REQUIREMENTS ANALYSIS

Dr. Tony D. Barber

Fall, 2024

TODAY'S CLASS OBJECTIVES AND OUTCOME

- Explain the course software development project
- Describe how software design relates to other life cycle activities, particularly with requirements engineering
- Explain what modeling is and how it can be used in software engineering and specifically in requirements analysis
- Read requirements, and develop requirements analysis models: domain, context, and use case models
- Use UML class, use case, and activity diagrams for representing requirements analysis models

OUTLINE

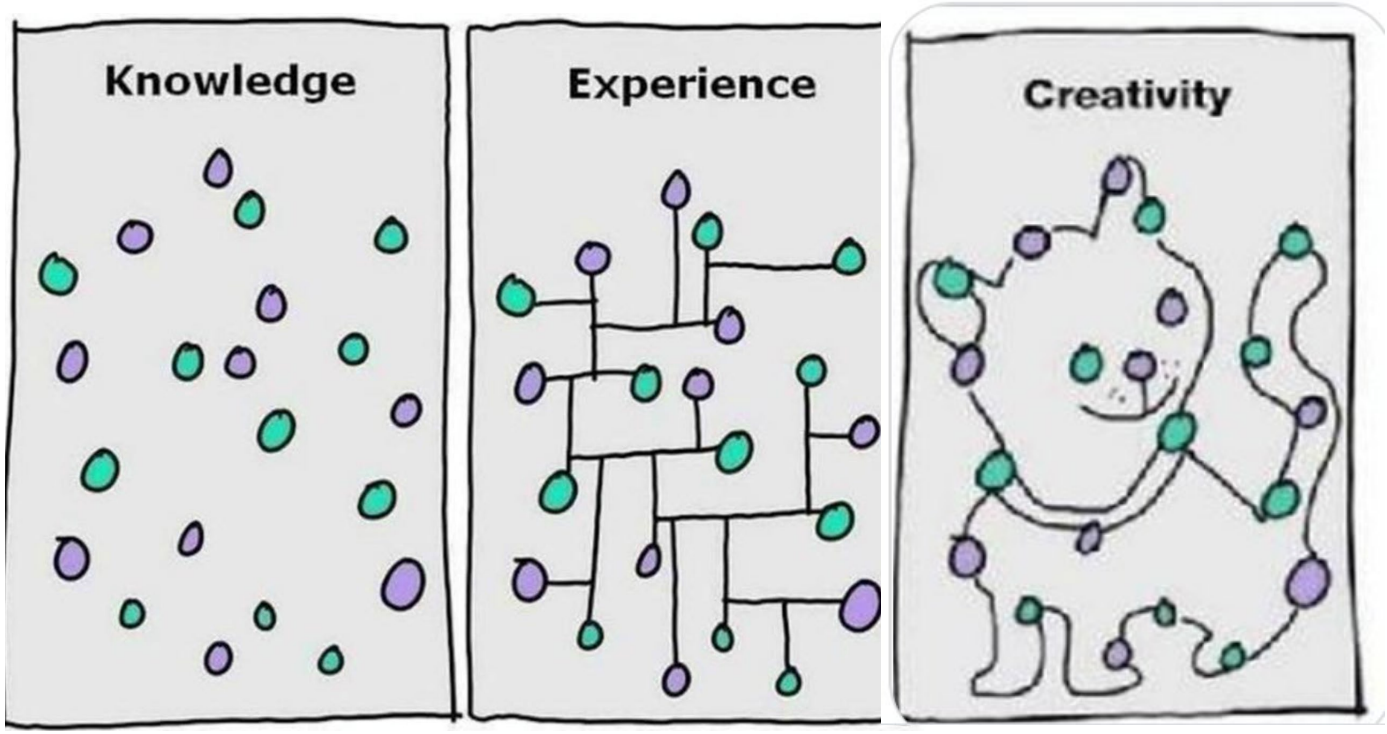
We are here

- Course project
- Lecture
  - Modeling in software development
  - Requirements engineering quick recap
  - Quick UML recap (class, use case, and activity diagrams)
  - Requirements analysis
    - Domain modeling
    - Context modeling
    - Use case modeling
    - Data modeling
- Assignments

SOFTWARE DEVELOPMENT PROJECT

- **Objectives/outcome**
  - Apply the knowledge progressively acquired in class throughout the semester, to the development of a software system, starting with requirements analysis, followed by architecture and detailed design, and implementation.
  - Learn/practice how to develop software in a team
    - This project will be conducted **in teams** (typically of 4-6 members)
    - Teams will review each other's artifacts
- **Type of software:** web based, Online Learning Management System
- **Domain**: of your choice – **BE CREATIVE!**

KNOWLEDGE, EXPERIENCE, AND CREATIVITY



SOFTWARE DEVELOPMENT PROJECT

- **Deliverables**:
  - Requirements analysis
  - User model & UI prototype
  - Software architecture document
  - Software architecture and detailed design document
  - Online system hosted on a platform of your choice
  - Source code
  - Test cases
  - Installation and Operation manual; User's manual
  - Project management documentation
- **Reviews:** Will be performed for each deliverable
- **Presentations**: For each phase
- **Project description** is posted in ELMS, together with due dates

All team deliverables are group assignments, and will be submitted by **one** student in the group, i.e., the lead of that activity/phase

SOFTWARE DEVELOPMENT PROJECT – 1ST ASSIGNMENT

- **Form project teams**
  - **Teams of 5 students**
  - Students can form teams, the instructor can help
  - Instructor reserve the rights to re-assign students to teams, for better balancing the number and knowledge/skills pool
    - Criteria: knowledge, experience, skills; availability to meet, proximity, …
- There are **Project Groups (Group 1 - 13)** in ELMS (under *People*)

SOFTWARE DEVELOPMENT PROJECT – 1ST ASSIGNMENT (CONTINUED)

- Determine an application domain and target stakeholders, and find a catchy name for your product
- For each team (only one student from each team):
  - **Post product name, as well as team member names and roles in the ELMS Discussion forum**
    - Recommended roles: project manager, user interaction lead, architecture lead, detailed design lead, implementation lead, testing lead.
      - Identify the knowledge and skills needed for each role
      - Assign students to roles
      - Note that a student can play more than one role
- Establish team's working and communication protocols
  - Meetings (on/offline, frequency, duration)
  - Select and set up communication and collaboration tools
- Develop the **Project Plan** – submit in ELMS by the due date (by the project manager)


MARYLAND APPLIED
GRADUATE ENGINEERING
THE A. JAMES CLARK SCHOOL of ENGINEERING
UNIVERSITY OF MARYLAND

OUTLINE

- Course project
- Lecture
  - Modeling in software development
  - Requirements engineering quick recap
  - Quick UML recap (class, use case, activity, diagrams)
  - Requirements analysis
    - Domain modeling
    - Context modeling
    - Use case modeling
    - Data modeling

We are here

ABSTRACTION AND MODELING IN DESIGN

- Abstraction is a fundamental design technique
  - *Abstraction* (from the Latin *abs*, meaning away from and *trahere* , meaning to draw) is **the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics**
    - "Essential" is relative to the purpose of abstraction
- Abstraction helps manage complexity
- Modeling (which relies on abstraction) is a basic design tool

MODELING IN GENERAL

- What is a model?
  - **Abstract** representation of a real entity, with a ***purpose***
- What is modeling?
  - "The process of developing abstract models of a system, with each model presenting a different view or perspective of that system" (Ian Sommerville)
- Why do modeling?
  - To cope with complexity (abstraction, refinement, decomposition, hierarchy)
  - For understanding, analyzing, designing, testing
- Where are models used?
  - Various domains and disciplines
  - Help us understand complex structure, relationships, and behavior

MODELING IN GENERAL

- A model is a **representation** (or "abstraction") of reality.
- **Representations** (abstractions) are always **simplifications**.
- A model focuses on representing those **aspects of reality** that the modeler believes to be the **most important** to capture in order to achieve the **model's purpose**.
  - E.g., if you are modeling a bridge in order to determine whether it can support a given load, it is generally does not model every atom (or proton, neutron, electron, and muon) that makes up the bridge.
- The term "**abstraction**" is often used as a synonym for "representation" or "model"
- The term "level of **abstraction**" is essentially synonymous with a model's "scope."
  - E.g., modeling a bridge as a collection of uniform structural elements, vs. specialized structural elements, vs. the atoms that make it up.

MODELING IN GENERAL

- While models can have many purposes, we will focus on identifying the principal engineering purpose of each.
  - The **principal engineering purpose** of **Modeling** is to **support engineering analysis.**
  - The **principal engineering purpose** of **engineering analysis** is to **solve a problem** (or to show that proposed solution to a problem works, or to determine which solution is best).
  - In supporting the solution of a problem, models are used to:
    - **Structure** a problem, concepts, approach to solving a problem, a potential solution, and/or associated information.
    - **Check (evaluate)** problem statements, concepts, solution approaches, solution structures, solutions and/or information for **clarity, consistency, and completeness**.
    - **Communicate** problem statements, concepts, solution approaches, solution structures, solutions and/or information **to others**.

TYPES OF MODELS

- **Text-Based Model:** A description of a system (and its context) in the form of text and/or tables.
  - Examples include documents that describe models of human social structure (e.g., government, religion, etc.) or human behavior (e.g., Maslow's hierarchy of needs, Myers-Briggs model of personality, etc.), biological taxonomies, concepts of operations, use case narratives, cost models, etc.
  - Aspects of these models are often also represented using schematic, mathematical, algorithmic, and/or data models.
- **Schematic Model:** A diagram or figure that represent a system, system elements, or processes and relationships between them.
  - These include organization charts, data flow diagrams, SysML diagrams, etc.
  - Students should be familiar with these from previous systems engineering, engineering, software development, and/or information technology courses.
  - Often used to express text-based models and as architectures for mathematical and/or physical models.
- **Mathematical Model:** A set of mathematical equations that represent some aspect of a system, set of systems, set of system elements, system environment, behaviors, processes, etc., and/or the relationships between these.
  - Examples include Newton's Laws expressed as equations (and their application to describing the behavior of systems), statistical distributions (and their application to describing systems), differential and difference equations, etc.

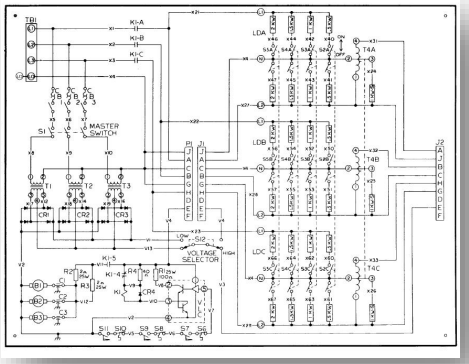
TYPES OF MODELS

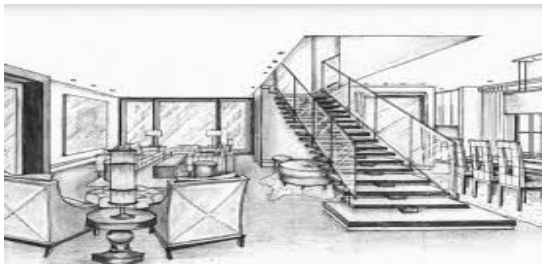
- **Algorithmic/Rule/Behavior Model:** A logic flow representation of the behavior of a system, system elements, system environment, processes, and/or the relationships between them.
  - These may be expressed as text-based models (e.g., pseudocode, computer code, rules tables, or structured English) or schematic models (e.g., flowcharts, activity diagrams, state machine diagrams).
  - All **computer programs/simulations** are instantiations of **algorithm models**.
  - Algorithm models often guide the application of mathematical models and are used to implement mathematical models in computer code.
- **Data Model**: capture information about elements of a domain, their behavior and relationships as data structures (instantiated in a database).
  - As an example, as one develops an integrated system architecture using a tool such as SysML, that architecture stored in a database as an example of a data model.
  - One often develops schematic models for data models (e.g., a class/object diagram ,or block definition diagram, or ER diagram).
  - These kinds of models serve to capture “knowledge” about a system or process in many artificial intelligence systems and “modeling languages.”
- **Meta-model/Ontology:** Data models that define the types of data entities that make up a data model.

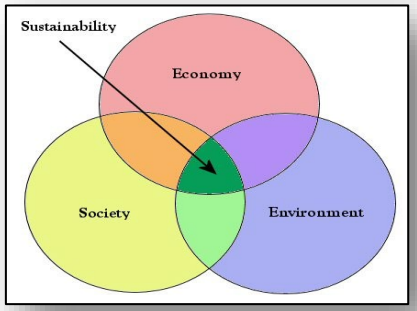
TYPES OF MODELS

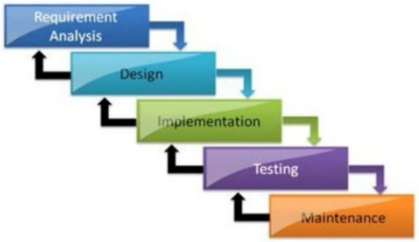
- **Physical Model:** Physical representations of a system or system element.
  - They generally model some of the most important physical aspects of a system.
  - These include scale models and full scale mockups (e.g., crash dummies), prototypes, etc.
  - Students may be familiar with these from experiences building model ships, planes, bridges, and/or “popsicle stick” houses.
- **Table Top Exercise/Model/Game:** Representations of situations in which one examines how people react and interact in response to given scenarios.

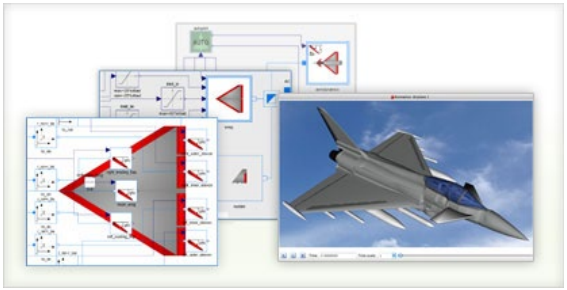
MODELING IN GENERAL – EXAMPLES

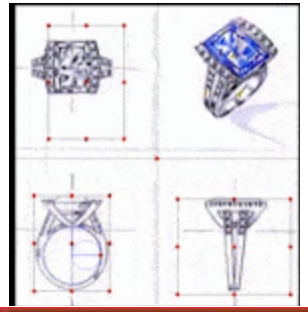


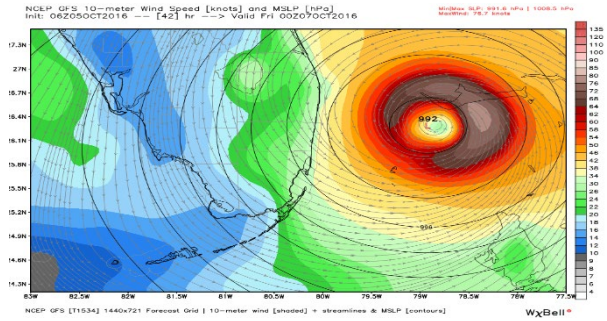






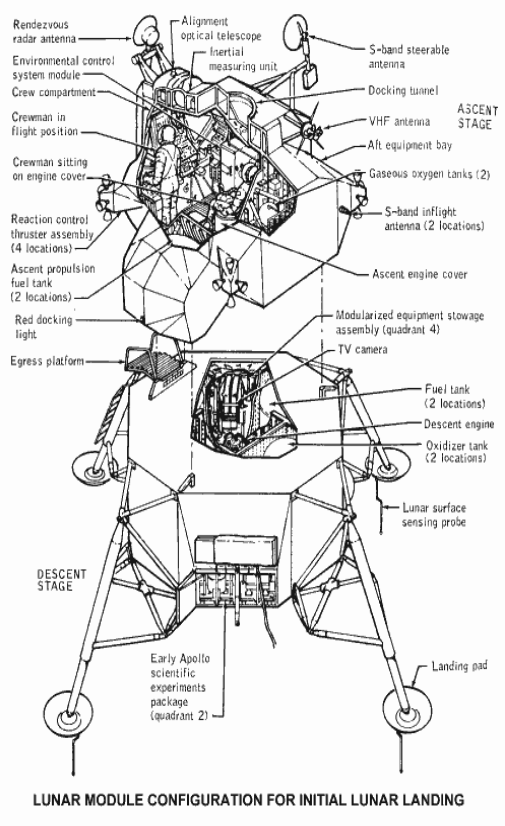




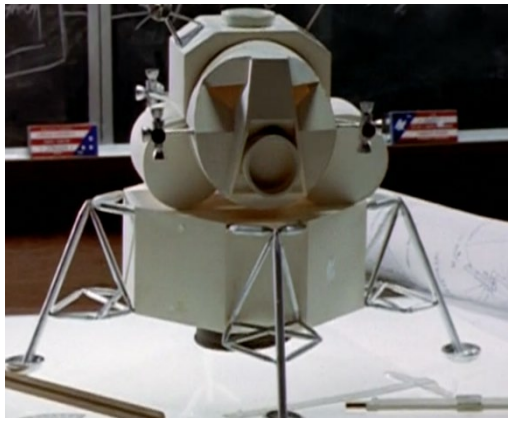


MODELING IN GENERAL – EXAMPLES (CONTINUED)

NASA Apollo Lunar Module (LEM)



LUNAR MODULE CONFIGURATION FOR INITIAL LUNAR LANDING



"From the Earth to the Moon" movie series
Also see http://www.andrewchaikin.com/nasa/

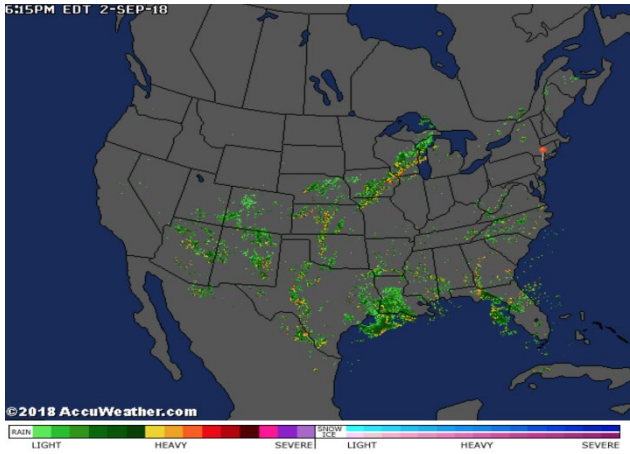
MODELING IN GENERAL – EXAMPLES (CONTINUED)

USA maps

Political

Physical/geographical

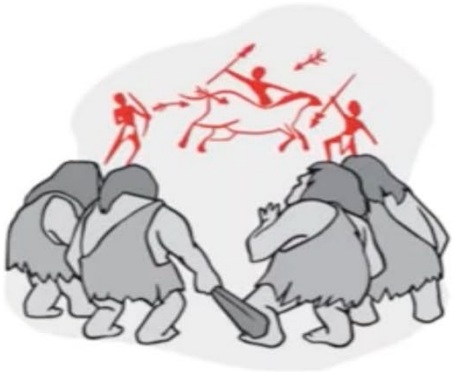
Weather

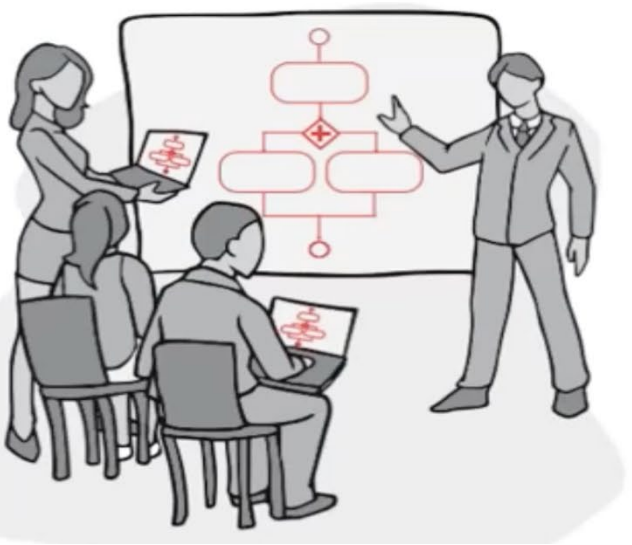


MODELING IN GENERAL – EXAMPLES (CONTINUED)

