CAT 1

Introduction to Software Engineering and Object Orientation

Alejandro Pérez Bueno

Mar 14, 2025

Table of Contents

Self-Responsibility Declaration

I certify that I have completed CAT1 completely individually and only with the help that the teaching staff of this subject considers appropriate, according to the instructions explained in the "Originality in the evaluation" section of the classroom. I understand that non-original work and/or the use of generative AI will mean that the submitted activity will not be marked and will automatically be assigned a grade of 0.

Module 1 Questions

Question 1

1. **Task:** Create an activity diagram in UML to represent the different execution flows of a use case.
   - **Role:** *Functional Analyst*. Unifies the different views of the domain in a single model that is clear, concise, and consistent. Activity diagrams are part of the modelling languages (UML) the functional analyst is expected to master.
   - **Activity:** *Modelling*, because this task involves creating a model of the system's behavior using UML.
2. **Task:** Define the layered structure of the system and how the layers will communicate with each other.
   - **Role:** *Architect*: responsible for defining the principal outline of a system design. Defining the layered structure is a key architectural decision.
   - **Activity:** *Modelling*. The architect uses models to represent the system's structure and communication pathways.
3. **Task:** Establish the project phases and estimate approximate start and end dates.
   - **Role:** *Project Manager*: organizes the project, planning, and ensuring compliance with objectives, including timelines.
   - **Activity:** *Project Management*: core part of project planning and falls under project management.
4. **Task:** Create the database for the project, including necessary tables and their relationships.
   - **Role:** *Programmer*. While database experts could be involved, programmers often handle database creation as part of the construction of the system.
   - **Activity:** *Construction and testing*, as creating the database is part of building the system and ensuring it functions correctly.
5. **Task:** Deliver the completed software system to the client.
   - **Role:** *Deployment Manager*. The deployment manager is responsible for packaging and delivering the software to the end users.
   - **Activity:** *Construction and testing*. Deployment is the final step in making the software available.

Question 2

Project 1: Gaming Entrepreneur

1. **Project Type (Section 3.2):** Type 2 (Clear Objective, Little-known Solution). The entrepreneur has a clear goal (launching a specific game), but lacks art and game design expertise. The reliance on demos and feedback indicates an exploratory approach to the solution.
2. **Development Method:** Lean and Agile Development: Agile is suitable for group 2 projects . The need for feedback and iterative improvements aligns perfectly with agile principles. Agile methodologies will allow the entrepreneur to adapt to user feedback and evolving design ideas, which is crucial

given her lack of expertise in art and design. The focus on demos and working software is also a key agile tenet .

Project 2: Hospital Emergency Management System

1. **Project Type (Section 3.2):** Type 2 (Clear Objective, Little-known Solution). The hospital has a clear objective (improving patient prioritization). However, they're open to new technologies and even replacing the system, suggesting the best solution is not yet known and they're willing to experiment. The need to compare the new system with the existing one and adjust features based on the results highlights the need for flexibility.
2. **Development Method:** Iterative and Incremental Life Cycle: This method provides an accelerated feedback . The iterative nature allows the hospital to test and validate improvements in stages while also integrating new technologies. The ability to test alongside the old system is crucial, and incremental development allows for features to be rolled out and adjusted based on real-world performance. Privacy and data protection requirements can be incorporated and tested in each iteration.

Project 3: Tourism Company and Augmented Reality

1. **Project Type (Section 3.2):** Type 4 (Unclear Objective, Little-known Solution). The company wants to *explore* the application of AR. There's no defined problem to solve, but rather a desire to see what possibilities AR offers. The solution is also unknown since it's about exploring AR's application.
2. **Development Method:** Lean and Agile Development: Type 4 projects need a flexible method . In this scenario, an agile approach is suitable, particularly lean development. Lean development prioritizes experimentation, learning, and delivering value quickly . The company needs a method that allows for rapid prototyping and testing of different AR concepts to determine what resonates with customers and aligns with the company's strategic goals.

Question 3

Example type 1 project:

> A company is looking to upgrade their small, well-defined Access database to SQL Server. The goal is straightforward - they want to transfer their existing data and functionality to a more robust platform that can scale better. The path forward is clear-cut: they'll use standard database migration tools to extract everything from Access, make any necessary format adjustments, and load it all into SQL Server. Since their database team has experience with both systems, the technical aspects shouldn't pose any surprises.
>
> This is a classic Type 1 project - the company has a precise objective and a proven method to achieve it. There's very little guesswork involved regarding what needs to be done, which technology to use, or how to define the project's boundaries. It's essentially a straightforward migration with clearly defined start and end points.

