Unit 1. Introduction

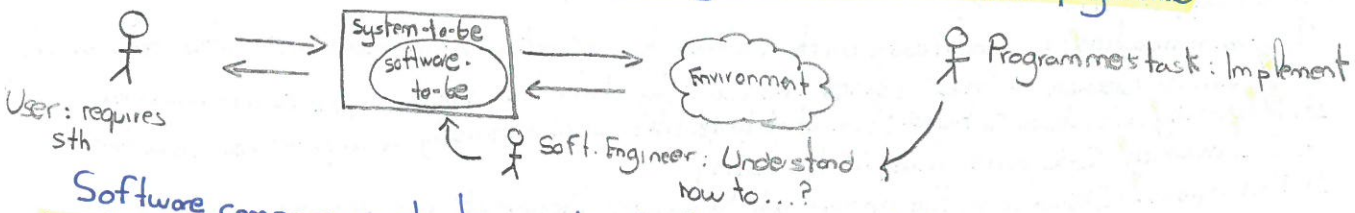
Software Engineering → Problem solving, modeling, knowledge acquisition, rationale management

- For problem solving we use:
  - Technique (methods):
    - Formal procedures
  - Methodologies:
    - Collection of techniques
  - Tools:
    - Instrument or automated systems

Software engineering: collection of techniques, methodologies and tools that help with the production of high-quality software with a given budget before a deadline, while change occurs

Scientific vs Engineer: Software Engineer works in multiple applications, has only 3 months, while changes occurs

"Software Engineering is the bridge between the customer and programmer



Software compasses instructions, data structure and documentation. It costs more to maintain than does to develop. It has some applications:

- System. software, application software, scientific, webApps, AI...

It also has new categories:

- Open world computing, netsourcing, data mining & open source

Software Engineering formal definitions:

- Seminal definition: Establishment and use of sound engineering principles in order to obtain economically sofware that is reliable and works efficiently on real machines.
- IEEE definition: Systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software

FAQs about software engineering:

What is software?: Computer programs, data structures and associated documentation

What are the attributes of good software?: Should deliver the required functionality and performance, should be maintainable, dependable and usable

What is software engineering?: Engineering disciple, concered with all aspects of software production

Difference between software engineering and computer science?: CS focuses on theory and fundamentals SE is concerned with practicalities of developing and delivering useful software

Difference between software engineering and system engineering?: SySEs concerned with all aspects of computer-based systems develop including hardware and software. SE is pot of this more general process

Essential attributes of good software

Maintainability: written in a way so it can evolve to meet the changing needs of customers

Dependability and security: Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access

Efficiency: Should not make wasteful use of system resources such as memory and processor cycles This includes responsiveness, processing time, memory utilisation...

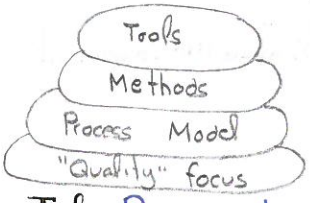
Acceptability: Must be acceptable to the type of uses for which it is designed. Understable and usable.

ISO 9126 software quality factors:

Functionality, reliability, usability, efficiency, maintainability and portability

Methodology: method + techniques

Layered technology



Quality focus: Fosters a continuous process improvement culture
Process layer: Defines a framework with activities for effective delivery of software engineering technology
Method layer: Technical how-to's for building software. It encompasses many tasks including communication, testing and support.
Tools: Provide automated or semi-automated support for the process and methods

Software process

A collection of activities, actions and tasks performed when a work product is to be created.
An appropriate set of work actions and tasks, in purpose to deliver software in a timely manner and with sufficient quality

Five activities of a generic process framework: Can be used to all software development regardless of the application domain, size of project, complexity of the efforts... Some of the following activities are applied iteratively

1. Communication: Communicate with customer to understand objectives and gather requirements
2. Planning: Creates a "map" defines the work by describing the tasks, risks and resources
3. Modeling: Create a "sketch", what it looks like architecturally, how parts fit together.
4. Construction: Code generation and the testing
5. Deployment: Delivered to the customer who evaluates the products and provides feedback.

We can complement those frameworks with even more like risk management, technical reviews, reusability management, tracking and control...
The process should be agile and adaptable, making helps for other projects.
George Polya outlines the essence of problem solving, suggests:

1. Understand the problem
2. Plan a solution
3. Carry out the plan
4. Examine the result for accuracy

CASE (Computer-Aided Software Engineering) Tools: Software used to support software process activities by automating some process activities and providing information. Used in every phase/workflow

- Case tool: such a design editor or a program debugger, used to support an activity
- Case workbench: an integrated set of CASE tools that work together to support a major process activity such as software design or configuration management

↳ Benefits:

- Improve software quality
- Reduction of time and effort

- Upper-CASE tools (Front-end-tools)
  - Things related to those things done early in development
  - Assist developer during requirements, analysis and design workflows or activities
- Lower-CASE tools (back-end tools)
  - Assist with implementation, testing and maintenance workflows or activities
  - Database generation and compilers
- Integrated CASE tools (I case)
  - Provide support for the full life cycle

Challenges of SE?
↳ Deal with legacy systems + Heterogeneity + Delivery deadlines

Areas covered by SE.
Project planning
Cost evaluation
Project management
Design
Testing
Maintenance
...

Software Artefacts
Requirement Specification
Source Code
Installation Instructions
User manuals
...

Unit 2. Requirements engineering

Problem domain: Context for requirements. High abstraction level, must answer the "WHAT?". Analysis:
- **Basic tasks**: Identification of needs and goals, negotiation, modelling and validation
- **Participants**: Users (clients) ⇒ Know the problem and communicates the needs
  Analysts (Requirements Engineer) ⇒ In charge to understand the problem and describe new reality

He is the link between clients and developers. He should hold some skills such as communication, conflict mediator, be in alert for new technologies...

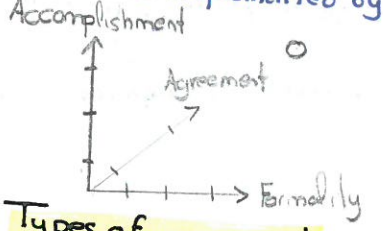
Requirements Engineering: Requirements for a product are defined, managed and tested systematically
Essencial to understand the requirements before trying to solve it. It establishes a solid base for design and construction, to improve the probability of meeting customer needs

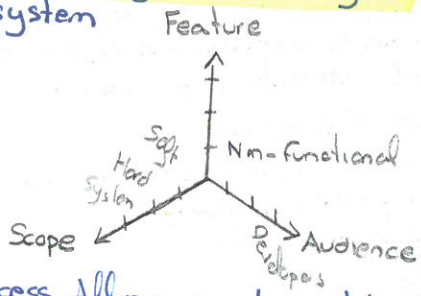
NIST (National Institute of Standards and Technology)
70% defects in the specification phase
30% in the technical solution
Only 5% fixed in the specification phase
95% After delivery, increasing 22 times the cost

Most common problems: Incomplete requirements, lack of user involvement or lack of resources.