Map and model of Washington DC

EVOLUTION OF INFORMATION EXCHANGE ☺





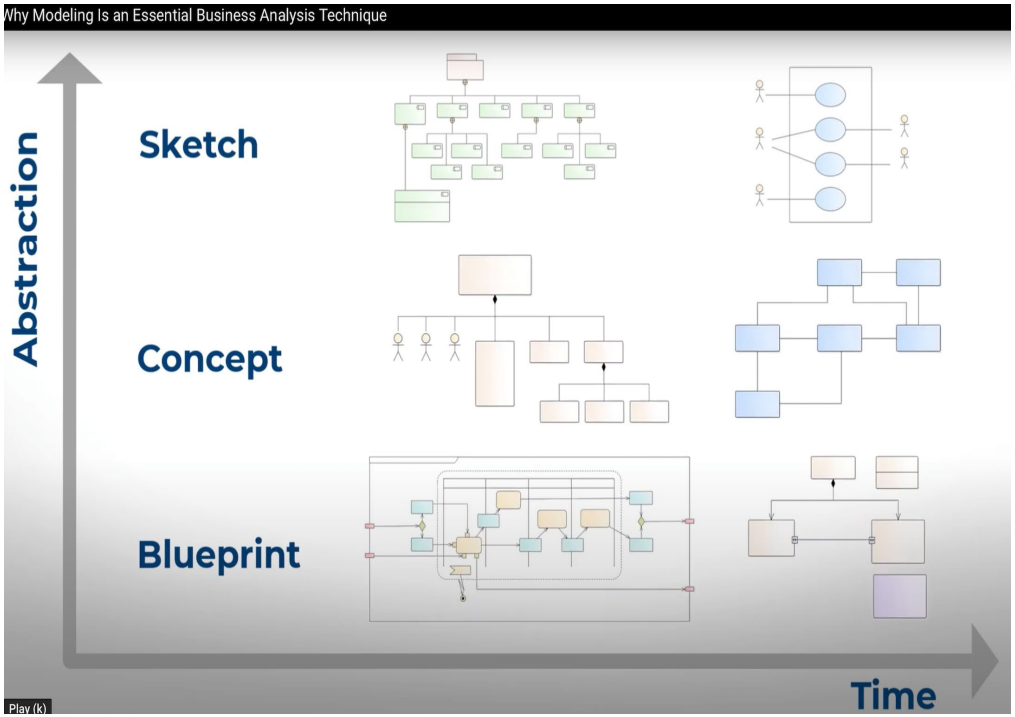
From: [Moving from Documents to Models]

MODELING IN SOFTWARE ENGINEERING

- **Why** do we model?
  - Understand, clarify, communicate, evaluate
- **Where** do we model?
  - Requirements, design, testing, project management
- **What** do we model?
  - Various aspects of SW artifacts and process
- **How** do we model?
  - Using various modeling techniques, notations, tools

MODELING - ESSENTIAL BUSINESS ANALYSIS TECHNIQUE

Watch:
https://www.youtube.com/watch?v=quFp2AMx5fU&feature=youtu.be



Modeling is used for:

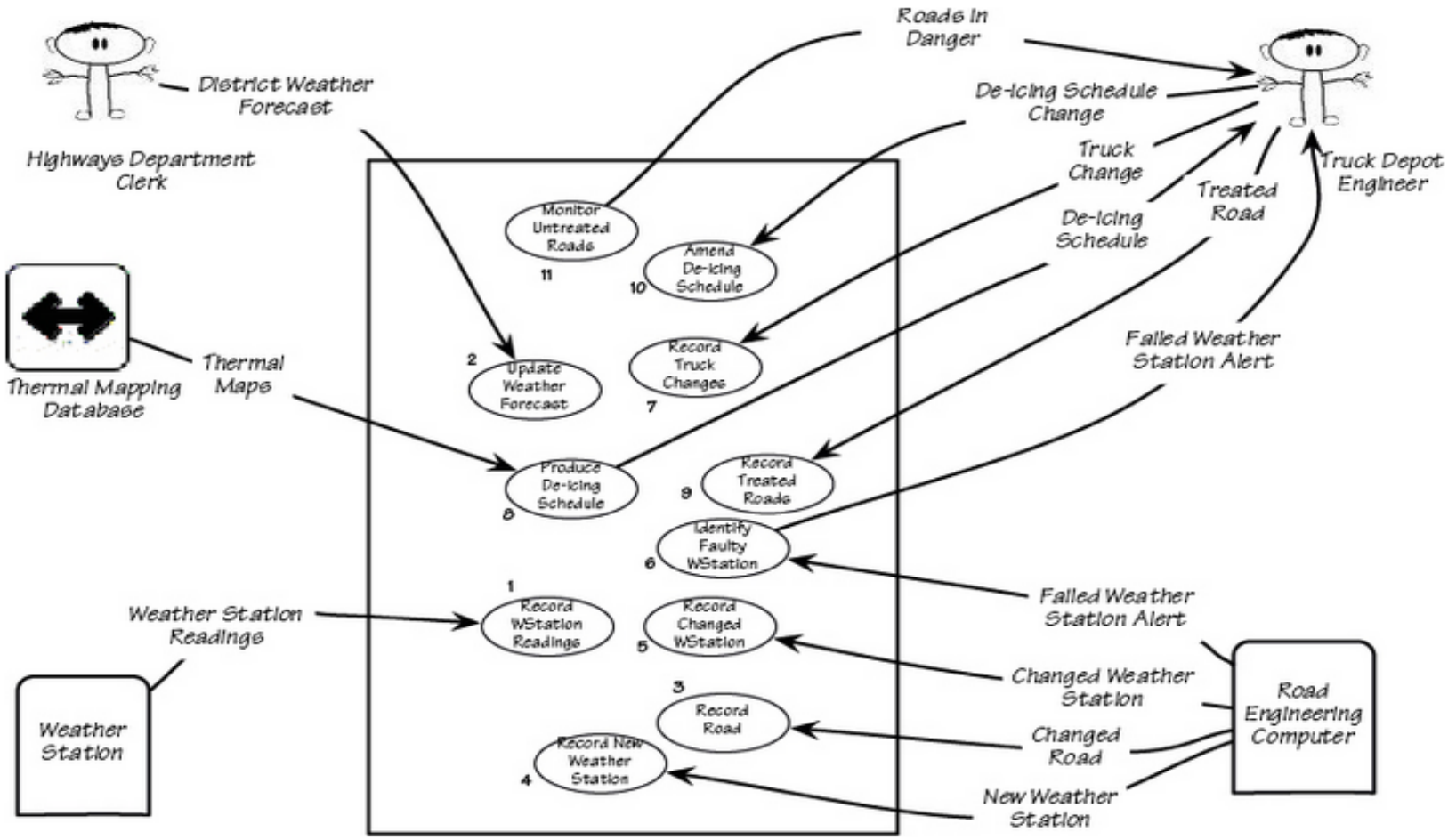
- System (SW) analysis
- Understanding the problem domain
- Designing and exploring solution approaches

MODELING IN SOFTWARE ENGINEERING (CONTINUED)

- How are model used?
  - For facilitating discussion about an existing or proposed system
    - These models may be **incomplete** (so long as they cover the key points of the discussion) and may use the modeling notation **informally**. This is how models are normally used in so-called 'agile modeling'
  - For documenting an existing system
    - These models **do not have to be complete** (could be for only parts of a system. However, these models have to be **correct** —they should use the notation correctly and be an **accurate** description of the system.
  - As a detailed system description that can be used to generate a system implementation
    - These models are used as part of a model-based development process, so they must be both **complete and correct/accurate**. Rigorous notation must be used.

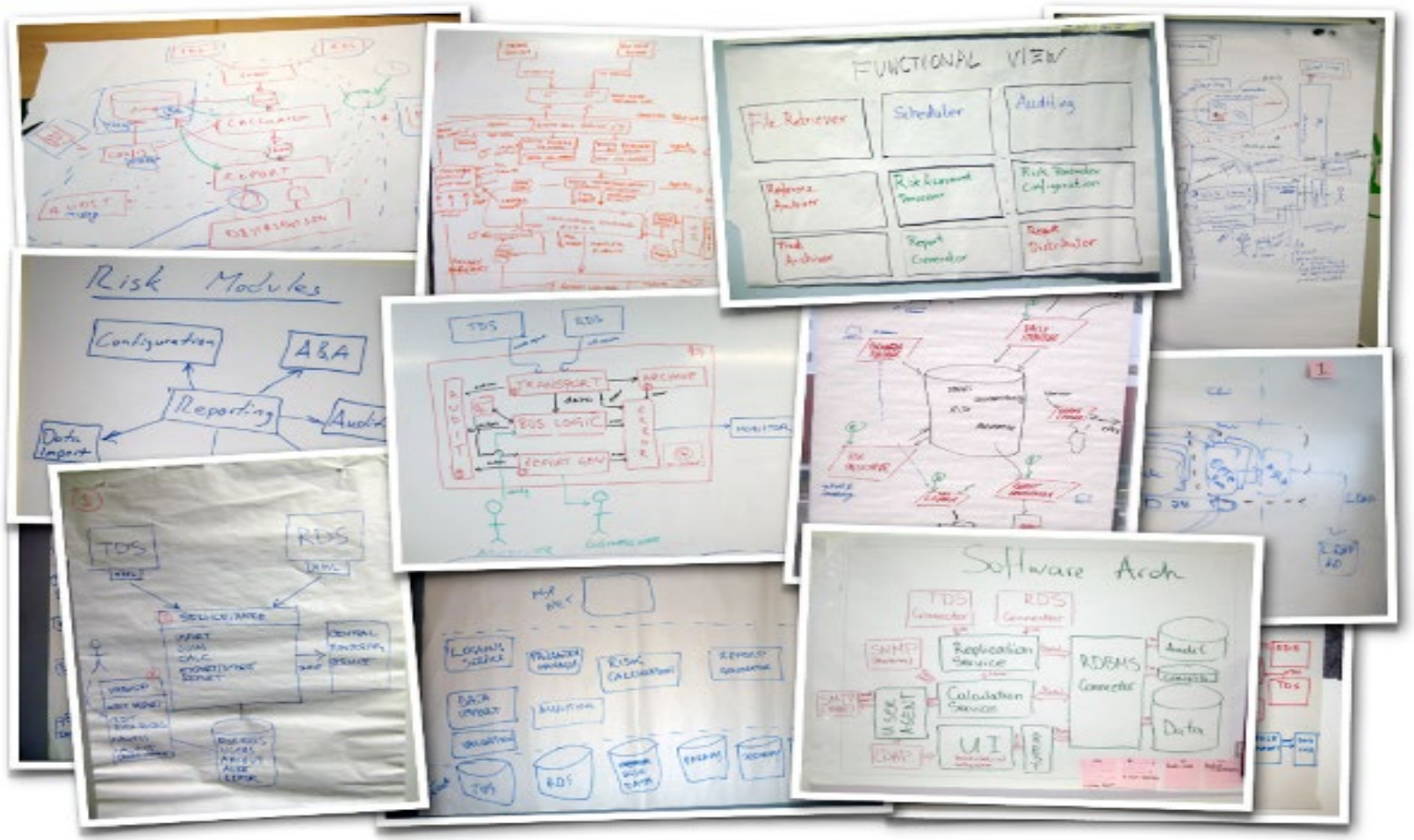
Completeness, Accuracy, Rigor

SOFTWARE MODELS – EXAMPLE USE CASE DIAGRAM

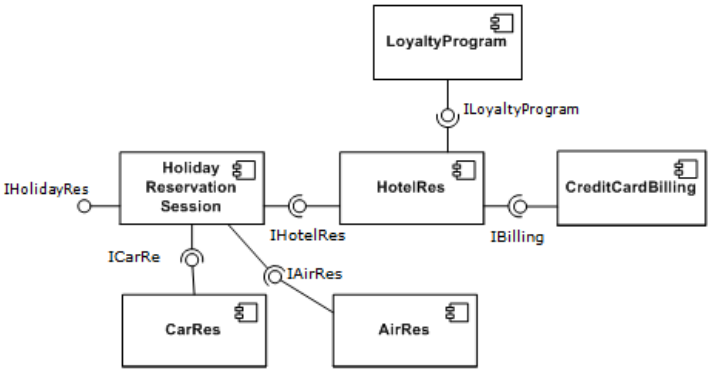


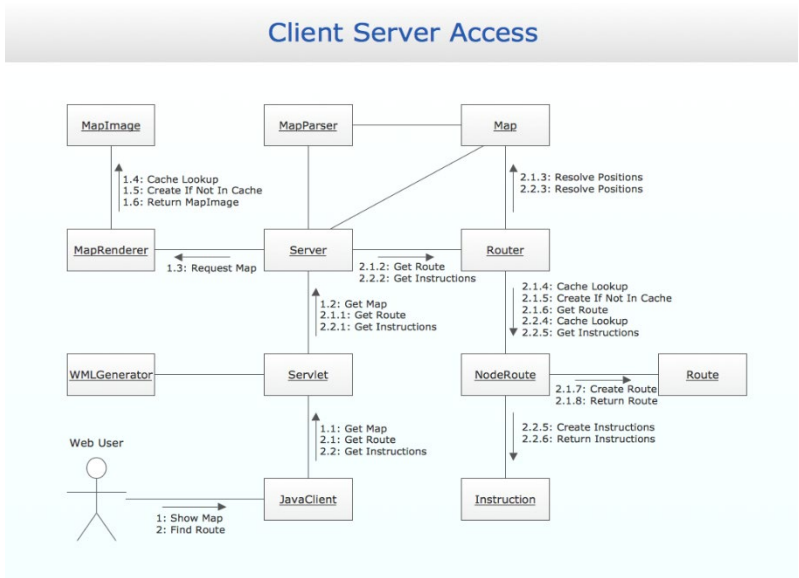
From: S. Robertson & J. Robertson: “Mastering the Requirements Process”

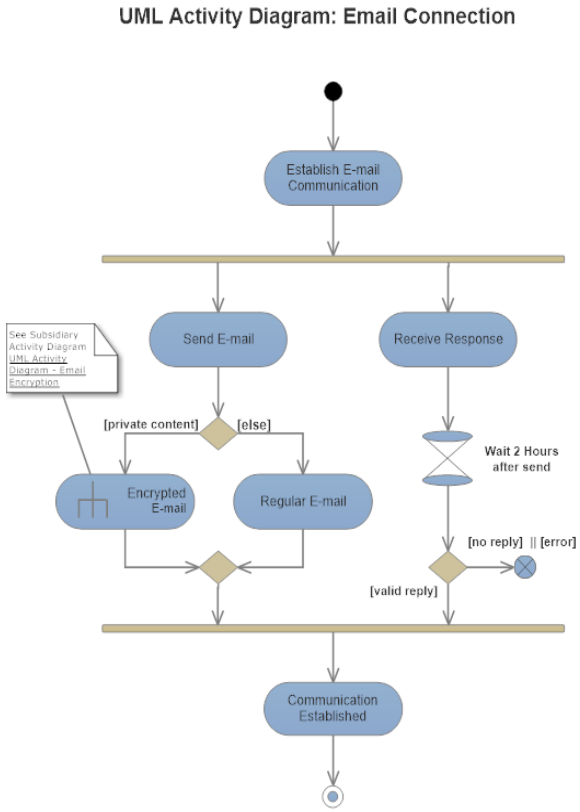
SOFTWARE MODELS – EXAMPLE DESIGN



SOFTWARE MODELS – EXAMPLE DESIGN (CONTINUED)







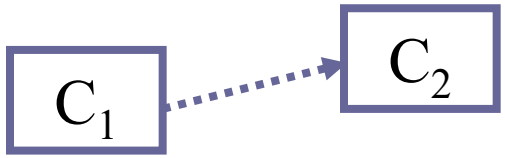
ELEMENTS OF DESIGN AS ACTIVITY

- *Design Process*—A collection of related tasks that transforms a set of inputs into a set of outputs
- *Design principles* - Characteristics of design that make them better
- *Design Heuristics*—Rules providing guidance, but no guarantee, for achieving some end
- *Design Notations*—A symbolic representational system
  - Modeling language

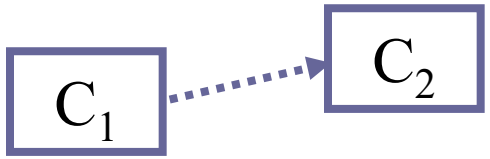
ELEMENTS OF A MODELING LANGUAGE

- **Symbols**: Standard set of symbols

- **Syntax**: Acceptable ways of combining symbols



- **Semantics**: Meaning given to language expressions



C_1 *is associated with* C_2

The modeling language we will use in this class in UML

OUTLINE

- Course project
- Lecture
  - Modeling in software development
  - Requirements engineering quick recap
  - Quick UML recap (class, use case, activity, diagrams)
  - Requirements analysis
    - Domain modeling
    - Context modeling
    - Use case modeling
    - Data modeling

We are here

REQUIREMENTS ENGINEERING – QUICK RECAP

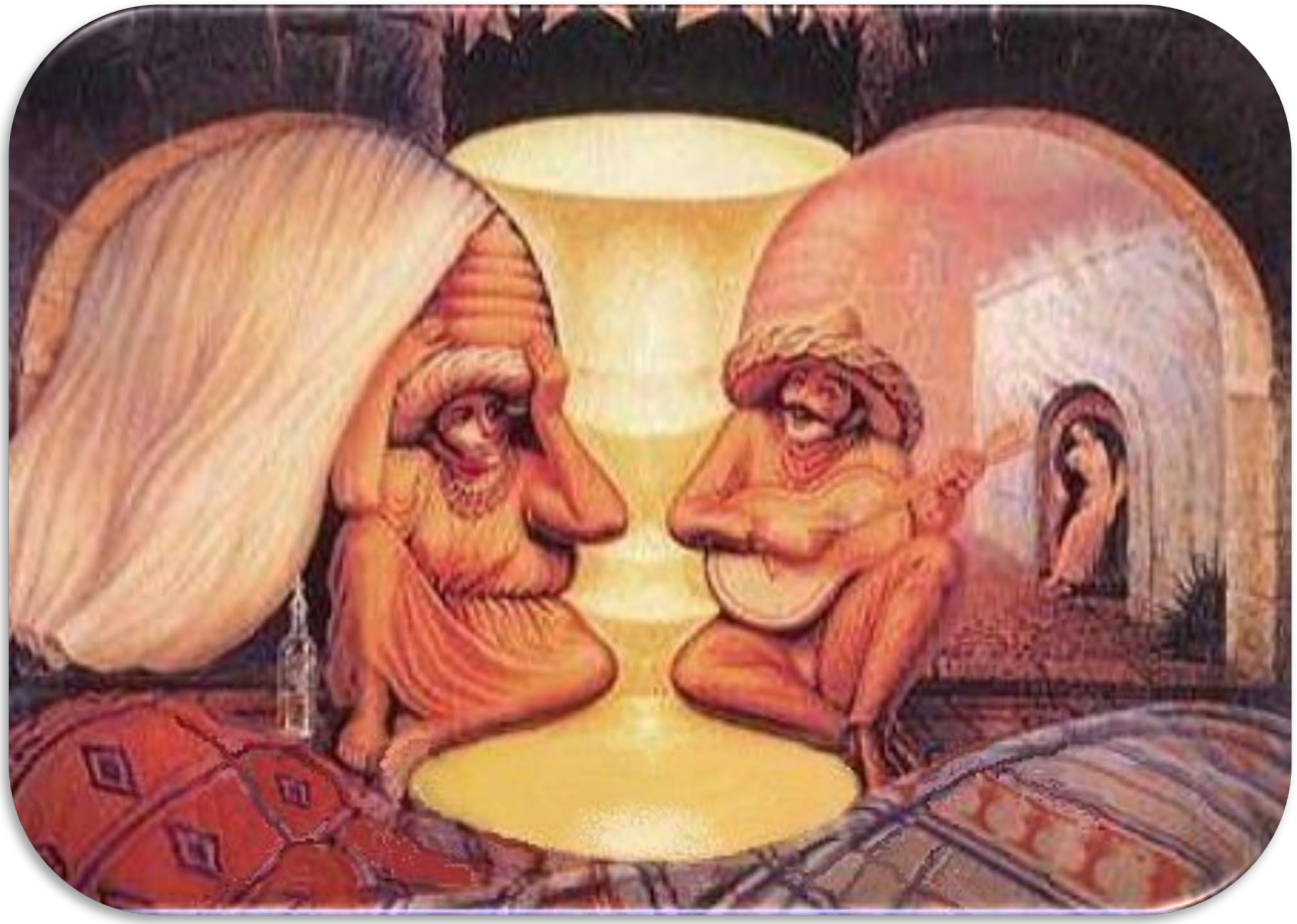
- **Requirements engineering** is creating, modifying, and managing requirements, over a product's lifetime
  - **Requirements development** - the portion of requirements engineering concerned with initially establishing requirements (aka product design)
  - **Requirements management** - the portion of requirements engineering concerned with controlling and propagating requirements changes

REQUIREMENTS ENGINEERING – QUICK RECAP

What do you see?