Module 2 Questions

Question 4

1. Trip

- **a. Subclasses:**
  - `BusinessTrip`: Represents a trip taken for work purposes.
  - `VacationTrip`: Represents a trip taken for leisure.
- **b. Association:** `Trip` can be associated with the class `Traveler`: A `Trip` is planned by one and only one `Traveler`, and a `Traveler` plans any number of `Trips`.
- **c. Attributes:**
  - `Trip`: `destination` (String)
  - `BusinessTrip`: `companyName` (String)
  - `VacationTrip`: `theme` (String)
- **d. Instances:**
  - `Trip`: `{destination: "London"}`
  - `BusinessTrip`: `{destination: "New York", companyName: "Acme Corp"}`
  - `VacationTrip`: `{destination: "Paris", theme: "Romantic Getaway"}`

2. Game

- **a. Subclasses:**
  - `VideoGame`: A game played on an electronic device (console, phone, PC, etc.)
  - `BoardGame`: A game played on a physical board with tangible pieces.
- **b. Association:** `Game` can be associated with the class `Publisher`: A `Game` is published by one `Publisher`, and a `Publisher` publishes one or more `Games`.
- **c. Attributes:**
  - `Game`: `title` (String)
  - `VideoGame`: `platform` (string)
  - `BoardGame`: `numberOfPlayers` (String)
- **d. Instances:**
  - `Game`: `{title: "Chess"}`
  - `VideoGame`: `{title: "The Last of Us", platform: "PlayStation 5"}`
  - `BoardGame`: `{title: "Monopoly", numberOfPlayers: "2-8"}`

Question 5

1. **Visibility of `name` in `Character`:** `Protected`. To be accessible by subclasses (`Warrior` and `Archer`) but not by unrelated classes (`Shield`), the `name` attribute needs `protected` visibility. `Protected` visibility allows access from the class itself and any subclasses, regardless of the package .
2. **Visibility of `protection` in `Shield`:** No visibility to Character, Archer, and Warrior is possible. `protection` is an attribute of the `Shield` class. To allow the Warrior and Archer classes access the protection attribute of the Shield class, you should first implement an association between both classes. For example, you could define a "wields" association between Character and Shield. To do this, access should be public.
3. **Visibility of `accuracy` in `Archer`:**
   - **a) Accessible only by `Archer` itself:** `Private`. A `private` attribute is only accessible within the class where it is defined .
   - **b) Impact of adding `Mage`:** No, the answer would not change. `Private` visibility ensures that even if a new subclass of `Character` is added, it will not have direct access to `accuracy` defined in the `Archer` class.
4. **Association between `Shield` and `Character`:**
   - **Association:** `wields`. A `Character` *wields* (or *carries*) zero or one `Shield`. A `Shield` is *wielded by* zero or one `Character`. The multiplicity would be `0..1` on both ends. This indicates that a character may not have a shield, and a shield may be unequipped. This is important because a shield will not be associated with a Character for its existence.
   - **Polymorphic Association?:** Yes, it's a polymorphic association. The `Character` end of the association is with the abstract `Character` class. Therefore, you can associate a `Shield` with instances of `Warrior`, `Archer`, or any other future subclass of `Character` (like `Mage`).
5. **Modeling `Bow` and `MagicShield` Relationships:**
   - **a. `Bow` and `Archer`:** Association. An `Archer` *uses* a `Bow`. The relationship between an archer and a bow is better modeled as an association because a bow is an external tool, and an archer can change bows.
   - **b. `MagicShield` and `Shield`:** Inheritance. `MagicShield` *is a* type of `Shield`. It will likely share all the base properties of a shield (name, type, protection) but add magical properties. Inheritance is the appropriate mechanism when you want to create a specialized version of an existing class, inheriting its properties and behaviors .

Question 6

a) Classes and attributes:

- **Property** (Abstract):
  - `identificationCode` (String)

  - `address` (String)
  - `surfaceArea` (double)
  - `availability` (enum: Rent, Sale, Both)
- **CommercialPremises** (Concrete, subclass of Property):
  - `hasFullBathroom` (Boolean)
- **ResidentialHome** (Concrete, subclass of Property):
  - `numberOfBedrooms` (int)
  - `numberOfFullBathrooms` (int)
  - `numberOfHalfBathrooms` (int)
- **Owner** (Concrete):
  - `idNumber` (Str)
  - `firstName` (String)
  - `lastName` (String)
  - `phoneNumber` (String)

b) Associations:

- **Owns:** Associates each instance of `Owner` with *one or more than one* instance(s) of `Property` and each instance of `Property` with *one or more than one* instance(s) of `Owner`.
  - It is polymorphic, since the association is with the `Property` class, which is abstract, allowing any subclass of `Property` (e.g., `CommercialPremises`, `ResidentialHome`) to be part of the association.
- **SimilarProperties:** Associates each instance of `Property` with *any number of* instance(s) of `Property` and each instance of `Property` with *any number of* instance(s) of `Property`.
  - It is polymorphic too: The association is between the `Property` class and itself. The fact that the association is performed in Property implies that we can relate ResidentialHomes to CommercialPremises through SimilarProperties. The similarity may be based on certain factors that are common to both and allow, for instance, to offer ResidentialHomes to those who are browsing CommercialPremises and vice versa.