What are "Requirements"?
↳ Capturing the purpose of a system, an expression of the ideas to be emboided in the system
According to IEEE: "a condition or capability needed by a user to solve a problem"

What is "Requirements Engineering"?
↳ The activity of development, elicitation, specification, analysis and management of the stakeholder requirements. Identify the purpose of a software system and the context
Captures real world needs of stakeholders affected by a software system and expresses them as artifacts that can be implemented by a computing system





Types of requirements
- **Goal**: Objective that guides the RE process. All requirements must be verifiable
- **Functional requirement**: Defines functions of the system. What should do?
- **Non functional requirement**: What the system must fulfil. Some sub-categories:
  - **Performance requirement**: system properties such as capacity, usability...
  - **Design contrains**: how the server should be designed and built. Lenguage, documentation...
  - **Commercial contrains**: such as development time frame and costs
- **User requirement**: is a desire goal that a user expect the system to achieve
- **Application domain requirement**: derived from bussiness practices within a given industrial sector.
- **System requirements**: for the system to be built, being this multidisciplinary
- **Software requirements**: derived from the system, being also the hardware part of the enviroment

Functional requirements
Inputs should accept
outputs should produce
data should store
computations should perform
Timing and synchronization

Non-Functional requirements
Also called quality requirements. If they are not met, the system is useless. Three main categories
- Performance requirements: usability, efficiency, reliability, maintainability, reusability and security
- Design constraints: enviroment and technology
- Commercial constaints: project plan and development methods

Characteristics of a good requirement:
Clear and unambiguas
Correct
Understandable
Verifiable
Complete
Complete
Consistent
Traceable
Modifiable

Requirements Engineering: General Activities

- Elicitation: Requirements are discovered through consultation with stakeholders
- Analysis and negotiation: Requirements are analyzed and conflicts resolved through negotiation
- Specification: A precise requirements document is produced
- Validation: Requirements document is checked for consistency and completeness

" " : Process Models

- Pohl model: Iterative and incremental. Stakeholders involved. For large companies
- Spral model: Like a real spral. Processes like planning, risk, analysis, design, test and evaluation
- SWEBOK model: Systematic and successful. A circle divided in 4 stages

There are three iteration cycles:

1. Elicitation - Analysis - Validation cycle:
2. Elicitation - Analysis cycle: For shortcomings and conflicts
3. Requirements Engineering Cycle - Rest of development: Possible to go back to any of the requirements

REM Activities:

- Elicitation: Between users and analysts
  Goals → know the problem domain, discover the real needs, reach agreements (Requirements - C)
- Analysis
  Goals → Detect Conflicts in Requirements-C, Deep problem domain, establish the cost for design (the rat)
- Validation: Confirm that Requirements-C match with the necessities
  Goals → Make sure requirements describe desired product, validate by users

Common Problems: Of scope, of understanding, of volatility

Techniques: Analysis of existing system, interviews, brainstorming, JAD, prototyping; use cases

JAD = (Joint Application Design): More intense brainstorming

Tenets ↳ Effective use of group dynamics, use of visual aids, defined process, standardized

Activities ↳ Preparation, working session, summary

Use Cases: Sequence of interactions between a system and external actors to complete an action

Requirements Elicitation Methodology

Task 1: Get info about the problem and the current system
Task 2: Prepare and perform elicitation
Task 3: Identify the system goals
Task 4: Identify IR
Task 5: Identify FR
Task 6: Identify NFR
(Can be done in parallel)
Task 7: Prioritize Goals and Requirements

The software Requirements Specification (SRS) → Traceable Matrix

Requirements Analysis: Discover conflicts in Requirements-C, deepen knowledge and completition and formality of the RE process

Requirements Analysis Methodology

Task 1: Analyse IR
↳ Goal: discover conflicts
Task 2: Analyse FR
↳ Goal: Construct behavioural models
Task 3: Analyse NFR
↳ Goal: Detect conflicts (technical impossibility. Choose among different features

Outcomes: Structural models + possible conflicts

System Architecture Design ⇒ Iterations between the rest of the process and the RE
Task 4: Construct Prototypes (In Elicitation or Validation)

Requirements Verification and Validation

Validation: Checks that the right product is being built
Verification: Checks that the product is being built right

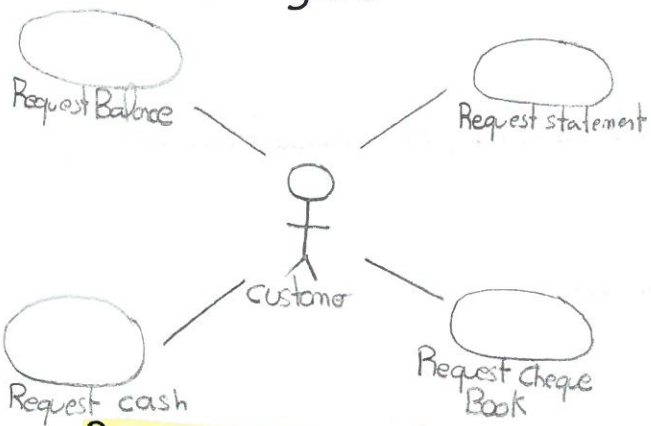
Requirements Validation Methodology

Task 1: Validate IR and FR
↳ Technique: walk through
Task 2: Validate NFR
↳ Reviewed by users. Cannot be integrated without a walk through
Task 3: Close Requirements Specification
↳ If no conflicts detected

Unit3. Requirements modelling. Use Cases

Use Cases: Describes the interaction of the user with the system (not the internal computation). Should cover the full sequence of steps from the beginning task to the end. Only include Actors ⇄ Computer. Every use-case describes a step-by-step sequence of operations, iterations and events.

Use Case Diagrams



What does the oval represent?

A unique use-case. It must be initiated by Actors. These are the high-level functionality. Each use-case is documented with a detailed description

Actors

External entity outside the domain of the system. Must be somebody/something that initiates an interaction with measurable benefit

Scenarios

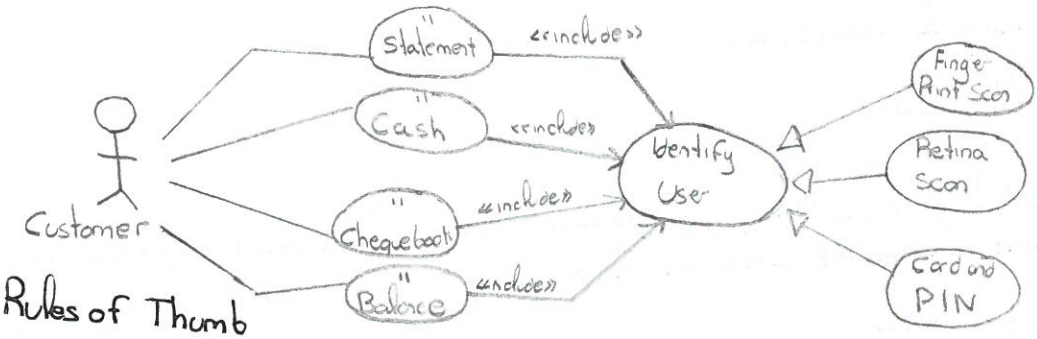
Primary and Secondary

- **Primary scenario:** Normal course of events
- **Secondary scenario:** Alternative/exceptional course of events
  - **Alternative:** Meets the intent but with a different sequence of steps
  - **Exceptional:** Differs from the norm and cases already covered

Alternative scenarios are NOT errors, they may reflect important business logic.
A scenario captures the many different possible interactions and outcome. All use-cases binds together a set of scenarios. Scenarios must not include Graphical User Interface or technical elements

Relationships in Use-Case Diagrams

- **Association:** Communication between an actor and a use case
  " ——— "
- **Generalization:** Relationship between general use case/actor and a special use case/actor (for special alternatives)
  " ——▷ " "kind of..."
- **Include:** When a chunk of behaviour is similar accross more than one case. (Algo obligatorio de hacer)
- **Extend:** May add behaviour to the base case.



Rules of Thumb

Don't over specify behaviour — Don't under specify behaviour — Avoid functional decomposition
Use case don't model order — Don't want too many use cases

Benefits of Use Case-Based Software Development

Can help to scope of the system — Used to develop and validate requirements. Used to plan development

Unit 4.0. Introduction to Software Modelling with UML

What is modeling?

↳ Building an abstraction of reality, simplifications because they ignore irrelevant details

↳ Aims to help us to visualize a system, give us templates and make the structure of a system

What is a Model?

↳ Simplification of the reality

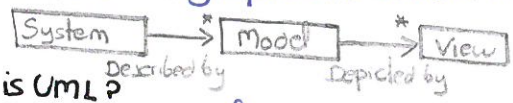
Why model software?

It gets more complex increasingly, code is not easily to understand, to communicate the idea easily and give us a big picture of the project

Model: Abstraction describing a subset of a system

View: Depicts selected aspects of a model

Notation: Set of graphical or textual rules for depicting views



What is UML?

