Lecture 1: Why we model

Chapter 1

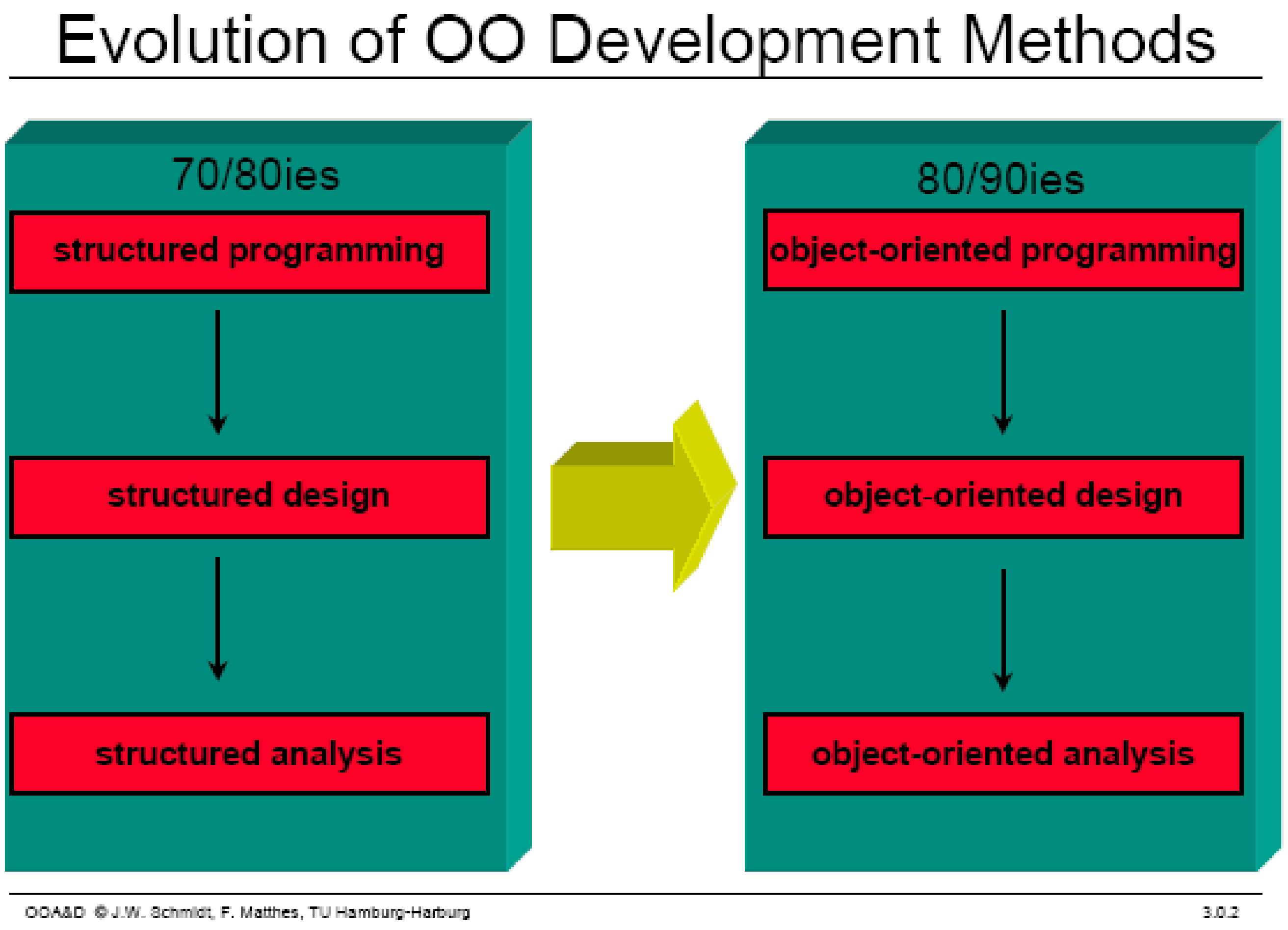
The Unified Modeling Language User Guide

By Grady Booch, James Rumbaugh,
Ivar Jacobson

Unified **Modeling** Language

- UML
  - 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/)
  - http://www.uml.org
  - What is Modeling?
  - A model is a simple representation of reality that helps us to understand how the system works.

