Software Architecture

Tamlin Love
Shamelessly stolen and adapted from
Rylan Perumal and Terence van Zyl

Recap

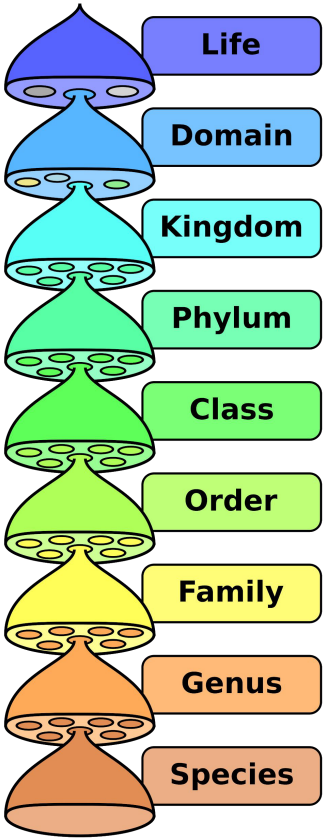
- Requirements Engineering
  - Gathering, Analysis and Specification
- Types of Requirements
  - Functional, Non-functional, On-screen appearance
- User Stories
  - Who-What-Why
- Acceptance Tests
  - UAT: Given-When-Then
- Sizing requirements with scrum

Today's Topics

- Software Architecture Definition
- Architectural Decisions & Key Concerns
- Architectural Styles (Brief Overview)
- Documenting Architecture: Views and UML

Hierarchical Organisation of Software

- **Software has structure**
  - Not just a long list of code
- Software can be organised hierarchically
  - Data and memory
  - Algorithms
  - Objects
  - Module
  - Component
  - System
  - System of systems
- How do we decompose software systems into parts?



Why Decompose Systems?

- Recall: Tackle complexity with **divide-and-conquer**
  - Little problems are easier to solve than big ones
- **Don’t reinvent the wheel**
  - Focus on creative parts; if something already exists, reuse it
- **Separation of concerns**
  - Support flexibility and future evolution by decoupling unrelated parts, so each can evolve separately

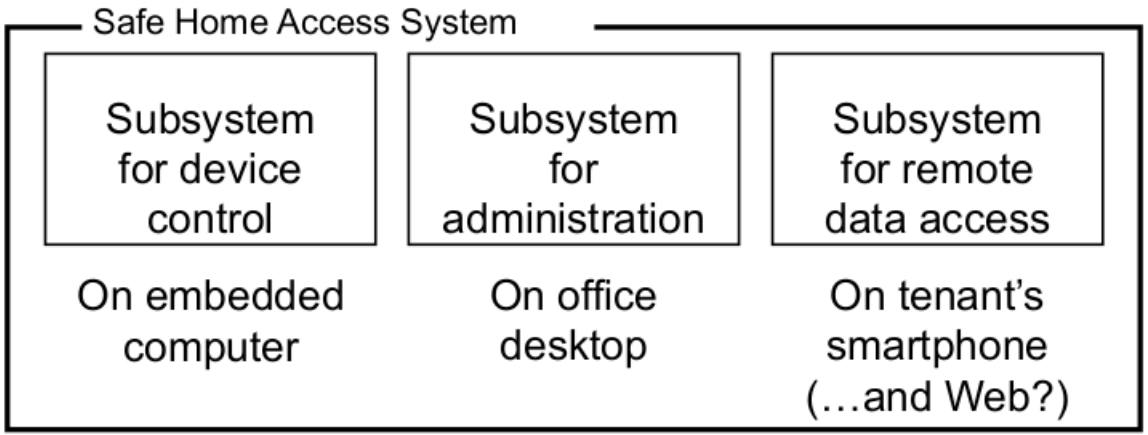
Software Architecture

- **Software architecture** refers to the set of high-level decisions that determine the structure of the software solution
  - Parts of the system
  - Relationships between parts
- These decisions are often made early in the design process and affect large parts of the system
  - Can be difficult to modify later
- Aim to use well-known solutions that are proven to work for similar problems
- Software architecture is not a phase of development, rather it refers to the structure of the software system at any point in development

Example Architectural Decisions

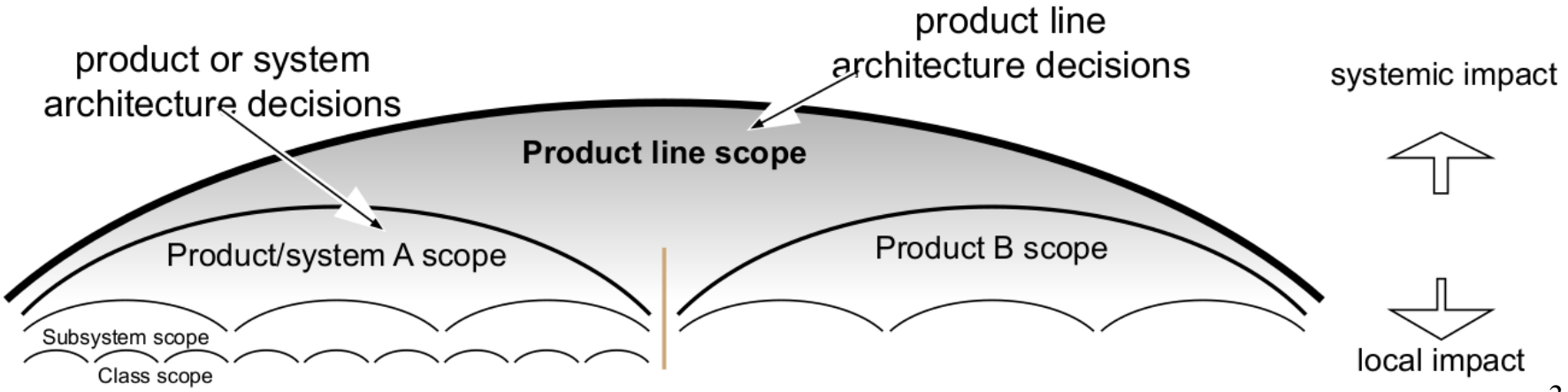
- Example decisions:
  - How do we decompose the system?
  - How do we map software to hardware?
  - What development platforms, deployment operating systems, etc. will be used?
  - How will these subsystems interact?
    - "Architectural style"

Example: Home security system



Architectural Decisions: Scope

- Given the current level of system scope, a decision is “architectural” if it can be made only by considering the present scope
  - i.e. could not be made from a more narrowly-scoped, local perspective
  - Architectural decisions should focus on high impact, high priority areas that are in strong alignment with the business strategy



Key Decisions

- System decomposition
  - How do we break the system up into parts?
    - What functionality/behavior to include?
  - Do we have all the necessary parts?
  - How do the parts interact?
- Cross-cutting concerns
  - Broad-scoped qualities or properties of the system
    - i.e. non-functional requirements
  - Usually a trade-off between qualities
- Conceptual integrity
  - Does the system make sense?

Architecture vs Design

- Architecture focuses on non-functional requirements and on the decomposition of a system
- Design focuses on implementing functional requirements
- Note: the border is not always clear

Compromise

- Architectural decisions always involve compromise
- The “best” design for a component considered in isolation may not be chosen when components considered together or within a broader context
  - e.g. car components may be “best” for racing cars or “best” for luxury cars, but will not be best together
- Additional considerations:
  - Business priorities
  - Available resources
  - Target customers
  - Competitors’ moves
  - Technology trends
  - Backward compatibility
  - Security needs
  - etc.

Architectural Styles

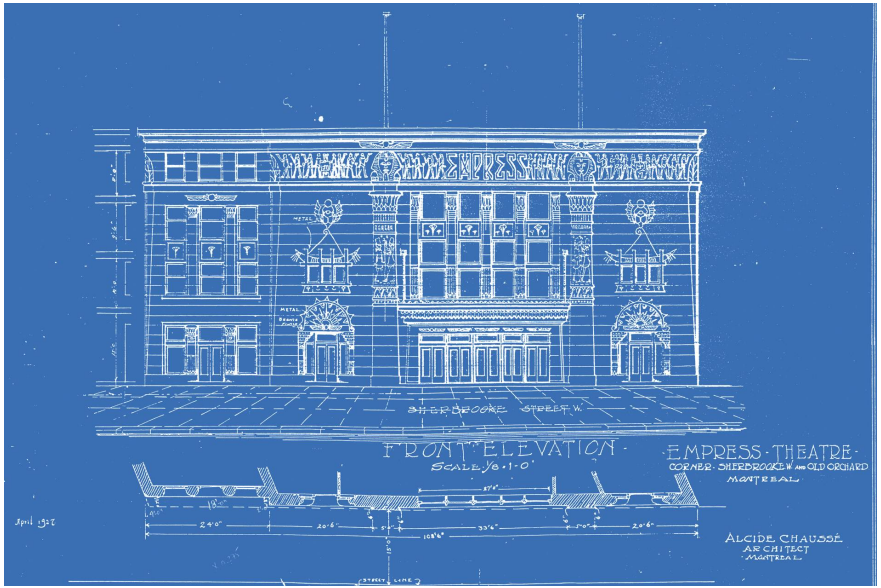
- An **architectural style** (or **architectural pattern**) is a general, reusable solution to a commonly occurring problem in software architecture within a given context
  - Different from, but related to, design patterns

Architectural Styles

- Some well known architectural styles include:
  - Monolithic
  - Client-Server
  - Peer-to-Peer
  - Multi-Tier
  - Pipe and Filter
  - Database-Centric
  - Event-Driven
  - Microservices
  - Microkernel
  - Representational State Transfer (REST)
  - Service-Oriented
  - Space-Based
- Not an exhaustive list!

