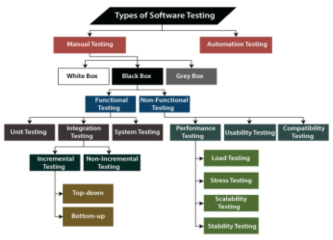
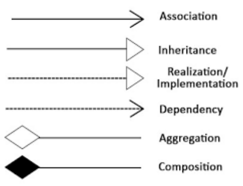
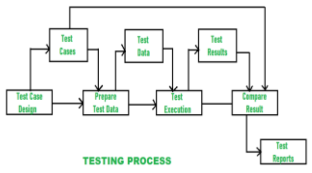
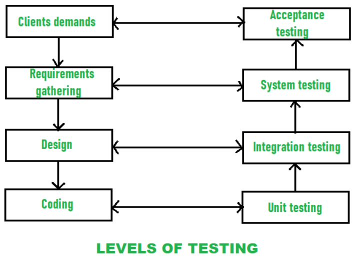






Software Design and Testing

Introduction to Modeling and UML



Outline

- The Importance of Modeling.
- Object-Oriented Modeling and Principles,
- An Overview and Conceptual Model of UML

Abstraction

- Abstraction allows us to ignore unessential details
- Two definitions for abstraction:
  - Abstraction is a *thought process* where ideas are distanced from objects
    - Abstraction as activity
  - Abstraction is the *resulting idea* of a thought process where an idea has been distanced from an object
    - Abstraction as entity
- Ideas can be expressed by models

3

Abstraction

- A model is an abstraction of a system
  - A system that no longer exists
  - An existing system
  - A future system to be built.

4

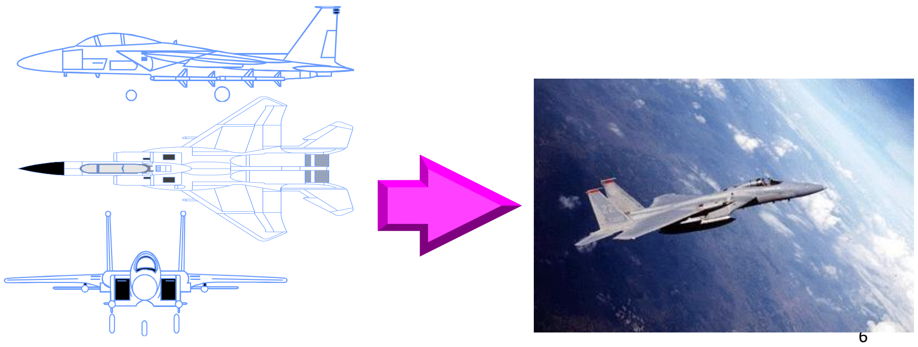
Why Modeling?

- To develop best quality software, you have to craft a solid architectural foundation that's resilient to change.
- To develop software rapidly, efficiently, and effectively, with a minimum of software scrap and rework, you have to have the right people, the right tools, and the right focus.
- To do all this consistently and predictably, you must have a sound development process that can adapt to the changing needs of your business and technology.
- Modeling is a central part of all the activities that lead up to the deployment of good software.
- We build models to communicate the desired structure and behaviour of our system. We build models to visualize and control the system's architecture.

5

Why Modeling?

- We build models to better understand the system we are building, often exposing opportunities for simplification and reuse.
- We build models to manage risk.
- *A picture is worth a thousand words!*
- A model is a simplification of reality.



6

We use Models to describe Software Systems

- Object model: What is the structure of the system?
- Functional model: What are the functions of the system?
- Dynamic model: How does the system react to external events?
- System Model: Object model + functional model + dynamic model

Other models used to describe Software System Development

- Task Model:
  - PERT Chart: What are the dependencies between tasks?
  - Schedule: How can this be done within the time limit?
  - Organization Chart: What are the roles in the project?
- Issues Model:
  - What are the open and closed issues?
    - What blocks me from continuing?
  - What constraints were imposed by the client?
  - What resolutions were made?
    - These lead to action items

What is Model?

- A model is a simplification of a reality.
- A model provides the blueprints of a system. A good model includes relevant elements and may implement level of abstraction for the elements of less importance.
- Every system may be described from different aspects using different models.
- A model may be Structural, emphasizing the organization of the system, or It may be Behavioral, emphasizing the dynamics of the system.
- We build models so that we can better understand the system we are developing. We build models of complex systems because we cannot comprehend such a system in its entirety

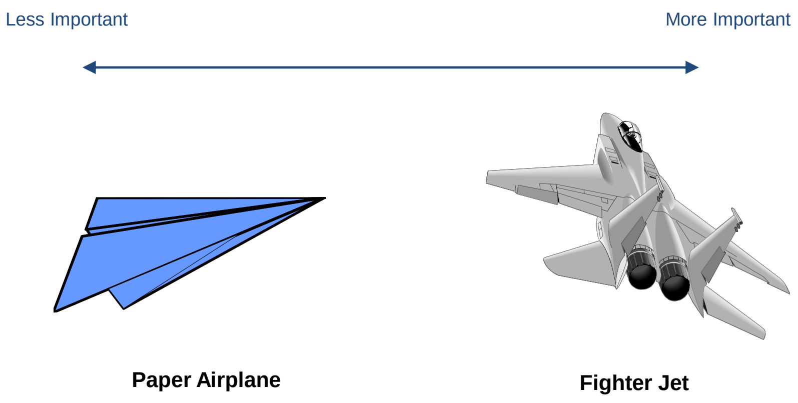
9

What is Model?

- Following are the four important aspects of the system which can be emphasized by implementing models using Unified Modeling Language.
  - Model helps us to visualize the system as it is or as we want it to be.
  - Models permit us to specify the structure or behavior of a system.
  - Models give us a template that guides us in constructing a system.
  - Models document the decisions we have made.
- There are limits to human ability to understand complexity.
- Modeling can narrow down the problem by focusing only on one aspect at a time.

10

Importance of Modeling



11

Importance of Modeling

- If you want to <u>build a dog house</u>, you can pretty much start with a pile of lumber, some nails, and a few basic tools, such as a hammer, saw, and tape measure.
  - In few hours with little prior planning with no one else's help you can build it.
- If you want to <u>build a house for your family</u>, you can start with a pile of lumber, some nails, and a few basic tools, but it's going to take you a lot longer, and your family will certainly be more demanding than the dog.
  - In this case, unless you've already done it a few dozen times before, you'll be better served by doing some detailed planning before you pound the first nail or lay the foundation.
  - At the very least you can make some sketches of how you want the house to look or draw some blueprints as well.

12

Importance of Modeling

  - As long as you stay true to your plans and stay within the limitations of time and money, you will be able to build the house which will satisfy you and your family members requirements.
- If you want to build a high-rise office building, it would be infinitely stupid for you to start with a pile of lumber, some nails, and a few basic tools.
  - Because probably other people are also investing in the building, they will demand to have input into the size, shape, and style of the building.
  - It is also possible that in the middle of the construction their minds change for specific facilities in the building.
  - In such a case , You will want to do extensive planning, because the cost of failure is high.

13

Importance of Modeling

- A lot of software development organizations start out wanting to build high rises but approach the problem as if they were knocking out a dog house.
- If you really want to build the software equivalent of a house or a high rise, the solution is not writing lots of software.
- **In fact, the trick is in creating the right software and in figuring out how to write less software.**
- **This makes quality software development an issue of architecture and process and tools.**
- There are many elements that contribute to a successful software organization; one common thread is the use of modeling.
- We may even build mathematical models in order to analyse the effects of winds or earthquakes on our buildings.

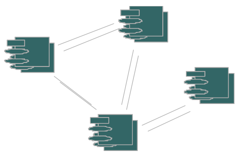
14

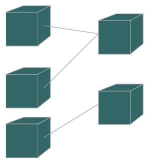
Principles of Modeling

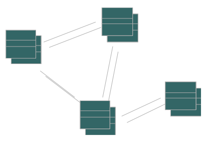
- The model you create influences how the problem is attacked.
- Every model may be expressed at different levels of precision.
- The best models are connected to reality.
- No single model is sufficient.

Principle 1: The Choice of Model Is Important

- The models you create profoundly influence how a problem is attacked and how a solution is shaped.
  - In software, the models you choose greatly affect your world view.
  - Each world view leads to a different kind of system.
  - For example, If you build a system through the eyes of a database developer (entity-relationship models) or
  - a structured analyst (algorithmic-centric models) or
  - an object-oriented developer (classes and their collaboration centric models)

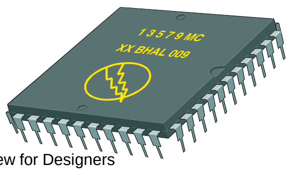

Process Model


Deployment Diagram

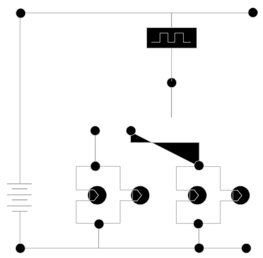

Design Model

Principle 2: Levels of Precision May Differ

- Every model may be expressed at different levels of precision.
  - The best kinds of models let you choose your degree of detail, depending on:
    - who is viewing the model.
    - why they need to view it.