Software Development Evolution



Evolution of OO Development Methods

UML History

- OO languages appear mid 70's to late 80's
- Between '89 and '94, OO methods increased from 10 to 50.
- Unification of ideas began in mid 90's.
    - Rumbaugh joins Booch at Rational '94
  - v0.8 draft Unified Method '95
    - Jacobson joins Rational '95
  - UML v0.9 in June '96
  - UML 1.0 offered to OMG in January '97
  - UML 1.1 offered to OMG in July '97
    - Maintenance through OMG RTF
  - UML 1.2 in June '98
  - UML 1.3 in fall '99
  - UML 1.5 http://www.omg.org/technology/documents/formal/uml.htm
  - **UML 2.0** underway http://www.uml.org/
- IBM-Rational now has
  - Grady Booch - Fusion
  - James Rumbaugh – Object Modeling Technique (OMT)
  - Ivar Jacobson – Object-oriented Software Engineering: A Use Case Approach (Objectory)
  - David Harel - StateChart
- Rational Rose http://www-306.ibm.com/software/rational/

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.

What is a Model?

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

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 full meaning 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.

Why 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. Rational Rose, ArgoUML)

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 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 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

Review

Discuss

Building blocks of the UML

- The vocabulary of the UML include three kinds of building blocks:
  1. **Things** - abstractions that are first-class citizens in a model
  2. **Relationships** - tie these things together
  3. **Diagrams** - group interesting collections of things

Things in the UML

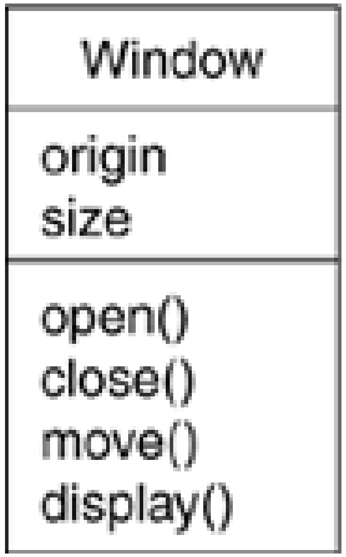
- Structural things
  - Nouns/static of UML models (irrespective of time).
- Behavioral things
  - verbs/dynamic parts of UML models
- Grouping things
  - organizational parts of UML models
- Annotation things
  - explanatory parts of UML models

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.

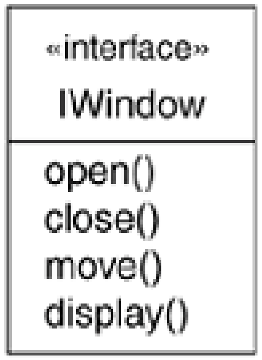
Structural things

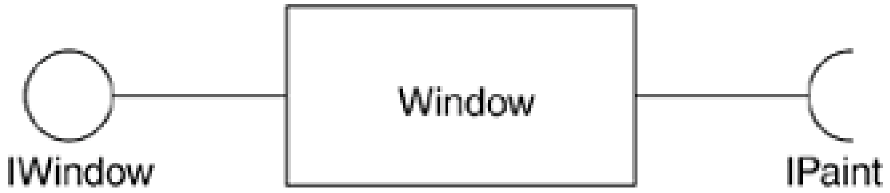
- **A class**
  - is a description of set of objects that share the same attributes, operations, relationships, and semantics.