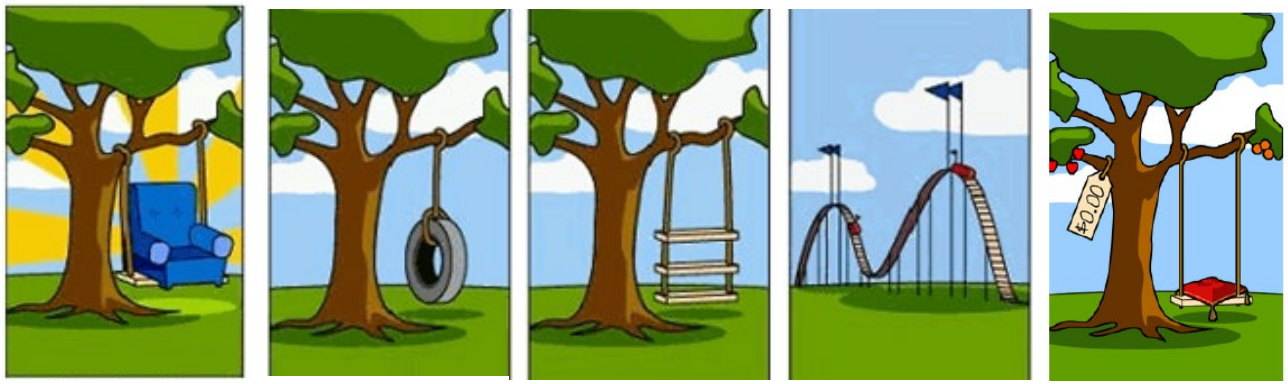
REQUIREMENTS ENGINEERING – QUICK RECAP

What do you see?



REQUIREMENTS ENGINEERING – QUICK RECAP

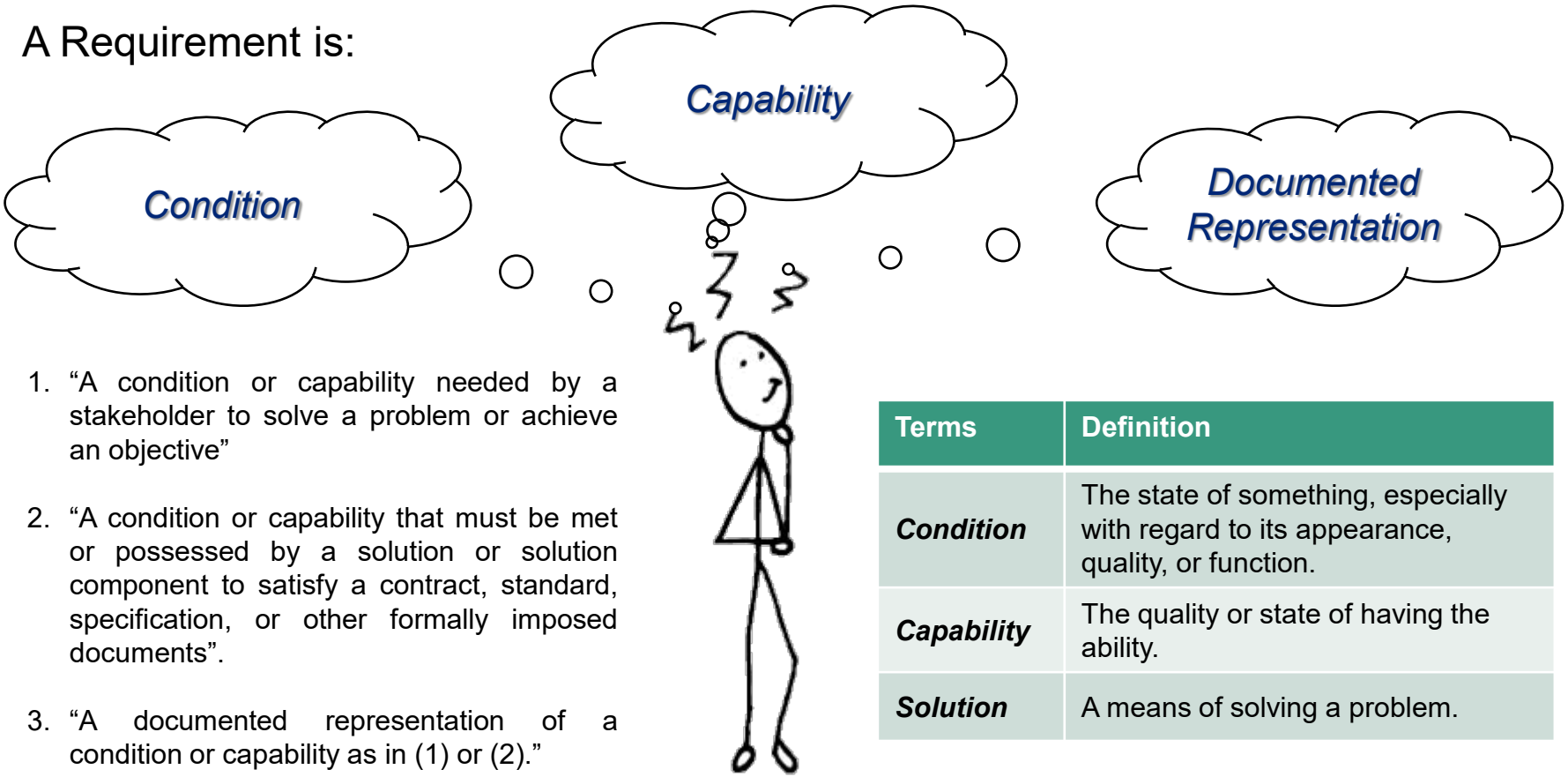
Requirements Management is difficult because it deals with perspective.



Why are there so many different interpretations?

REQUIREMENTS ENGINEERING – QUICK RECAP

A Requirement is:



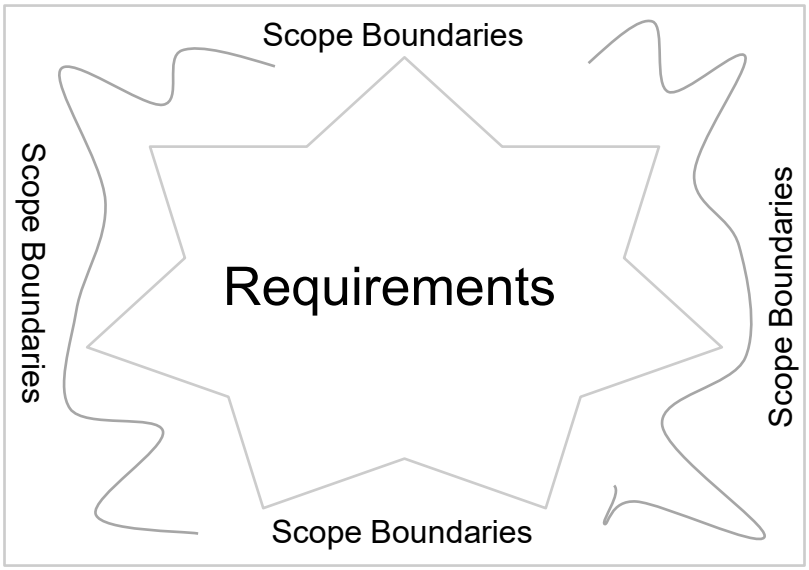
1. "A condition or capability needed by a stakeholder to solve a problem or achieve an objective"
2. "A condition or capability that must be met or possessed by a solution or solution component to satisfy a contract, standard, specification, or other formally imposed documents".
3. "A documented representation of a condition or capability as in (1) or (2)."

Terms	Definition
Condition	The state of something, especially with regard to its appearance, quality, or function.
Capability	The quality or state of having the ability.
Solution	A means of solving a problem.

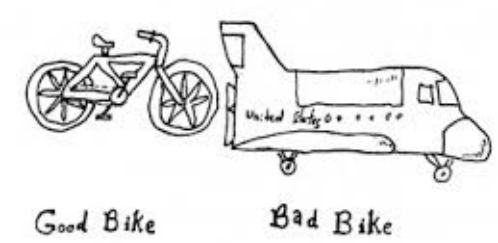
Requirements Management is the process of documenting, analyzing, tracing, prioritizing and agreeing on requirements.

REQUIREMENTS ENGINEERING – QUICK RECAP

Requirements Management is often confused with Scope Management.



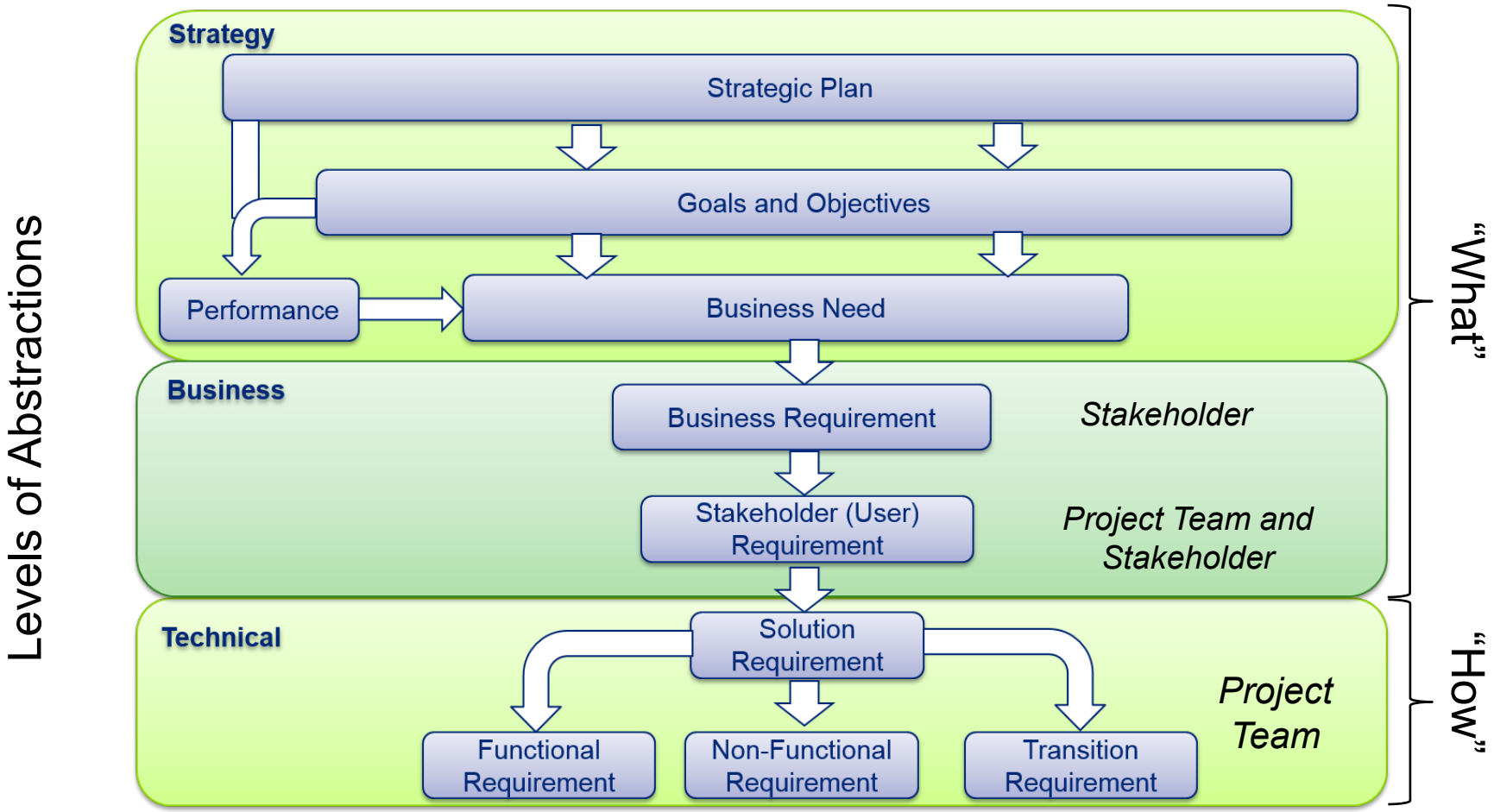
Terms	Definition
Scope	The extent or breadth of a project or product.
Gold plating	Giving the user extras (i.e., extra functionality, higher quality components, and extra scope or better performance) ☹
Scope Creep	Uncontrolled changes or continuous growth in a project's scope ☹



Scope Management is the process of defining what work is required and ensuring only ***that*** work is done.

Note: The Project Team will manage scope to prevent Scope Creep and/or Gold plating.

REQUIREMENTS ENGINEERING – QUICK RECAP

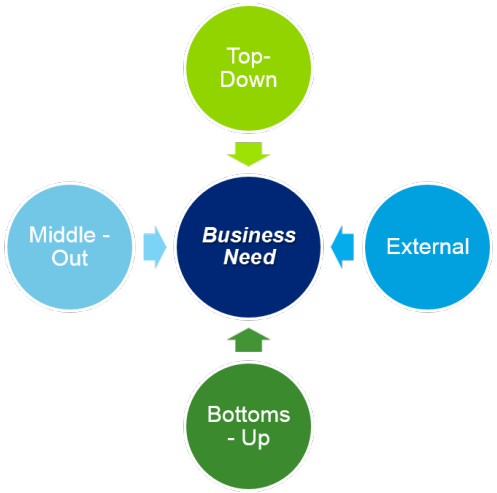


Your role is to enable the capabilities needed to support the business; do not restate or change the intent of the business need.

REQUIREMENTS ENGINEERING – QUICK RECAP

Driven by the Goals and Objectives

Optimizations based on management for efficient operations



Changes required by regulators, business partners, and/or competitors

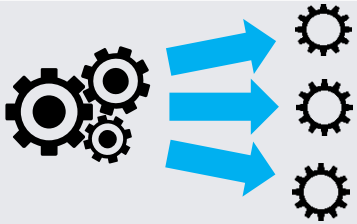
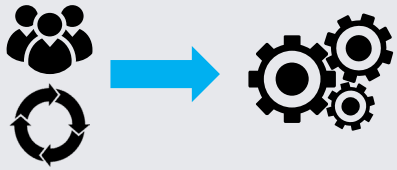
Process or Solution Malfunctions identified by the User

Terminology
Business Requirements – Describes business activities that must be performed to meet organizational objectives / address business needs.
Stakeholder (User) Requirements – States needs of stakeholders, specifically how stakeholders or classes of stakeholders will interact with a solution.

A Business Need is a high-level business requirement that is a statement of a business objective, or an impact the solution should have on its environment.

REQUIREMENTS ENGINEERING – QUICK RECAP

Terminology
Solution (System) Requirements – Characteristics of a solution to meet business and stakeholder requirements made up of Functional and Non-Functional Requirements.
Functional Requirements – Features and functions the solution should possess and provide. (e.g., processes, data, interactions, etc. the solution will have).
Non-functional Requirements – Qualities the system must have, such as performance, security, quality, availability, etc.
Transition Requirements - Activities executed to support the migration from current state to future state of business / systems operations. These includes training, role realignments, process changes, technology sun-sets, scheduling, and planning activities, to name a few.

Requirements Decomposition	Versus	Requirements Derivation
Requirements Decomposition is the technique of breaking down derived requirement into its component parts.		Requirements Derivation is the technique of translating business and technical requirements to its lowest level of abstraction to support implementation.
		
• *Best leveraged to compartmentalize features for reuse.*		• *Best leveraged to translate business requirements into technical taxonomy.*

REQUIREMENTS ENGINEERING – QUICK RECAP

Criterion of a "Good" Requirement	Checklist
Necessary	❑ Can the system meet the need without it and does it reflect a true need, not a perceived constraint?
Verifiable	❑ Can someone ensure that the requirement is met within the solution?
Feasible	❑ Can the requirement be met within the solution under development?
Unambiguous	❑ Can the requirement be interpreted in more than one way?
Complete	❑ Are all conditions stated, under which the requirement applies?
Consistent	❑ Are there common format and structure (System requirements begin with, "The system shall…") and not in conflict with other requirements?
Traceable	❑ Is the origin of the requirement known?
Concise	❑ Is the requirement stated simply and clearly? [No Compound Requirements]
Design Independent	❑ Does not contain colors or screen locations or the fact something is a button or a link?
Atomic	❑ Does the requirement address one and only one thing and does not contain words like and / or?
Unique	❑ Is not redundant or a restatement of a similar requirement?

Why Shall Statements?

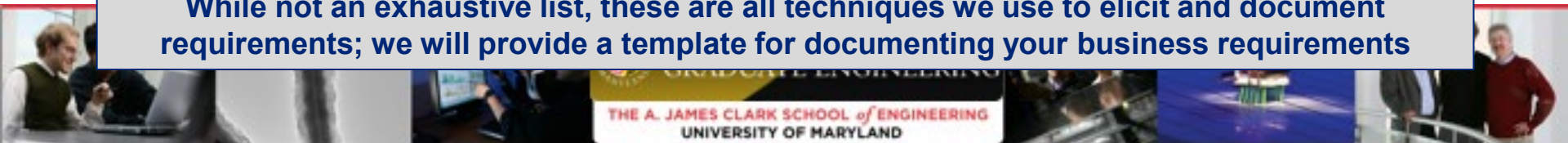
- **Shall** is used to indicate a requirement that is contractually binding, meaning it must be implemented, and its implementation verified.
- **Should** is used to indicate a goal which must be addressed by the design but is not formally verified.
- **Must and might** is synonymous with shall. Shall has been adopted as best practice for what is required.

