Lecture 8: Introduction to UML

Learning Goals

- Describe the purpose of the **Unified Modelling Language** (UML)
- Argue why modelling is **important** in System Software Engineering
- List several types of **diagrams**, and give a short description of their purpose
- Outline instances where and how UML is used in practice

UML: Unified Modelling Language

UML is a general modelling **language** in software engineering to provide a standard way to **visualize** the **design** of a system.

Its main purpose is to help **communicate** and plan out

- structure
- behaviour
- interactions

in a software system. This becomes essential when working on large projects, and when working in a team where everyone needs to be on the same page.

Why Model at All?

If you are hammering together a simple bookcase with a few pieces of wood and a handful of nails, you may not need to spend much time on design.

By Gryffindor (Own work) [CC BY-SA 3.0]

Imagine building this one without a plan.

The Leistler Bookcase took over a year to build, and was shown off at the Great Exhibition in London (1851) as a demonstration of the “wonders of industry”.

Why Model at All?

Software systems are becoming more and more **complex**. With the ubiquity of computers (smart phones, smart homes, smart clothing, automation), the desire for everything to be connected, and the incredible computing power available, more and more is being **demanded** of software engineers.

Have you ever been faced with a complex problem and had to draw something out? How about when trying to explain something to a friend?

Diagrams allow us to **decompose** complex systems into simpler **abstract** components and **visualize** them. This can help us **understand** and better **communicate** the "big picture" ideas behind a system, and give us a road-map for development.