UML = Unified Modeling Language, used for specifying, visualizing, constructing and documenting software systems. Increase communication of products to customers and developers

UML Main Features

UML is a graphical notation. It's neither a process model nor a methodology, useful for non-software systems

Diagrams in UML: General Overview

- **Use Case Diagrams**
  Describe behaviar of the target system, with the help of FR (scenarios)
- **Class Diagrams**
  Classes of the system, their interrelationships and the operat+ attributes of classes. Static view of system
- **Activity Diagrams**
  What happens in a workflow, activities done in parallel. Dynamic view + Model order
- **State diagram**
  Shows transitions. Dynamic and model behaviar
- **Sequence diagram**
  Time ordering
- **Communication Diagrams**
  Structural organization that send and receive messages
- **Object Diagrams**
  Static view of a system, but from a prototypical cases
- **Component Diagrams**
  Dependencies among components. Static view
- **Deployment Diagrams**
  Run-time processing. Static view

Stereotypes

↳ Extension of the vocabulary of the UML, creating similar blocks as the ones already created

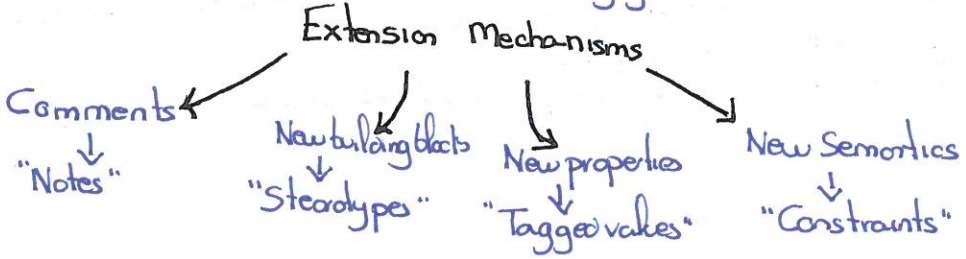
Define a new model element in terms of another model element

Tagged Values

↳ Define new element properties

Constraint

↳ Extension of semantics of a UML, allowing ya to create new rules



Unit 4.1. Object Oriented Analysis With UML: Class Diagram

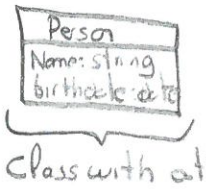
Aim to describe objects (a concept, abstraction). All objectives have identity and are distinguishable

Classes

A class describes a group of objects with the same properties (attributes), behaviour (operation), kind of relationships... With a common semantic

Value: Piece of data

Attribute: Named property of a class that describes a value held by each object of the class



* Don't list object identifiers: they are implicit in models*

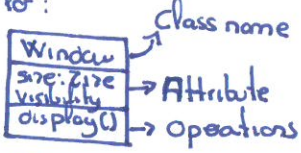
Polymorphic: Same operation may apply to many different classes

A class diagram is a diagram describing the structure of a system

Essentials in UML Class Diagram → Classes, Attributes, Operations → Relationships

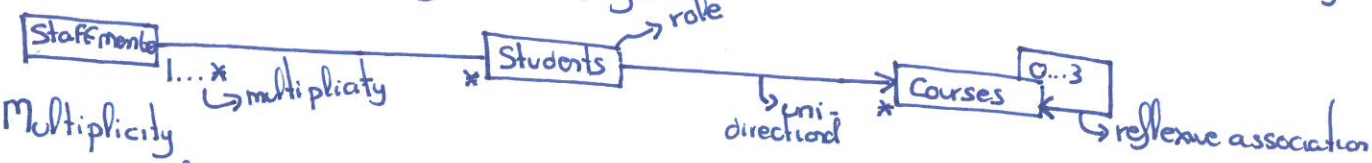
Class: Describes a set of objects having similar:

- Attributes
- Operations
- Relationship with other classes



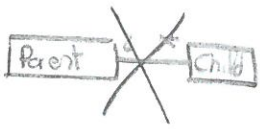
Links and associations: Denoted by a line. May have name if ambiguous.

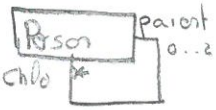
An association between two classes indicates that objects at one end of an association "recognize" objects at the other end and may send messages



Multiplicity

Exactly one	1
Zero or more (unlimited)	*(0...*)
One or more	1...*
Zero or one	0...1
Specified range	2,...4
Multiple	2,4...,6,8





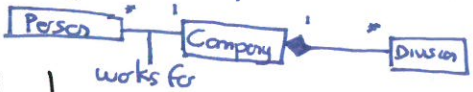
Aggregation

Special form of association between a whole-part and its parts

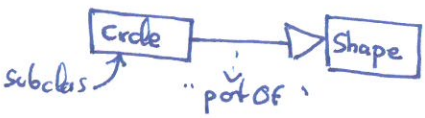
"is a part of..."

Composition

Strong aggregation. Composition must be one or zero in the whole



Generalization



4.2. Activity Diagram

- What is an AD?

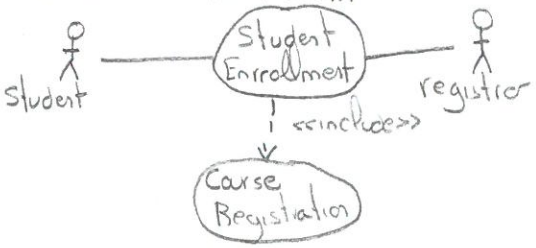
Represents dynamic (behavioral) view of a system
Model the logic captured by a single use-case or usage scenario
↳ + the flow across use cases or a particular use case
They include activities, decision points, transitions...

Use them for:
Analysing Use Cases
Understanding work flows
Dealing with multi-threaded apps

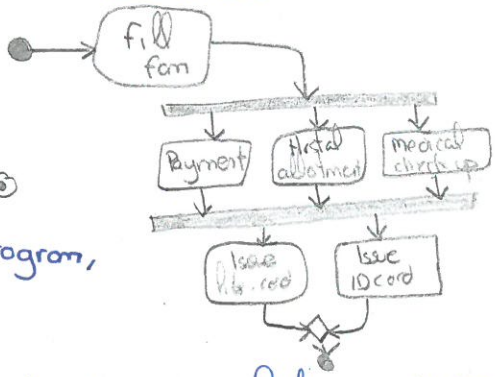
Don't use them for:
See how objects collaborate
See how a object behaves over its lifetime

Example

Student Enrollment in Uni



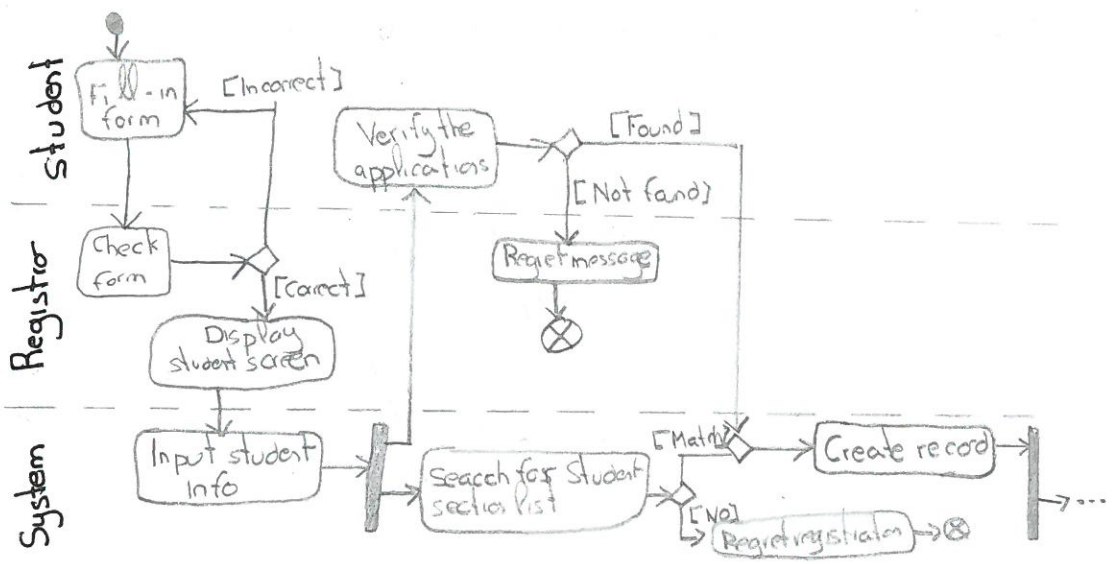
→ Activities
→ Fill form
→ Pay fee
→ Medical fom
→ Hostel fom
→ Issue library card
→ Issue ID card