Describing Architecture

- How do we describe the components of a system and how they interact?
- Each component must be identified and its interface described. Interfaces must be:
  - Fully documented
  - Semantics, not just syntax
  - Understandable
  - Unambiguous
  - Precise
- Adding semantics:
  - Informal description
  - Design models (UML diagrams)
  - Pre/Post conditions

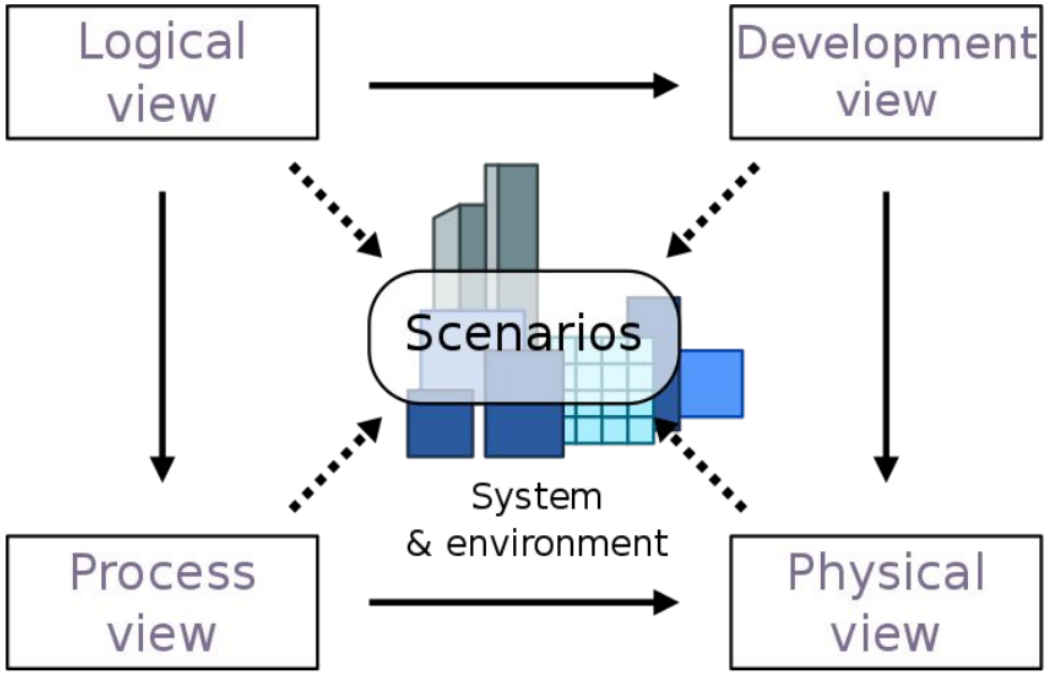


Architectural Views

- **Views** are different kinds of “blueprints” created for the system-to-be
  - e.g. Blueprints for buildings: construction, plumbing, electric wiring, heating, air conditioning, etc.
  - Different stakeholders have different information needs
- Each view has a purpose, and only shows the information related to that purpose
  - Unnecessary information would clutter the view, making it unintelligible

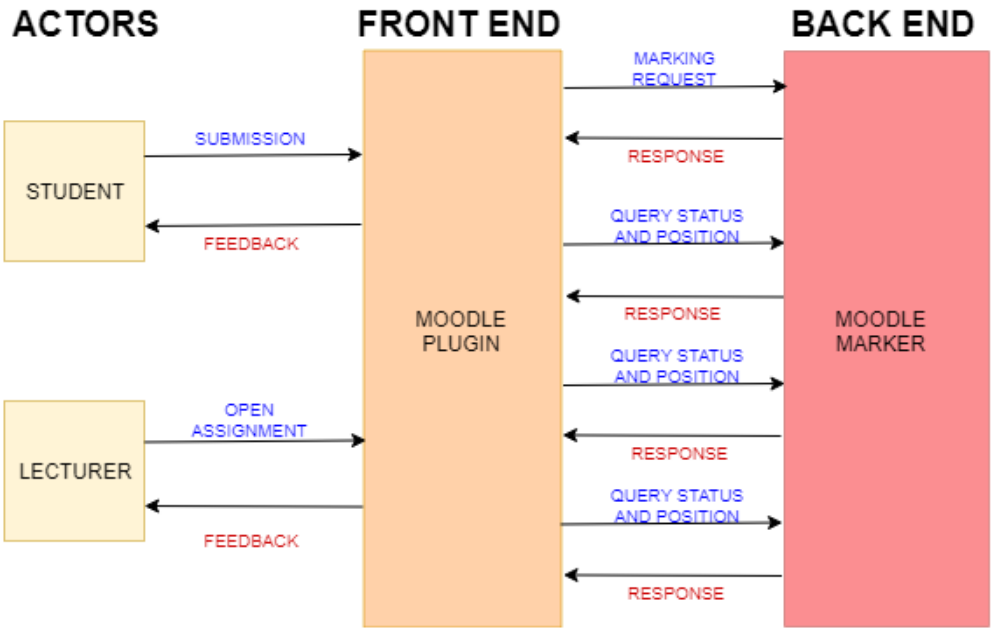
Architectural Views

- **4 + 1 Architectural View Model**
  - **Logical View**
    - State Diagram
    - Class Diagram
  - **Development View**
    - Component Diagram
  - **Process View**
    - Activity Diagram
    - Sequence Diagram
  - **Physical View**
    - Deployment Diagram
  - **Scenarios**
    - Use Case Diagram



Running Example

- Wits Moodle Marker
  - Lecturers create assignments and test cases
  - Students submit code
  - Marker runs code and gives feedback

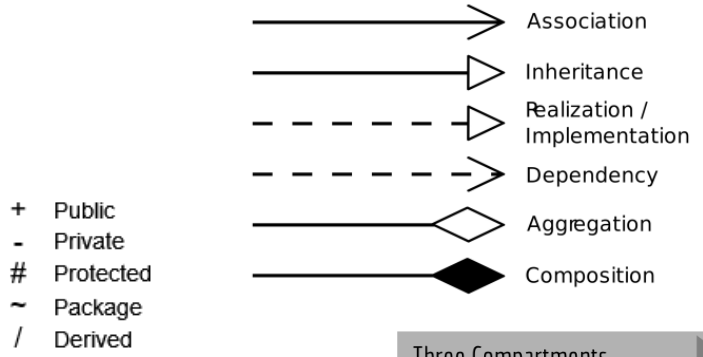


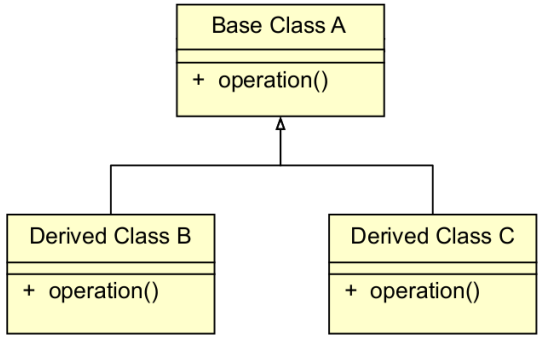
Logical View

- The **logical view** is concerned with the functionality provided to users
- Focuses on the decomposition of the system into a set of key abstractions, namely objects and classes
- Represented in UML using **class diagrams** and **state diagrams**

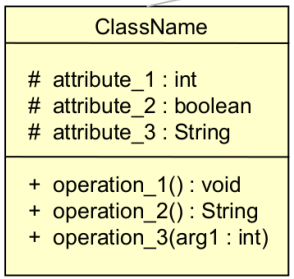
Logical View – Class Diagrams

- See Lecture 2 – The Object Model
- Static view of the classes/objects in a system
- Can be generated from code
  - See tools like Umbrello, Smartdraw, etc.