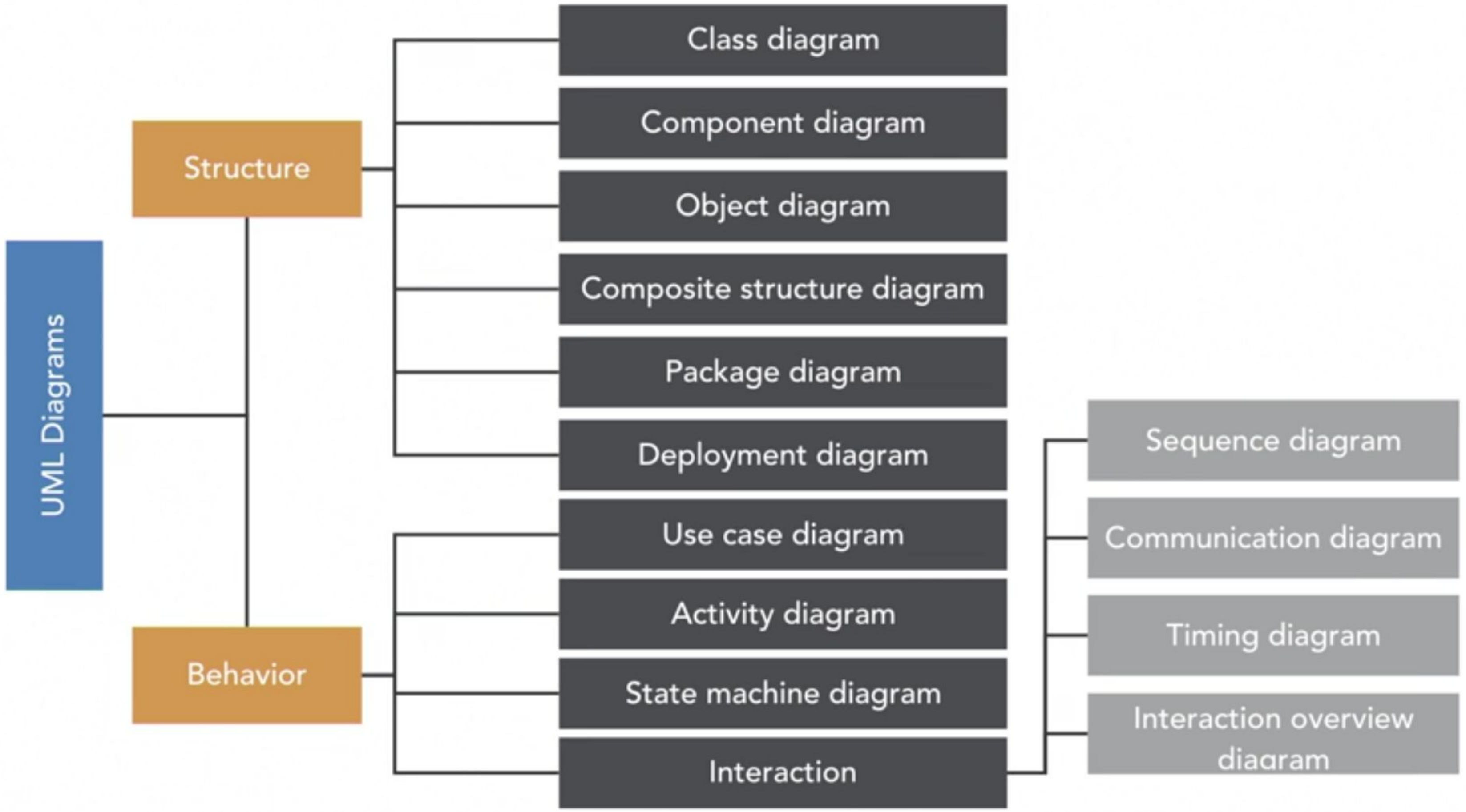
UML: Unified Modelling Language

Mathematicians use algebraic symbols to communicate, electrical engineers have circuit notation, software engineers have evolved their own **notation** for describing the **architecture** and **behaviour** of software systems.

UML, the **Unified Modelling Language**, is that standardized language. It has **syntax** and **semantics** to convey