View for Designers



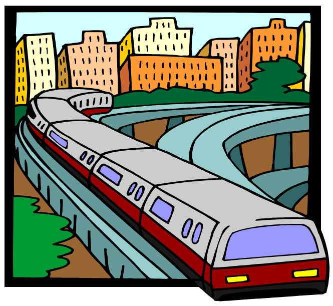
View for Customers

  - An analyst or an end user will want to focus on issues of what; a developer will want to focus on issues of how.
  - Both of these stakeholders will want to visualize a system at different levels of detail at different times.

17

Principle 3: The Best Models Are Connected to Reality

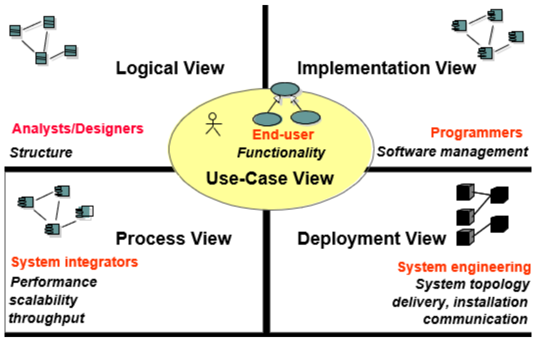
- All models simplify reality.
- A good model reflects potentially fatal characteristics.
- All models simplify reality; the trick is to be sure that your simplifications don't mask any important details.



18

Principle 4: No Single Model Is Sufficient

- No single model is sufficient. Every non-trivial system is best approached through a small set of nearly independent models.
  - Create models that can be built and studied separately, but are still interrelated.



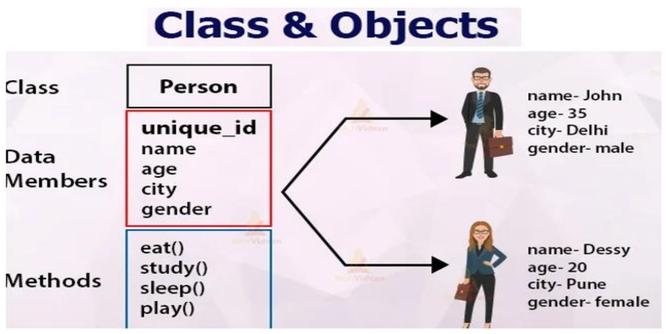
Object Oriented Modeling

- In software, there are several ways to approach a model. The two most common ways are from an algorithmic (functional) perspective and from an object-oriented perspective.
- The traditional view of software development takes an algorithmic perspective. In this approach, the main building block of all software is the procedure or function.
  - This view leads developers to focus on issues of control and the decomposition of larger algorithms into smaller ones.
  - As requirements change (and they will) and the system grows (and it will), systems built with an algorithmic focus turn out to be very hard to maintain.
- In an object-oriented perspective, the main building block of all software systems is the object or class.

21

Object Oriented Modeling

- An object is a thing, generally drawn from the vocabulary of the problem space or the solution space; a class is a description of a set of common objects.
- Every object has identity (you can name it or otherwise distinguish it from other objects), state (there's generally some data associated with it), and behaviour (you can do things to the object, and it can do things to other objects, as well).

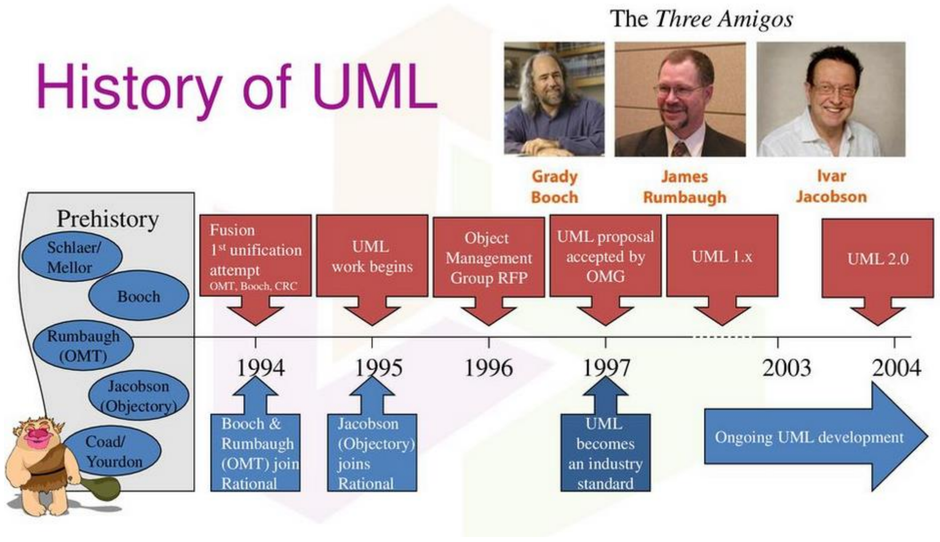


22

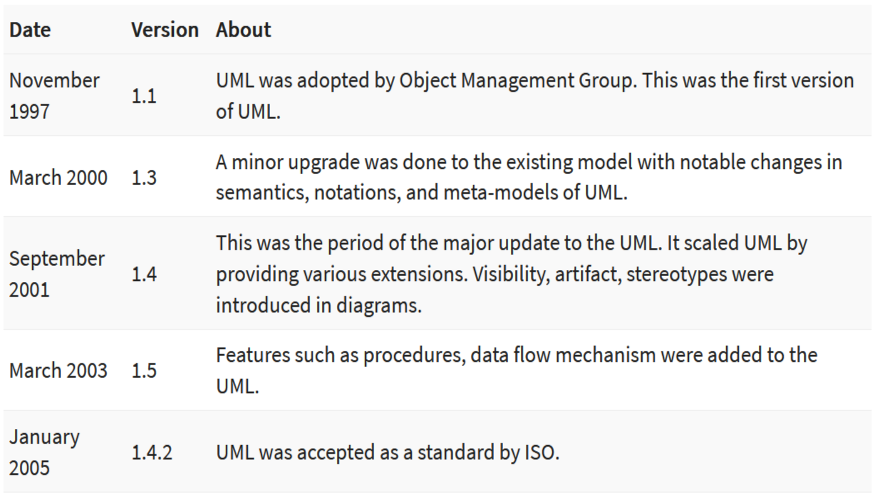
Object Oriented Modeling

- For example, consider a simple three-tier architecture for a billing system, involving a user interface, middleware, and a database.
  - In the user interface, you will find concrete objects, such as buttons, menus, and dialogue boxes.
  - In the database, you will find concrete objects, such as tables representing entities from the problem domain, including customers, products, and orders.
  - In the middle layer, you will find objects such as transactions and business rules, as well as higher-level views of problem entities, such as customers, products, and orders.

23

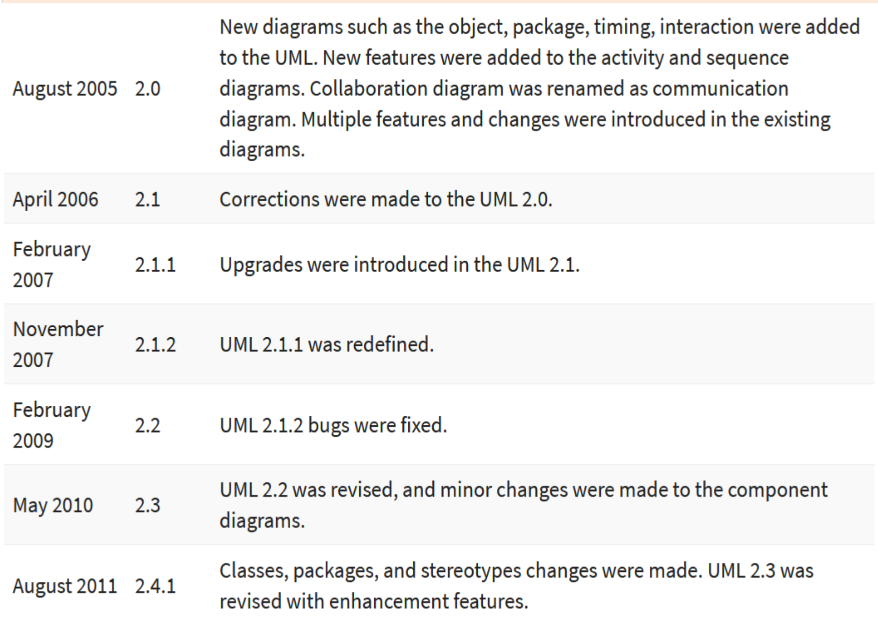


UML Versions



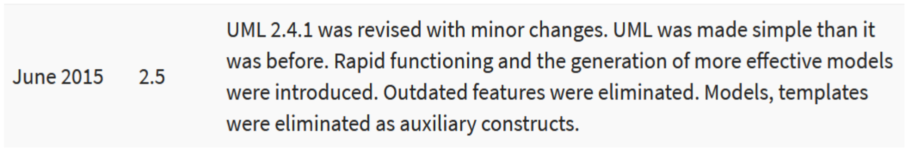
Date	Version	About
November 1997	1.1	UML was adopted by Object Management Group. This was the first version of UML.
March 2000	1.3	A minor upgrade was done to the existing model with notable changes in semantics, notations, and meta-models of UML.
September 2001	1.4	This was the period of the major update to the UML. It scaled UML by providing various extensions. Visibility, artifact, stereotypes were introduced in diagrams.
March 2003	1.5	Features such as procedures, data flow mechanism were added to the UML.
January 2005	1.4.2	UML was accepted as a standard by ISO.