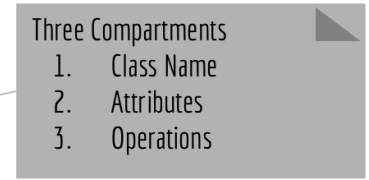




Inheritance

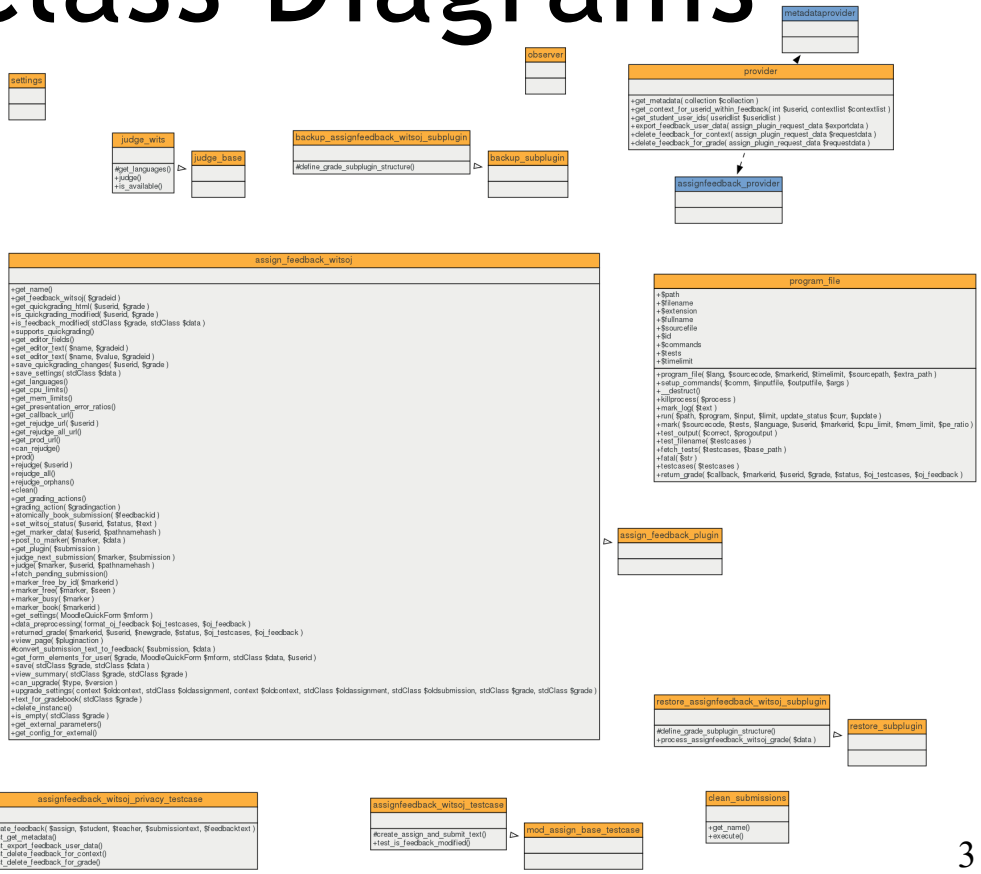


Software Class



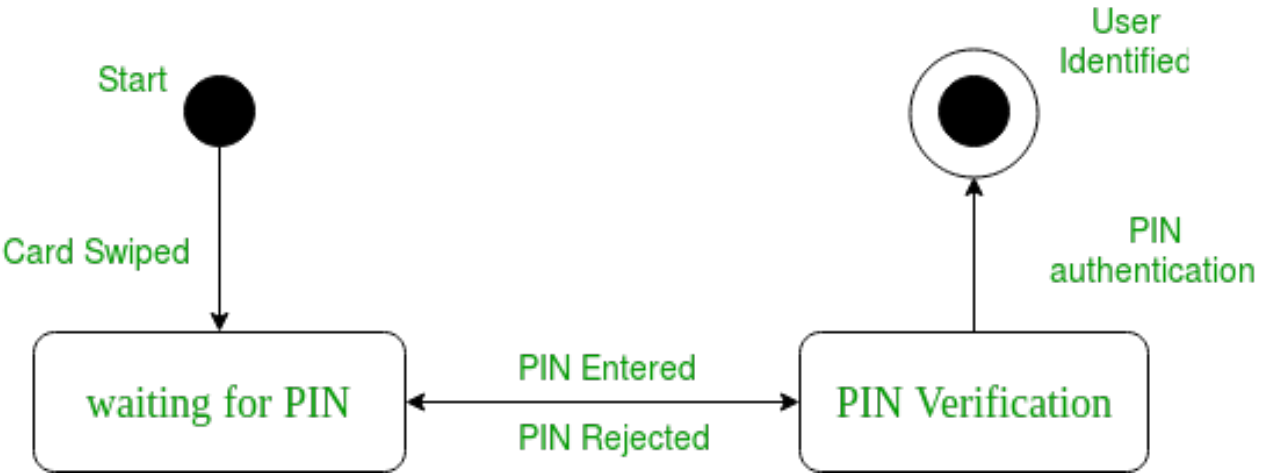
Logical View - Class Diagrams

- Example: Wits Moodle Marker
- Note: automatically generated



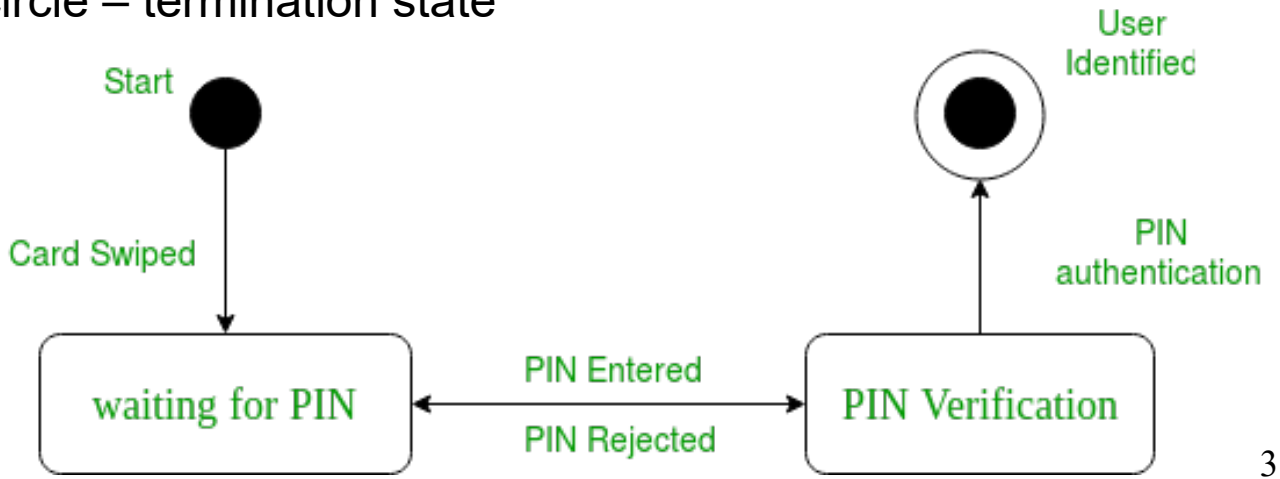
Logical View – State Diagrams

- **State diagrams** (also known as **statechart diagrams**) give a dynamic view of objects
- Define states for each object and how objects transition between states
- Models and object's lifetime from creation to termination



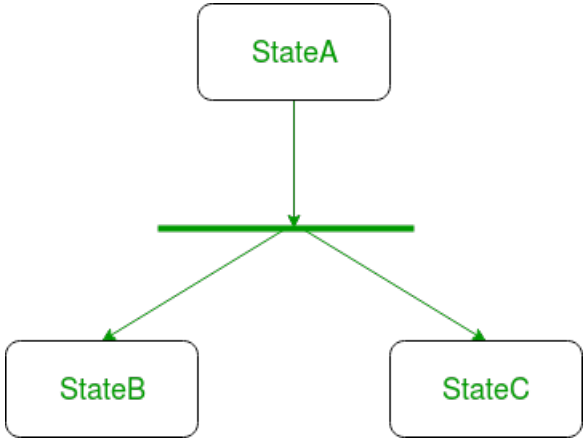
Logical View – State Diagrams

- How to draw:
  - Circle – initial state
  - Rectangle - state
  - Arrow – transition between state, labelled with text
  - Circle with outer circle – termination state