REQUIREMENTS ENGINEERING – QUICK RECAP

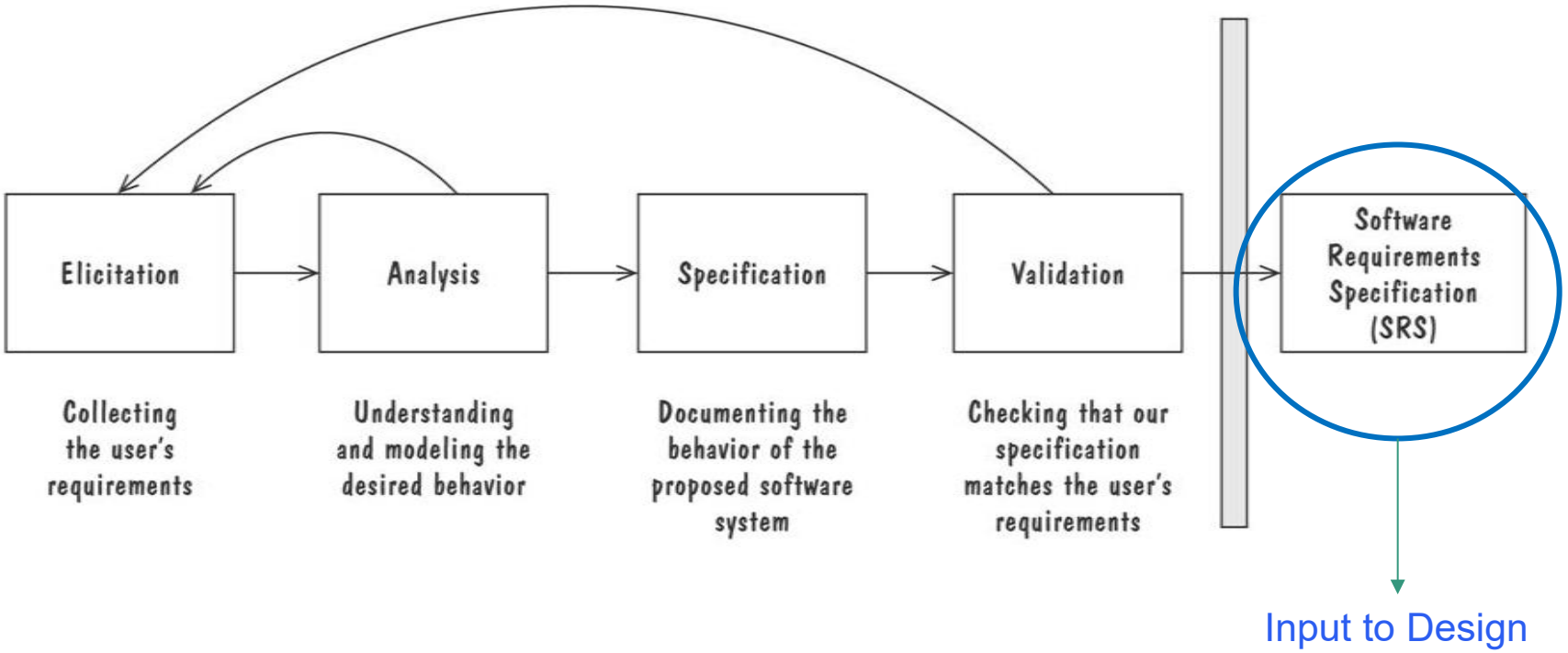
Requirements Gathering Techniques

Acceptance and Evaluation Criteria Definition	Benchmarking	Brainstorming
Business Rules Analysis	Data Dictionary and Glossary	Data Flow Diagrams
Data Modeling	Decision Analysis	Document Analysis
Estimation	Focus Groups	Functional Decomposition
Interface Analysis	Interviews	Lessons Learned Process
Metrics and Key Performance Indicators	Non-Functional Requirements Analysis	Observation
Organization Modeling	Problem Tracking	Process Modeling
Prototyping	Requirements Workshops	Risk Analysis
Root Cause Analysis	Scenarios and Use Cases	Scope Modeling
Sequence Diagrams	Structured Walkthrough	Survey / Questionnaire
SWOT Analysis	User Stories	Vendor Assessments

While not an exhaustive list, these are all techniques we use to elicit and document requirements; we will provide a template for documenting your business requirements



SOFTWARE REQUIREMENTS DEVELOPMENT

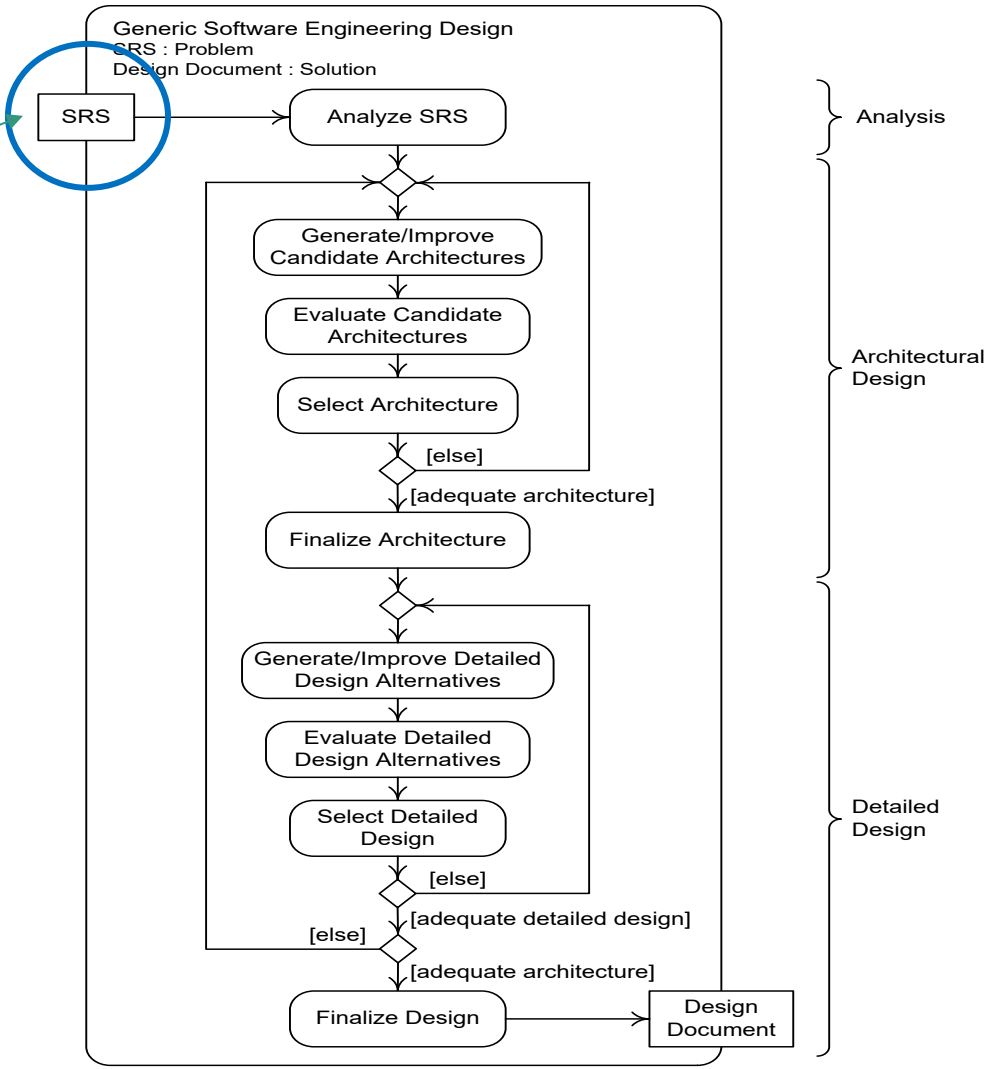


[Source: Pfleeger & Atlee]

SOFTWARE DESIGN

From Requirements development

SRS is input to software design



SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT (SRS)

- SRS – is a project deliverable
- IEEE Standard 830-1998, "IEEE Recommended Practice for Software Requirements Specifications"

1. Introduction to the Document
   - 1.1 Purpose of the Product
   - 1.2 Scope of the Product
   - 1.3 Acronyms, Abbreviations, Definitions
   - 1.4 References
   - 1.5 Outline of the rest of the SRS

2. General Description of Product
   - 2.1 Context of Product
   - 2.2 Product Functions
   - 2.3 User Characteristics
   - 2.4 Constraints
   - 2.5 Assumptions and Dependencies

3. Specific Requirements
   - 3.1 External Interface Requirements
     - 3.1.1 User Interfaces
     - 3.1.2 Hardware Interfaces
     - 3.1.3 Software Interfaces
     - 3.1.4 Communications Interfaces
   - 3.2 Functional Requirements
     - 3.2.1 Class 1
     - 3.2.2 Class 2
     - …
   - 3.3 Performance Requirements
   - 3.4 Design Constraints
   - 3.5 Quality Requirements
   - 3.6 Other Requirements

4. Appendices

REQUIREMENTS TYPES

Requirement type	Example

• Functional requirement:
  • describes required behavior in terms of required activities.
• Quality requirement or nonfunctional requirement:
  • describes some quality characteristics that system must posses.
• Design constraint:
  • a design decision such as choice of platform or interface components.
• Process constraint:
  • a restriction on the techniques or resources that can be used to build the system.
	• The system shall issue a paycheck for each employee in the payroll list.
	• The system shall pull the employee information from an existing database.
	• The system shall be available 90% of the time during workdays.
	• Formal inspections will be used for all code.
	• The project will use IEEE Standards 830 for requirements and 42010 for design.

[From ENPM612]

Exercise: Match each example on the right with the appropriate type of requirements

REQUIREMENTS TYPES (CONTINUED)

Requirement type

- Functional Requirement:
  - describes required behavior in terms of required activities.
- Quality / Nonfunctional Requirement:
  - describes some quality characteristics that system must posses.
- Design Constraint:
  - a design decision such as choice of platform or interface components.
- Process Constraint:
  - a restriction on the techniques or resources that can be used to build the system.

Example

- The system shall issue a paycheck for each employee in the payroll list. **(F)**
- The system shall pull the employee information from an existing database. **(DC)**
- The system shall be available 90% of the time during workdays. **(Q / NF)**
- Formal inspections will be used for all code. **(PC)**
- The project will use IEEE Standards 830 for requirements and 42010 for design. **(PC)**

[From ENPM612]

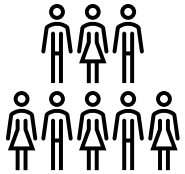
REQUIREMENTS SPECIFICATION NOTATIONS

- **Natural language**
  - (+) Easy to understand
  - (-) Prone to ambiguity and vagueness
- **Semi-formal** notations
  - (+) More precise than natural language
  - (-/+) Not defined with mathematical rigor (most UML diagrams)
  - (+) (relatively) Easy to understand
- **Formal notations**
  - (+) Mathematical and logical notations -> No ambiguity and vagueness
  - (+) Very precise
  - (-) Hard to understand

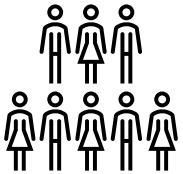
Resources: Software Engineering, Ian Sommerville, Ch 27 https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/WebChapters/PDF/Ch_27_Formal_spec.pdf
https://www.slideshare.net/SharifSalem/1-formal-methods-introduction-for-software-engineering

REQUIREMENTS STAKEHOLDERS

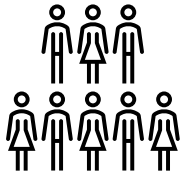
Requirements



- Requirements Analyst
- Requirements Engineer
- Requirements Manager
- Requirements Specialist
- …



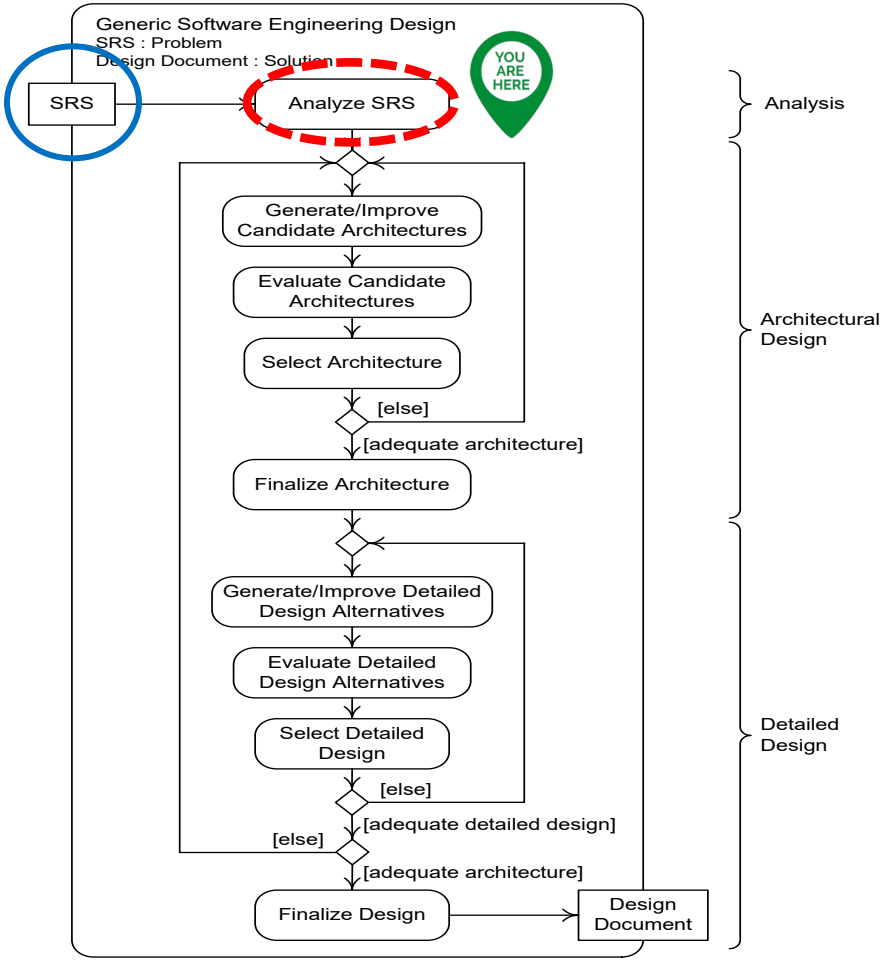
- Program / Project Manager
- Solutions Architect
- Designer
- UX Specialist
- Software Developer
- Software Engineer
- …



- Customer
- User
- Client
- External System
- Operations
- Staff
- …

SOFTWARE DESIGN

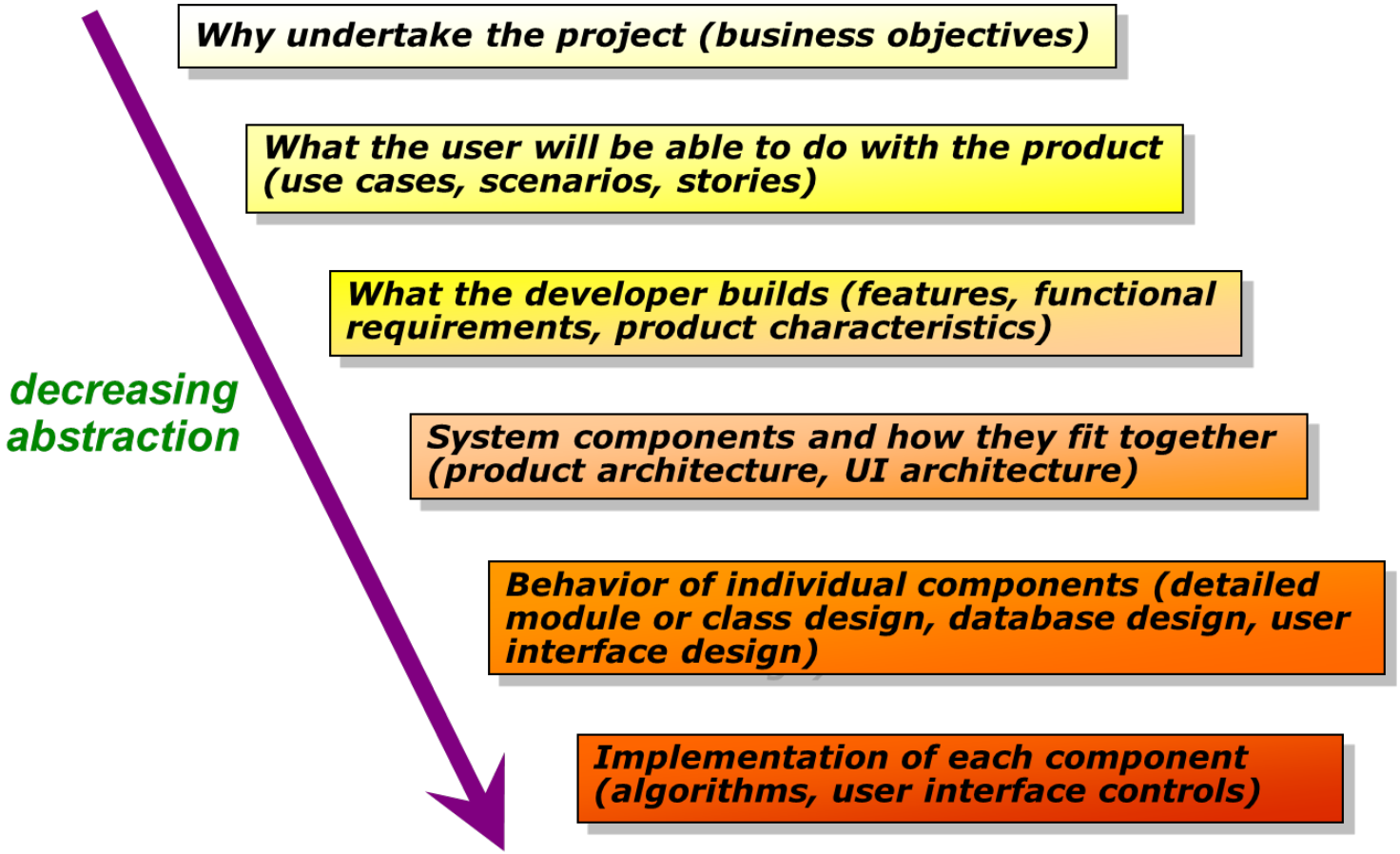
Software design starts with analyzing the SRS



SRS ANALYSIS GOALS, INPUTS, AND ACTIVITIES

- Goal: to **understand** the **problem** for engineering design, using
  - SRS
  - Requirements models
- *Analysis* is for:
  - Understanding a problem by breaking it down
  - Clarification
  - Agreement between requirements author/producer and consumer(s)
    - Can lead to requirements update/correction
- Understanding can be achieved by
  - Studying the SRS
  - Studying Requirements models
    - If *modeling was used for as a requirements development approach* (e.g., in ENPM612)
  - Developing analysis models
    - ***Modeling is used in this case as an analysis approach***

THE LINE BETWEEN REQUIREMENTS AND DESIGN



The Fuzzy Line Between Requirements and Design, by Karl Wiegers

OUTLINE

- Course project
- Lecture
  - Modeling in software development
  - Requirements engineering quick recap
  - Quick UML recap (class, use case, activity, diagrams)
  - Requirements analysis
    - Domain modeling
    - Context modeling
    - Use case modeling
    - Data modeling