UML Versions



August 2005	2.0	New diagrams such as the object, package, timing, interaction were added to the UML. New features were added to the activity and sequence diagrams. Collaboration diagram was renamed as communication diagram. Multiple features and changes were introduced in the existing diagrams.
April 2006	2.1	Corrections were made to the UML 2.0.
February 2007	2.1.1	Upgrades were introduced in the UML 2.1.
November 2007	2.1.2	UML 2.1.1 was redefined.
February 2009	2.2	UML 2.1.2 bugs were fixed.
May 2010	2.3	UML 2.2 was revised, and minor changes were made to the component diagrams.
August 2011	2.4.1	Classes, packages, and stereotypes changes were made. UML 2.3 was revised with enhancement features.

UML Versions



June 2015	2.5	UML 2.4.1 was revised with minor changes. UML was made simple than it was before. Rapid functioning and the generation of more effective models were introduced. Outdated features were eliminated. Models, templates were eliminated as auxiliary constructs.

Introduction to UML

- The Unified Modeling Language (UML) is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software intensive system.
- The UML is appropriate for modeling systems ranging from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems.
- Learning to apply the UML effectively starts with forming a conceptual model of the language, which requires learning three major elements:
  - the UML's **basic building blocks**,
  - the **rules** that dictate how these building blocks may be put together, and
  - some **common mechanisms** that apply throughout the language.

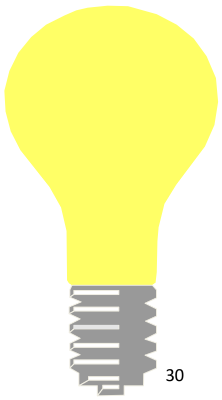
Introduction to UML

- The UML is only a language and so is just one part of a software development method.
- The UML is process independent, although optimally it should be used in a process that is use case driven, architecture-centric, iterative, and incremental.

29

The UML Is a Language for Visualizing

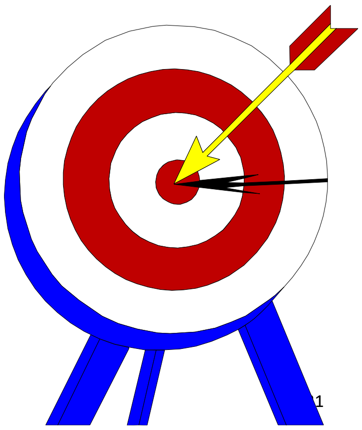
- Communicating conceptual models to others is prone to error unless everyone involved speaks the same language.
- There are things about a software system you can't understand unless you build models.
- An explicit model facilitates communication.



30

The UML Is a Language for Specifying

- The UML builds models that are precise, unambiguous, and complete.

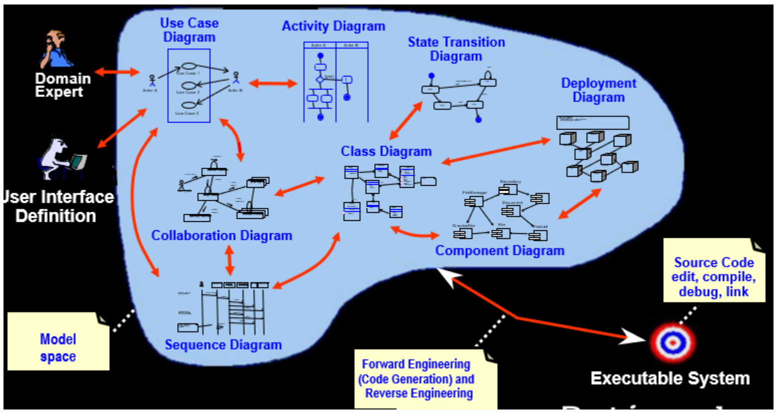


The UML Is a Language for Constructing

- UML models can be directly connected to a variety of programming languages.
  - Maps to Java, C++, Visual Basic, and so on
  - Tables in a RDBMS or persistent store in an OODBMS
  - Permits forward engineering
  - Permits reverse engineering

The UML Is a Language for Documenting

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



33

The UML Language

- The UML addresses documentation of system architecture, requirements, tests, project planning, and release management.
- A language provides a vocabulary and the rules for combining words in that vocabulary for the purpose of communication.
- A modeling language is a language whose vocabulary and rules focus on the conceptual and physical representation of a system.
- A modeling language such as the UML is thus a standard language for software blueprints.

34

The UML Language

- The vocabulary and rules of a language such as the UML tell you how to create and read well-formed models, but they don't tell you what models you should create and when you should create them.
- That's the role of the software development process.
- A well-defined process will guide you in deciding what artifacts to produce, what activities and what workers to use to create them and manage them, and how to use those artifacts to measure and control the project as a whole.

35

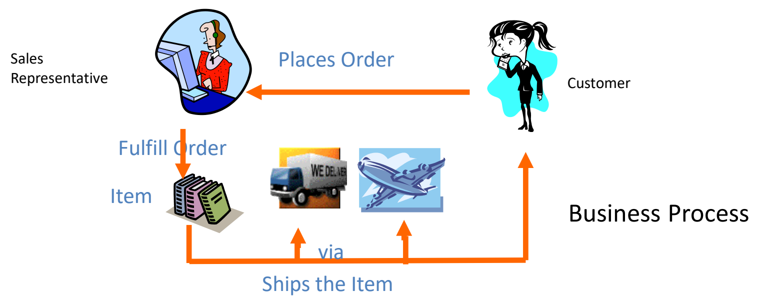
Where Can the UML be used?

- The UML is intended primarily for software-intensive systems. It has been used effectively for such domains as
  - Enterprise information systems
  - Banking and financial services
  - Telecommunications
  - Transportation
  - Defence/aerospace
  - Retail
  - Medical electronics
  - Scientific
  - Distributed Web-based services

36

Where Can the UML be used?

- The UML is not limited to modeling software. In fact, it is expressive enough to model non-software systems, such as workflow in the legal system, the structure and behaviour of a patient healthcare system, and the design of hardware.



37

A Conceptual Model of the UML

- To understand the UML, you need to form a conceptual model of the language, and this requires learning three major elements:
  - the UML's **basic building blocks**,
  - the **rules** that dictate how those building blocks may be put together, and
  - some **common mechanisms** that apply throughout the UML.

38

The basic building blocks of the UML

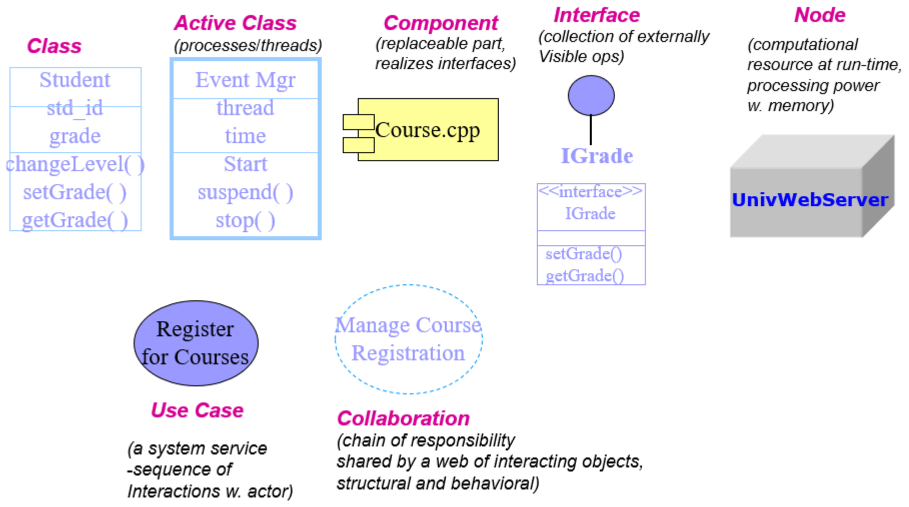
- The vocabulary of the UML encompasses three kinds of building blocks:
  1. Things - important modeling concepts
  2. Relationships - tying individual things
  3. Diagrams - grouping interrelated collections of things and relationships
- Things are the abstractions that are first-class citizens in a model;
- relationships tie these things together;
- diagrams group interesting collections of things.