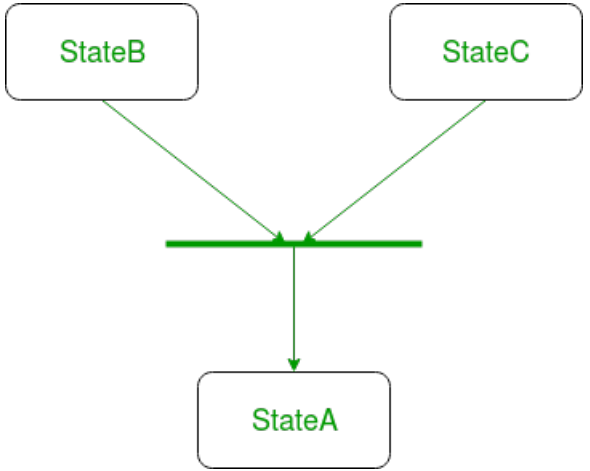


Logical View – State Diagrams

- We sometimes represent forks and joins with a bar



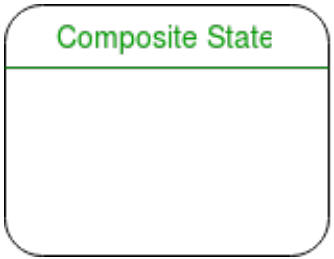
Fork

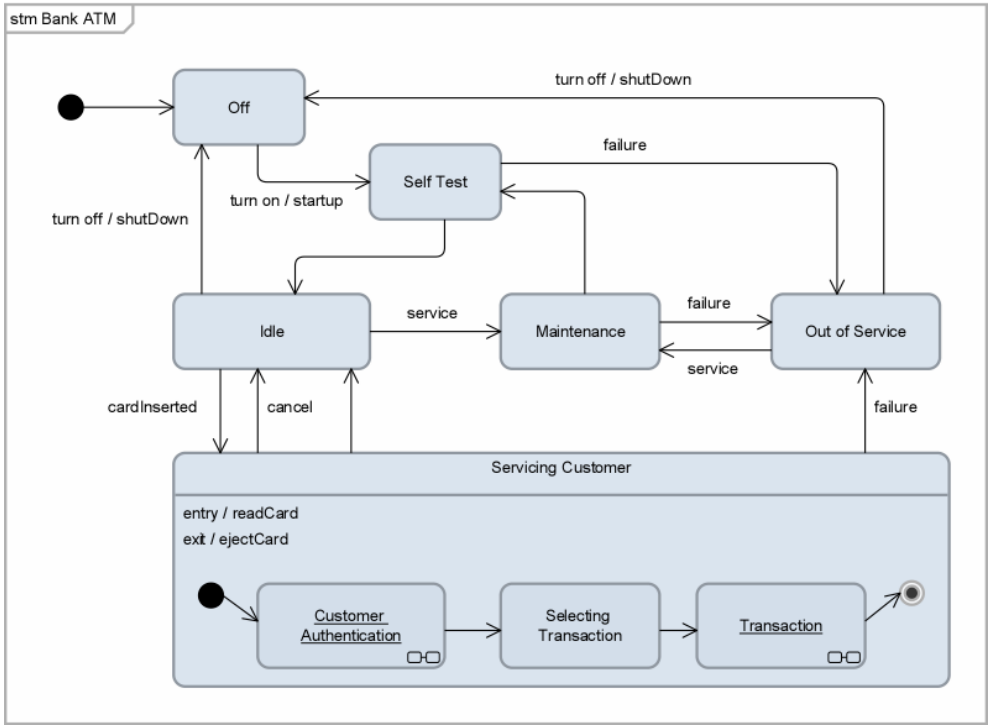


Join

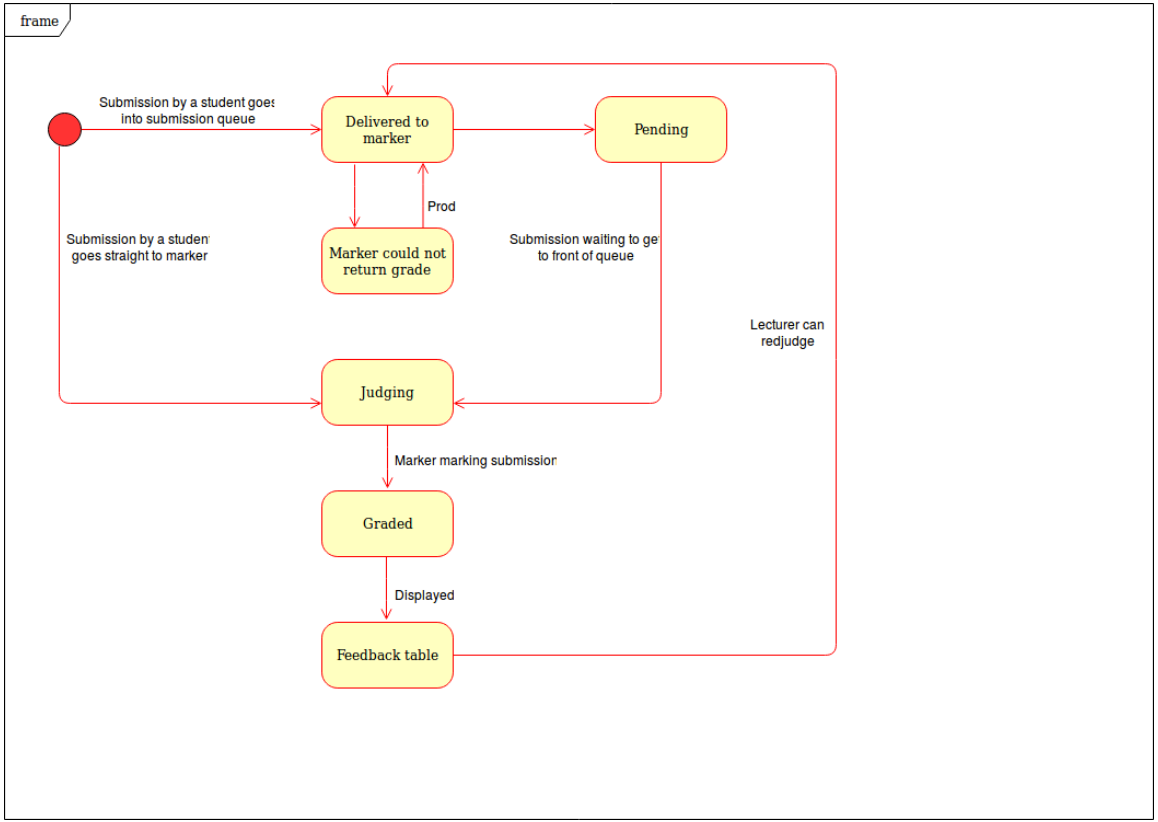
Logical View – State Diagrams

- We can represent subprocesses using composite states





Logical View – State Diagrams

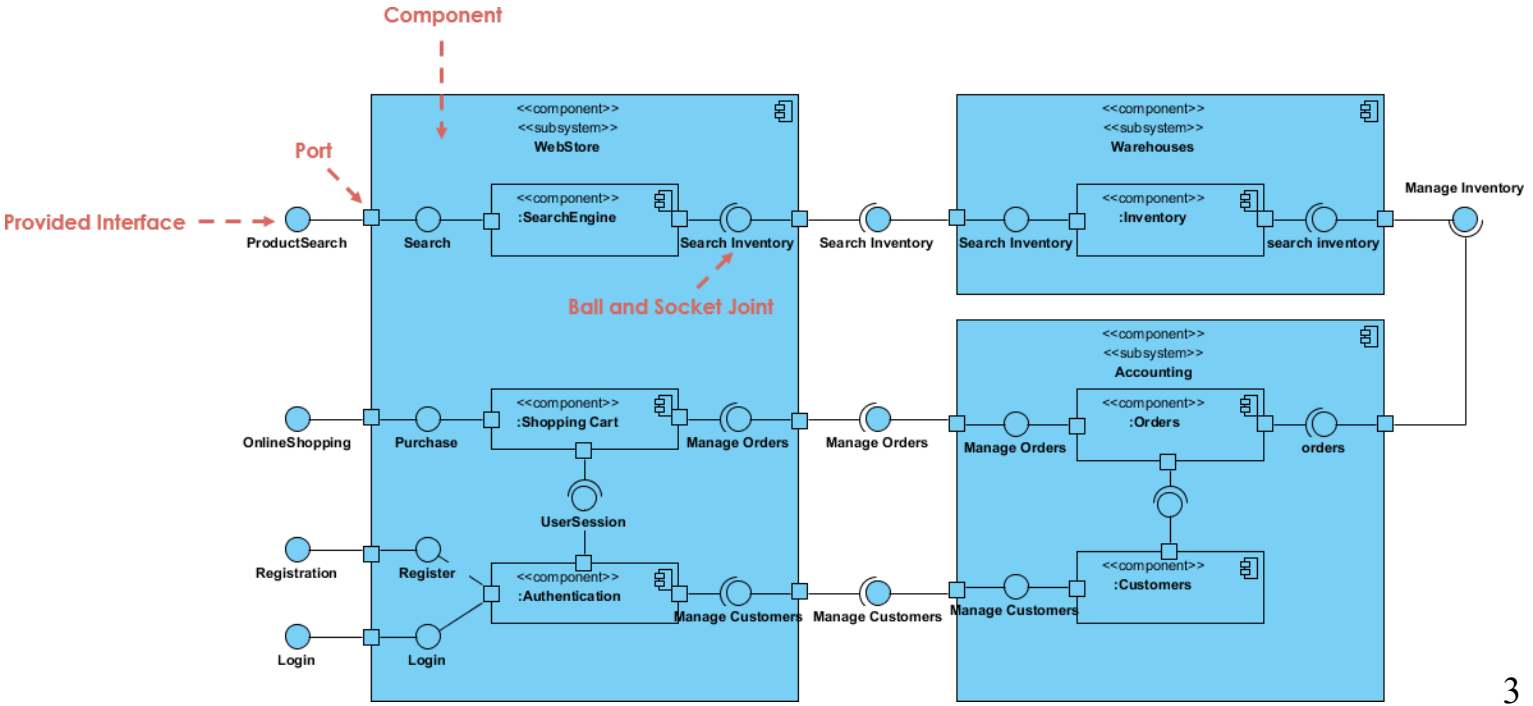


Development View

- The **development view** is concerned with software administration from the point of view of the programmer
- Focuses on **components**: modules of classes that represent independent systems or subsystems with the ability to interface with the rest of the system.
- Represented in UML using **component diagrams**