We are here

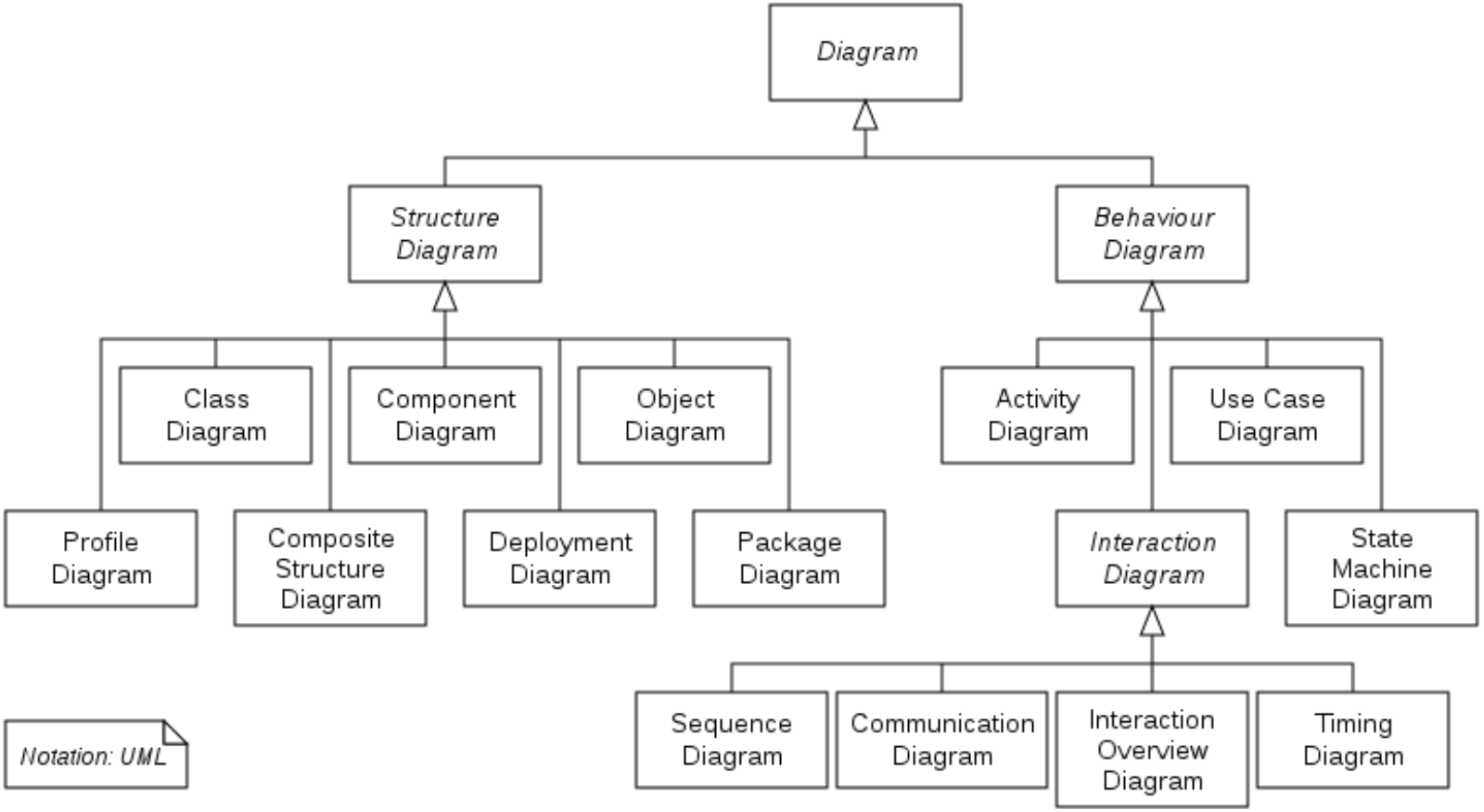
UNIFIED MODELING LANGUAGE (UML) OVERVIEW

- UML is a nonproprietary, standard, general purpose, graphical language for **modeling software systems**
  - Used for visualizing, specifying, constructing, and documenting the artifacts of software-intensive systems
- UML was the result of convergence of various notations used in object-oriented methods
- Typically used for OO development
- UML is a *language*, **not** a method or procedure
- You need to *first* identify the **type of information** to be captured in a model, and *then* the type of notation/diagram that best represents that information

UML OVERVIEW (CONTINUED)

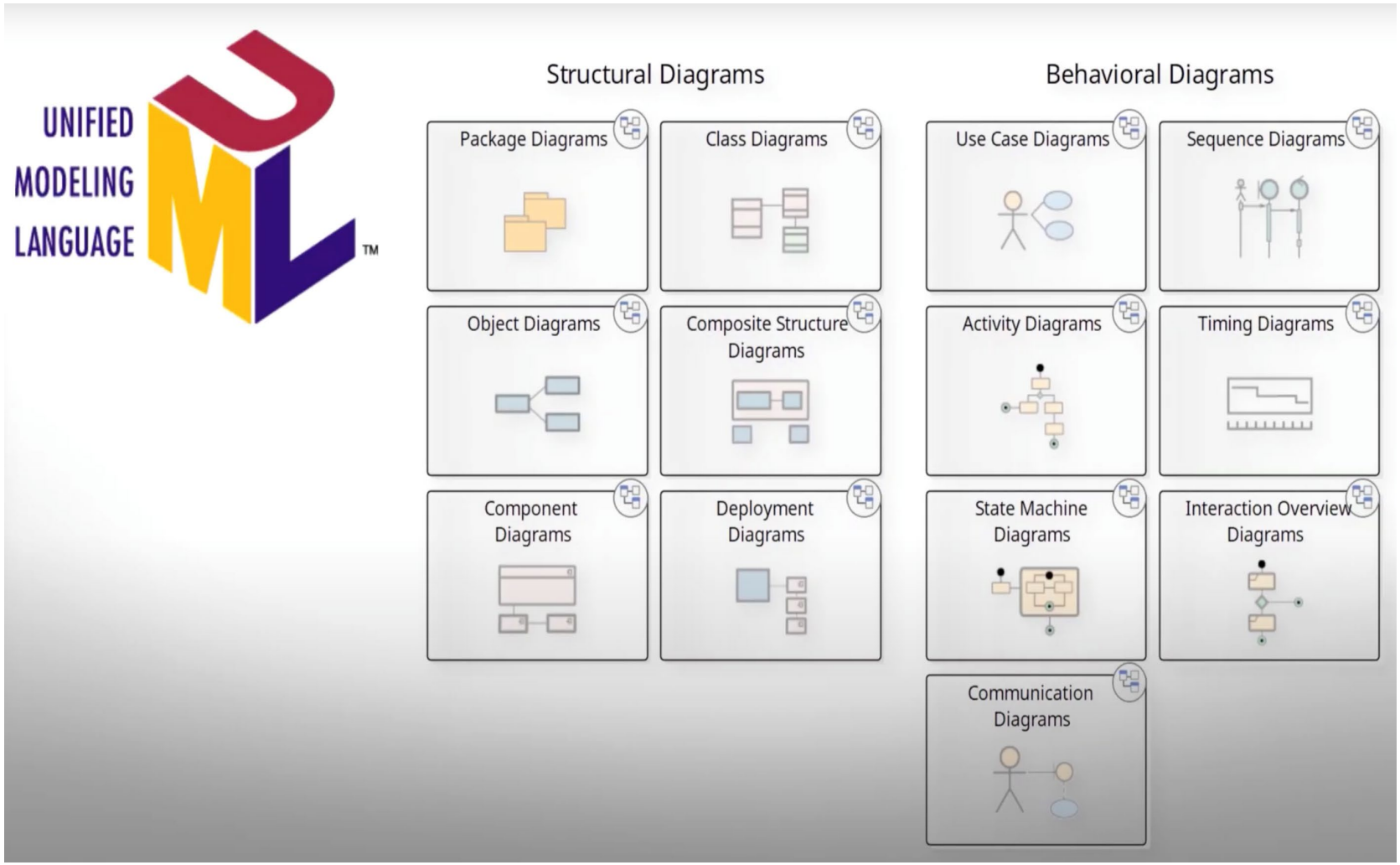
- UML specification (standard) is updated and managed by the Object Management Group (OMG) http://www.omg.org
  - International, open membership, not-for-profit technology standards consortium
  - Latest version: 2.5
- SysML (Systems Modeling Language) is a UML profile for *systems* modeling
- There are many tools that support UML modeling – *Model Based Software Engineering*
  - Example: Rhapsody (IBM), Enterprise Architect (Sparx), MagicDraw/Cameo (NoMagic), Visual Paradigm
  - Reduce the costs to develop and maintain models
  - Enable automatic code and documentation generation from models
  - Enable reverse engineering from code to design
  - Other drawing tools support drawing UML diagrams (e.g., yEd, StarUML, Lucidchart, Visio)

UML NOTATION - DIAGRAMS



For UML 2.5 diagrams see: https://www.uml-diagrams.org/uml-25-diagrams.html

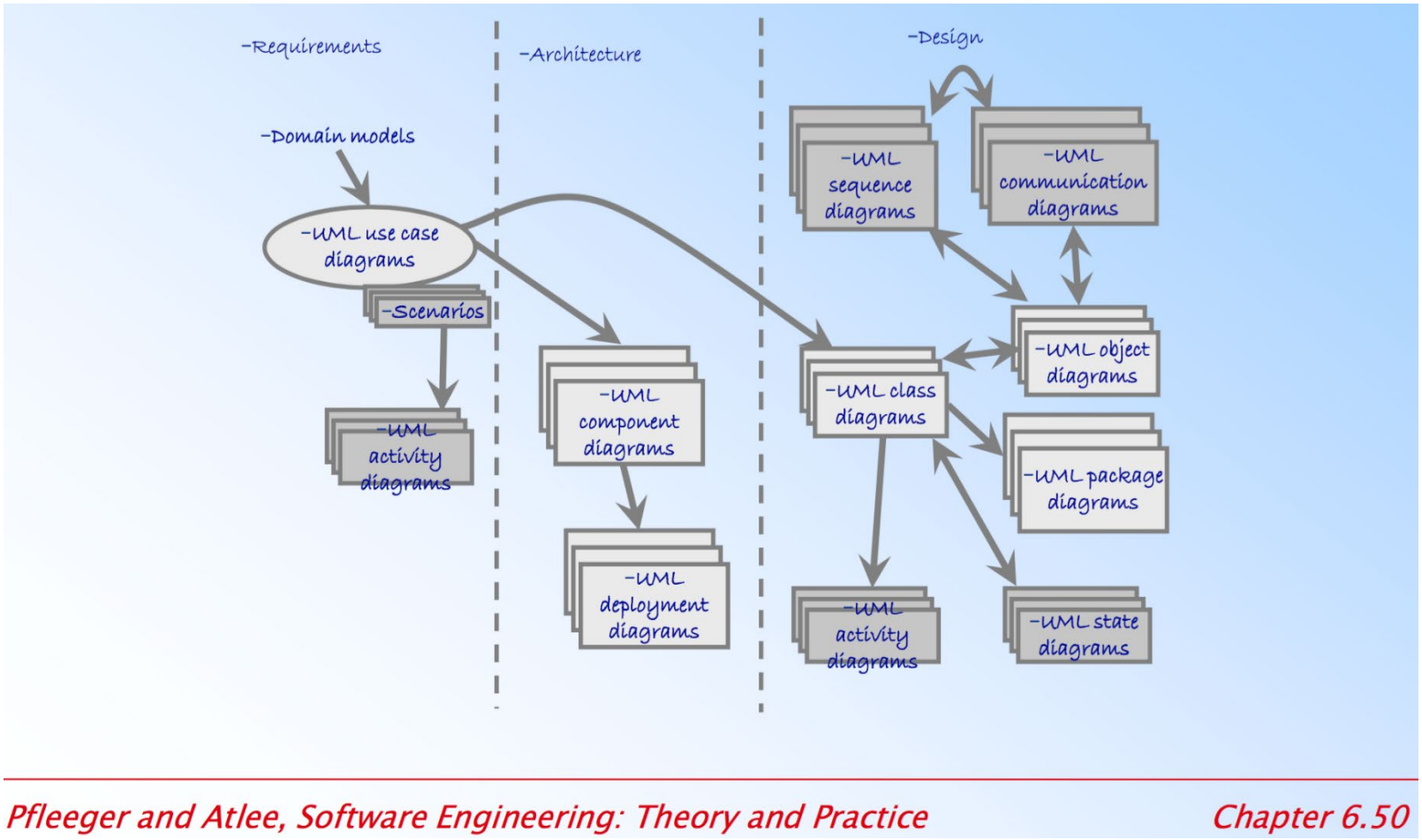
UNIFIED MODELING LANGUAGE



USING UML IN SOFTWARE DEVELOPMENT

- Where (for what models) can UML be used in software development?

USING UML IN SOFTWARE DEVELOPMENT



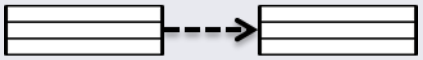
Pfleeger and Atlee, Software Engineering: Theory and Practice *Chapter 6.50*

UML CLASS DIAGRAM

- One of the most important (and mostly used) UML analysis and design notations
- Almost universally used for object-oriented analysis and design
- Is a *structural* diagram
- Used throughout the development lifecycle at different levels of abstraction to model:
  - The problem - the concepts and their relationships
  - The solution - the design elements and their relations

UML CLASS DIAGRAM NOTATION

Class relations

Notation	Relationship	Definition	Uses
	**Association**	- a family of links. A Binary association is normally represented as a line. Can link any number of classes. Represents a static relationship shared among the objects of two classes. **"A Student has one or More Instructors"**	• Bidirectional • Uni-Directional
	**Generalization / Inheritance**	- any instance of the subtype is also an instance of the superclass. "**is a kind of", "an A is a B", "a Human is a mammal", "a mammal is an animal".**	• Strong Lifecycle Dependency
	**Realization / Implementation**	- a relationship between two model elements in which one model element (the client) realizes (implements or executes) the behavior that the other model element (the supplier) specifies.	• Weak Lifecycle Dependency
	**Dependency**	- a type association where there is a semantic connection between dependent and independent model elements. **Changes to one element may cause changes to the other.**	• Uni-Directional
	**Aggregation**	- a variant of the **"has a"** association relationship. Aggregation is more specific than association. An Association that represent a part-whole or part-of relationship. **"A Professor has a Class".** May not involve more than two classes.	• Uni-Directional • Bi-Directional
	**Composition**	- a special case of association that describes a relationship between a whole and its existential parts. **"A Car has a Wheel"**	• Binary • Bilateral • Strong Form

CLASS ATTRIBUTE AND OPERATION EXAMPLES

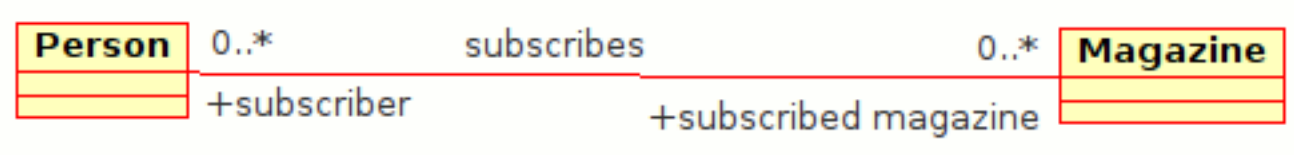
Player
roundScore : int = 0 totalScore : int = 0 words : String[*] = ()
resetScores() setRoundScore(in size : int) findWords(in board : Board) getRoundScore() : int getTotalScore() : int getWords() : String[*]

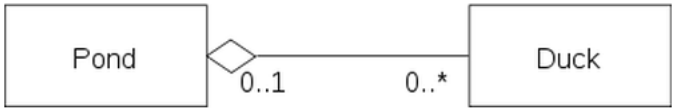
WaterHeaterController
mode : HeaterMode = OFF occupiedTemp : int = 70 emptyTemp : int = 55
setMode(newMode : Mode = OFF) setOccupiedTemp(newTemp : int) setEmptyTemp(newTemp : int) clockTick(out ack : Boolean)

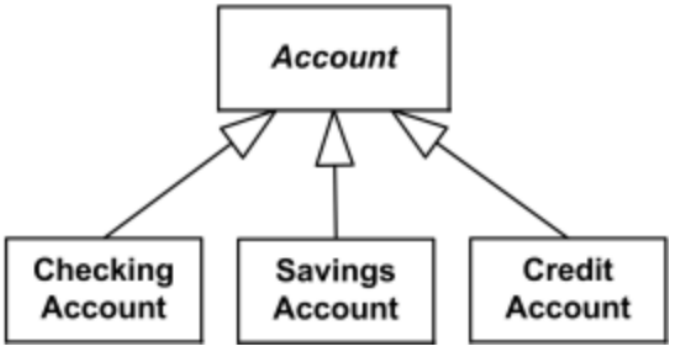
CLASS RELATIONS EXAMPLES

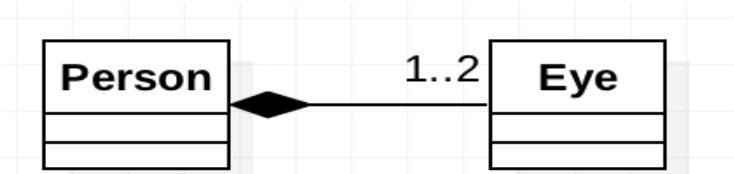
Exercise: Name the relations in these diagrams

a) Association
b) Composition
c) Aggregation
d) Generalization
e) Dependency
f) Navigation



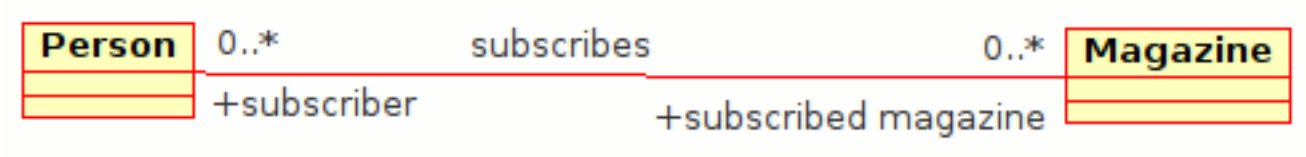




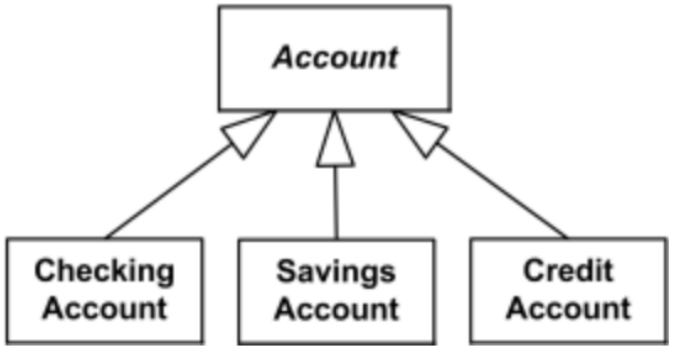


CLASS RELATIONS EXAMPLES

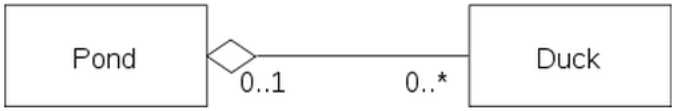
Association



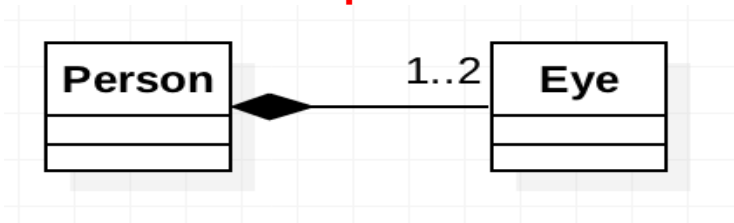
Generalization



Aggregation



Composition



ASSOCIATION MULTIPLICITY

- **Multiplicity** – an indication of how many objects may participate in the given relationship or the allowable number of instances of the element.