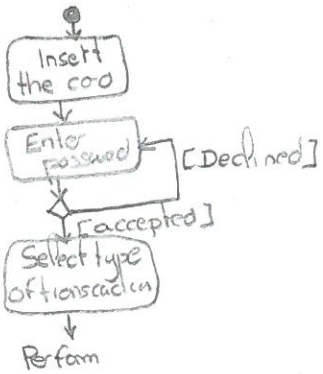
Initial node
Final node : Can have zero or more
Activity : Not neccessarily a program, may be a manual thing also
Flow/Edge : Arrows in the diagram
Fork : Black bar with one flow going into it and several leaving it. Beginning of parallel activities
Join : Black bar with several flows entering it and one leaving it. End of parallel activities. All must be done
Merge : Diamond with several entering and one leaving. It can be executed two or more times
Decision/Branch : outflows of yes/no state
Flow final : Process stops at this point

Branching
↳ Specifies alternate paths taken based on Boolean expressions

Activity Diagram of SE with Swim Lanes



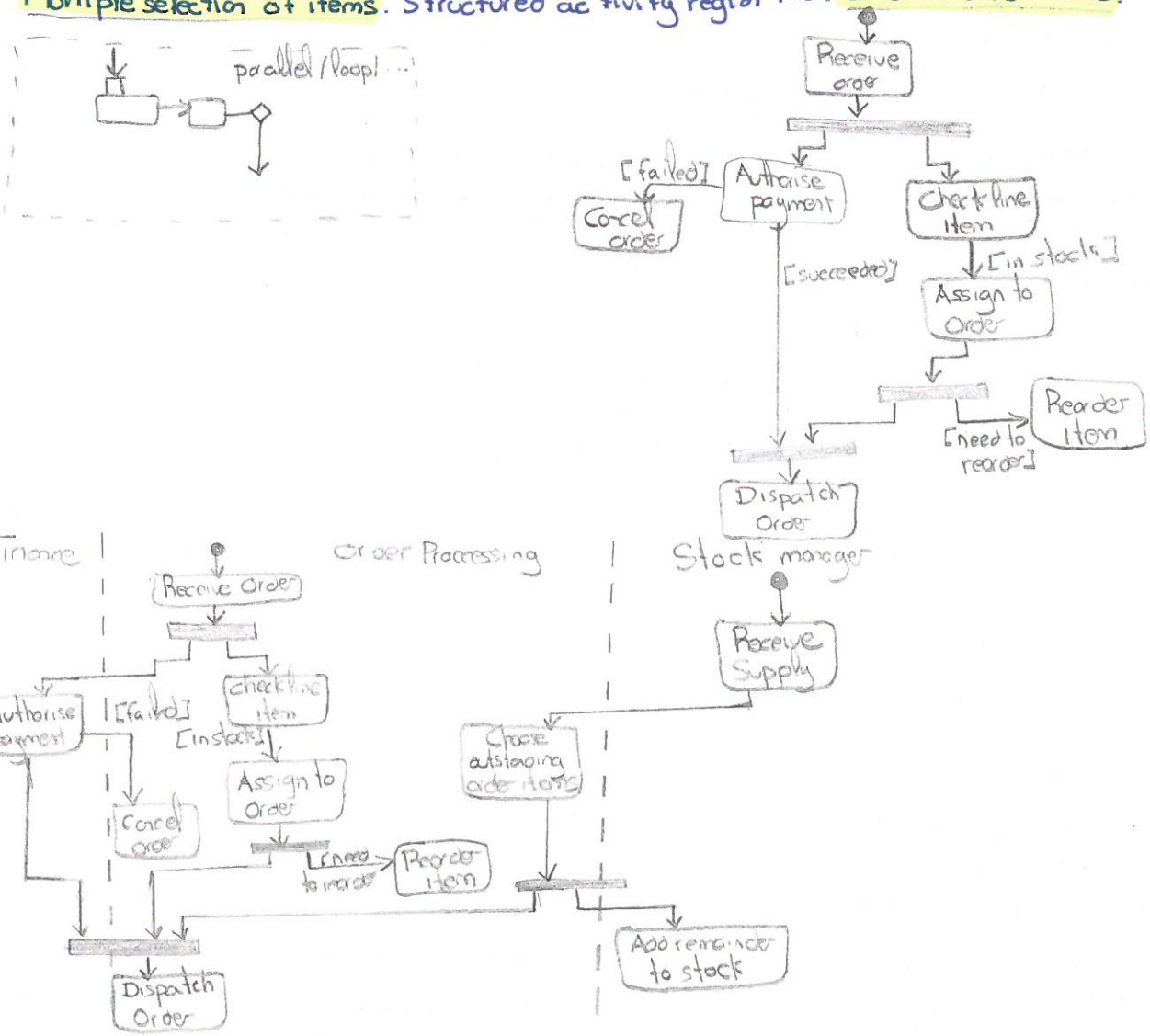
Member arrives ATM



Object VS Object Flow

Expansion region

Multiple selection of items. Structured activity region that executes several times.



Importance of Activity Diagram

- Can depict a model in several ways
- Can depict "basic course of actions" as well as "detailed courses"
- Can cross several use cases, or a small part of one
- Employed in business process modeling. Initial states of requirement analyses and specification
- Used to develop interaction diagrams which helps to allocate activities to classes

5.1 State Diagrams

- Solution Domain Models
  - ↳ Getting closer to the software implementation
  - They should answer HOW, not just What
- Design Class, what's?
  - Supplements info of the problem domain classes detected during the Analysis stage
  - Detects other possible classes to perform a better implementation of the system

  Basic Features: Name, attributes, operations, relations

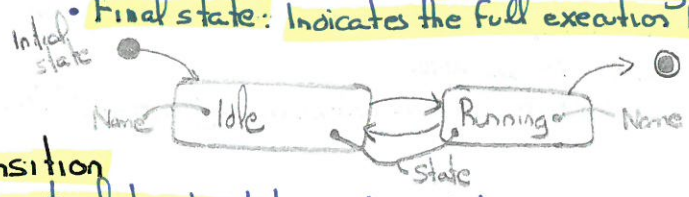
  Advanced Features: Visibility, multiplicity, polymorphims...
- State Diagrams:
  - Models the dynamic aspects of the system by showing the flow of control from state to state for a particular class
  - Every class will have its own state diagram
    - → Only classes with interesting or complex internal behavior will be modeled with state diagram
    - → State diagrams are not shared among classes

Flow of control from activity to activity ← Activity Diagram ⟷ State Diagram → Flow of control from event to event

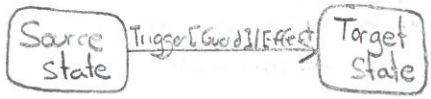
  - They show how an object reacts to external events or conditions during its lifetime

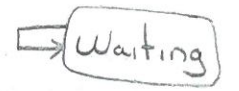
Term and concepts

- State
  - Condition during the life of an object in which satisfies some condition, performs an activity, waits...
  - May include: Name, entry/exit actions, activities, substates...
    - Initial state: Indicates the initial starting state for a state machine
    - Final state: Indicates the full execution has completed