meaning, allowing two people fluent in that language to communicate and understand the intention of the other.

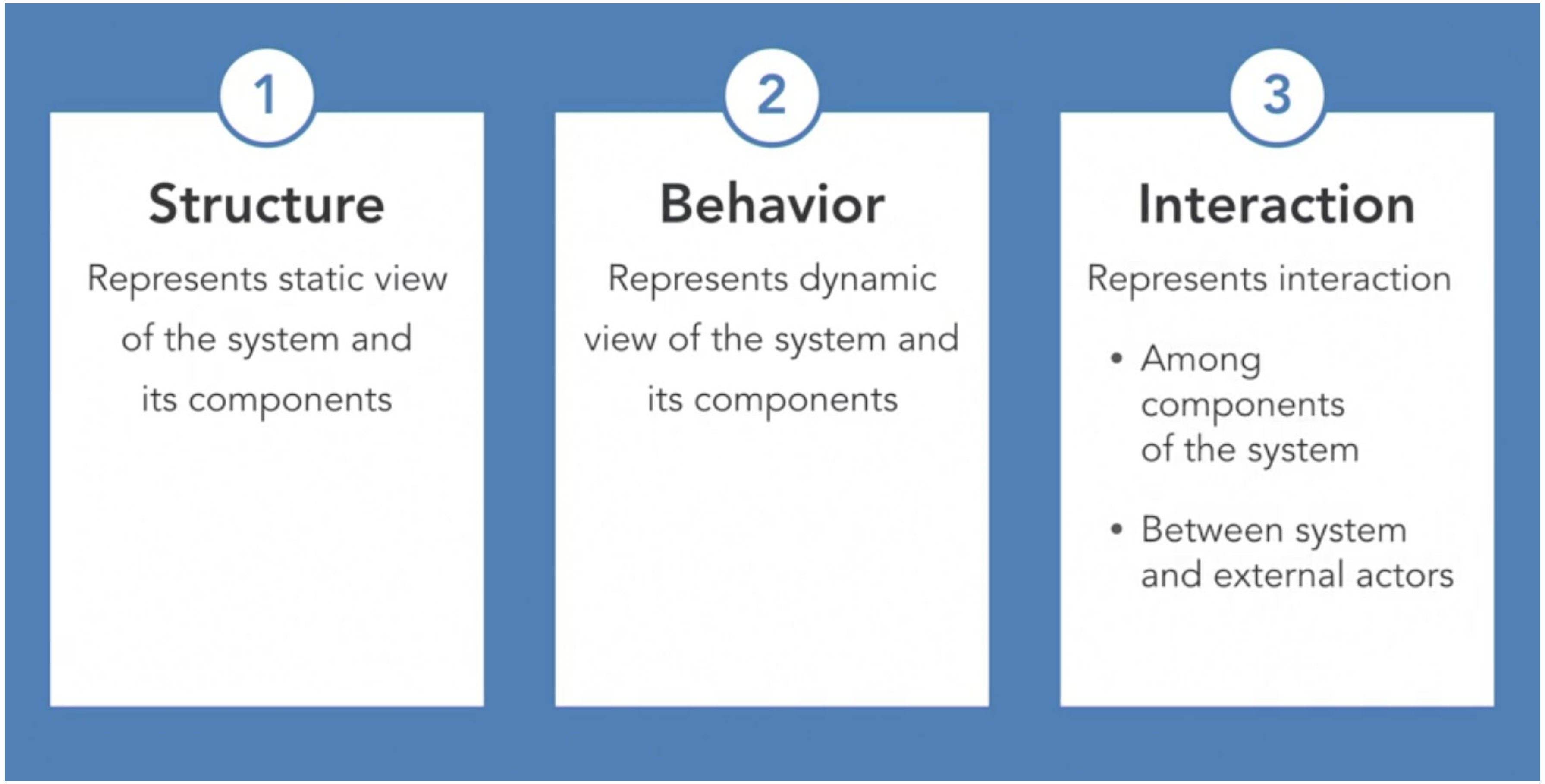
There are a set of **13** essentially **graphical** notations, supplemented by text, designed to capture **requirements** and **design alternatives**. You don't always need all 13 diagrams; you choose the ones that capture important information about the system you are working on.