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

CLASS DIAGRAMS REPRESENT STRUCTURAL MODELS

- Capture the *static structure* of the system
  - Entities (e.g., classes, interfaces, components, nodes)
  - Relationship between entities
  - Internal structure
- Do **not** capture
  - Behavior
  - Temporal information (Timing)
  - Runtime constraints

USE OF UML CLASS DIAGRAM

- Analysis or conceptual models—Important *entities* or *concepts* in the problem, their *attributes*, important *relationships*
- (Architecture) Design models—Components in a software system, attributes, operations, associations, but no implementation details
- Detailed design (Implementation) models—Classes in a software system with implementation details

- Requirements analysis models represent the *problem* (or *needs* that the software will address – the "what")
- Architecture and detailed design models represent the *solution* (the "how" the software will address the need)

OUTLINE

- Course project
- Lecture
  - Requirements engineering quick recap
  - Modeling in software development
  - Quick UML recap (class, use case, activity, diagrams) and their use
  - Requirements analysis
    - Domain modeling
    - Context modeling
    - Use case modeling
    - Data modeling

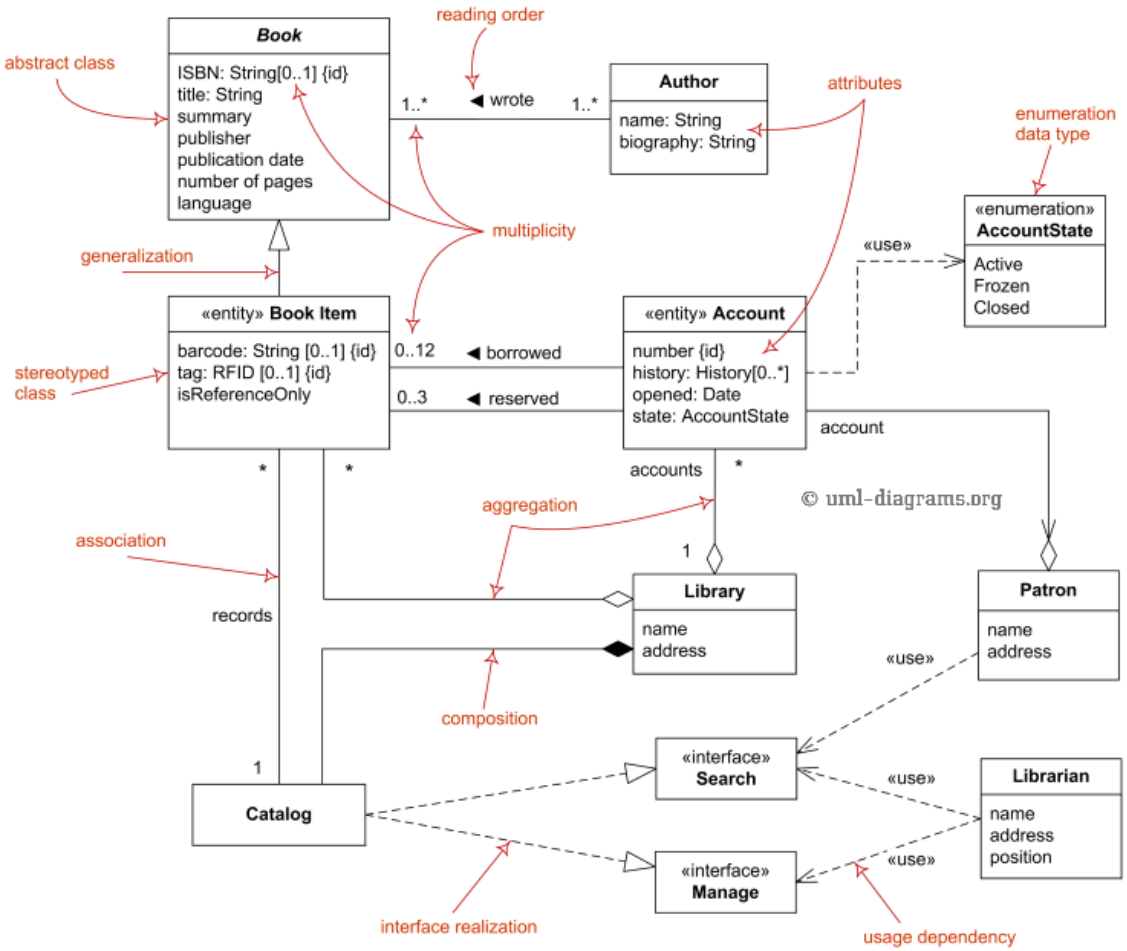
We are here

DOMAIN MODEL

- An abstraction the “world” (domain) in which the software will execute
- Purpose of *understanding* the problem/need the software will address
- Represents the important *entities* in a **problem**, their *responsibilities* or *attributes*, the important *relationships* among them, (and perhaps their behaviors)
- A **static** model

- Very useful when we develop software in a new (unfamiliar) domain

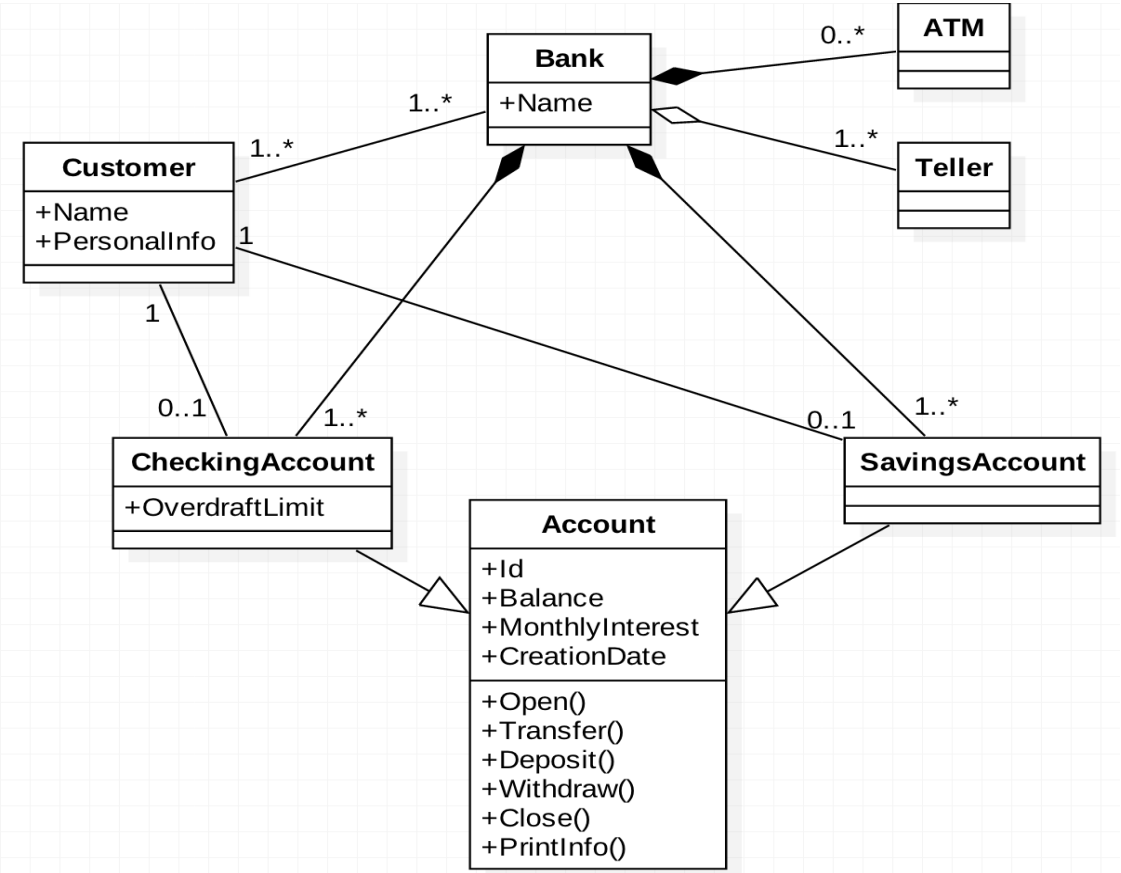
EXAMPLE – ONLINE LIBRARY DOMAIN MODELING



EXAMPLE 1 - BANK ACCOUNT SYSTEM DOMAIN

- Banks offer individual customers bank accounts.
- Each customer can have one checking and one savings account with any given bank.
- A customer can have accounts with more than one bank, and each bank has multiple customers.
- A customer can open an account, deposit money in it, withdraw money from it, and close it.
- The deposit and withdraw operations can be done by using an ATM, or by interacting with a human teller.
- Opening an account and closing it must be done by a teller (cannot be done by using an ATM).
- A checking account has an overdraft limit (of $5,000), but a savings account cannot be overdrawn.
- A customer can transfer funds between the checking and savings accounts.
- Customer can print the following information for the checking and savings account:
  - current balance (after withdrawal and deposit),
  - the monthly interest, and
  - the date when the account was created.

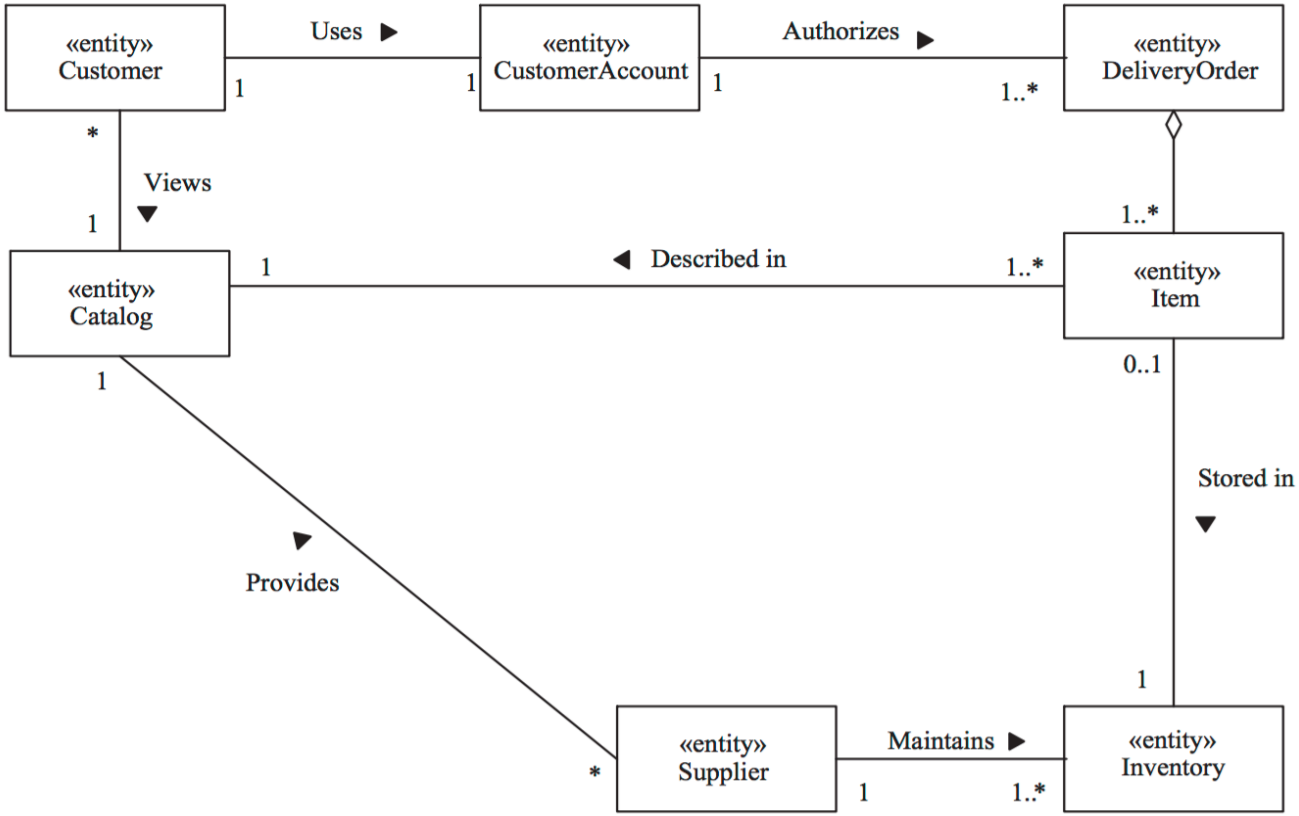
EXAMPLE 1 - BANK ACCOUNT SYSTEM (BAS) *DOMAIN MODEL* USING UML CLASS DIAGRAM NOTATION



EXAMPLE 2 - ONLINE SHOPPING SOFTWARE DOMAIN

- In the Web-based Online Shopping System, customers can request to purchase one or more items from the supplier.
- The customer provides personal details, such as address and credit card information.
- This information is stored in a customer account.
- Using a valid credit card, a delivery order is created and sent to the supplier.
- The supplier checks the available inventory, confirms the order, and enters a planned shipping date.
- When the order is shipped, the customer is notified, and the customer's credit card account is charged.

EXAMPLE 2 – ONLINE SHOPPING SOFTWARE *DOMAIN MODEL* USING UML CLASS DIAGRAM NOTATION



From: *Software Modeling and Design* by Hasan Gomaa

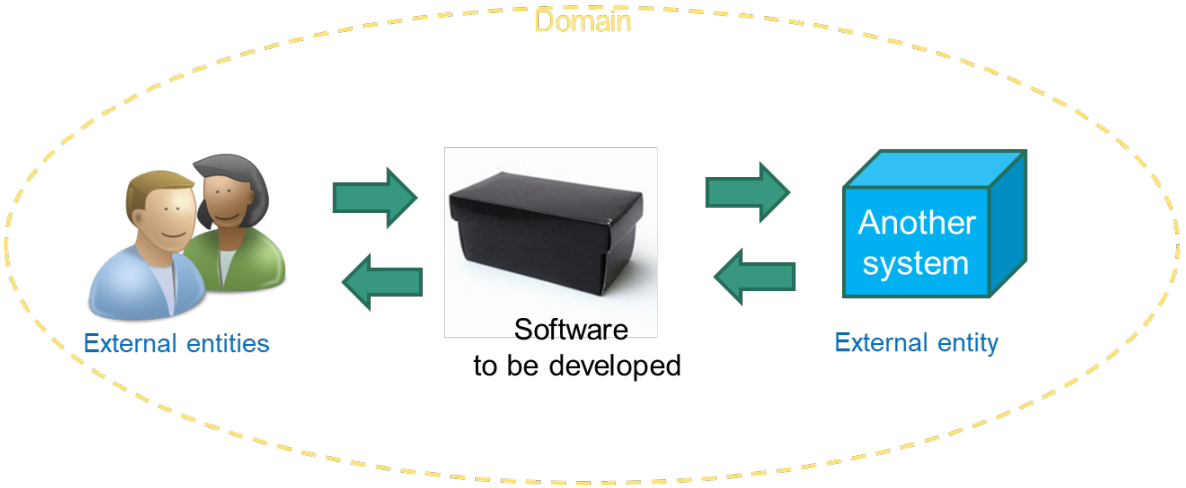
OUTLINE

- Course project
- Lecture
  - Requirements engineering quick recap
  - Modeling in software development
  - Quick UML recap (class, use case, activity, diagrams) and their use
  - Requirements analysis
    - Domain modeling
    - Context modeling
    - Use case modeling
    - Data modeling

We are here

CONTEXT MODEL

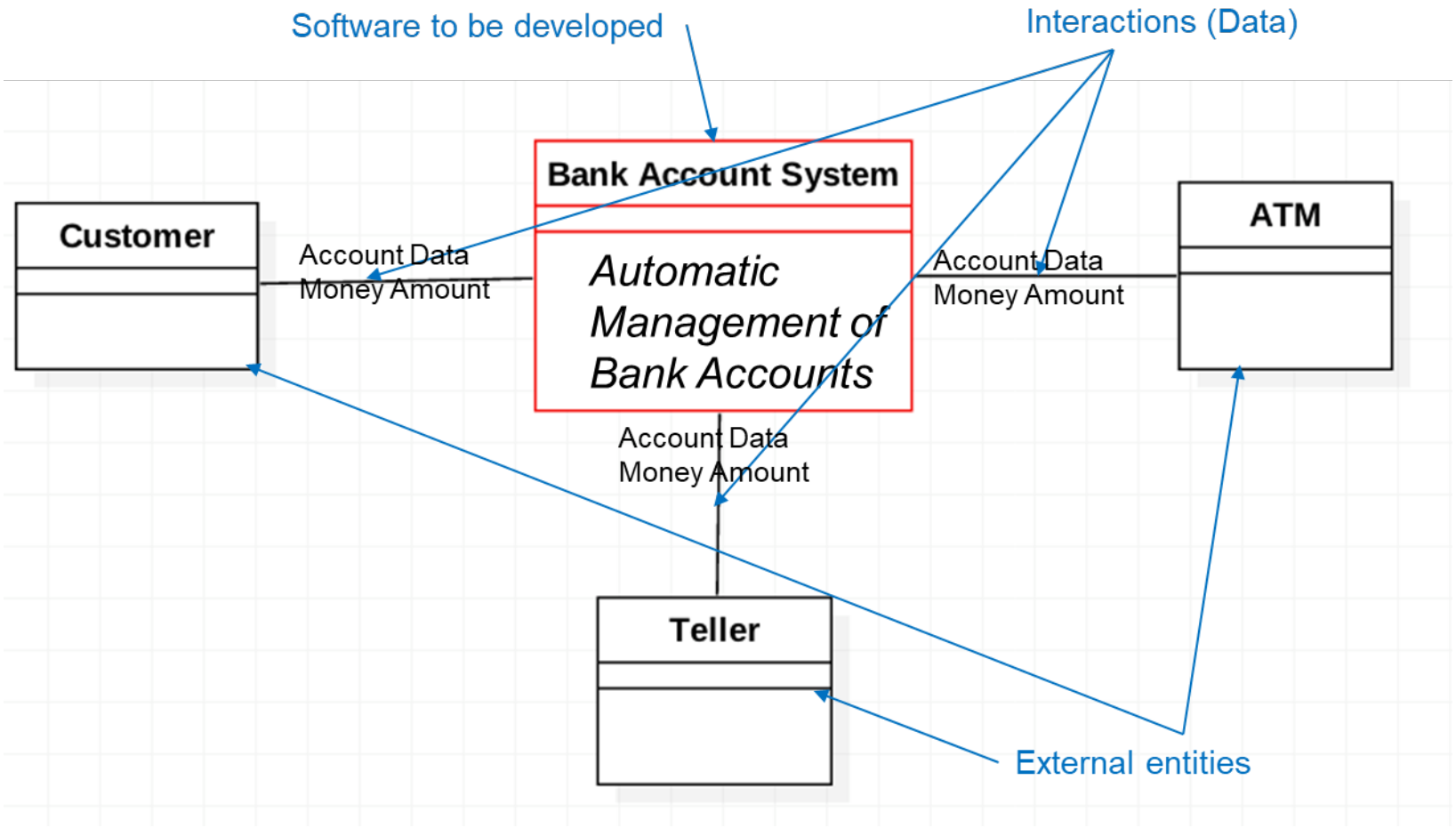
- The most abstract view of the software system:
  - Identifies the *boundaries,* the *scope* of the system and its external *interactions*
    - Treat the system as a **black box** and look only at its interactions with the environment