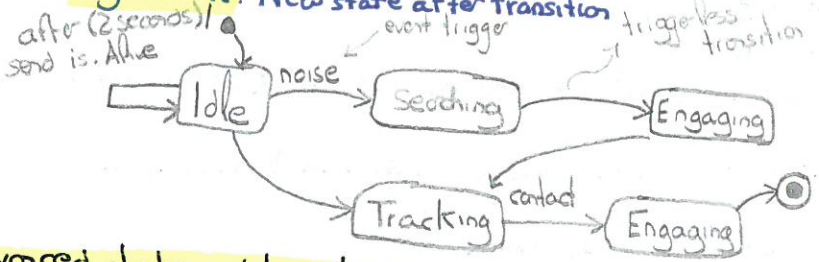


- Transition
  - Direct relationship between two states
  - Contains five parts:
    - Source state: Before transition
    - Event trigger: External stimulus
    - Guard condition: Condition that must be satisfied to continue
    - Action: (Effect)
    - Target state: New state after transition

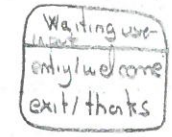


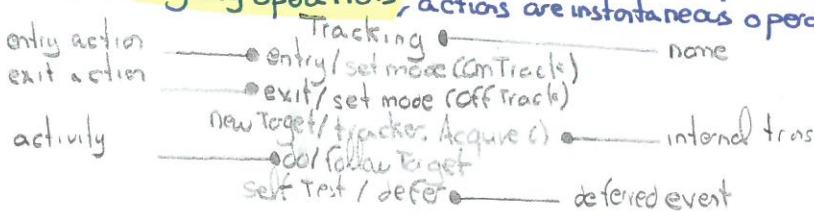


Event[Condition]/Action

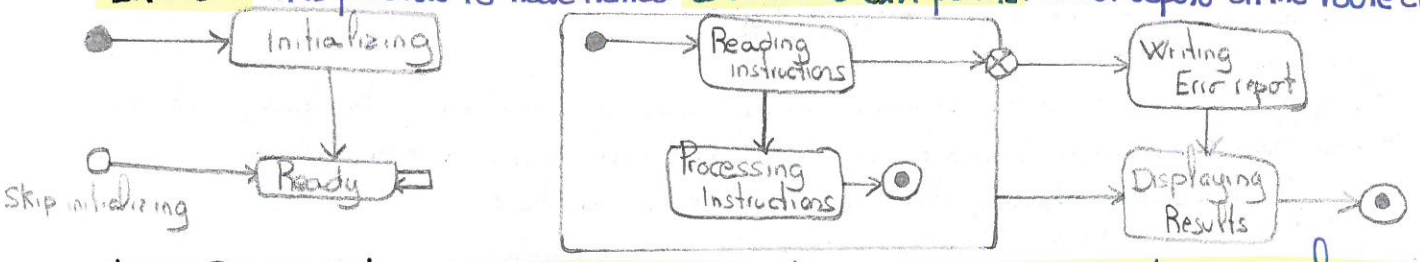


- Advanced states and transitions
  - Entry action: Upon each entry, a specified action is automatically executed
  - Exit action: Prior to leaving a state, " " " " " "
  - Internal transitions: The handling of an event without leaving the current state
    - → Used to avoid a states entry and exit actions
  - Activities: Ongoing work that an object performs while in a particular state. The work finishes when leaves
    - → Activities are ongoing operations; actions are instantaneous operations



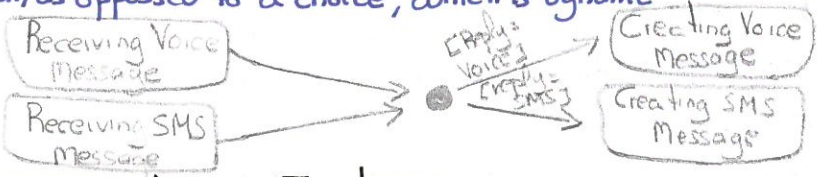


- State machine diagrams
  - Entry Point: If you don't find necessary to perform the initial state, you can skip it to avoid entering the normal initial state of the submachine
  - Exit Point: It is possible to have named alternative exit points. It can depend on the route chosen

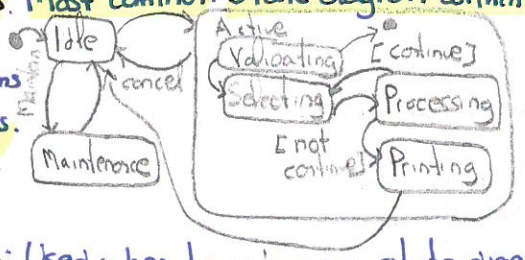


  - Choice Pseudo-State: Shown as a diamond with one arrow arriving and two or more leaving

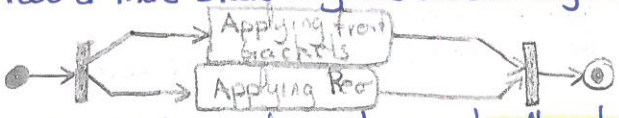
  - Junction Pseudo-state: To chain together multiple transitions. Can have one or more incoming, and one or more outgoing. A guard can be applied to each transition. They are semantic free. Junction which splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice, which is dynamic



- Advanced States and Transitions
  - Simple states: With no substates
  - Composite states: With substates
    - Substate: State nested inside another state. Allows to see different levels of abstraction. May be sequential or concurrent. May be nested to any level
    - Sequential Substates: Most common. State diagram within a single state. The containing state becomes an abstract state. Using this simplifies state diagrams since reduces the number of transition lines. May have at most one initial state and one final state



    - Concurrent Substates: Used when two or more state diagrams are executing in the same object. Allows an object to be in multiply states simultaneously. Must start and end simultaneously. If one finishes first, it must wait for the other



  - History states: Allows an object to remember which substate was last active, so it will reenter directly

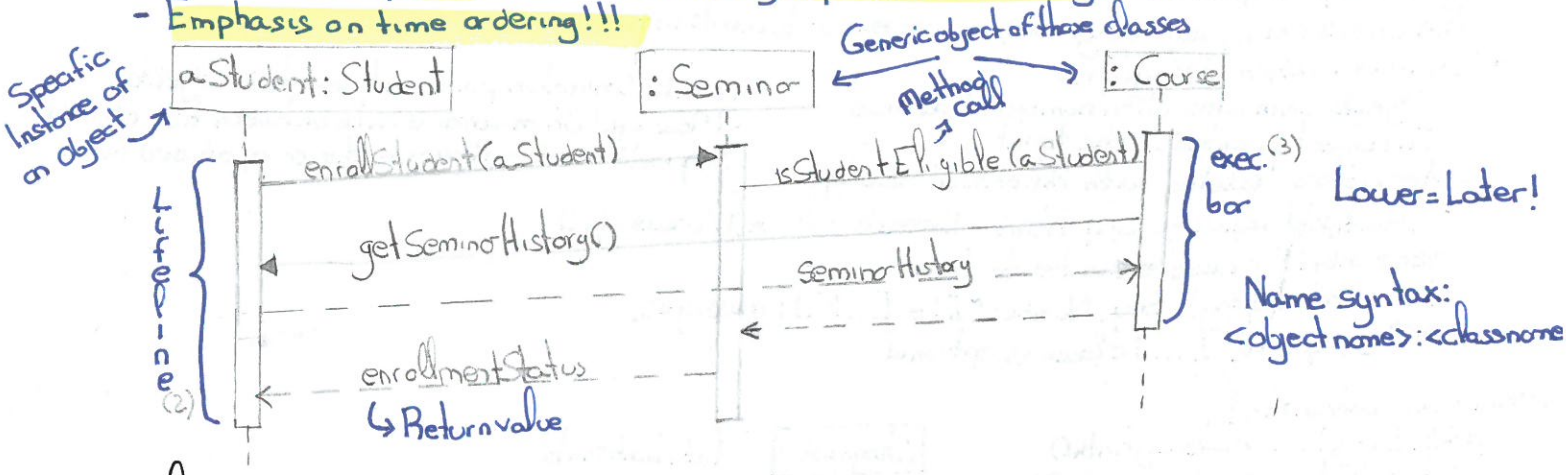
- Event
  Is the specification of a significant occurrence that has a location in time and space
- Activity
  Is an ongoing non-atomic execution within a state machine
- Action
  Atomic execution within a state machine that results in a change of state or the return of a value