The basic building blocks of the UML - Things

- UML 1.x
  - Structural — nouns/static of UML models (irrespective of time).
  - Behavioral — verbs/dynamic parts of UML models.
  - Grouping — organizational parts of UML models.
  - Annotational — explanatory parts of UML models.

Structural Things in UML - *7 Kinds (Classifiers)*

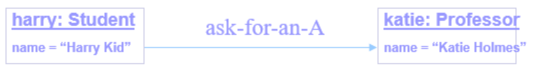
- Nouns.
- Conceptual or physical elements.


Class
Student
std_id
grade
changeLevel( )
setGrade( )
getGrade( )
Active Class
(processes/threads)
Event Mgr
thread
time
Start
suspend( )
stop( )
Component
(replaceable part, realizes interfaces)
Course.cpp
Interface
(collection of externally Visible ops)
IGrade
<<interface>>
IGrade
setGrade()
getGrade()
Node
(computational resource at run-time, processing power w. memory)
UnivWebServer
Register for Courses
Use Case
(a system service -sequence of Interactions w. actor)
Manage Course Registration
Collaboration
(chain of responsibility shared by a web of interacting objects, structural and behavioral)

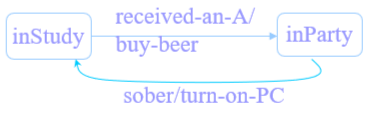
Behavioral Things in UML

- Verbs.
- Dynamic parts of UML models: "behavior over time"
- Usually connected to structural things.
- Two primary kinds of behavioral things:

❑ ***Interaction***
a set of objects exchanging messages, to accomplish a specific purpose.



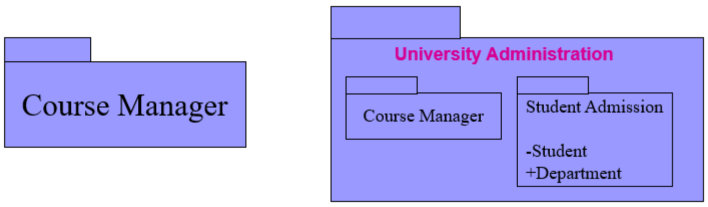
❑ ***State Machine***
specifies the sequence of states an object or an interaction goes through during its lifetime in response to events.



12

Grouping Things in UML – *Packages*

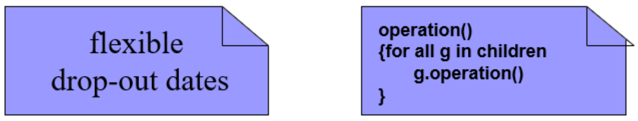
- For organizing elements (structural/behavioral) into groups.
- Purely conceptual; only exists at development time.
- Can be nested.
- Variations of packages are: Frameworks, models, & subsystems.



43

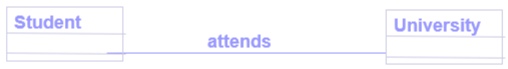
Annotational Things in UML – *Notes*

- Explanatory/Comment parts of UML models - usually called adornments
- Expressed in informal or formal text.



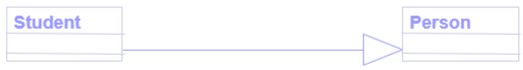
44

Basic building blocks of UML – *Relationship*



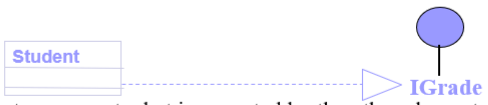
1. Associations

Structural relationship that describes a set of links, a link being a connection between objects.
(UML2.0: The semantic relationship between two or more classifiers that involves connections among their instances.)
variants: aggregation & composition



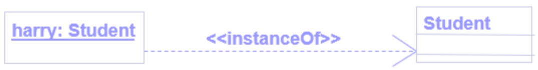
2. Generalization

a specialized element (the child) is more specific the generalized element.



3. Realization

one element guarantees to carry out what is expected by the other element.
(e.g, interfaces and classes/components; use cases and collaborations)



4. Dependency

a change to one thing (independent) may affect the semantics of the other thing (dependent).
(direction, label are optional)

Basic building blocks of UML – *Diagrams*

A connected graph: Vertices are things; Arcs are relationships/behaviors.

UML 1.x: 9 diagram types.

Structural Diagrams

Represent the static aspects of a system.

- Class;
  Object
- Component
- Deployment

Behavioral Diagrams

Represent the dynamic aspects.

- Use case
- Sequence;
  Collaboration
- Statechart
- Activity

UML 2.0: 12 diagram types

Structural Diagrams

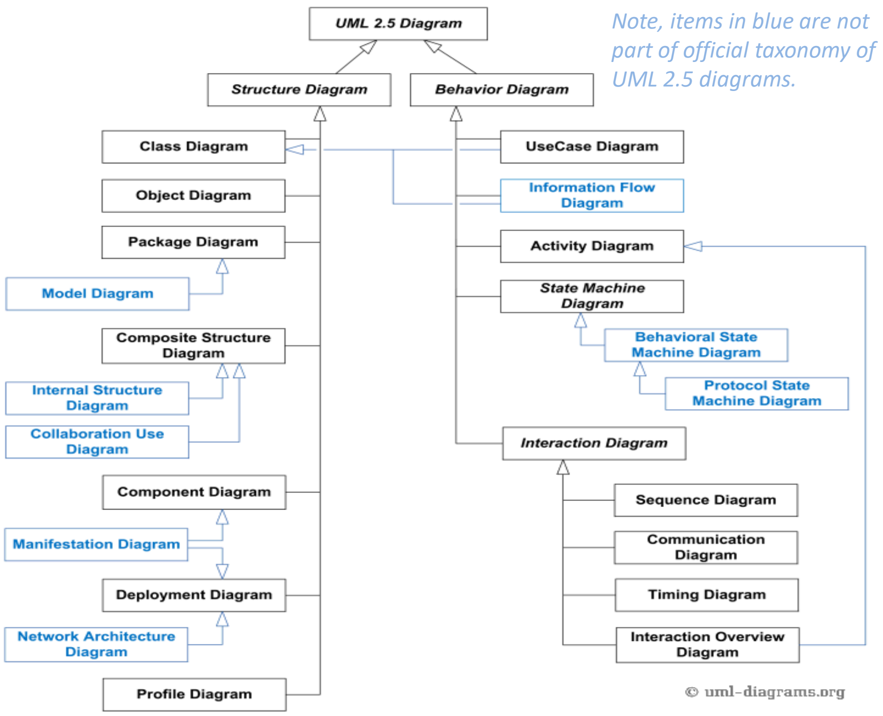
- Class;
  Object
- Component
- Deployment
- Composite Structure
- *Package*

Behavioral Diagrams

- Use case
- Statechart
- Activity

Interaction Diagrams

- Sequence;
  Communication
- Interaction Overview
- Timing



Types of UML Diagrams

- **Class diagram** – A class diagram is a static diagram that describes the structure of a system by showing its classes and their properties and operations, as well as the relationships between objects.
- **Use Case Diagram** – A use case diagram consists of use cases, roles, and the relationships between them. It shows how users interact with the system and defines the specifications of the use cases.
- **Sequence diagram** – A sequence diagram is a model for communication between objects in a sequential manner. It shows the exact order of objects, classes and roles and information involved in a scenario. It consists of vertical lines belonging to lifelines and horizontal lines of messages.

Types of UML Diagrams

- **Activity diagram** – An activity diagram is a behavior diagram that shows a scenario in terms of the flow of actions. It models a sequence of actions, condition-based decisions, concurrent branches and various loops.
- **Communication diagram** – A communication diagram shows the interaction between objects and parts in the form of messages, which are represented by lifelines. A communication diagram is a modified form of a UML sequential diagram, but differs from it in that its elements do not need to be horizontally ordered and can have any position in the diagram.
- **State Machine Diagram** – A state machine diagram describes the state of an entity (device, process, program, software, module, etc.) and the transitions between states. The conditions specify when a transition from one state to another can be used.