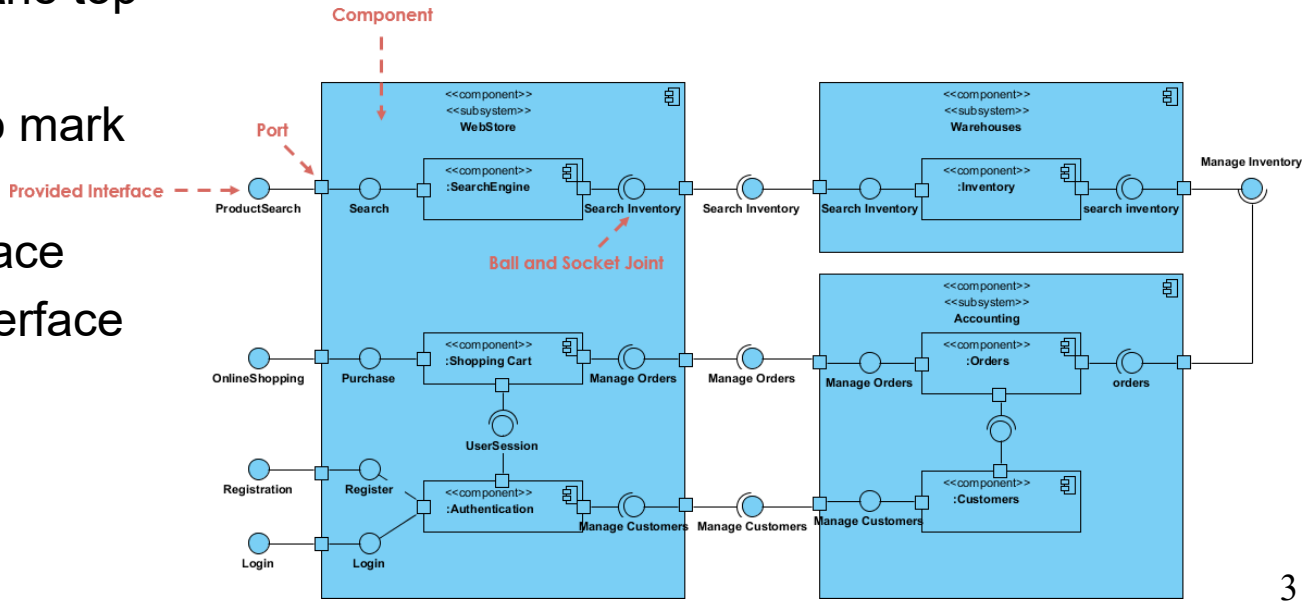
Development View – Component Diagrams

- **Component diagrams** show how different components in a system interact

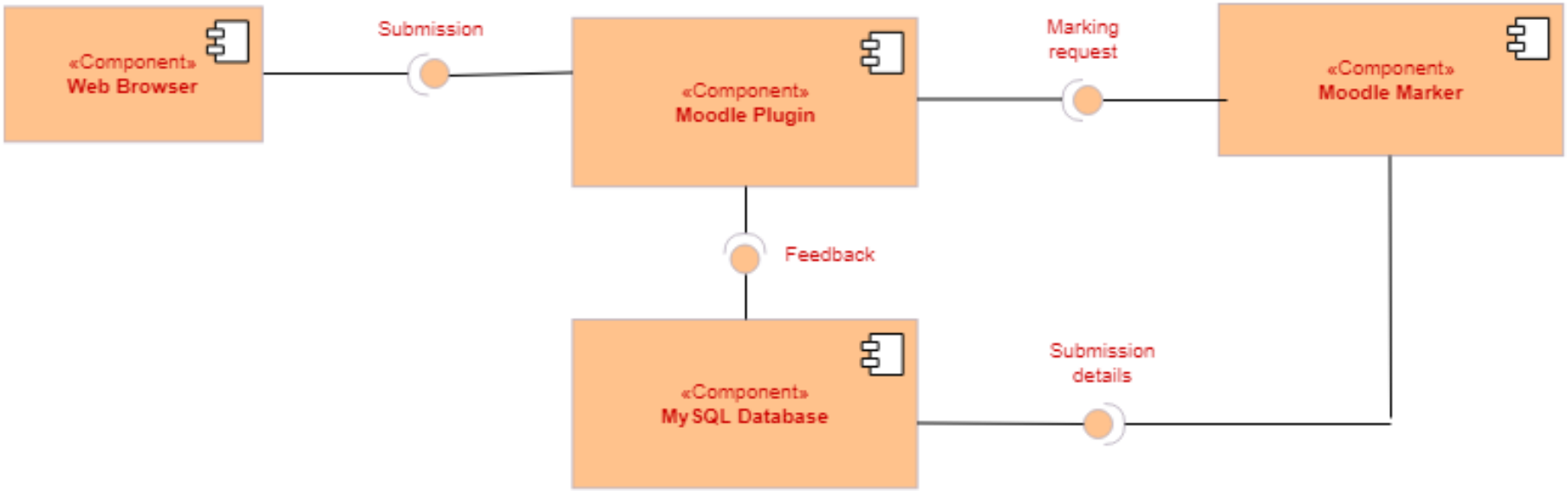


Development View – Component Diagrams

- Rectangles = components
  - Symbol in top right corner
  - Type and name at the top
- Squares = ports
  - Sometimes used to mark interfaces
- Ball = provided interface
- Socket = required interface



Development View – Component Diagrams

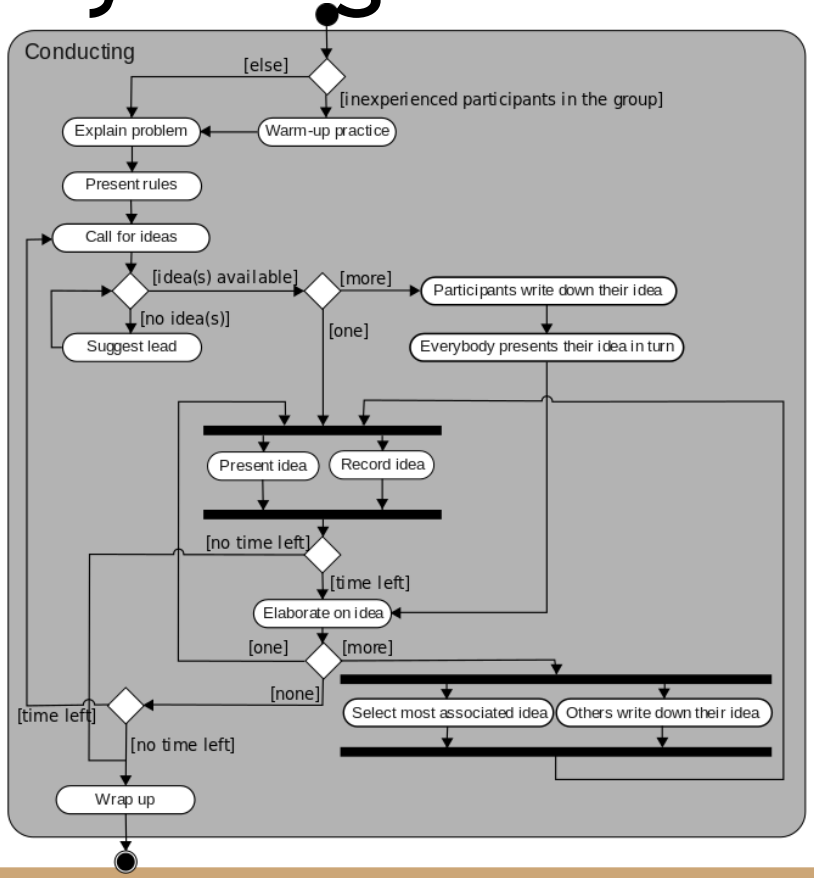


Process View

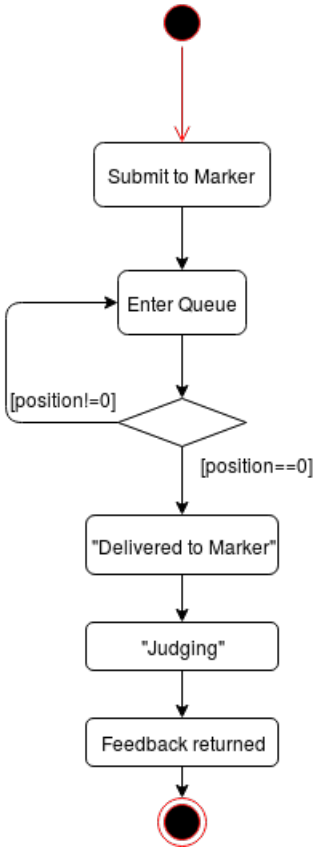
- The **process view** is concerned with the dynamic aspects of the system, explaining the system processes and how they communicate
- Focuses on the runtime behaviour of the system
- Represented in UML using **activity diagrams** and **sequence diagrams**