UML Types



Software Design: Modeling with UML, by Neelam Dwivedi

UML Category Types



1

Structure

Represents static view of the system and its components

2

Behavior

Represents dynamic view of the system and its components

3

Interaction

Represents interaction

- Among components of the system
- Between system and external actors

Software Design: Modeling with UML, by Neelam Dwivedi

More Common Types of Diagrams

Structure:

- **Class Diagrams:** outline the different entities in a system, and their relationships with each other. It shows the structural breakdown of the software.
- **Deployment Diagrams:** show where each of your software modules are deployed in the physical system and how they communicate.

Behaviour:

- **Use-Case Diagrams:** document high-level functional requirements, and relationships with users and other systems (actors). These outline every **observerable** function your system must perform.

More Common Types of Diagrams

Behaviour (cont.):

- **State Diagrams:** outline the time-dependent changes in state and transitions of each major object or interaction in your system.
- **Activity Diagrams:** model high-level activities and transitions between system states, shows concurrency of activities.

Behavior-Interaction:

- **Sequence Diagrams:** show the detailed flow of execution of events, and relative timings between them; these model the behaviour and the interactions between collaborating objects.

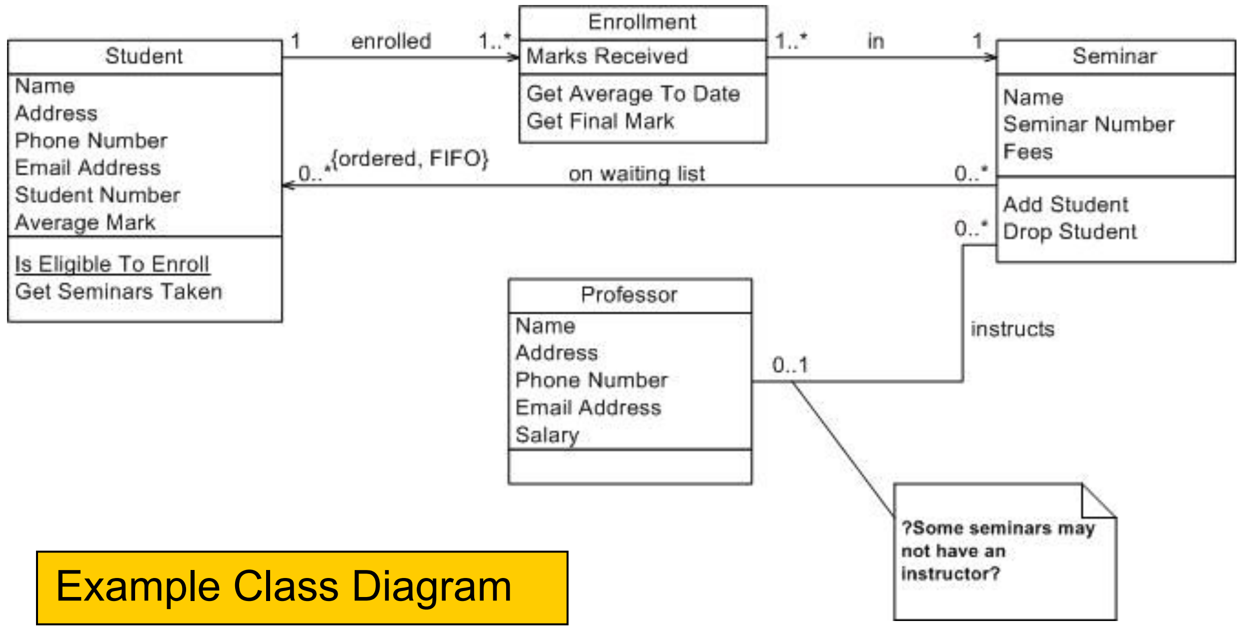
http://www.agilemodeling.com/essays/umlDiagrams.htm

