a system encompasses the use cases that describe the behavior of the system as seen by its end users, analysts, and testers. With the UML, the static aspects of this view are captured in use case diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams, and activity diagrams.

UML ARCHITECTURE

2. The **design view** of a system encompasses the classes, interfaces, and collaborations that form the vocabulary of the problem and its solution. This view primarily supports the functional requirements of the system, meaning the services that the system should provide to its end users. With the UML, the static aspects of this view are captured in class diagrams and object diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams, and activity diagrams.

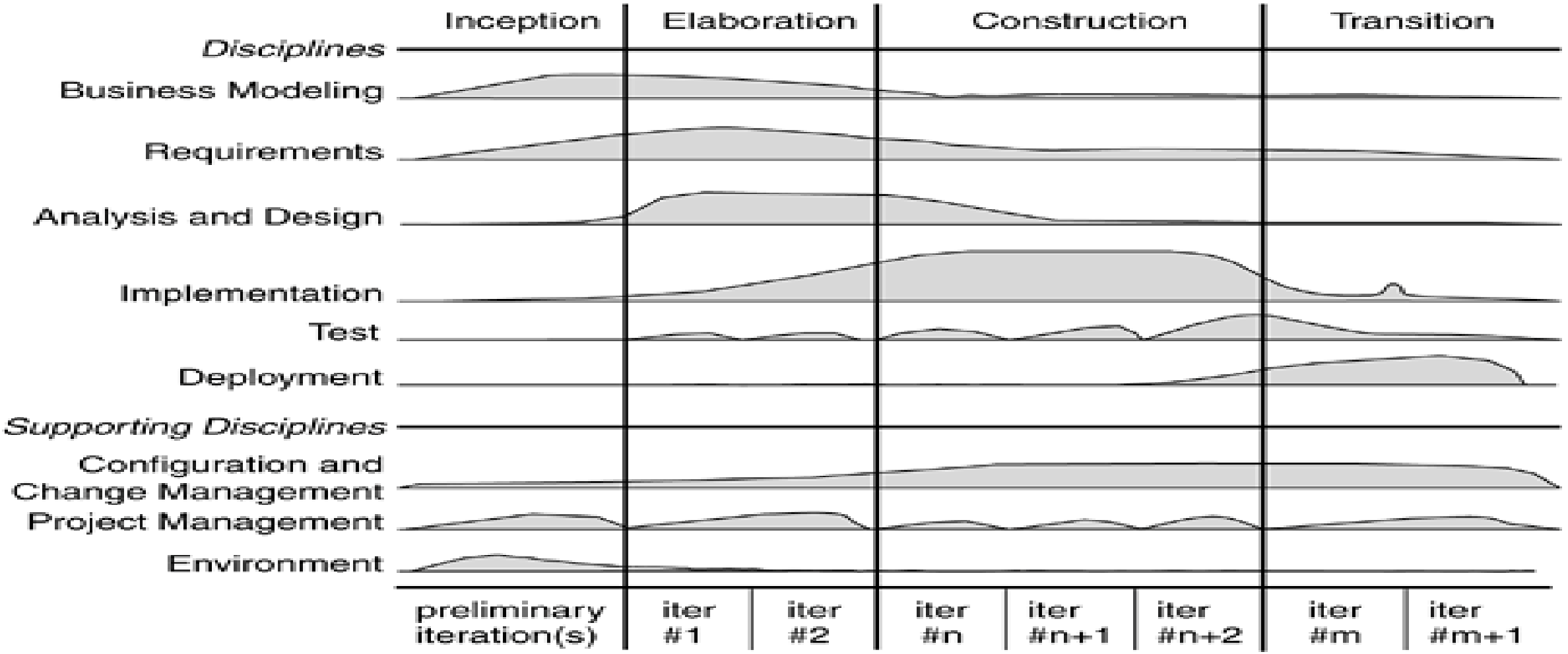
3. The **interaction view** of a system shows the flow of control among its various parts, including possible concurrency and synchronization mechanisms. This view primarily addresses the performance, scalability, and throughput of the system. With the UML, the static and dynamic aspects of this view are captured in the same kinds of diagrams as for the design view, but with a focus on the active classes that control the system and the messages that flow between them.

UML ARCHITECTURE

4. The **implementation view** of a system encompasses the artifacts that are used to assemble and release the physical system. This view primarily addresses the configuration management of the system's releases, made up of somewhat independent files that can be assembled in various ways to produce a running system. It is also concerned with the mapping from logical classes and components to physical artifacts. With the UML, the static aspects of this view are captured in artifact diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams, and activity diagrams.

5. The **deployment view** of a system encompasses the nodes that form the system's hardware topology on which the system executes. This view primarily addresses the distribution, delivery, and installation of the parts that make up the physical system. With the UML, the static aspects of this view are captured in deployment diagrams; the dynamic aspects of this view are captured in interaction diagrams, state diagrams, and activity diagrams.

SDLC – RATIONAL UNIFIED MODEL (RUP)



SDLC – RATIONAL UNIFIED MODEL (RUP)

- **Inception** is the first phase of the process, when the seed idea for the development is brought up to the point of being at least internally sufficiently well-founded to warrant entering into the elaboration phase.
- **Elaboration** is the second phase of the process, when the product requirements and architecture are defined. In this phase, the requirements are articulated, prioritized, and baselined. A system's requirements may range from general vision statements to precise evaluation criteria, each specifying particular functional or nonfunctional behavior and each providing a basis for testing.
- **Construction** is the third phase of the process, when the software is brought from an executable architectural baseline to being ready to be transitioned to the user community. Here also, the system's requirements and especially its evaluation criteria are constantly reexamined against the business needs of the project, and resources are allocated as appropriate to actively attack risks to the project.

SDLC – RATIONAL UNIFIED PROCESS (RUP)

- **Transition** is the fourth phase of the process, when the software is delivered to the user community. Rarely does the software development process end here, for even during this phase, the system is continuously improved, bugs are eliminated, and features that didn't make an earlier release are added.
- **Iteration** is a distinct set of work tasks, with a baselined plan and evaluation criteria that results in an executable system that can be run, tested, and evaluated.

REFERENCES

- Booch, G., Rumbaugh, J. & Jacobson, I. (2005). *The unified modeling language user guide*. Pearson Education India.