Process View – Activity Diagrams

- **Activity diagrams** show the workflow of a system
  - Blocks represent actions
  - Diamonds represent decisions
  - Bars represent forks/joins of concurrent activities
  - Circle represents start
  - Outlined circle represents end of process
- Similar to state diagrams, but show system activities rather than object states

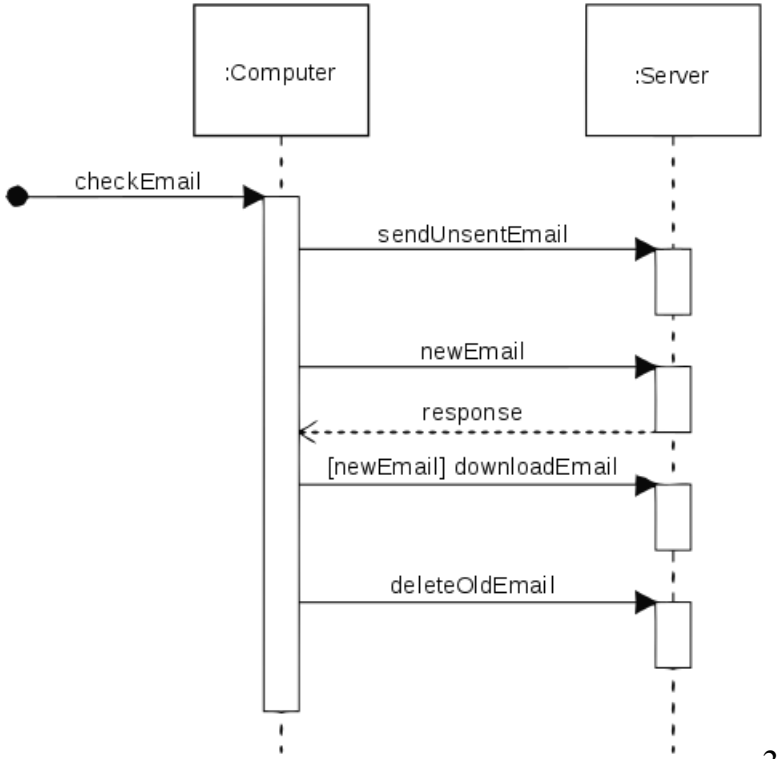


Process View – Activity Diagrams



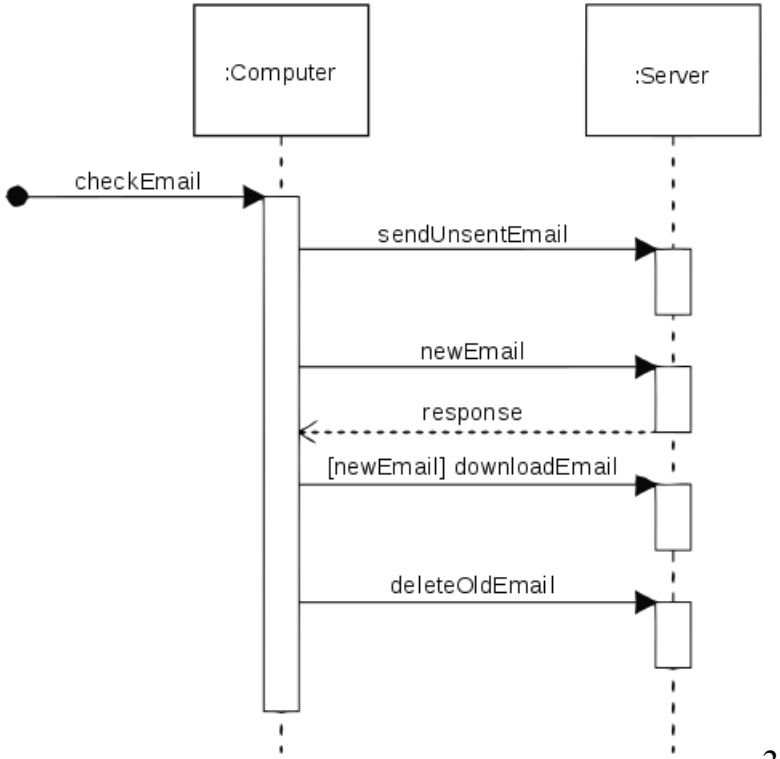
Process View – Sequence Diagrams

- **Sequence diagrams** show how components/objects in a system interact over time
- Typically associated with the use case realisation of the logical view



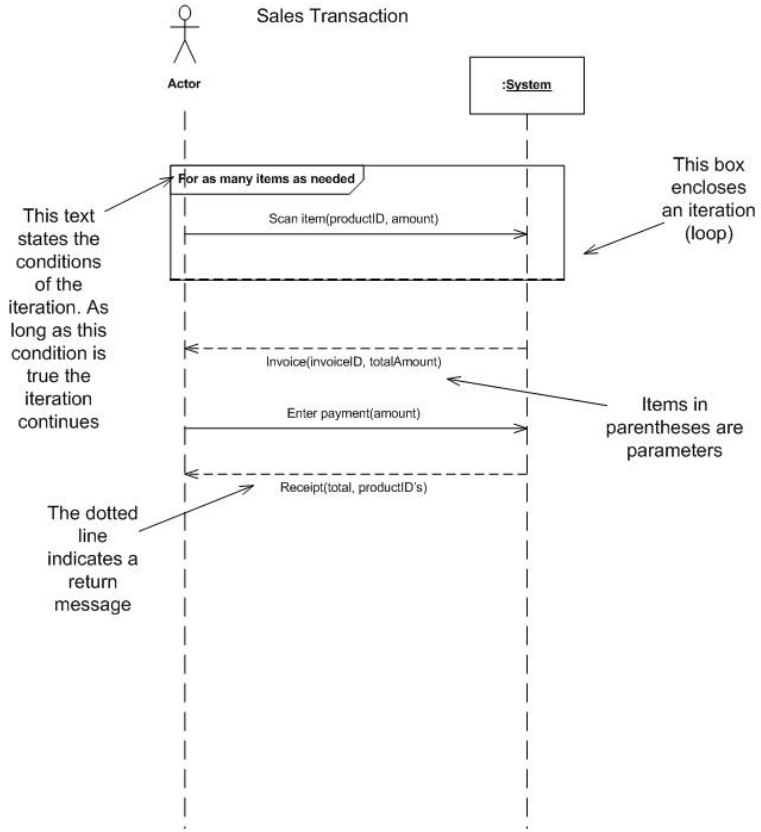
Process View – Sequence Diagrams

- Box on top represents an object/component
- Stick figure on top represents an external actor
- Dotted vertical line represents lifeline of system
- Blocks along lifeline represent processes
- Solid arrows represent communications (method calls)
- Dotted arrows represent returns

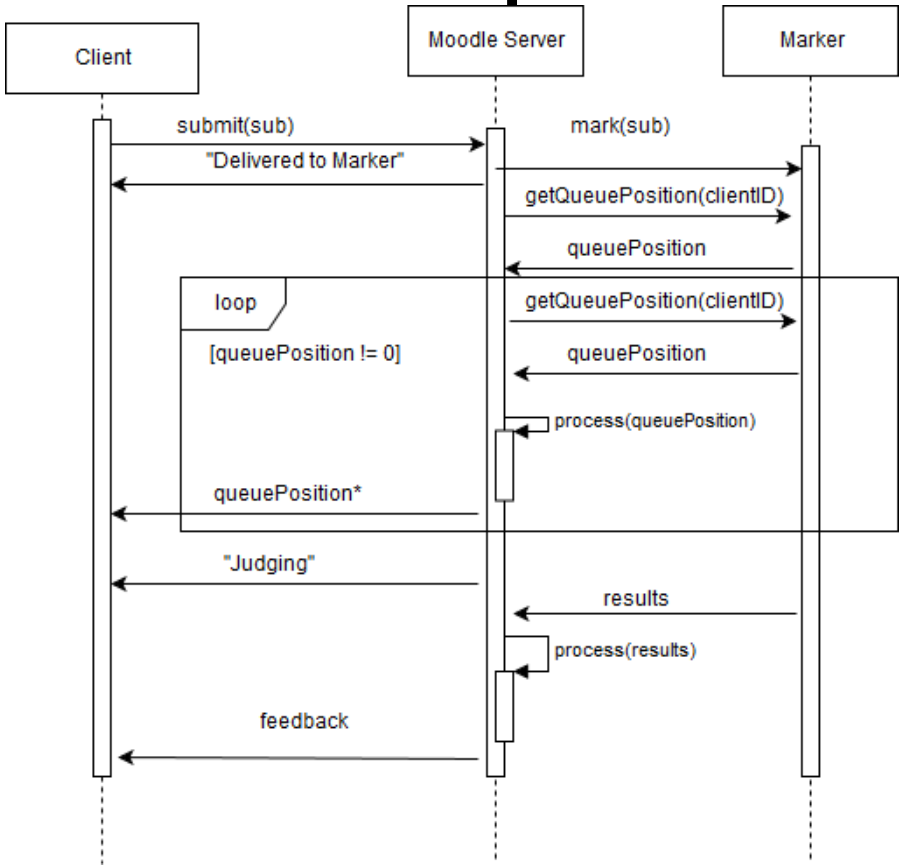


Process View – Sequence Diagrams

- Large box represents loops
  - Loop conditions in top left corner
- Text above/below lines indicates methods and parameters