Class Diagram

Relationship between objects/classes.

Can be more abstract and design-centered, or specific and implementation-centered.

Explores software **architecture**, functionality and relationships between objects in our system (i.e. instances of classes).

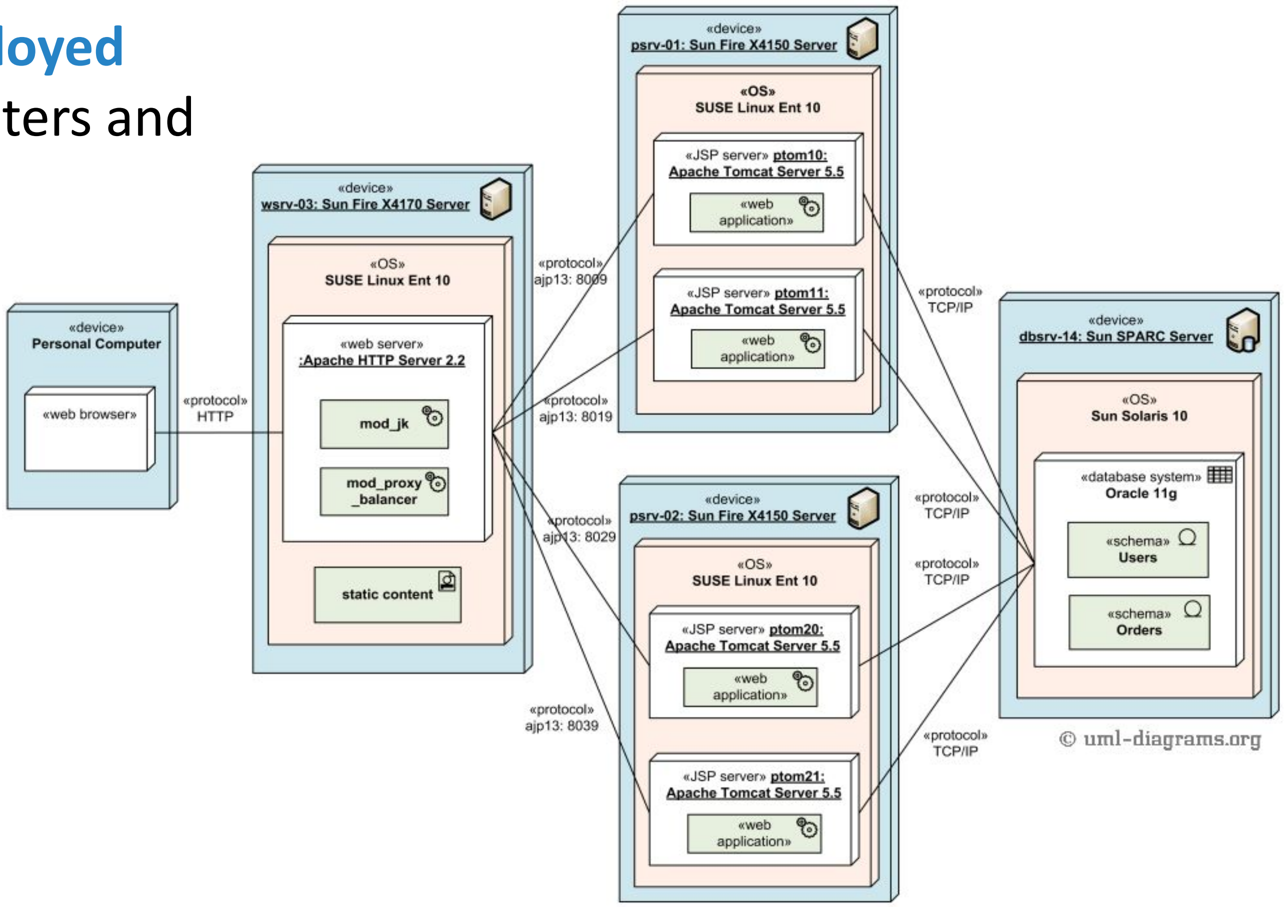


Example Class Diagram

Deployment Diagram

Show how complex software will be **deployed** (installed) across a distribution of computers and networks.