Structural things cont'd

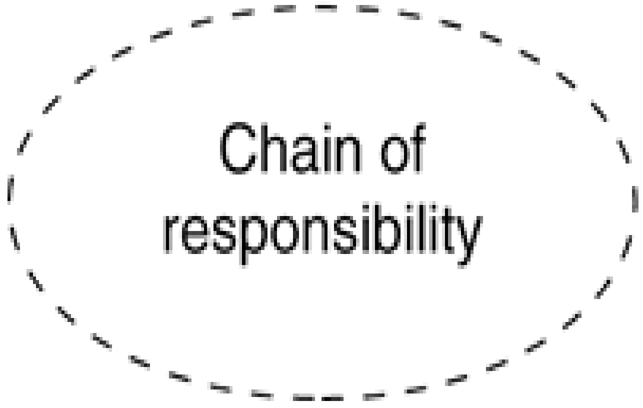
- **An interface**
  - is a collection of operations that specify a service of a class or component. An interface therefore describes the **externally** visible behavior of that element





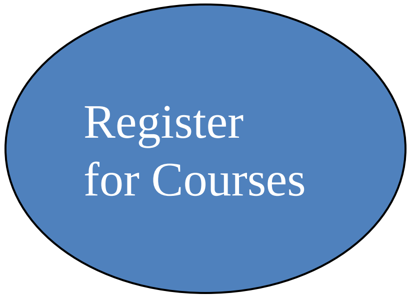
Structural things cont’d

- **A collaboration**
  - Chain of responsibilities shared by interacting objects



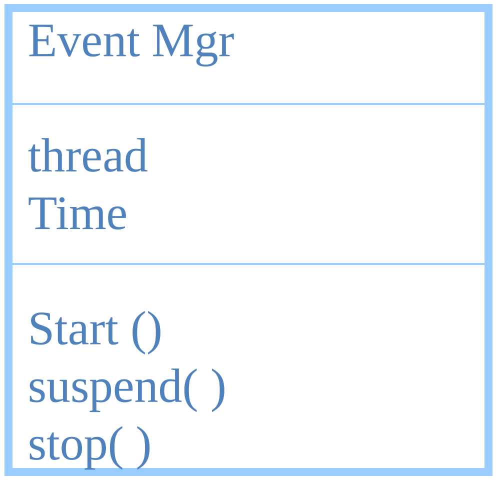
Structural things cont'd

- 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**.



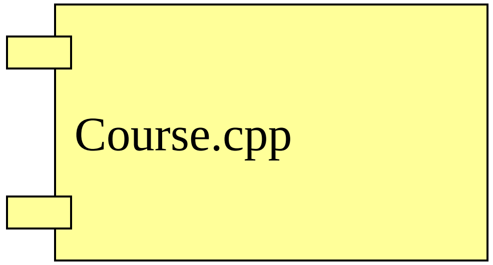
Structural things cont'd

- **An active class**
  - is a class whose objects own one or more processes or threads and therefore can initiate control activity. An active class is just like a class except that its objects represent elements whose behavior is concurrent with other elements.



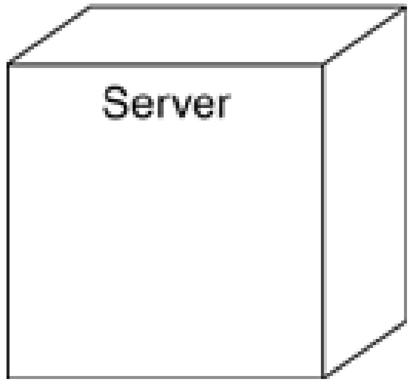
Structural things cont’d

- **A component**
  - A replaceable part, realizes interfaces.



Structural things cont'd

- **A node**
  - Computational resource that exists at run time
  - e.g., memory, processing capability.