Process View – Sequence Diagrams

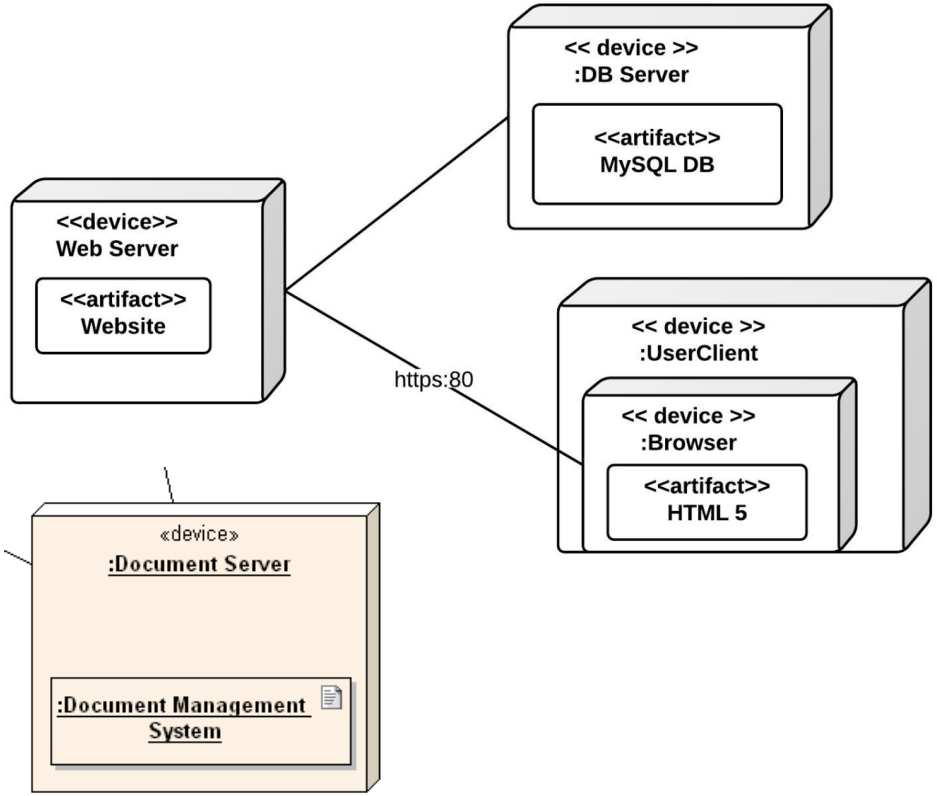


Physical View

- The **physical view** is concerned with the physical distribution of the system, as well as the physical connections between distributed parts
- Depicts the system from a system engineer's point of view
- Represented in UML using **deployment diagrams**

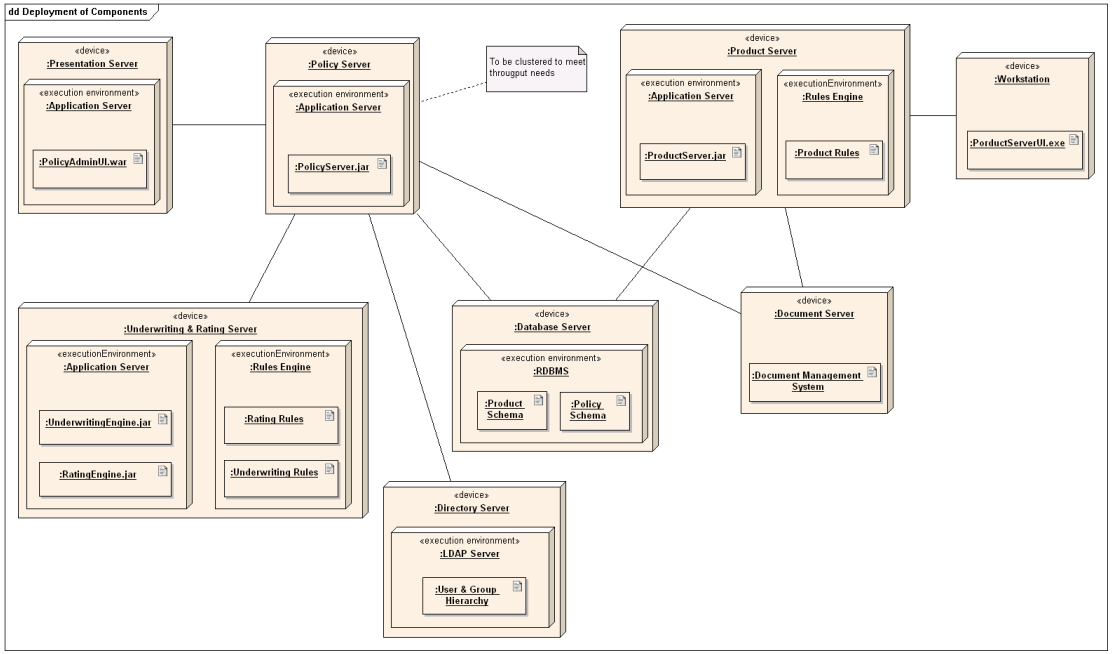
Physical View – Deployment Diagrams

- **Deployment diagrams** model the physical deployment of **artifacts** (software components) on **nodes** (hardware components)
  - Artifacts modeled by rectangles
    - Sometimes a symbol in top right
    - Databases sometimes modeled by cylinders
  - Nodes modeled by 3D boxes
  - Communication modeled by lines

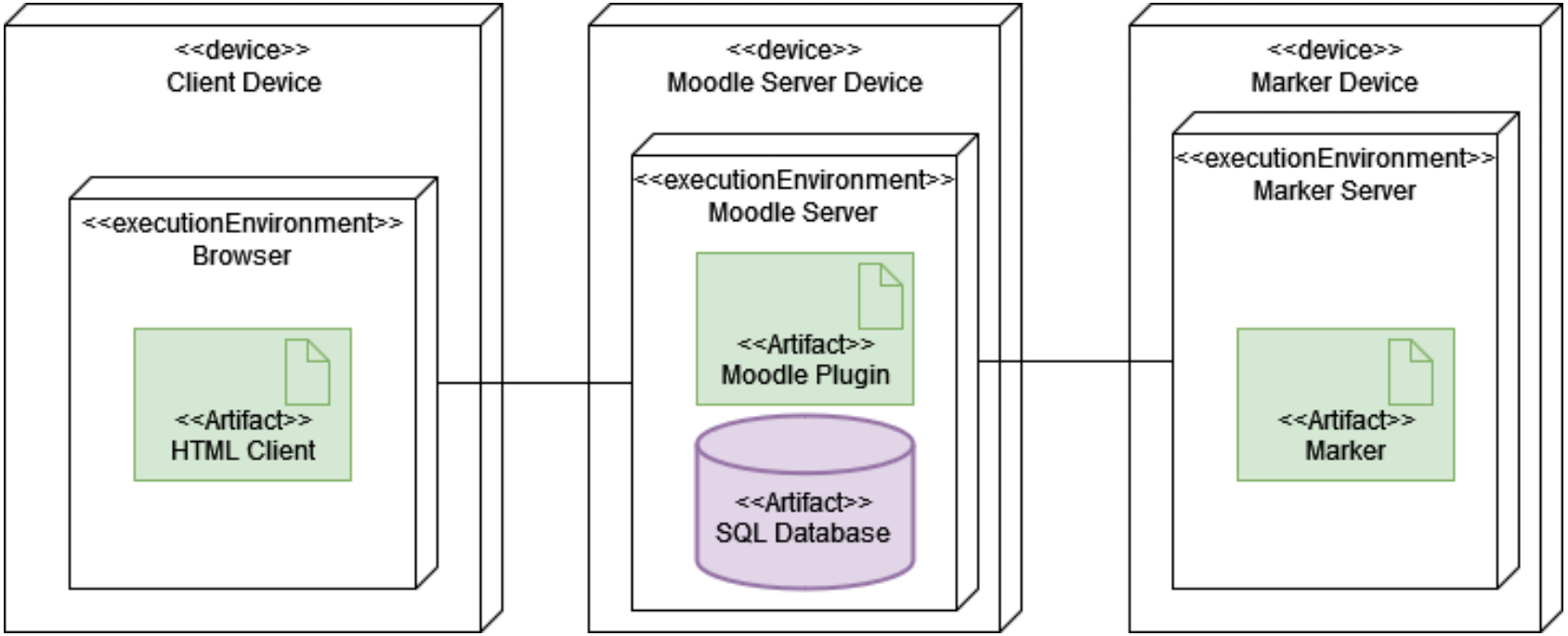


Physical View – Deployment Diagrams

- Two types of nodes:
  - **Device nodes** – physical devices that run software
    - e.g. computer, phone
  - **Execution Environment Node** (**EEN**) – software resource that runs on a device and can execute other software
    - e.g. server, browser
- Distributed nodes can be conceptually represented as one