Gives an indication of the kinds of runtime resources are required



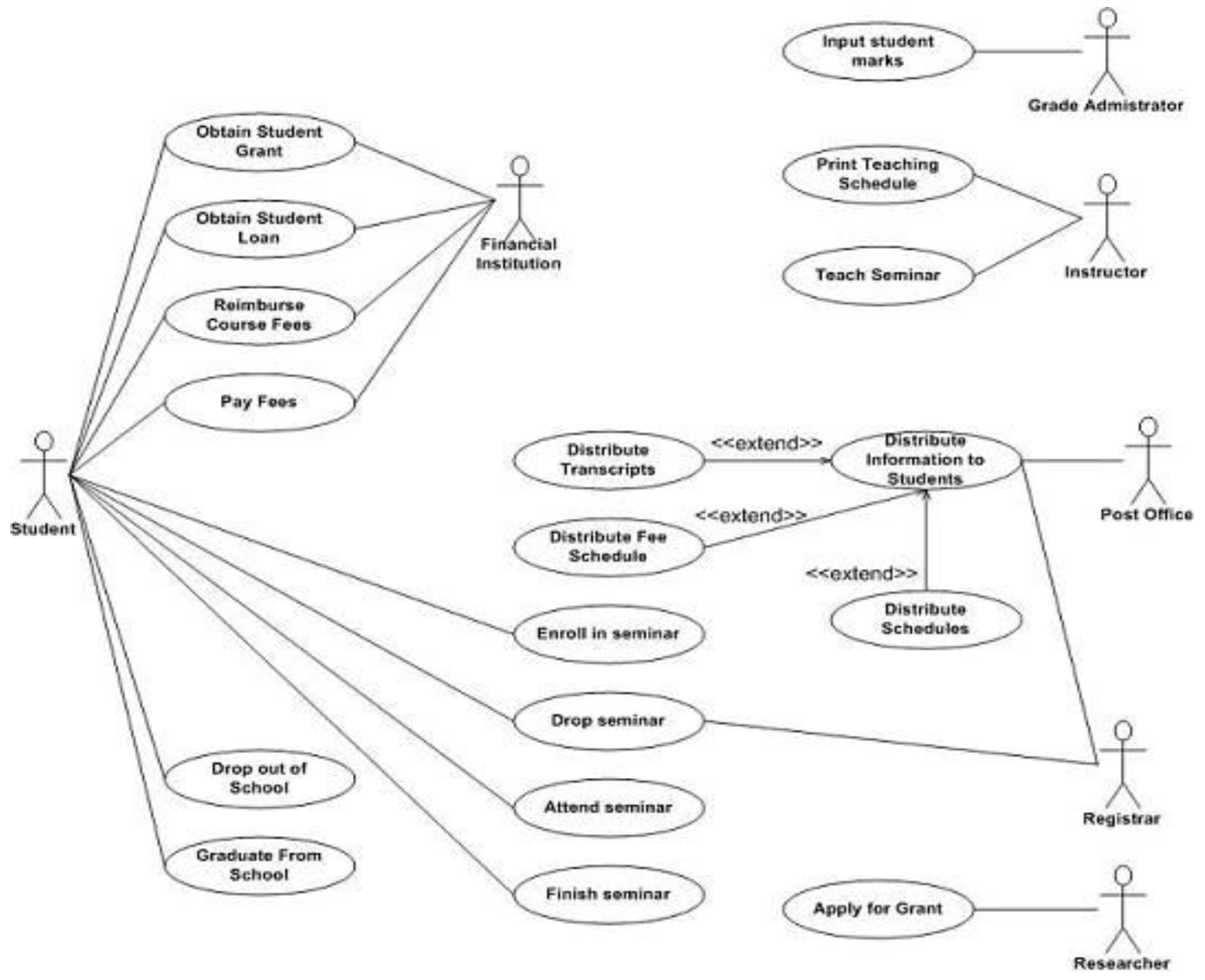
Use Case Diagram

Used for analyzing **requirements**, exploring user interactions.

Documents:

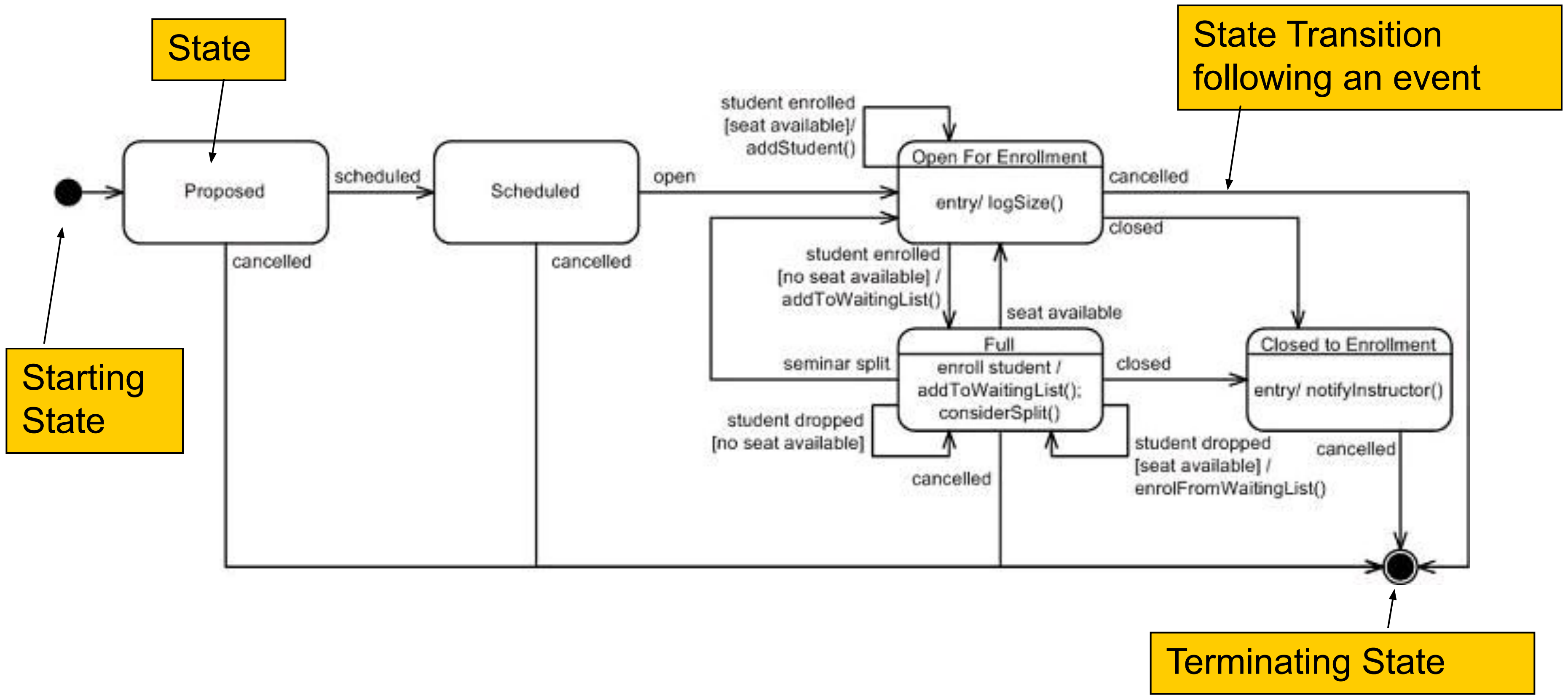
- Who initiates an interaction
- What information enters the system
- What information comes out
- Measurable benefits to the user

Requirements analysis uncovers **functionality** the system must provide to satisfy its users.