Types of UML Diagrams

- **Object diagram** – An object diagram is a structured UML diagram. It describes a system or its parts at a particular time. It models instances, their values and relationships. It can be used to show examples of data structures.
- **Package diagram** – A package diagram shows the dependencies between packages in a model. It describes the structure and organization of large-scale projects.
- **Component diagram** – A component diagram provides a view of a complex system. It describes the interfaces provided and/or required by the various parts of the system and the relationships between the parts. These parts are represented by components and other artifacts.

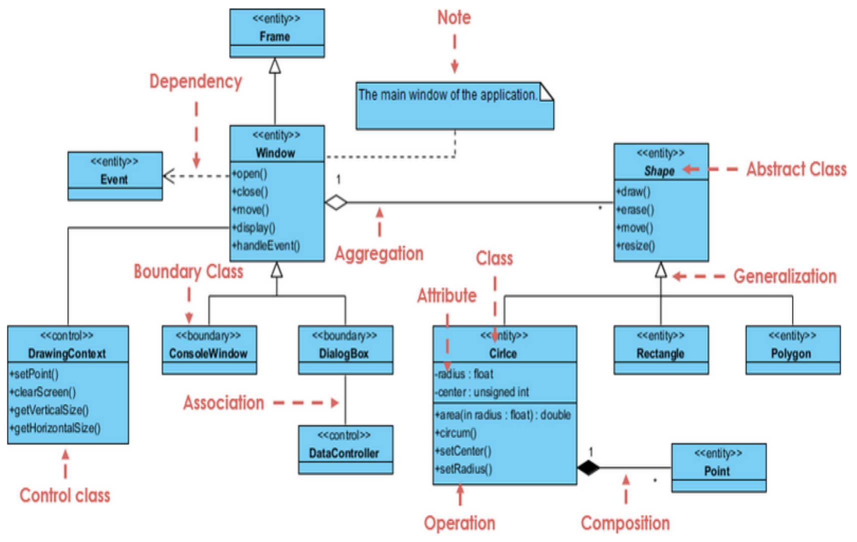
Types of UML Diagrams

- **Deployment Diagram** – The deployment diagram describes the deployment of artifacts on a network node. It is used to show the location of artifacts (software, systems, modules, etc.) on physical nodes (hardware, servers, databases, etc.) and the relationships between specific parts of the solution.
- **Composite Structure Diagram** – The composite structure diagram shows the internal structure of a classifier, its parts and ports, through which it communicates with its environment. It models collaboration, where each element has its defined role.
- **Interaction Overview Diagram** – The Interaction Overview Diagram provides a high level view of the interactions in a system or subsystem. It describes processes in a similar way to activity diagrams, but it uses other interaction diagrams and interaction references rather than action nodes.

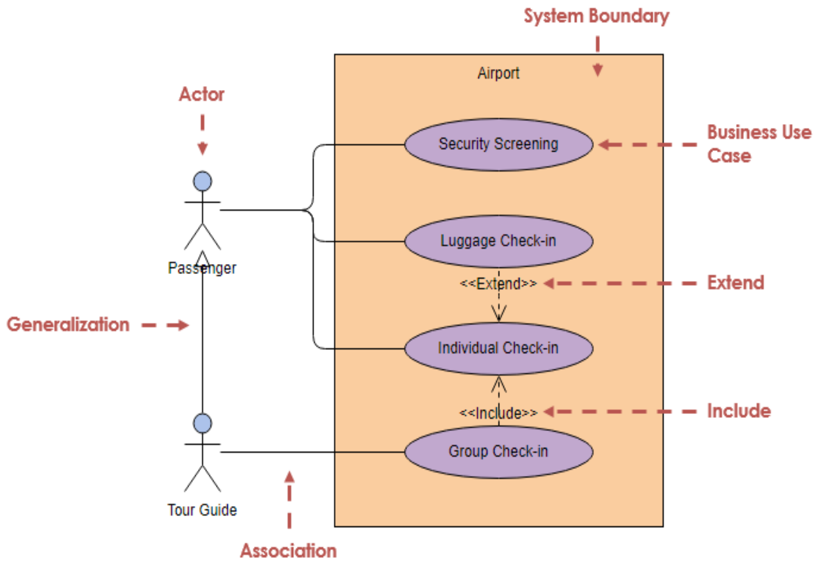
Types of UML Diagrams

- **Timing Diagrams** – The timing diagram focuses primarily on time, and it describes the changes in the classifier on a timeline. The timelines are stacked vertically, with time increasing from left to right.
- **Profile Diagram** – The Profile Diagram describes and defines extensions to the UML language. The extension mechanism allows you to adapt the language to a specific domain or platform. Extensions are defined by stereotyping.