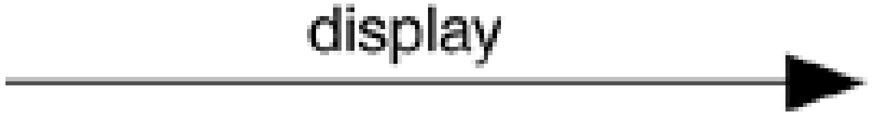


Behavioral Things

- 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

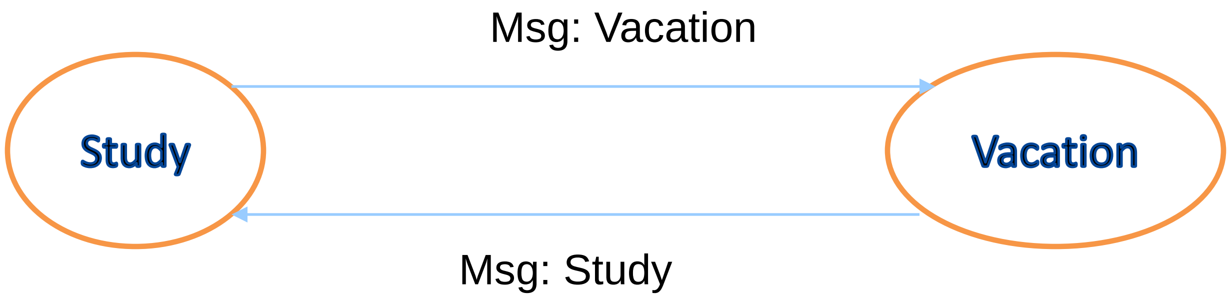
Behavioral Things Cont’d

- **Interaction**
  - A set of object exchanging messages, to accomplish a specific purpose accomplish a specific purpose.



Behavioral Things Cont'd

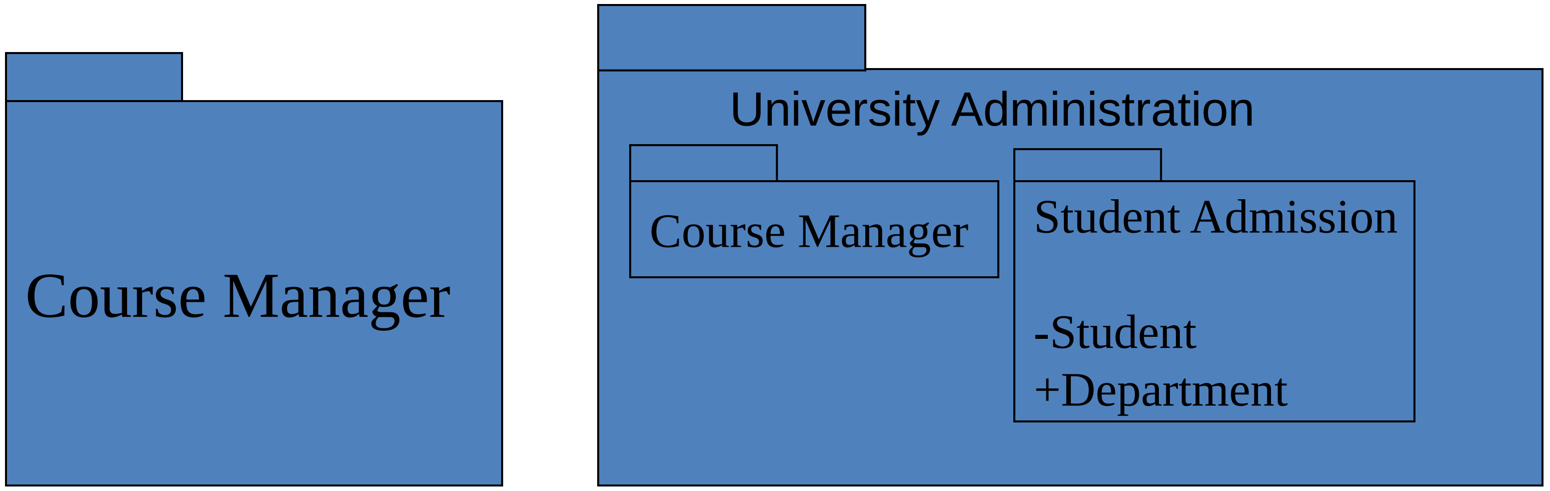
- **A state machine**
  - specifies the sequence of states an object or an interaction goes through during its lifetime in response to events.