CONTEXT MODEL (CONTINUED)

- Can be documented using
  - A UML class diagram
    - Software to be developed and external entities represented as classes
    - Interactions represented as relations
      - Label interactions with names of exchanged data
  - **And** a textual description of the system to be developed and of the external entities

EXAMPLE 1 – BAS CONTEXT MODEL USING UML CLASS DIAGRAM NOTATION



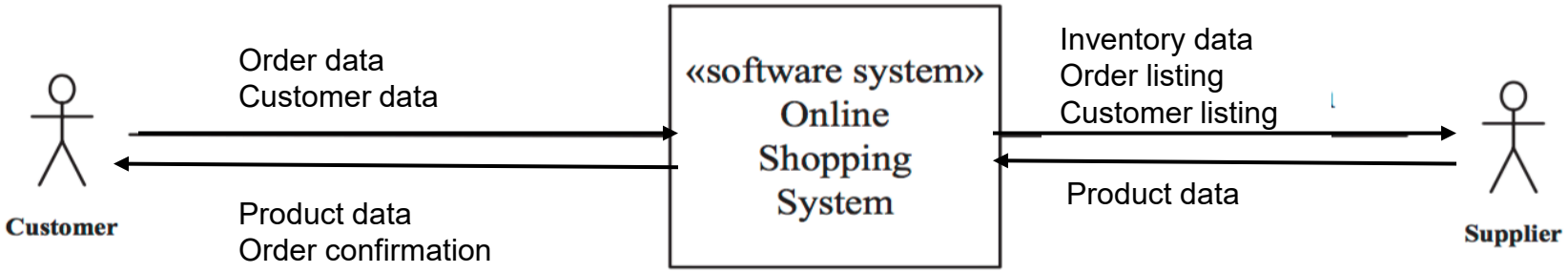
EXAMPLE – BAS CONTEXT MODEL DESCRIPTION

- BAS description: The BAS software supports customers to manage their bank accounts, and perform financial transactions, with the assistance of tellers and ATMs

- External entities description
  - Customer: A human who use the BAS to manage his/her account(s)
  - Teller: A bank employee who interacts with BAS to assist customers
  - ATM: A bank machine who interacts with BAS to assist customers

EXAMPLE 2 – ONLINE SHOPPING SOFTWARE CONTEXT MODEL USING UML CLASS DIAGRAM NOTATION

Diagram



Adapted from: *Software Modeling and Design* by Hasan Gomaa

Textual description

Customer: A person who buys items online

Supplier: A person/company who sells items online and sends them to the customer

Online shopping system: Facilitates ordering and purchasing of one or more items from supplier by customer and maintains online items inventory.

OULTINE

- Course project
- Lecture
  - Requirements engineering quick recap
  - Modeling in software development
  - Quick UML recap (class, use case, activity, diagrams) and their use
  - Requirements analysis
    - Domain modeling
    - Context modeling
    - Use Case modeling
    - Data modeling

We are here

USE CASES

- A representation of the interactions and behavior of a software system, **visible from the outside of the system**
  - Who interacts with the system?
  - How? What behavior do they expect from the system?
- A **use case** is a type of complete **interaction between a product and its environment**

ACTORS

- An **actor** is a type of agent that interacts with a product
- External entity - human or other software/system
- Types of actor:
  - Primary: initiator of the interactions
  - Secondary: actors that also interact with the system in a use case

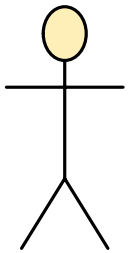
USE CASE MODEL

- Use case (UC) model is the set of all use cases that completely describe the functionality of the system and the actors

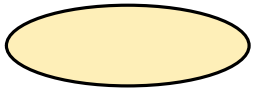
- Working with use cases helps organize, **analyze**, generate, and/or evaluate *functional* requirements

- For representing use case models we will use:
  - UML Use Case diagram **AND**
  - Use case description
    - Textual or
    - Graphical (e.g., using UML activity diagram)

UML USE CASE DIAGRAM NOTATION



Actor Name



Use Case Name

Actor Symbol

Association Line

Use Case Symbol

RELATIONS IN A USE CASE MODEL

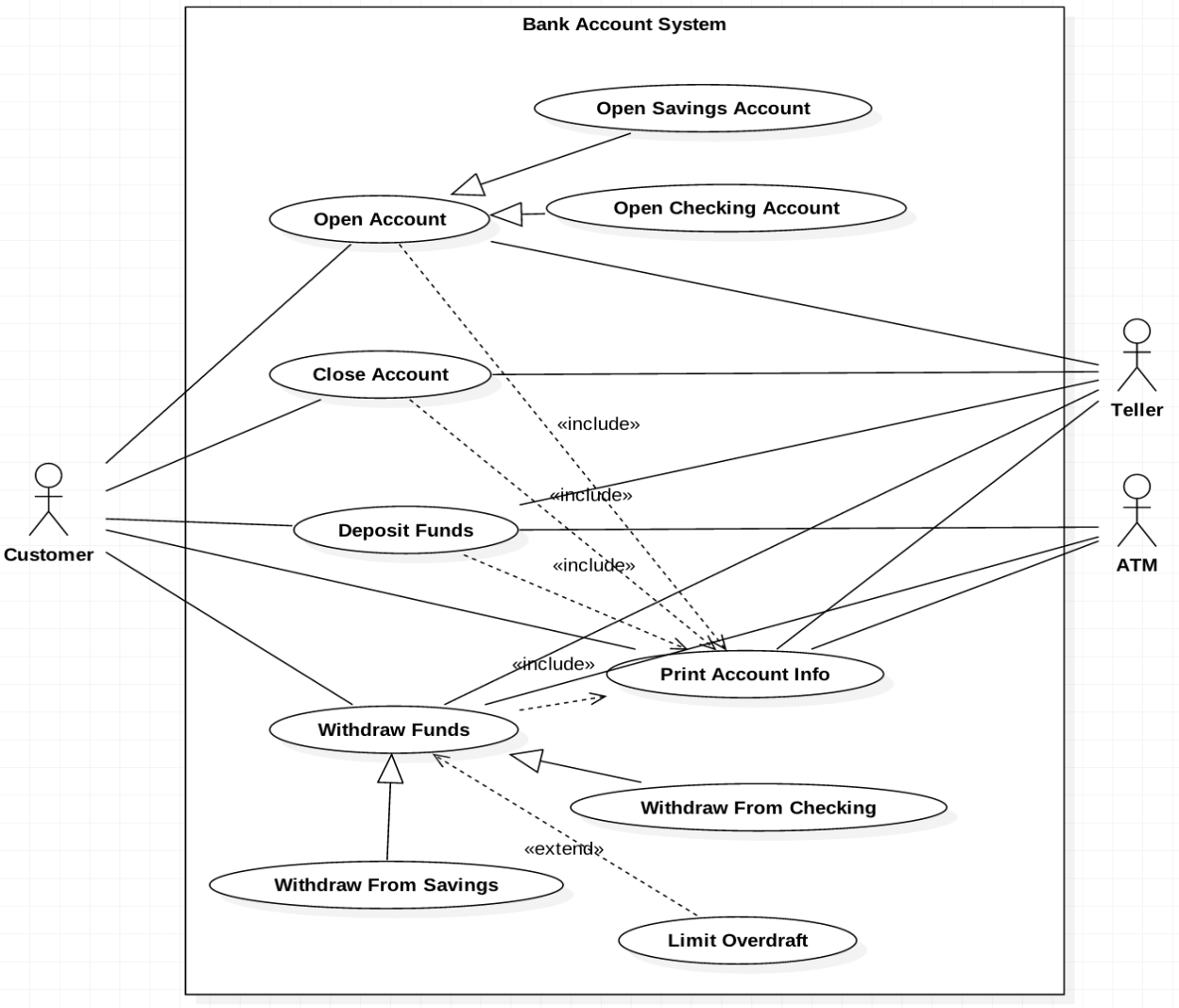
- <<includes>> relation exists between a use case A and use case B if the process of doing A always involves doing B at least once
  - Promotes reuse
- <<extends>> relation exists between a use case A and use case B, if B inserts “behavior fragments” during the performance of A (under certain conditions)
  - B may have no independent existence except as an augmentation of A
  - An extension may be characterized by *extension points* and *conditions of extensions*
- *Generalization* - Specializes abstract use cases to more concrete ones

BEWARE OF MISUSING USE CASE RELATIONS!

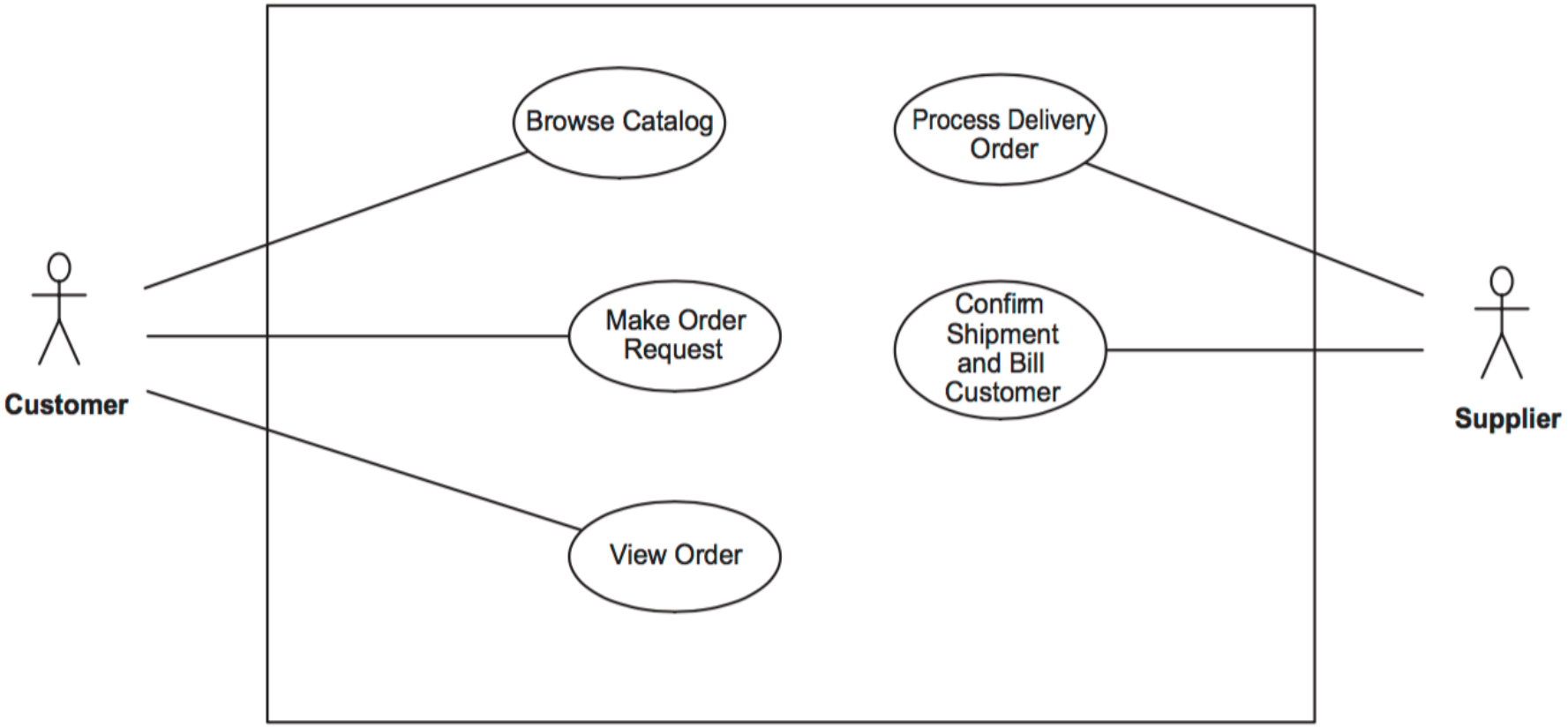
<<include>>

- should be used where there is an **opportunity for reuse** of the "included" use case, between two or more "including" use cases
- but should **not** be used to:
  - decompose a use case
  - indicate the order of which use cases occur or
  - show some vague connection between use cases

BANK ACCOUNT SYSTEM USE CASES - USING UML USE CASE DIAGRAM (UCD) NOTATION

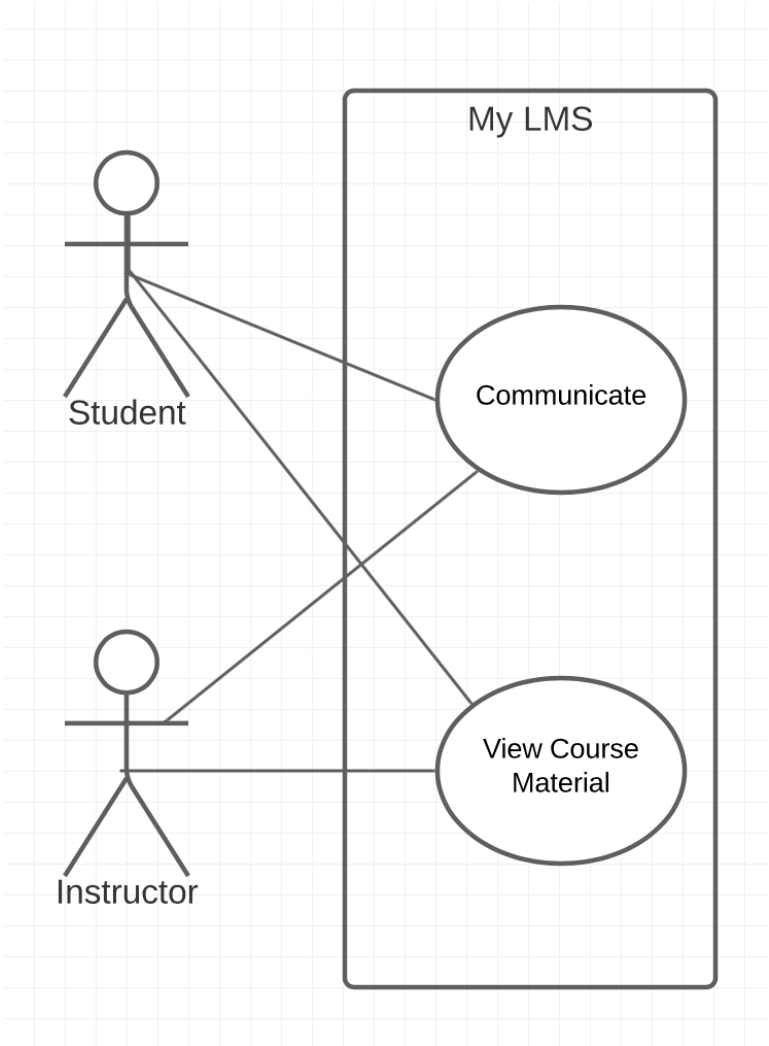


ONLINE SHOPPING SOFTWARE USE CASE DIAGRAM



From: *Software Modeling and Design* by Hasan Gomaa

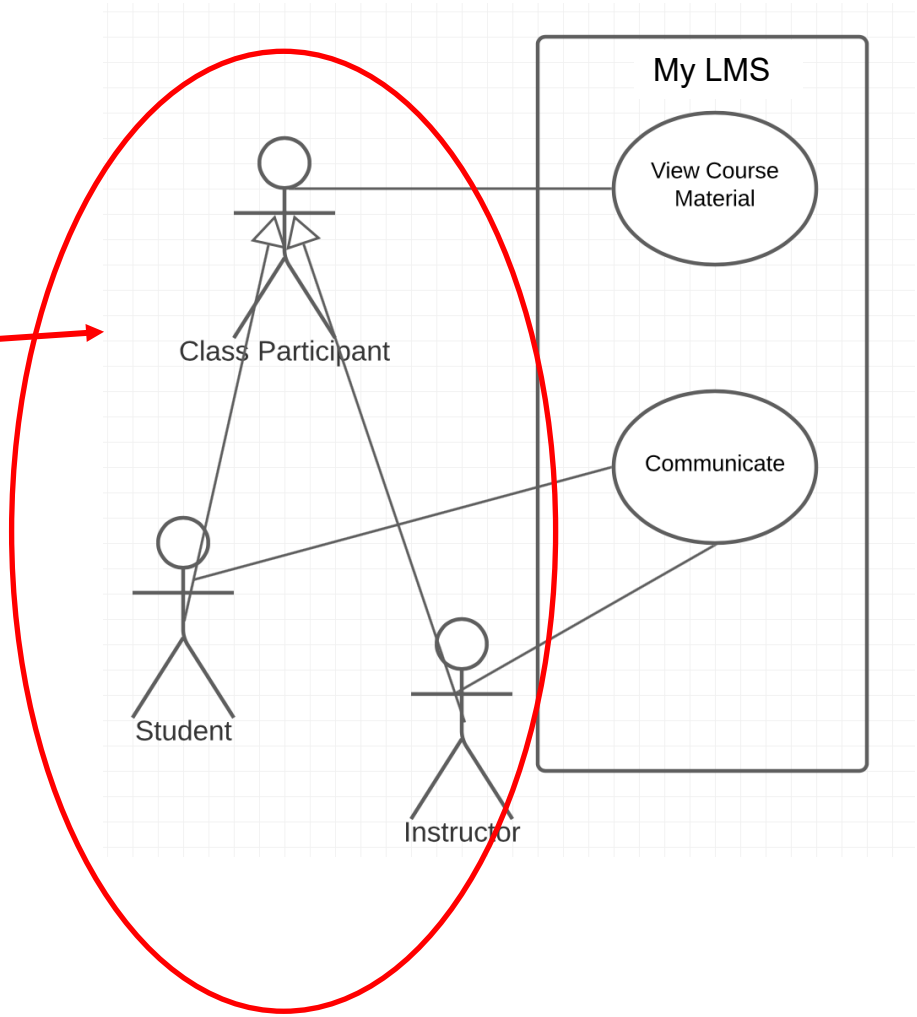
ACTORS PARTICIPATING TO A USE CASE – VERSION1



- How do read this diagram?
- Do Student and Instructor actors participate *together* in both use cases?
- If not, then is this diagram accurate?

ACTORS PARTICIPATING TO A USE CASE – VERSION 2

Generalization relation between actors



USE CASE FORMATION RULES

- Every use case diagram must have
  - At least one use case
  - At least one actor
  - Name for every actor and use case
  - At least one actor associated with each use case
  - At least one use case associated with each actor
  - No association line between actors
  - No association line between use cases

VERIFYING A USE CASE MODEL

- Review the stakeholders list to make sure no actors are missing
- Review the needs list to make sure no uses cases are missing
- Review constraints and limitations to make sure they are not violated
- Generate an event list and check that all events are handled
- Check that the collection of use cases covers all externally visible behavior
- Check the diagram against the use case heuristics

USE CASE *DESCRIPTION*

- Adds details, to describe the behavior of the software and its use by the actors
- Focuses on the *event flow* between *actor(s)* and *system*

- Textual description or
- Graphical description – using UML activity diagram (or an interaction diagram, e.g., sequence diagram)

USE CASE TEXTUAL DESCRIPTION TEMPLATE

- **Name**: the name of the use case
- **Actors:** the primary and secondary actors that participate in the use case
- **Trigger**: what external event happens that triggers the start of the use case
- **Preconditions**: any prerequisites before the use case can be started
- **Postconditions:**
  - **Success postconditions**: what is considered a successful end to the use case
  - **Failure postconditions:** what is considered a failed end to the use case

USE CASE TEXTUAL DESCRIPTION TEMPLATE (CONTINUED)

- **Basic flow:** (aka main course, basic course, normal flow, primary scenario, main success scenario, and happy path). Shows the actions if all goes well (if no mistakes and nothing unusual happens)
- **Alternative flow:** Shows alternative normal case arising from intentional choices made by the user
- **Exception case flow**: Sometimes considered as a type of alternative case scenario. Shows the actions taken to safely rejoin the normal case after an exception occurred. Exception is a condition that prevents a task from succeeding. An implementation of alternative case scenario can be deferred, but not of the exception case scenario.