5.2. Interaction Diagrams

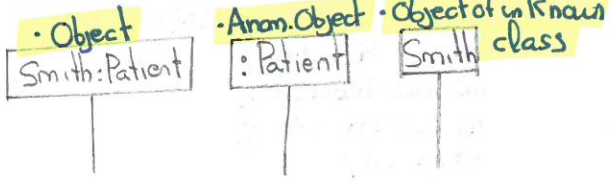
→ Dynamic aspects of the system
- Messages moving among objects/classes
- Flow of control among objects
- Sequences of events

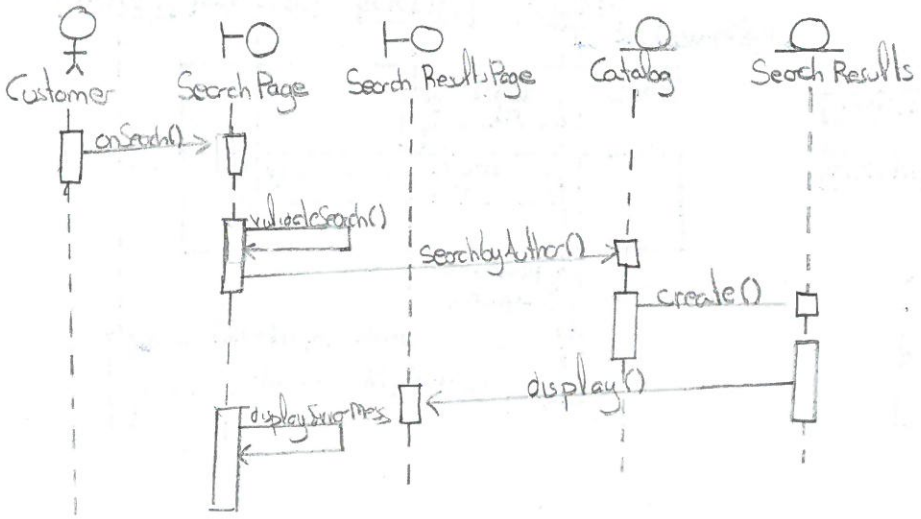
• Interaction diagrams
Set of objects or roles and the messages that can be passed among them
- Sequence Diagrams: Emphasize time ordering. Illustrate object interactions arranged in time sequence
- Communication Diagrams: Emphasize structural ordering. Illustrate object interactions organized around the objects and their links to each other
  * Commu. Diagrams emphasize the structural organization, while sequence diagrams on time ordering
  * Commu. Diagrams explicitly show object linkages, while links are implied in Sequence Diagrams

• Sequence Diagrams
- Describe the flow of messages, events, actions between objects
- Show time sequences that are not easily depicted in other diagrams
- Emphasis on time ordering!!!



- Elements:
  - Method call lines:
    Must be horizontal
    "Lower equals Later"
  - Arrows
    Synchronous ——▶, waiting for a return value
    Asynchronous ——▷, not waiting for a response
    Return call ◁- - - -
  - Creation
    Arrow with "new" written above it
  - deletion
    An X at the bottom of object's lifeline
- Lifeline (2)
  Dotted vertical line
- Execution bar (3)
  Bar around lifeline when code is running

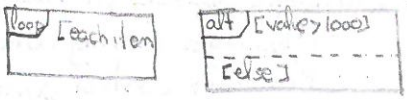


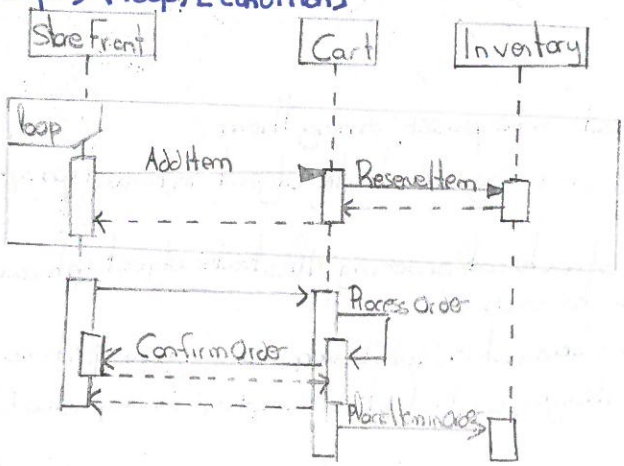


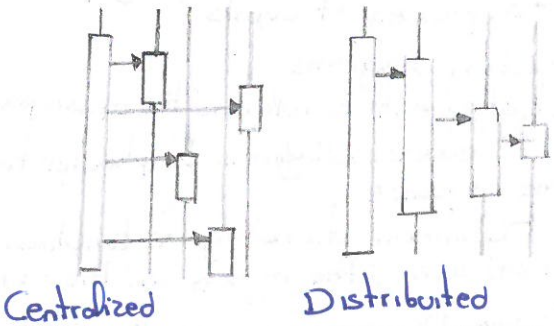
- Frame

Box around part of a sequence diagram to indicate selection or loop

- If → (opt) [condition]
- If/else → (alt) [condition], separated by horizontal dashed line
- Loop → (loop) [condition]







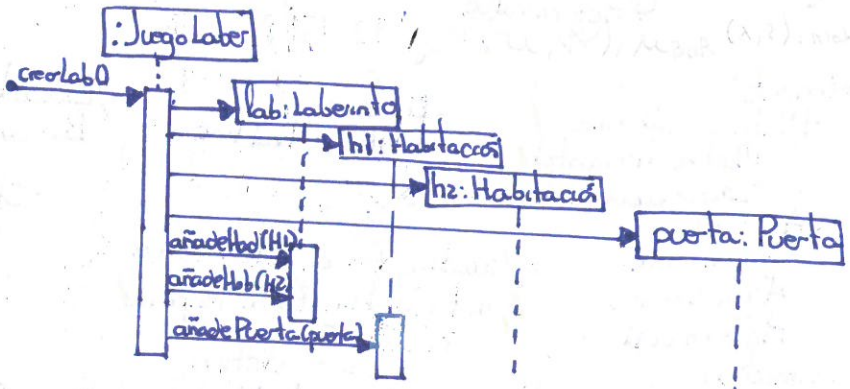
- Why not coding?

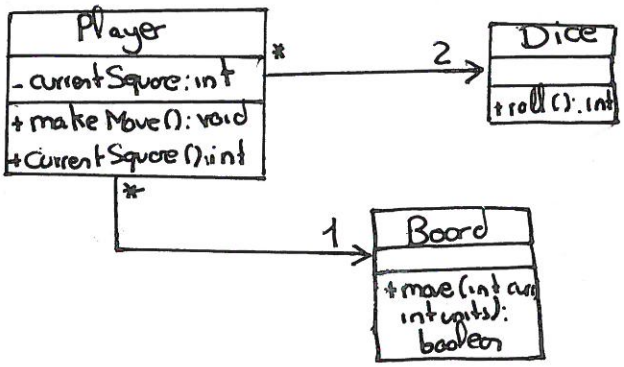
A good seq. diagram is still a bit above the level of the real code. They can be implemented in different languages)
Non coders can do seq. diagrams, and are easier to work in teams

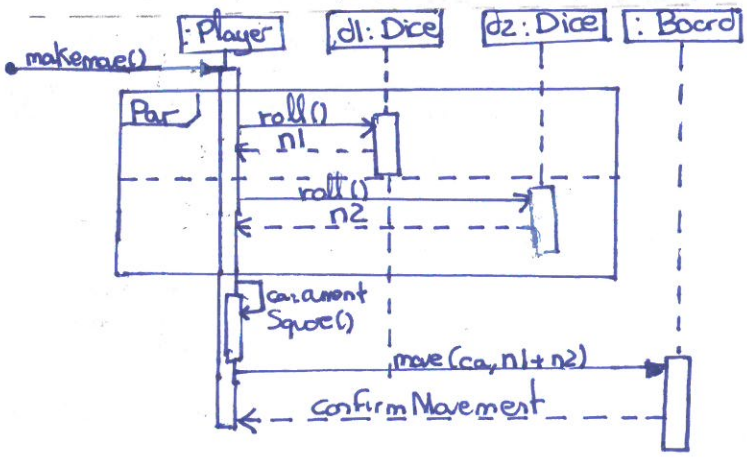
- Communication Diagrams
  - Objects connected with numbered arrows
  - The object pointed is the target
  - Arrows are labeled with the passed message
    - Conditional message: Seq. Number [variable = value]: message()
    Sent only if clause evaluates to true
    - Iteration (looping): Seq. Number * [i = 1...N]: message()
    "*" is required; [...] clause is optional