Grouping Things

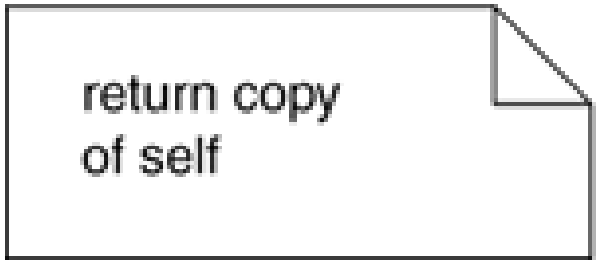
- Grouping things
  - are the organizational parts of UML models. These are the boxes into which a model can be decomposed. There is one primary kind of grouping thing, namely, **packages**.
- Organizing elements (structural/behavioral) into groups.
- Purely conceptual. only exists at development time.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

Grouping Things Cont'd



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**.



Review

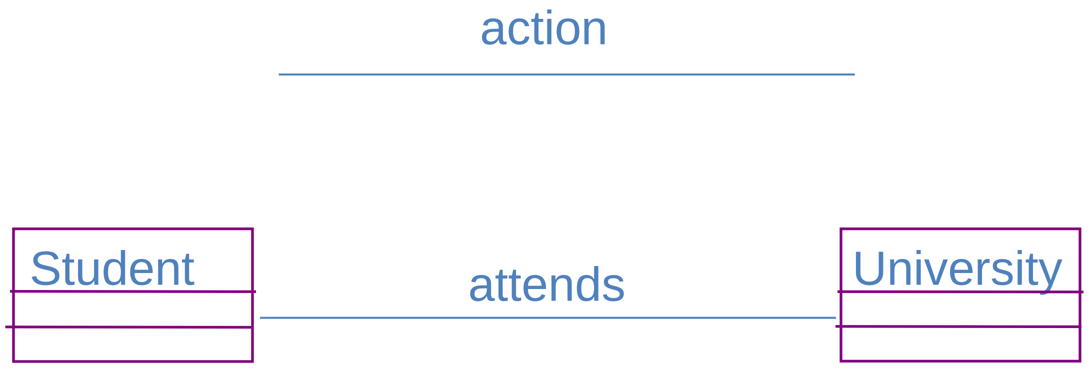
Discuss

Relationships in the UML

- There are four kinds of relationships in the UML:
  1. Association
  2. Generalization
  3. Realization
  4. Dependency

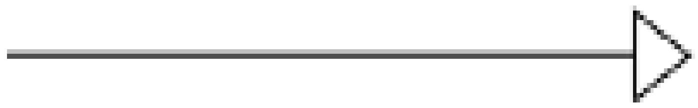
Relationships in the UML Cont’d

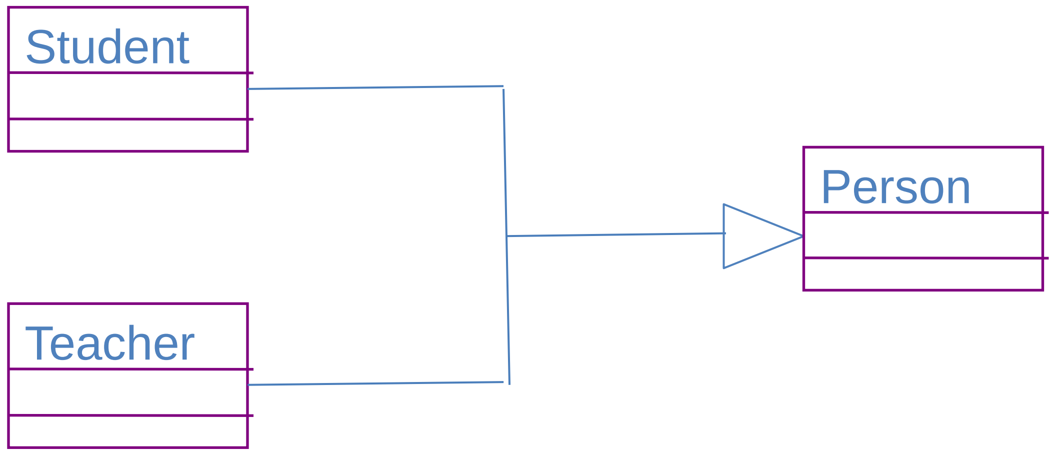
- **An Association** is a structural relationship that describes a set of links. A link is a **connection between objects**.
  - Variants: aggregation & composition