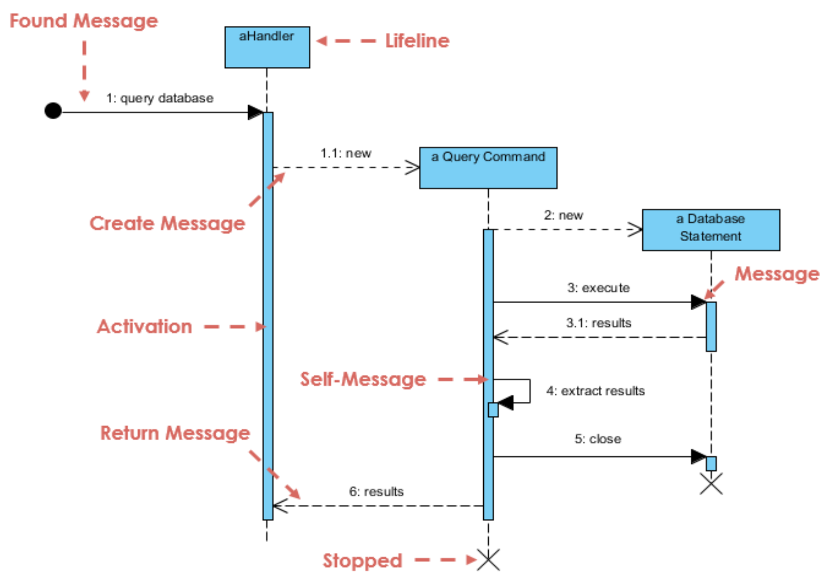
Class Diagram



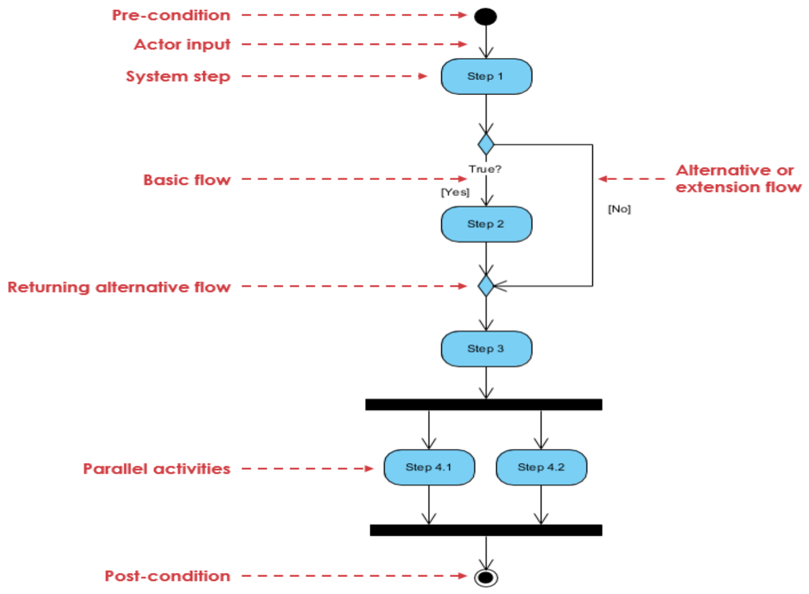
Use Case Diagram



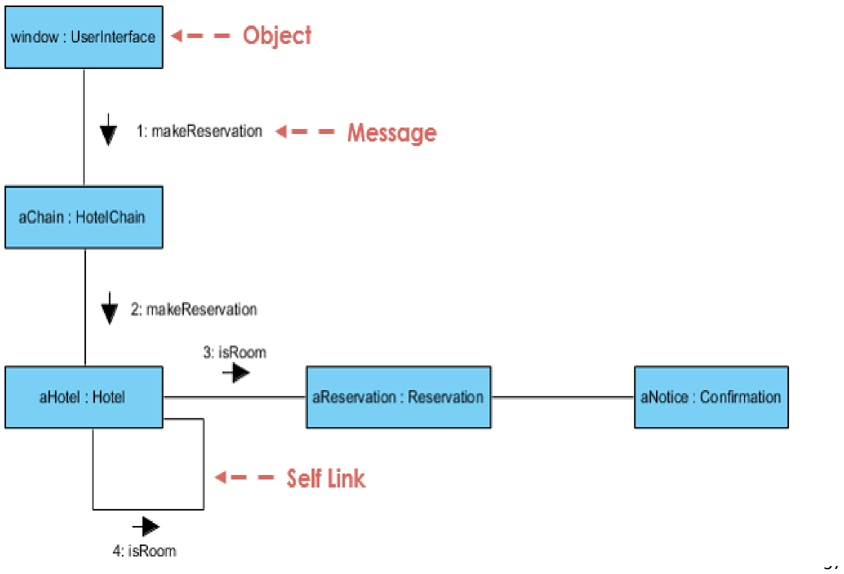
Sequence Diagram



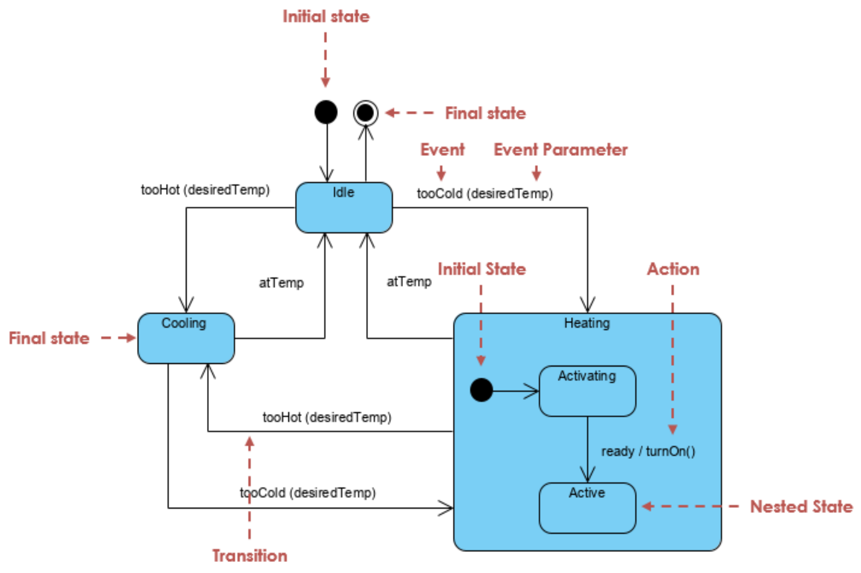
Activity Diagram



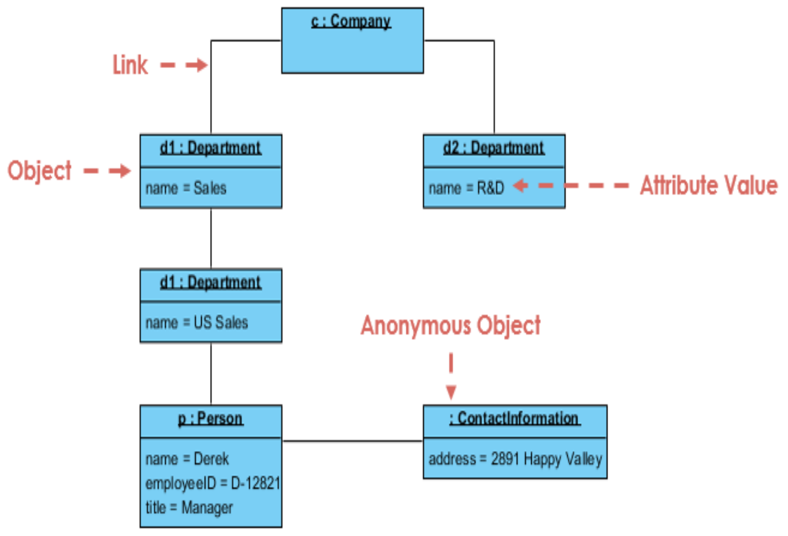
Communication Diagram



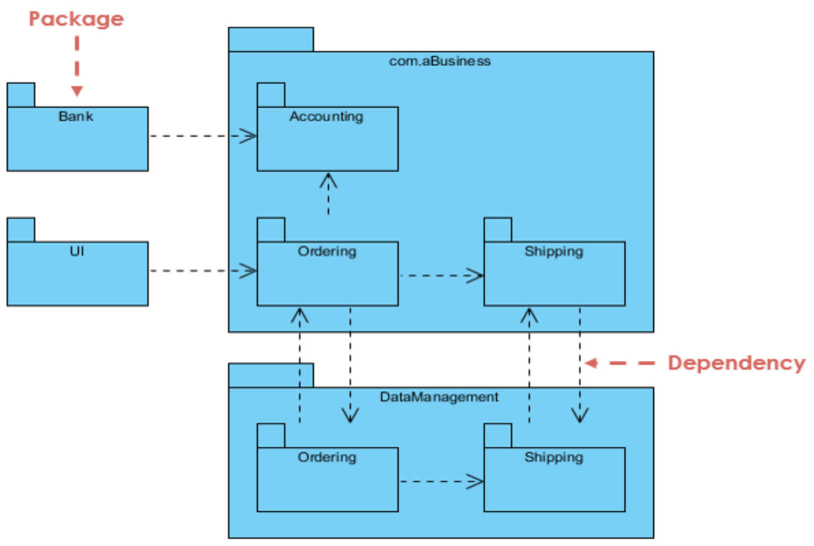
State Machine Diagram



Object Diagram

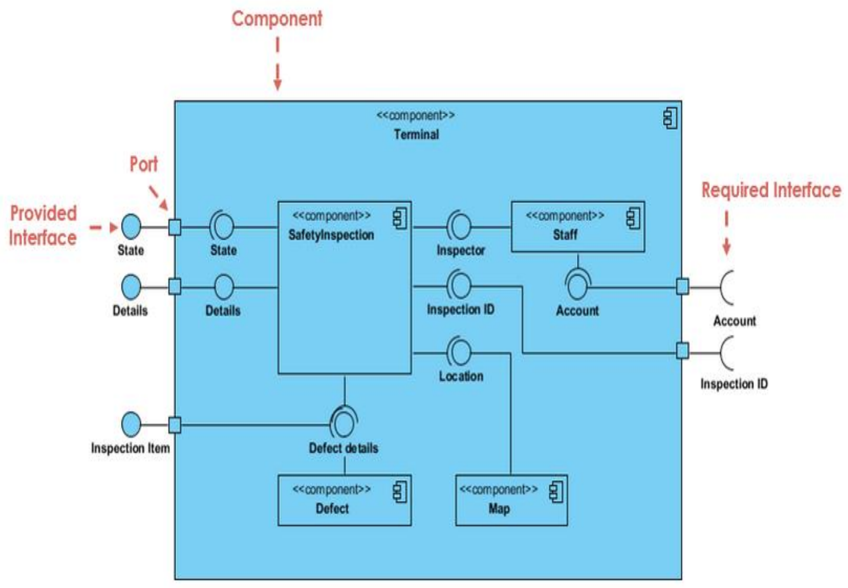


Package Diagram

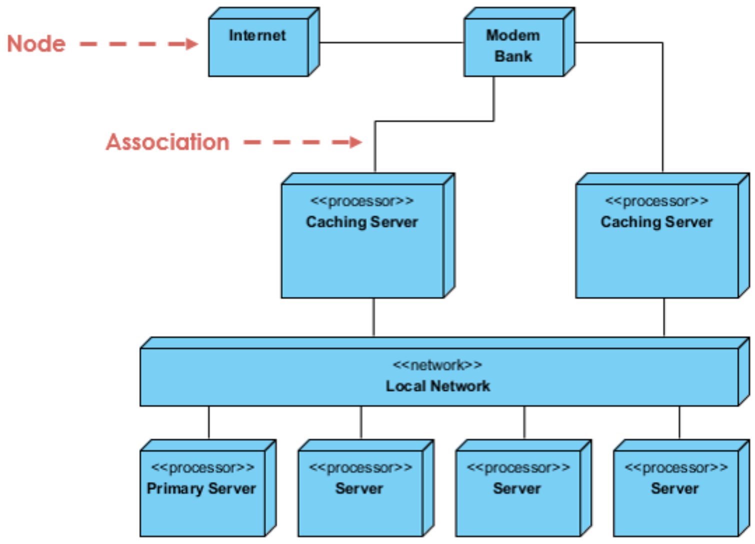


60

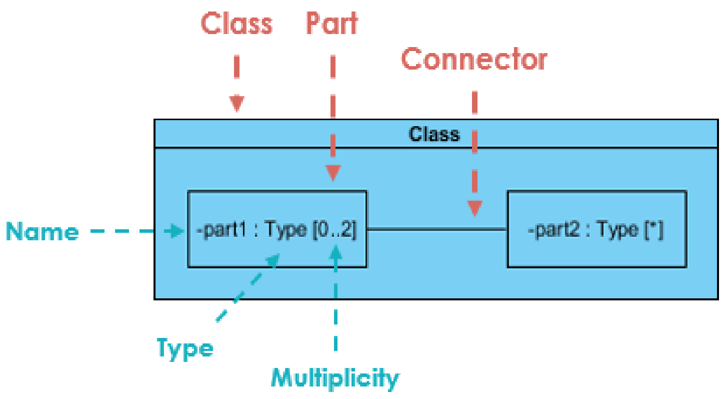
Component Diagram



Deployment Diagram

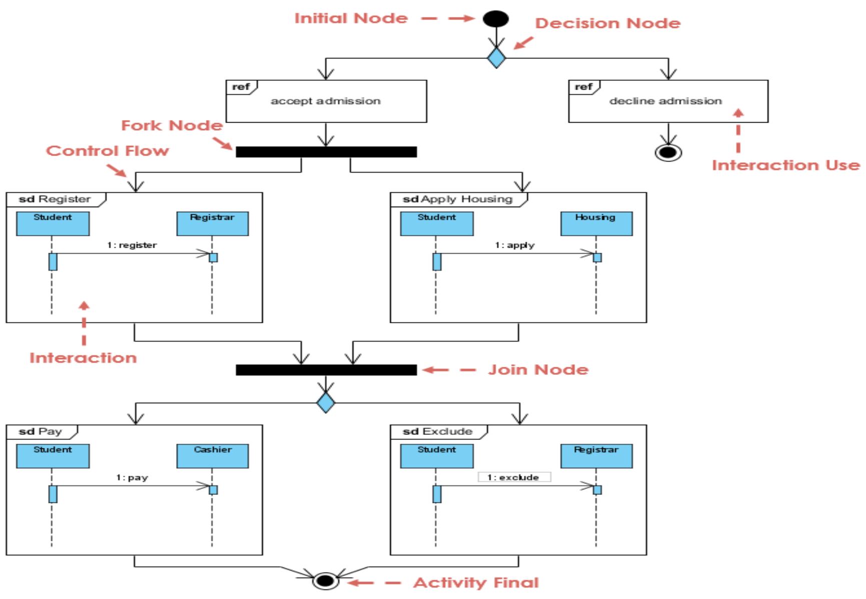


Composite Structure Diagram

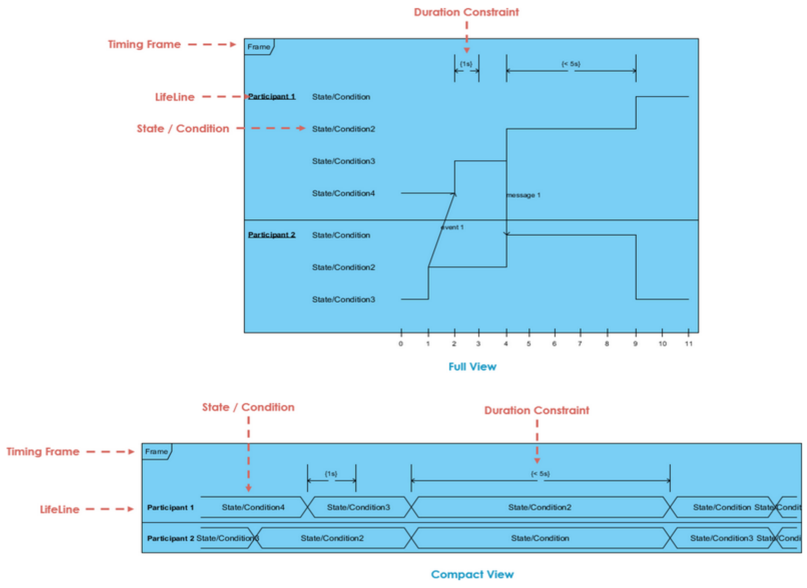


63

Interaction Overview Diagram

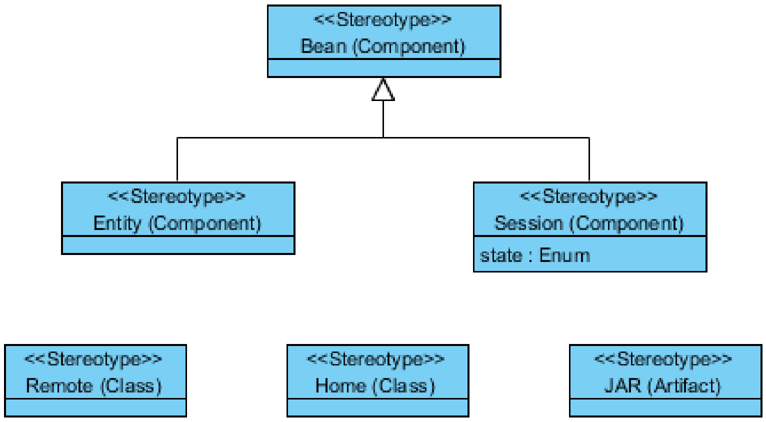


Timing Diagram



65

Profile Diagram


<<Stereotype>> Bean (Component)
<<Stereotype>> Entity (Component)
<<Stereotype>> Session (Component)
state : Enum
<<Stereotype>> Remote (Class)
<<Stereotype>> Home (Class)
<<Stereotype>> JAR (Artifact)

66

Rules of UML

- The UML's building blocks can't simply be thrown together in a random fashion. Like any language, the UML has a number of rules that specify what a well-formed model should look like.
- A well-formed model is one that is semantically self-consistent and in harmony with all its related models.
- The UML has semantic rules for

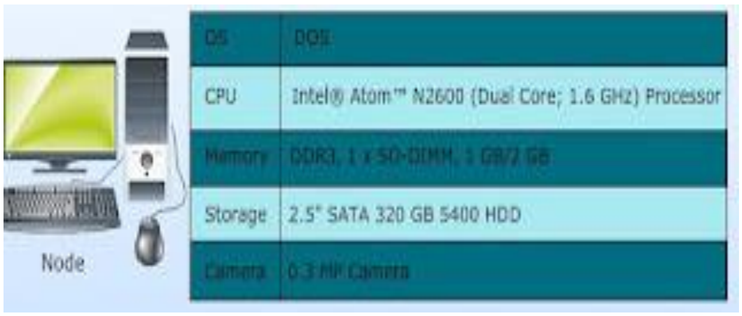
Names	What you can call things, relationships, and diagrams
Scope	The context that gives specific meaning to a name
Visibility	How those names can be seen and used by others
Integrity	How things properly and consistently relate to one another