- Link: Connection path between two objects
- Message: On an arrowed line between two objects
- Sequence number: Represents the order of used flows

```
public class Labyrinth {
    public Labyrinth createLabyrinth()
    Labyrinth lab = new Labyrinth();
        Room h1 = new Room();
        Room h2 = new Room();
        Door door = new Door(h1,h2);
        lab.addRoom(h1);
        lab.addRoom(h2);
        h1.addDoor(door);
        return lab;
    }
}
```





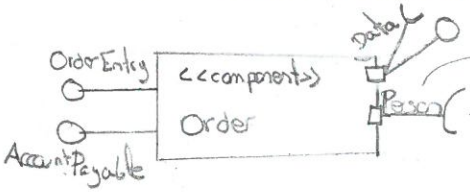


5.3. Implementation & Deployment Model
(Component)

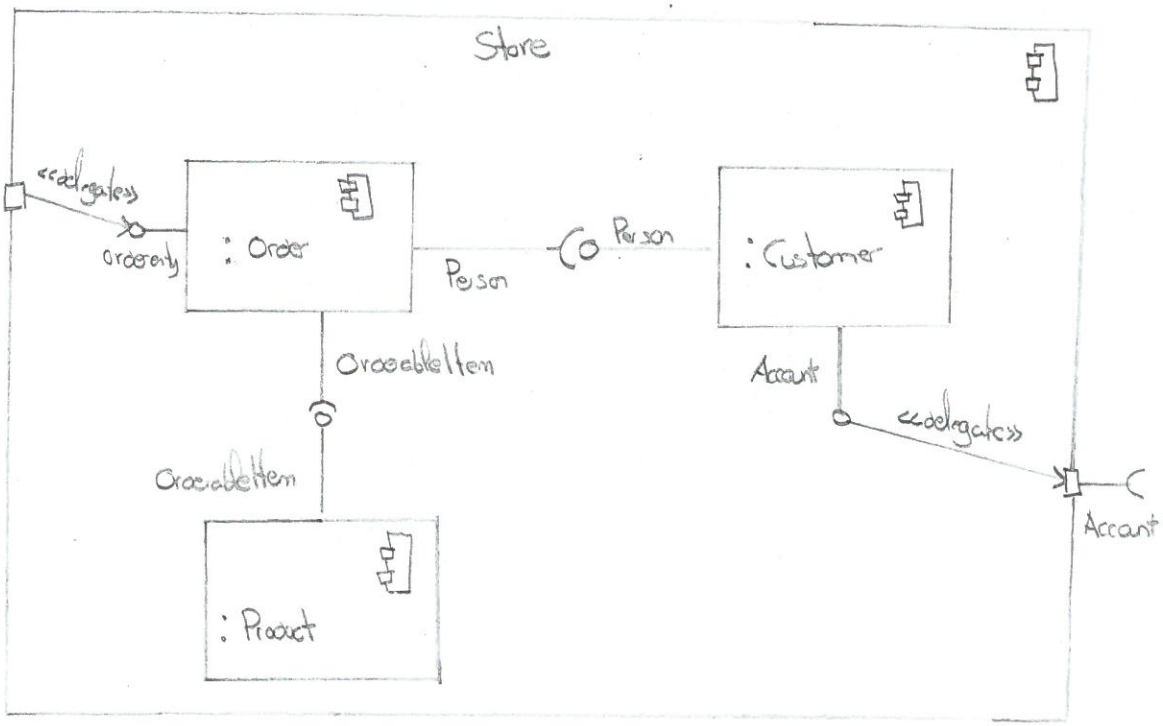
- **Component Diagram**

To show the structural relationships between the components of a system. Model the high-level software components. Dependency of softwares into other softwares

It can be shown with the components provided and required



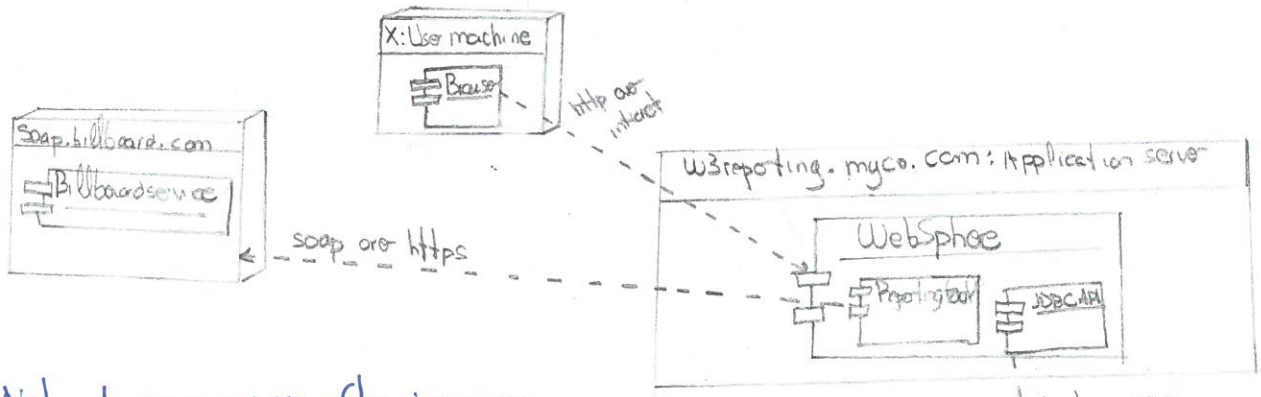
- Port: Feature that specifies a distinct interaction point between the classifier and its environment. Small squares on the sides of classifier. Can be named. Unidirectional or bi-directional



- **Deployment Diagram**

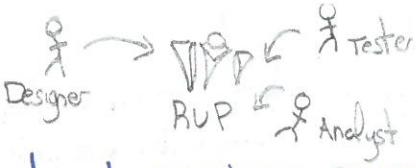
To show how a system will be physically deployed in the hardware environment. To show where are they and how are going to be communicated

- Node: Represents either a physical machine or a virtual machine node. To draw it, a 3D cube with the name
- Artifact: Actual implementation of a component.



Network diagrams are often drawn using software-based drawing tools

6.1 Unified Process



- Rational Unified Process (RUP)
  - Team-Unifying Approach: The RUP unifies a software team by providing a common view of the development process and a shared vision of a common goal

  Rup has four phases:
  1. Inception: Define the scope of project
  2. Elaboration: Plan project, specify features, baseline architecture
  3. Construction: Build the product
  4. Transition: Transition the product into end user community

  Iteration: Each phase into a complete development loop.
  Risks mitigated, reuse, better acoall qual.

- Inception → Goals

  Establishing the project's software scope and conditions:
  - What is intended in the product and what is not.
  - Acceptance criteria

  Discriminating
  - The critical use cases of the system
  - Primary scenarios with major design

  Defining at least one candidate architecture

  Estimating
  - The overall cost, the schedule and more detailed phases. Potential risks...

- Inception → Essential Activities

  Formulating the scope of project, planning and preparing a bussines case, choosing a candidate architecture...

- Inception → Artifacts
  - Vision: Projects core requirement, key features...
  - Glossary: Defines important terms used by the project
  - Bussines Case: Provides info to check if the project is worth or not
  - Software Development Plan: All info required to manage the project
  - Use-case model: Model of the system's intended functions and its enviroment

- Elaboration → Goals

  To ensure stability of:
  - Architecture, requirements and plans

  To be able to predictably determine:
  - Cost and schedule + All risks

  To establish a baseline architecture

  To produce a prototype...