Physical View – Deployment Diagrams

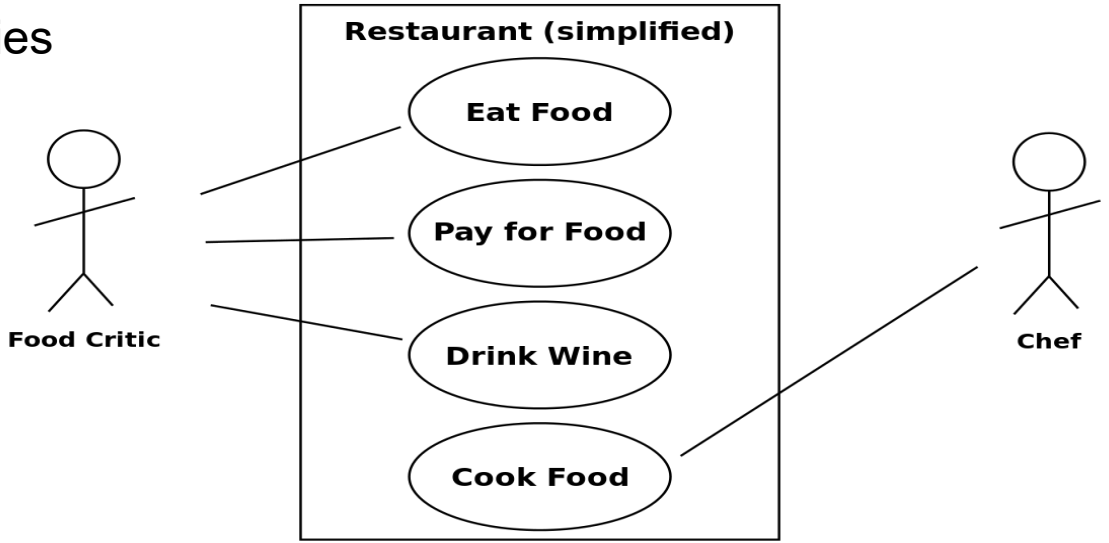


Scenarios

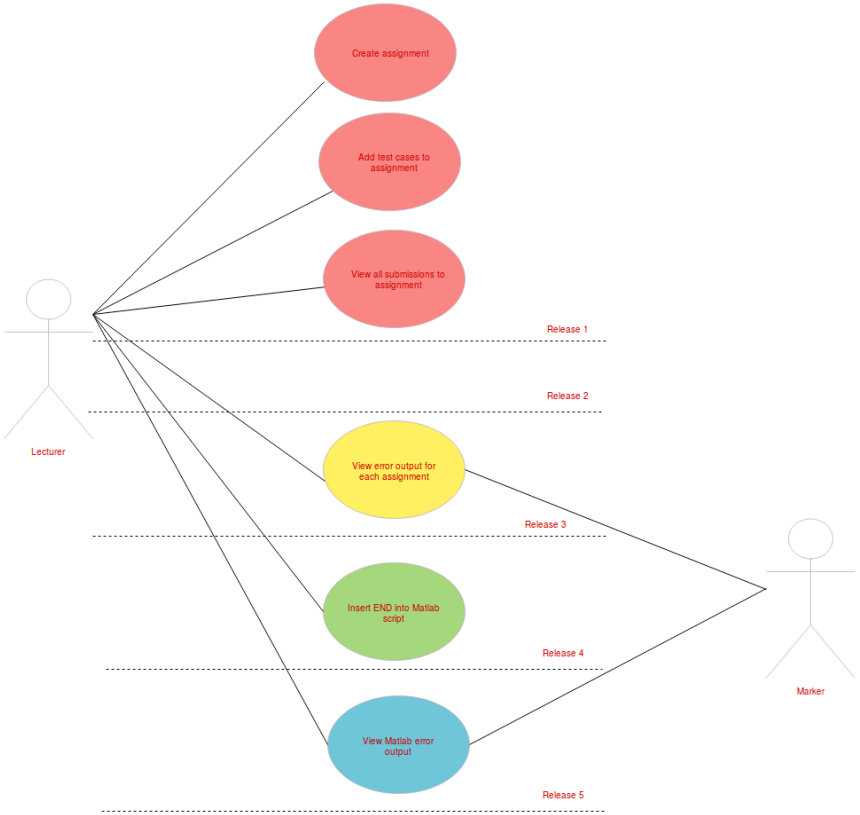
- **Scenarios**, sometimes known as the **use case view**, describe the system's architecture through **use cases** – the list of steps defining interactions between a role and a system to achieve some goal
  - See Lecture 3 – Requirements
- Represented in UML using **use case diagrams**

Scenarios – Use Case Diagrams

- Ovals = use cases
- Stick figures = actors
- Boxes = system boundaries
- Lines = associations

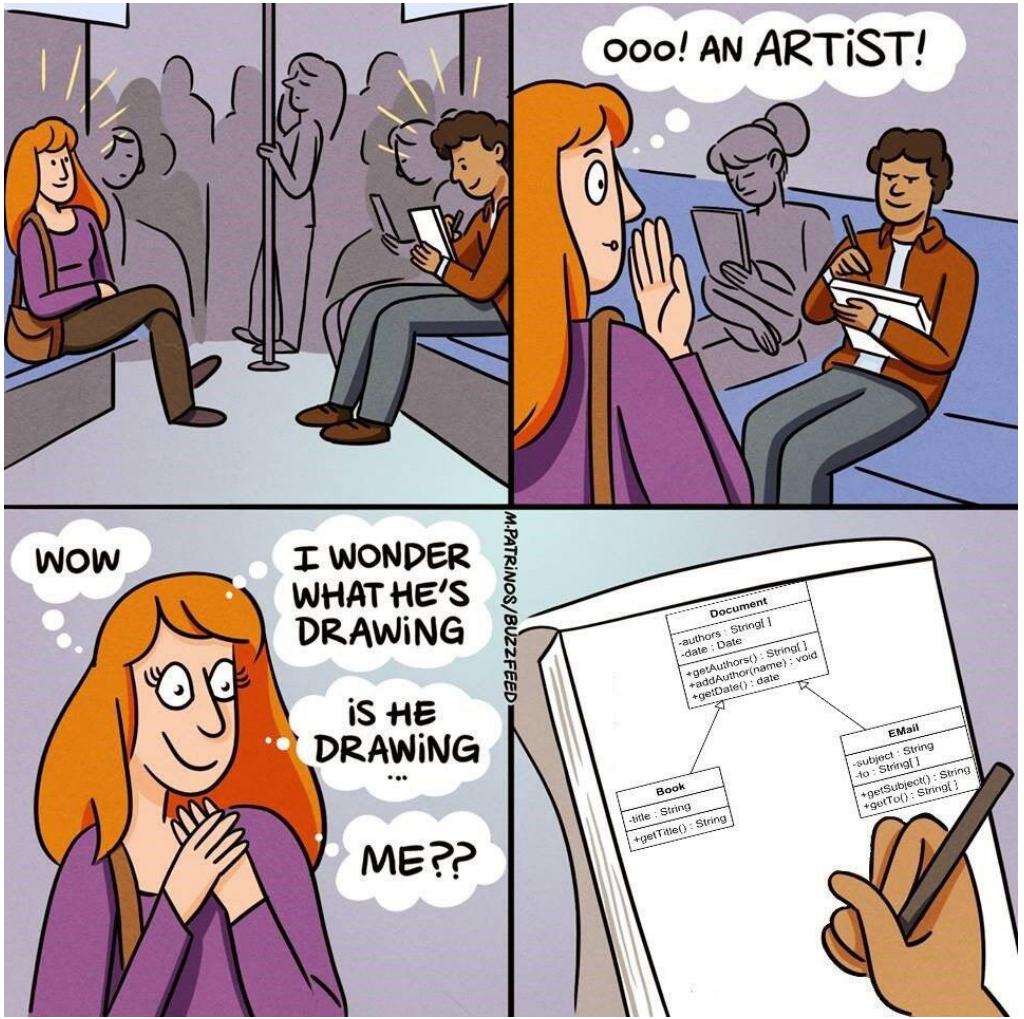


Scenarios – Use Case Diagrams



UML Tips

- Use drawing tools
  - draw.io
  - Umbrello
  - Lucidchart
  - ...and many many more
- Incorporate colour to make diagrams less confusing
- For more information, see:
  - GeeksforGeeks
  - Tutorialspoint



Architecture in your Projects

- Sprint 1:
  - High level architecture needs to be described
  - No need to describe viewpoints or use UML
- Sprint 2:
  - Every viewpoint needs to be described (not necessarily with UML, though UML is recommended)
- Sprint 3+:
  - Every viewpoint needs to be described with UML diagrams
- Refer to rubrics in course outline