Relationships in the UML Cont’d

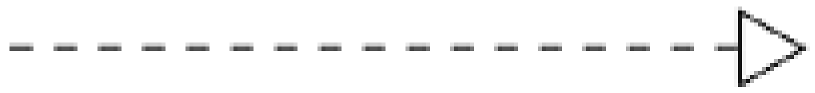
- **A Generalization**
  - can be defined as a relationship which connects a specialized element with a generalized element. It basically describes **inheritance relationship** in the world of objects.

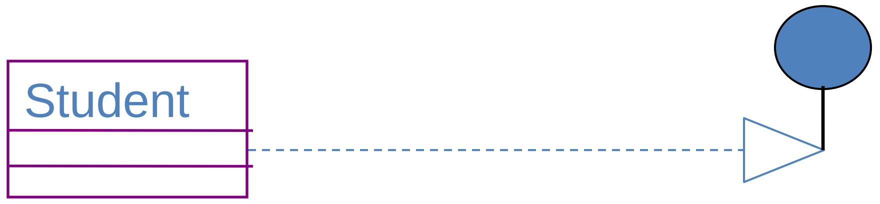




Relationships in the UML Cont’d

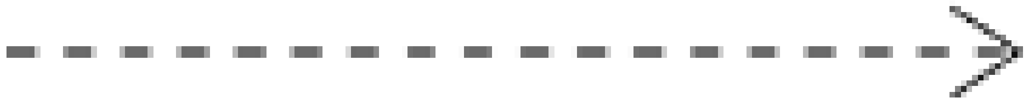
- **A realization**
  - can be defined as a relationship in which two elements are connected. One element describes some responsibility which is not implemented and the other one implements them. This **relationship exists in case of interfaces**.

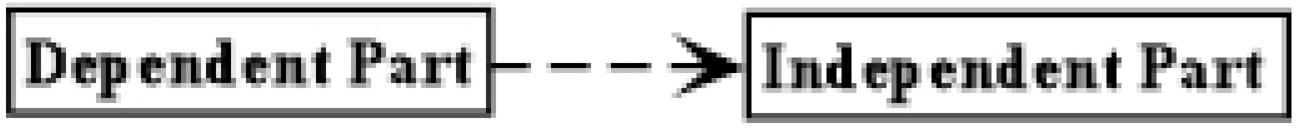




Relationships in the UML

- **A dependency**
  – is a relationship between two things in which change in one element also affects the other one.





Diagrams in the 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.

Diagrams in the UML Cont'd

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

Review

Discuss

Diagrams in the UML

- Class diagram
- Object diagram
- Component diagram
- Composite structure diagram
- Use case diagram
- Sequence diagram
- Communication diagram
- State diagram
- Activity diagram
- Deployment diagram
- Package diagram
- Timing diagram
- Interaction overview diagram

Diagrams in the UML

- **A 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.
- **An 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.

Diagrams in the UML

- A **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.

Diagrams in the UML

- **A 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 the 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.
- **Sequence diagrams** emphasize temporal ordering, and **communication diagrams** emphasize the data structure through which messages flow.