- Elaboration → Activities

  Defining, validating the baseline architecture
  Refining vision, creating detailed plans

- Elaboration → Artifacts
  - Software Architecture Document: Provides a comprehensive architectural overview of the system. With diff. views
  - Prototypes: One or more executable architectural prototypes to explore critical functionality
  - Design model: Object model describing the realization of use cases, and serves as an abstraction of the model
  - Data model: Subset of the implementation model which describes the logical and physical representation of persistent data in the system

  Testing mechanisms and refining Iterations artifacts.

- Construction → Goals
  Completing the analysis, design, development and testing of all required functionality
  Achieving useful versions, with adequate quality
  To the decide if the software, the sites, and the uses are all ready for the application to be deployed
  Minimize development costs by optimizing resources
  Archieve some degree of parallelism in the team work
- Construction → Activities
  Resource management and optimization
  Complete component development and testing
  Assesments against the vision
- Construction → Artifacts
  - The system: Executable system itself
  - Training materials: Used in programs to assist the end-users with product use
  Testing results and refining previus Iterations artifacts
- Transition → Goals
  Beta testing to validate new system
  Training of users and maintainers
  Roll-out of the marketing and sales forces
  Bux fixing...
  Achieving user self-supportability + stakeholder conarrance
- Transition → Activities
  Executing deployment plans
  Finalizing end-user support material
  Getting user feedback
- Transition → Artifacts
  - Product
  - Release notes: Identify changes and known bugs in a version avaible for use
  - Installation artifacts: Instructions to install the product
  - End user Support material: For learning, using and operating
  Testing results of previus Iteration's artifacts

- Process Work flows
  Set of activities performed by various roles, describing a meaningful sequence of activities, showing interactions
  3 key elements: Roles "who", Activities "how", Artifacts "what"
  - Roles: Individual may play on the project, responsible for producing artifacts and *different from roles*
  - Activities: Tasks performed by people representing particular roles to produce artifacts
  - Artifacts: A piece of info produced or used by a process. It can be a model, a model element or a document
- Business Modelling
  Understand target, problems, structure. Ensure customers and end users
  We will need artifacts like business vision, business architecture Document, Bussiness Glossary...
- Requirements
  To Establish agreement with the customers. To provide a basis. To get a better understanding
  Artifacts needed such as glossary, use-case model, vision document and user interface prototype
- Analysis and design
  Trans form requirements into a design.
  Evolve a robust architecture
  Artifacts: Design and data model
- Test
  Verify integration between objects,
  check if all requirements are completed.
  Artifacts: Test plan, results, ideas list...
- Implementation
  Define organization of the code. Integrate results into an executable system
  Artifacts: Build, implementation model
- Deployment
  Provide a stan installation, software over internet...
  Artifacts: Deployment plan, end-user support material

AD → Represent dynamic (behavioural) view of a use case or usage scenario. And the flow between them + Analysis stage

- → Initial node
- → Final node
- Activity
- → Flow final

Fork — Beginning of parallel activities
Join — End of parallel activities. All must be done
Merge → Executed two or more times
Decision / Branch → [yes] / [no] — Boolean answers

Expansion region: Group of items that repeats several times

Importance of AD
- Depict a model in several ways
- Detailed or basic
- Several use cases or portofore
- To develop diagrams
- Requirement analysis and specification
- Interact.

Domain models
↳ Closer to software implementation. Answer HOW, not just what

Basic features: Name, Attributes, Operations, Relations

Design Class
↳ Supplement info detected during Analysis stage

Advanced features: Visibility, Polymorphism

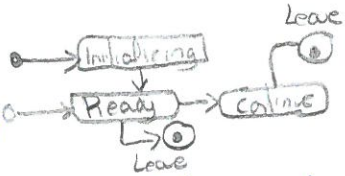
State diagram
↳ Dynamic flow from state to state for a particular class. Not shared among classes

Terms:
- State: Indicates which condition has been satisfied (activity, waits...)

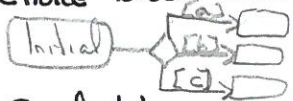
- Transition
  From state to state.
  5 parts. Source state, event trigger, Guard condition, Action, target state
  Event[Condition]/Action

- Advanced states and transitions
  + Entry/Exit actions
  + Internal transitions, to avoid
  → Activities

- State machine diagrams
  Entry point: To avoid the initial state
  Exit point: To have alternative leavings. Depends on the route chosen



- Choice Pseudo-State



- Junction Pseudo-State

To chain multiple transitions

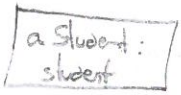
- Simple state: No substates
- Composited states: With substates (state nested inside another one). To see diff. levels of abstractions
  → Sequential substate: Most common. One inside another. To avoid too much transitions lines.
  At most has one initial and state and one final state
  → Concurrent substates: When two states are running at the same time. If one finishes first, waits for the other.
  ⇝ History states: Allows an object to remember which substate was last active

Flow Activities Activity AD ⇄ SD → Flow Event ↔ Event

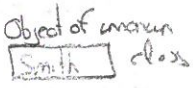
Sequential Diagrams
↳ Time ordering → Not easy in other diagrams
Links are implied
Flow of messages, actions or events

Communication Diagrams
↳ Structural orderings
Focus on links

Specific instance of an object



Generic object of those classes



Element

Synchronous ——▶ "Waiting for a response"
Asynchronous ——→ "Not waiting for a response"
Return call <- - -

Creation → with "new"

event Σ condition/action
Relationship Composition in Comp Dia
↳ Dependency relationship

Rup/workflow
These activities include both core and support
AD -> behav + Analysis
State D -> Always has initial state
Unified activated by -> use-case owner, level as in each case

Seq diagrams

"We can specify control structure though the use of Fragment frames"

Comp Diagrams -> struct. diagrams

Diagram in general cannot be control
Because entity class

Seq diag -> behav + design

Software Engineering I
Practice Assessment – 2nd Part

Academic Course 2023-2024

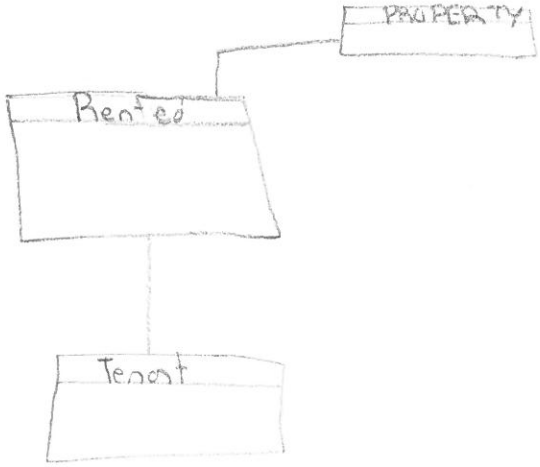
Consider the following extension of the practice coursework and create the following diagrams:

1) Model the new classes together with their corresponding relationships, that should be incorporated into the **Domain Class Diagram**. In the solution provided, include only the part of the Domain Diagram needed to model the new aspects, not the whole Diagram.

2) Model the **Sequence Diagram** corresponding to the Use Case "Register New Rent". Consider that the neighbour has previously logged into the system, so we already know the neighbour who starts the process of registering a new rental.

Many neighbours of the community have their flat rented to another person who makes use of all the services and facilities of the Community. We consider now that it is convenient that the neighbours could register these rents into the system and know, at all times, which flat is rented to whom. In case of problems with the tenant, it is necessary to know their data, which would be registered in the system database, and notify the owner of the corresponding flat.

We want the system to allow the neighbours to register new rents, indicating the rented flat, the person to whom it is rented (tenant), whose data should also be incorporated into the system and the start and end date of the rental. This resulted in the identification of a new Functional Requirement: Register new rent, which can be done by the neighbours directly.



Rented

notifyProblems()
getTenant()
getFlatInfo()

Flat

registerNewRent() ~~> E