Example Use-case Diagram for a student database

State Chart Diagram



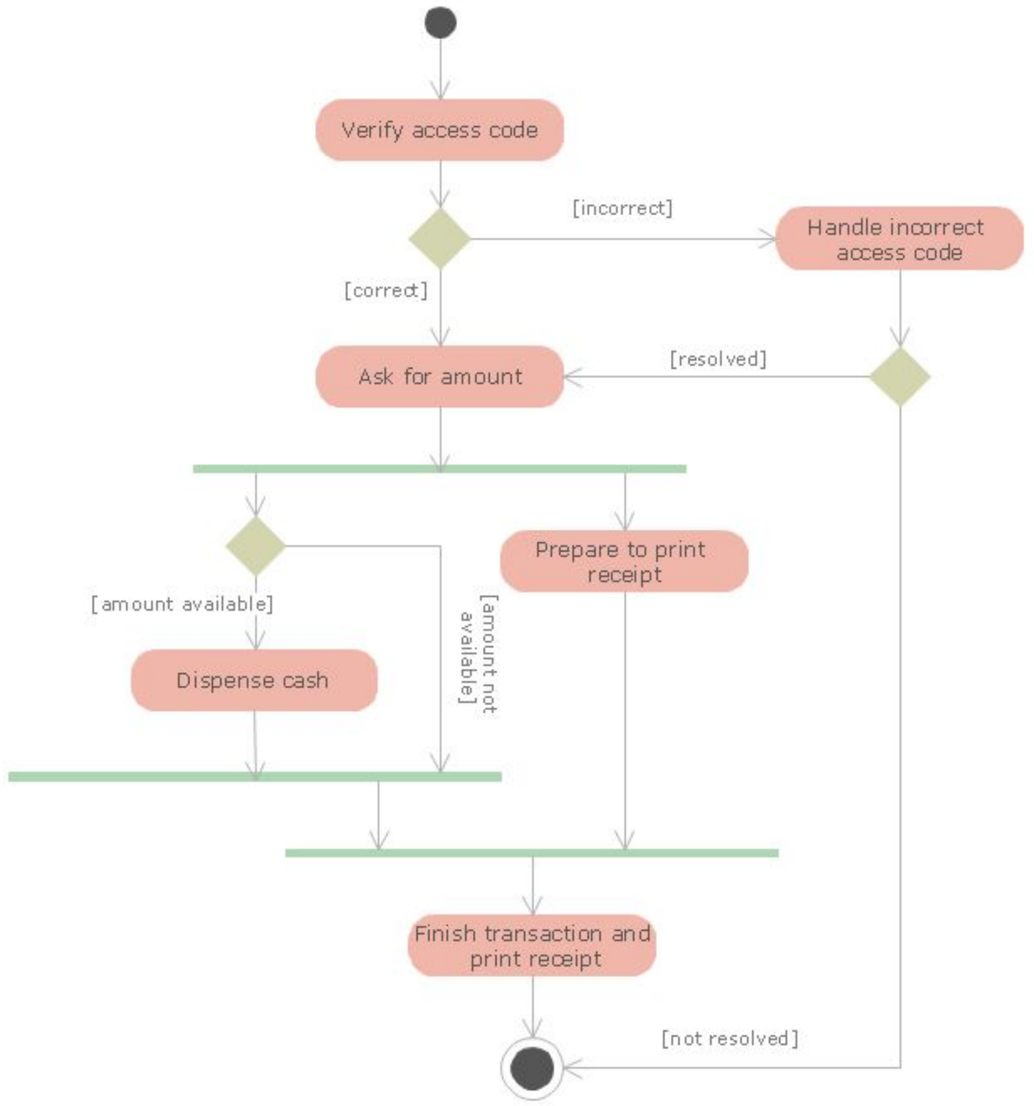
State
State Transition following an event
student enrolled [seat available]/ addStudent()
Proposed
scheduled
Scheduled
open
Open For Enrollment
entry/ logSize()
cancelled
closed
cancelled
cancelled
student enrolled [no seat available] / addToWaitingList()
seat available
seminar split
Full
enroll student / addToWaitingList(); considerSplit()
closed
Closed to Enrollment
entry/ notifyInstructor()
student dropped [no seat available]
cancelled
student dropped [seat available] / enrolFromWaitingList()
cancelled
Starting State
Terminating State

Model the **time dependent behaviour** of objects or systems in response to messages sent to it over a period of time.