Diagrams in the UML

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

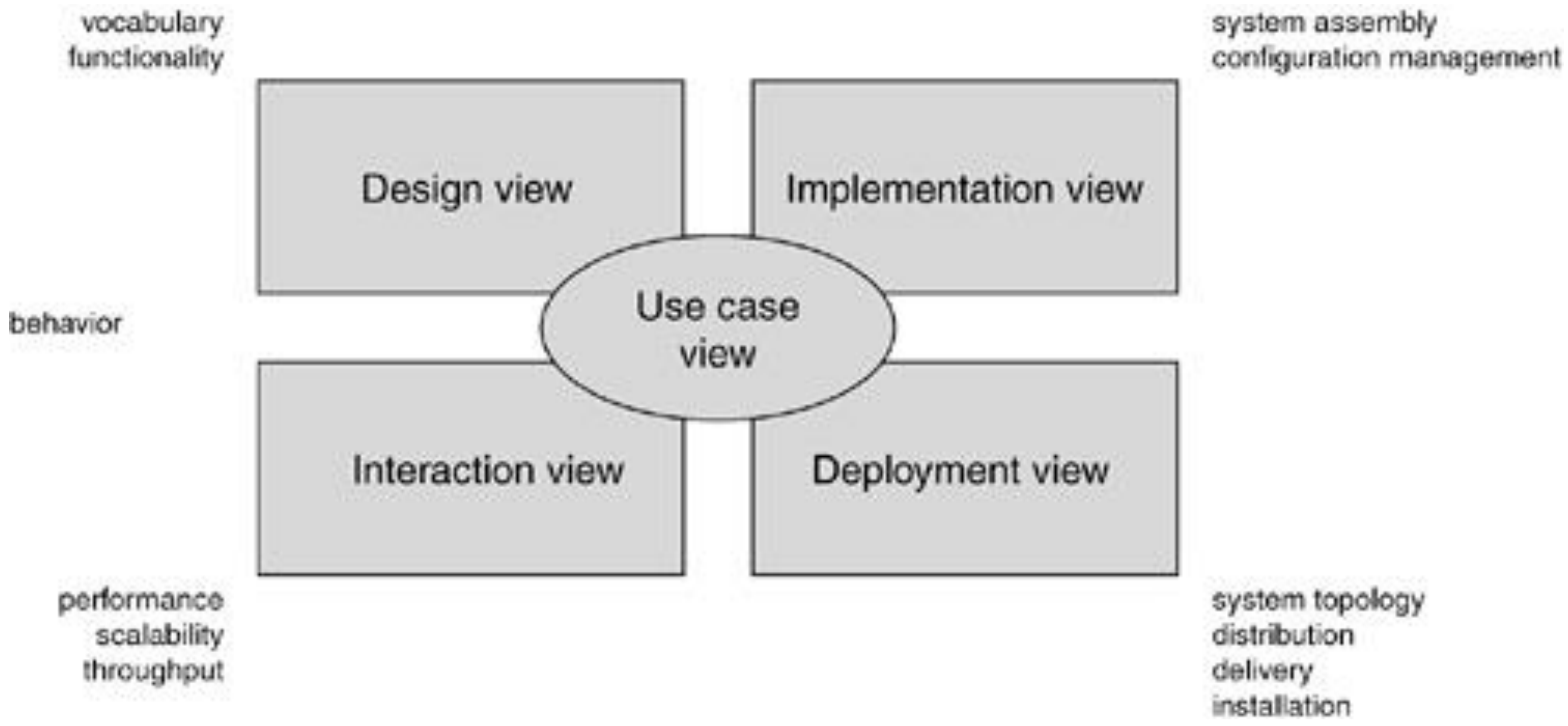
Diagrams in the UML

- A **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. A node typically hosts one or more artifacts.
- An **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. Artifacts

Diagrams in the UML

- A **package diagram** shows the decomposition of the model itself into organization units and their dependencies.
- A **timing diagram** is an interaction diagram that shows actual times across different objects or roles, as opposed to just relative sequences of messages.
- An **interaction overview diagram** is a hybrid of an activity diagram and a sequence diagram.

Modeling a System's Architecture



The UML Architecture

- Visualizing, specifying, constructing and documenting a software intensive system demands that the system be viewed from a number of perspectives.
- 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.

The UML Architecture

- 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.
- 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.