USE CASE TEXTUAL DESCRIPTION EXAMPLE FOR THE BANK SYSTEM

- **Use Case Name**: Withdraw from Savings
- **Actors**: Customer (Primary) and ATM (Secondary)
- **Preconditions**: Customer has Savings Account and card
- **Postconditions**:
  - Success postconditions: Customer withdraws the desired amount and gets card
  - Failure postconditions: Customer does not withdraw the desired amount or card gets stuck in ATM
- **Trigger**: Customer initiates ATM transaction to withdraw from his/her savings account

USE CASE TEXTUAL DESCRIPTION EXAMPLE FOR THE BANK SYSTEM

- Basic flow:
  - 1. Customer inserts card in ATM
  - 2. ATM accepts card
  - 3. System checks the card
  - 4. ATM displays message to input PIN
  - 5. Customer types PIN and presses ”Enter” button
  - 6. System checks for PIN validity
  - 7. ATM displays message to input amount to be withdrawn
  - 8. Customer inputs the amount to withdraw and presses “Enter” button
  - 9. System subtracts the amount from the account
  - 10. ATM dispenses cash and receipt and releases card
  - 11. Customer retrieves the card, cash, (optionally) receipt

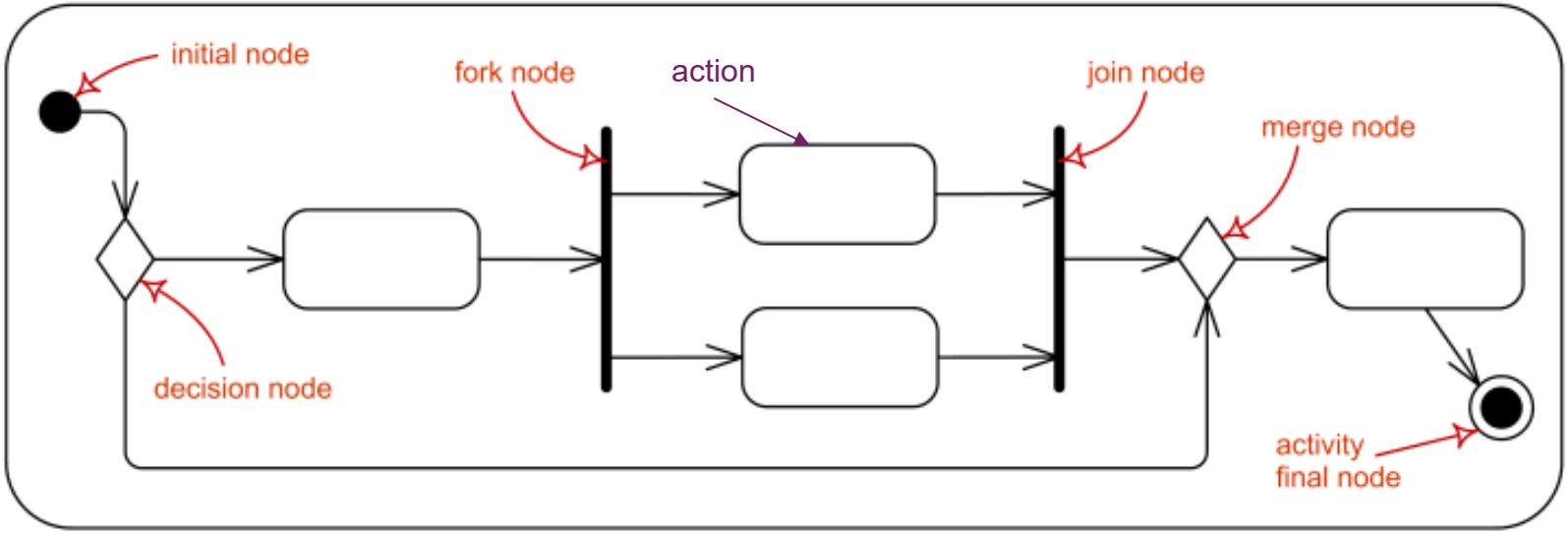
USE CASE TEXTUAL DESCRIPTION EXAMPLE FOR THE BANK SYSTEM (CONTINUED)

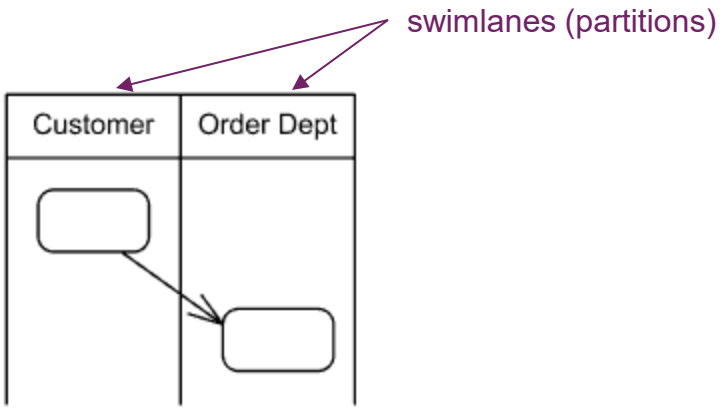
- Alternate flows:
  - 5.a Customer presses “Cancel” button to end transaction
  - 5.b ATM returns the card
  - 5.c Customer takes the card
- Exception flows:
  - 3.a System detects invalid Card
  - 3.b ATM returns card and displays error message
  - 3.c Customer takes the card
  - 9.a System detects insufficient funds
  - 9.b ATM displays error message and message to input amount to be withdrawn
  - ...

UML ACTIVITY DIAGRAM

- Used to represent a behavioral model (dynamic)
- Represents *actions*, the *flow of control* between actions, their *order*, *conditions*, *repetitions*, *parallelism/concurrency*
  - Can also represent *data flow* between actions

UML ACTIVITY DIAGRAM NOTATION



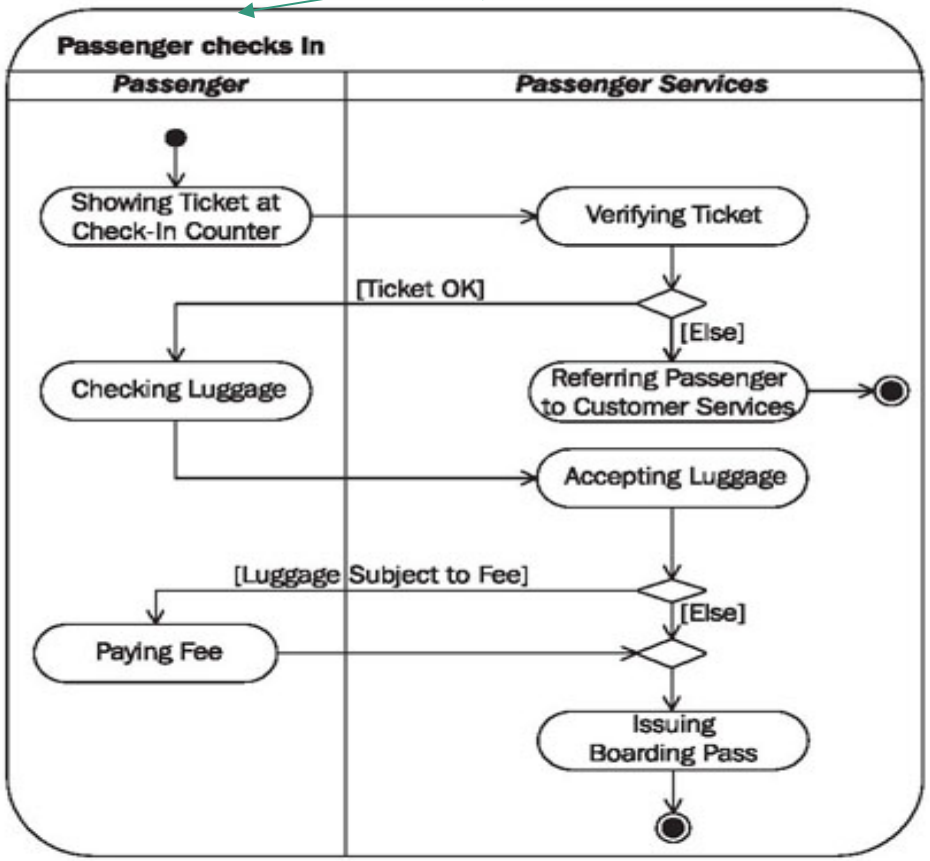


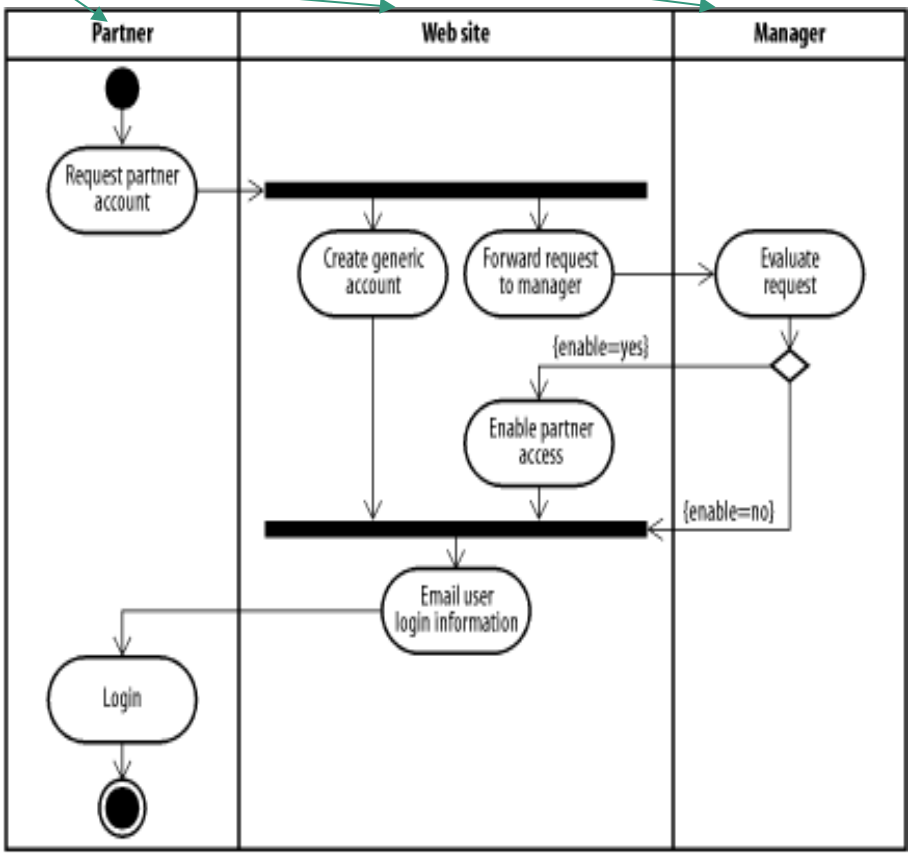
USING UML ACTIVITY DIAGRAM FOR USE CASE DESCRIPTION

- Each actor is represented in a swimlane
- The software to be developed is represented in a swimlane
- Use UML activity diagram elements as follows:
  - *Actions* represent steps in the use case
  - *Decision nodes* represent branching conditions, preconditions, postconditions, and trigger
  - *Merge nodes* represent flows merging
  - *Fork* and *join* (as needed) to represent parallel actions
  - *Initial* and *final node* to represent the start and end of a use case

EXAMPLES - USE CASE DESCRIPTION USING UML ACTIVITY DIAGRAM

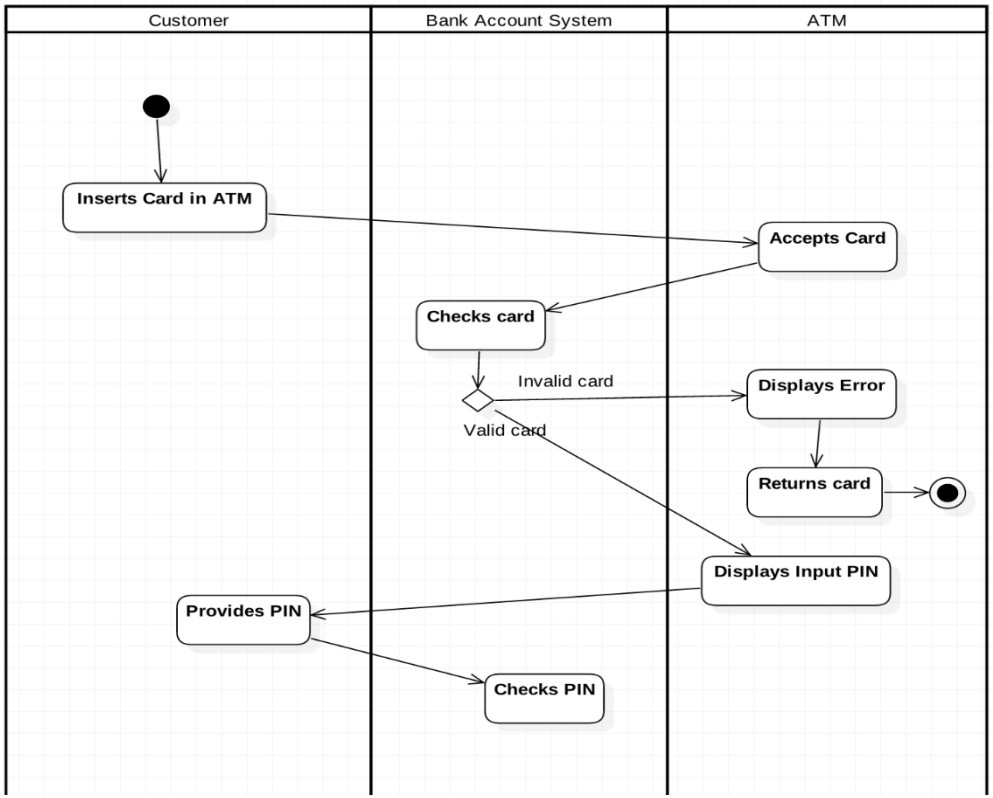
The *system* to be developed and each *actor* are represented in a separate swimlane





BAS USE CASE GRAPHICAL DESCRIPTION USING UML ACTIVITY DIAGRAM

Use case: Withdraw from account



Note: this example activity diagram does not represents the entire use case

USE CASE SCENARIOS

- A use case *scenario* is
  - An interaction between a system and particular actor(s)
  - An instance of a use case; a path through a use case
- A use case is a cohesive set of related scenarios

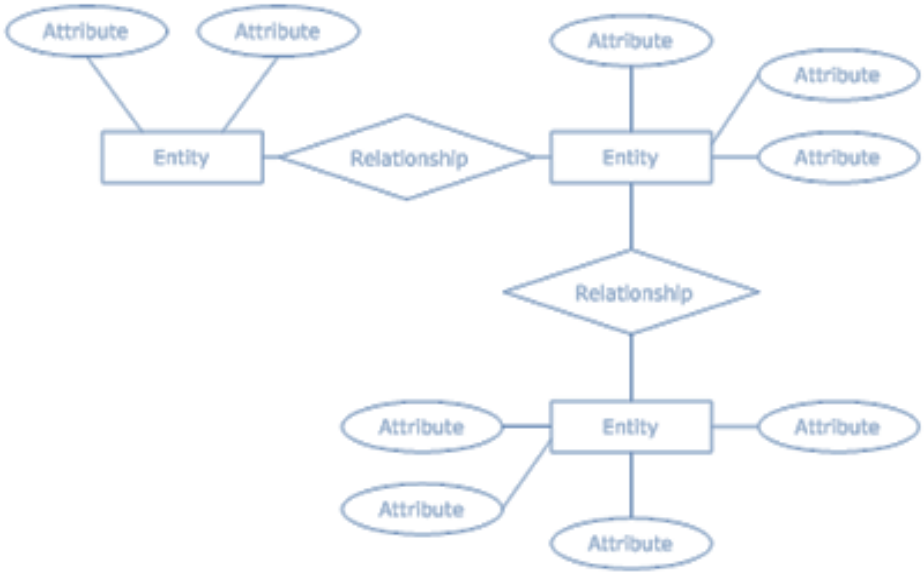
OUTLINE

- Course project
- Lecture
  - Requirements engineering quick recap
  - Modeling in software development
  - Quick UML recap (class, use case, activity, diagrams) and their use
  - Requirements analysis
    - Domain modeling
    - Context modeling
    - Use case modeling
    - Data modeling

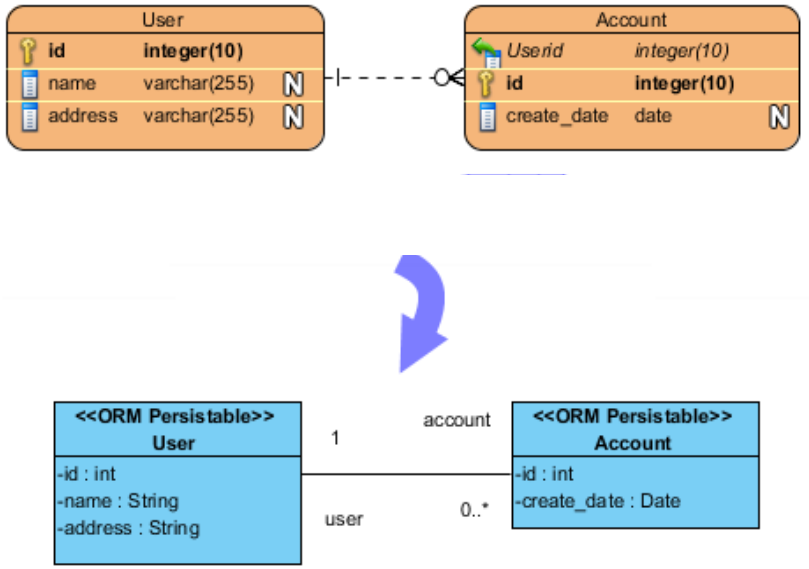
We are here

DATA MODEL

- Abstract conceptual **data model** (or semantic **data model**) used to represent structured **data**
- **Notations:** Entity relationship (ER) diagram or UML class diagram



ER diagram



ER diagram -> UML class diagram

SUMMARY

- Reviewed the use of models and modeling in software engineering
- Requirements analysis as beginning of design
- Modeling for requirements analysis
- UML diagrams to represent requirements analysis models
- Created requirements analysis models

ASSIGNMENTS

- Quizzes
  - ~~Syllabus Quiz~~
  - ~~Ethics Quiz~~
  - Requirements Analysis Quiz
- Discussions
  - ~~Week 1~~
  - ~~Week 2~~
  - Week 3
- Discussions
  - ~~Students Introduction for Team Formation~~