Activity Diagram

Show the **procedural flow** of control while processing an activity, modelling the logic in a use-case or use-case scenario.

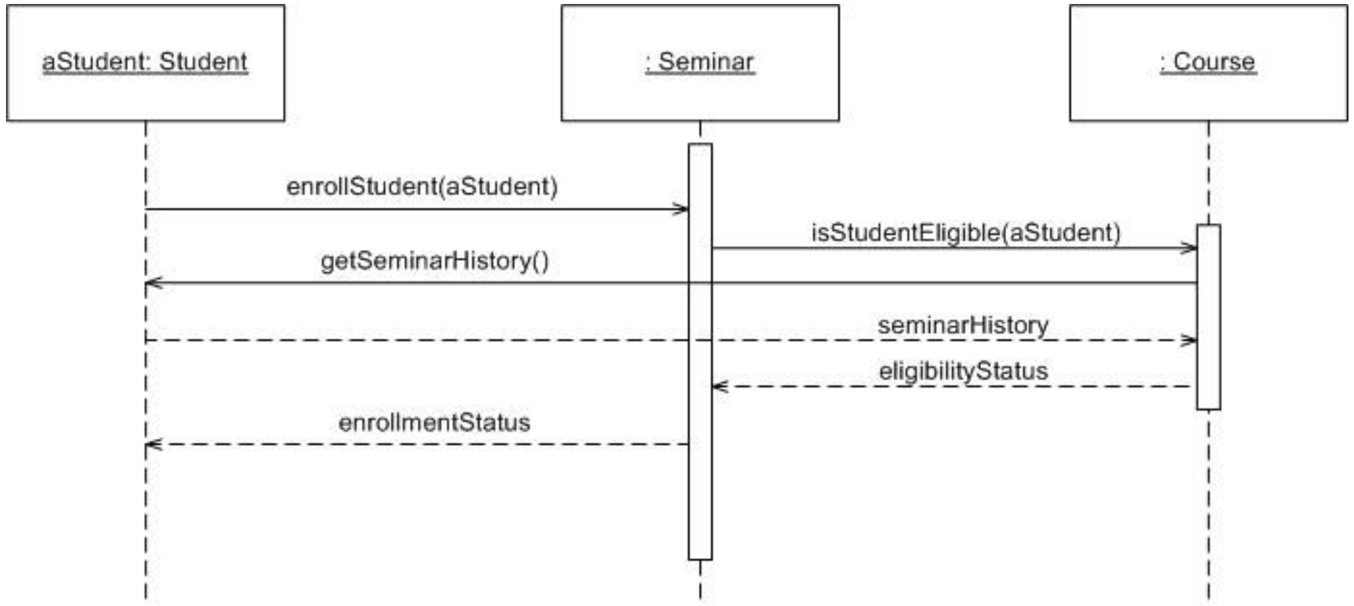
Green bars indicate creation and joining of parallel sections.



Sequence Diagram

Model the **interaction** of collaborating objects using **message passing** as they attempt to achieve the functionality expressed in one or more use cases.

This models the **behaviour** of the system in response to inputs from the external world.



Example Sequence Diagram

UML In Practice

Is UML and modelling actually **used** in practice?

It depends…

- In military, security, and safety critical applications, a full detailed design with documentation is often required *before* you even start programming.
- Some firms insist on a formal UML approach. This can be useful when you want to generate code automatically (or using interns) from a model. This is known as **Model-Driven Development (MDD)**.
- Some clients/customers may insist on full documentation with formal UML.
- Many developers use UML *informally*, documenting enough ideas to allow the team to understand what and how things will work.

Homework

Download the **Lecture** examples L2-L7 and practice the multi-threading and mutex concepts

https://github.com/alimousavifar/Lecture_examples_public