Execution	What it means to run or simulate a dynamic model

Rules of UML

- Avoid models that are
  - Elided — certain elements are hidden for simplicity.
  - Incomplete — certain elements may be missing.
  - Inconsistent — no guarantee of integrity.
- Common Mechanisms in the UML
  - UML is made simpler by the presence of four common mechanisms that apply consistently throughout the language.
  - Specifications
  - Adornments
  - Common divisions
  - Extensibility mechanisms

Common Mechanisms in the UML

- **Specifications** : Specification provides a textual statement describing interesting aspects of a system

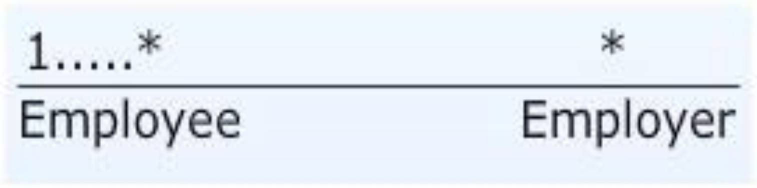


OS	DOS
CPU	Intel® Atom™ N2600 (Dual Core; 1.6 GHz) Processor
Memory	DDR3, 1 x SO-DIMM, 1 GB/2 GB
Storage	2.5" SATA 320 GB 5400 HDD
Camera	0.3 MP Camera

69

Common Mechanisms in the UML

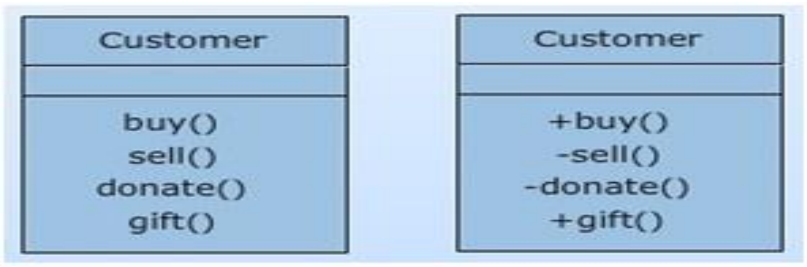
- **Adornments :** Textual/graphical items added to the basic notation of an element
  - They are used for explicit visual representation of those aspects of an element that are beyond the most important
  - **Examples:** The basic notation of association is line, but this could be adorned with additional details, such as the role names and multiplicity of each end



70

Common Mechanisms in the UML

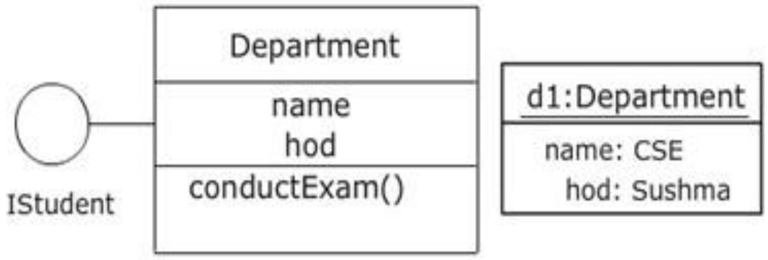
- **Adornments :** Textual/graphical items added to the basic notation of an element
  - Similarly, a class notation may highlight most important aspects of a class, i.e., name, attributes and operations. To show access specifies for the attributes and methods of a class adornments such as +, -, # are used.