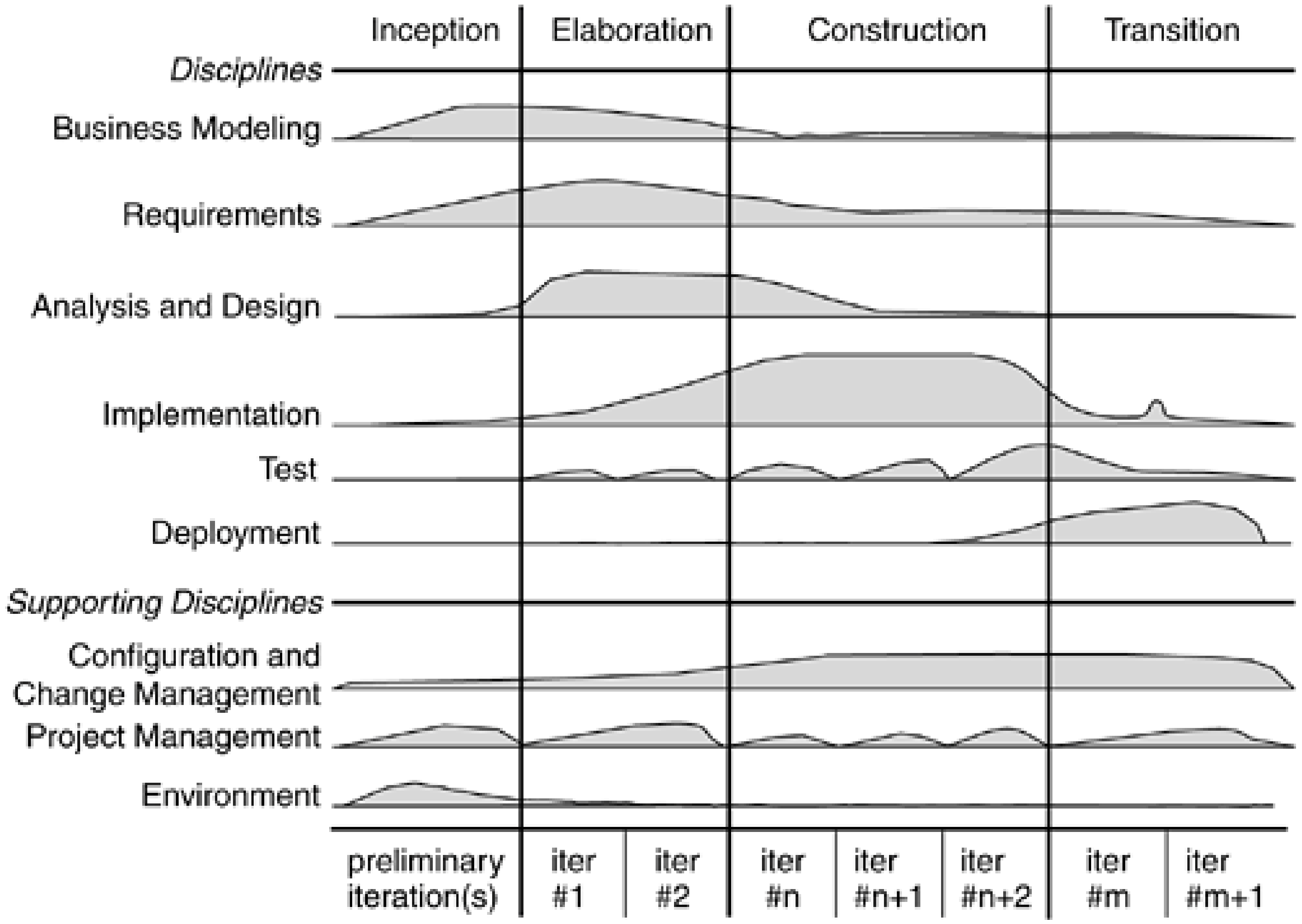
The UML Architecture

- 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.
- 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.

The UML Architecture

- 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.

Software Development Life Cycle



Software Development Life Cycle

- **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.
- **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 eradicated, and features that didn't make an earlier release are added.
- An **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

1. http://www.cs.uah.edu/~rcoleman/Common/SoftwareEng/UML.html
2. https://pdfs.semanticscholar.org/a420/e56d5ad5aedf9740ffc7a1f62193cd0a8ba0.pdf

Review

Discuss

End of First Lecture