71

Common Mechanisms in the UML

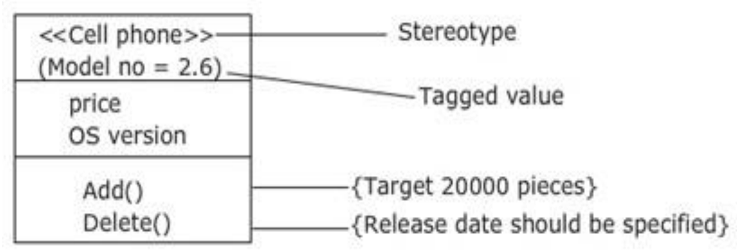
- **Common Divisions:** In modeling, object-oriented systems get divided in multiple ways.
- For example, class vs. object, interface vs. Implementation
- An object uses the same symbol as its class with its name underlined



72

Common Mechanisms in the UML

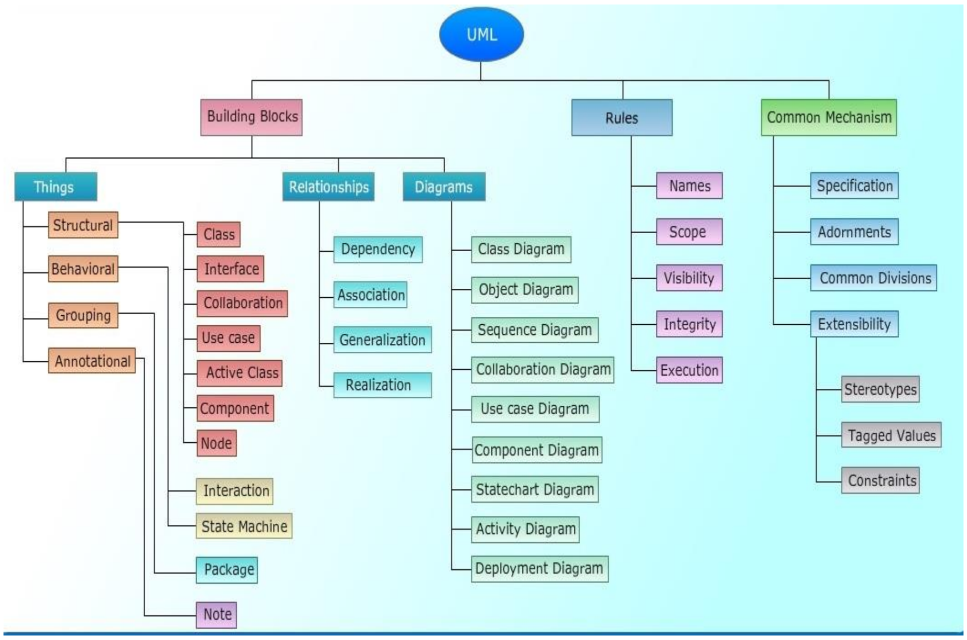
- **Extensibility Mechanisms :** Extensibility mechanisms allow extending the language in controlled ways. They include Sterotypes, Tagged Values and Constraints



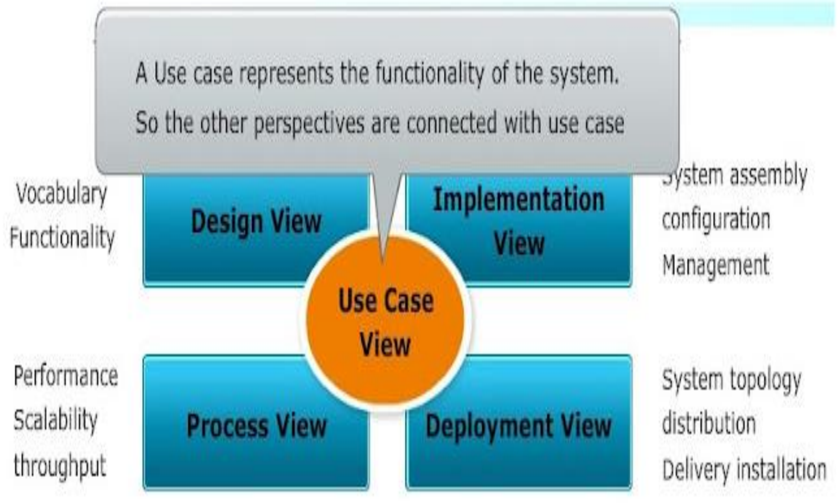
- Extensibility mechanisms can be classified in to
  - *Stereotypes*
  - *Tagged Values*
  - *Constraints*

73

Conceptual Models of UML



Architecture of UML



75

Software Development Life Cycle

- The UML is largely process-independent, meaning that it is not tied to any particular software development life cycle. However, to get the most benefit from the UML, you should consider a process that is
  - Use case driven
  - Architecture-centric
  - Iterative and incremental
- Use Case driven means that use cases are used as a primary artifact for establishing the desired behaviour of the system, for verifying and validating the system's architecture, for testing, and for communicating among the stakeholders of the project.
- Architecture-centric means that a system's architecture is used as a primary artifact for conceptualizing, constructing, managing, and evolving the system under development.

76

Software Development Life Cycle

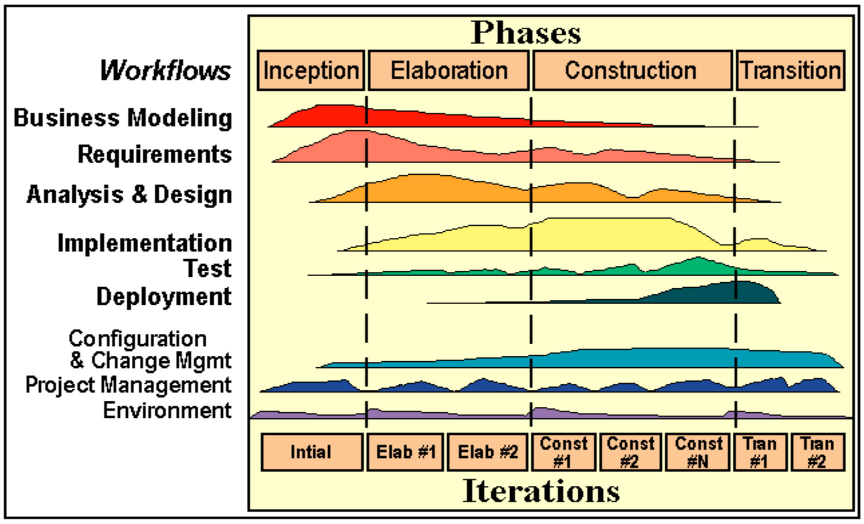
- An iterative process is one that involves managing a stream of executable releases. An is one that involves the continuous integration of the system's architecture to produce these releases, with each new release embodying incremental improvements over the other.
- Together, an iterative and incremental process is risk-driven, meaning that each new release is focused on attacking and reducing the most significant risks to the success of the project

Software Development Life Cycle

- This use case driven, architecture-centric, and iterative/incremental process can be broken into phases.
- A phase is the span of time between two major milestones of the process, when a well defined set of objectives are met, artifacts are completed, and decisions are made whether to move into the next phase.
- There are four phases in the software development life cycle:
  - inception,
  - elaboration,
  - construction, and
  - transition.

Software Development Life Cycle

- In the diagram, workflows are plotted against these phases, showing their varying degrees of focus over time.



Software Development Life Cycle

- *Critical activities in each phase:*
- *Inception:* Business case is established
  - 20% of the critical use cases are identified
- *Elaboration:* Develop the architecture
  - Analyze the problem domain (80% of use cases are identified)
- *Construction:* Source code
  - User manual
  - Verification and validation of code
- *Transition:* Deployment of software
  - New releases
  - Training

80

THANKS