CAT 1

Introduction to Software Engineering and Object Orientation

Alejandro Pérez Bueno

Oct 18, 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

Question 1

- **Task 1** The role is the **Architect** and the activity is **Modelling**. Choosing a database is a huge structural decision that impacts the whole system's performance and ability to scale later. This is a core responsibility for an architect, who makes these high-level design choices at the start of a project.
- **Task 2** This sounds like a **Technical Analyst** (or Organic Analyst) doing **Modelling**. They are taking the high-level architecture and creating a detailed blueprint for one part of it. The goal is to make the instructions so clear that a programmer can build it without having to guess or make their own structural decisions.
- **Task 3** The role is **Functional Analyst** and the activity is **Requirements Identification**. This person's job is to talk to different business departments, figure out what they each want, and create a single, clear set of requirements that the tech team can work from. They act as a mediator to resolve conflicts.
- **Task 4** I believe this is the **Deployment Manager**, and the activity falls under **Construction and testing**. Their task is to take the finished, tested software and release it to the end users. This includes packaging the application and getting it running on the production environment.
- **Task 5** This is a **Quality Expert (tester)** and the activity is **Maintenance and re-engineering**. Since a bug was found *after* the software was deployed, the task is to analyze and fix the problem. Creating an automated test to prevent it from happening again is a key part of the quality role, and it's considered corrective maintenance.

Question 2

Project 1

For the payroll system, I'd choose the Classic or cascade life cycle (Group 1). This is because the project is very well-defined; the goal is to automate a payroll system and all the rules and regulations are already known. The waterfall method makes sense here because each step has to be completed and reviewed before moving to the next. For something like payroll that has to be auditable and precise, you want to minimize risks and avoid changes mid-project.

Project 2

This project seems like a good fit for Group 2, using an Iterative and incremental life cycle. The goal—a patient portal—is clear, but the exact solution isn't. You'd need to get a lot of feedback from users to figure out the best design. This approach lets you build the portal in small, functional pieces. You can release one part, see how users react, and make adjustments. It's a good way to handle the technical challenges of connecting to older legacy systems and gives users something to test and train on early.

Project 3

The AI startup is definitely a Group 4 project, so it needs a Lean and agile development method. Here, both the objective and the solution are unclear. It's basically a research and development effort. Using an agile framework like Scrum lets the team build small experiments, release them quickly, and get feedback to see what's valuable. The whole point is to respond to change rather than follow a rigid plan, and pivot if needed. This avoids spending a lot of money building the wrong thing, which is crucial for a startup trying to find product-market fit.

Question 3

Project Proposal: An Integrated "Household Hub" App for Nextcloud

Nextcloud is a popular open-source platform that lets users self-host their own cloud for files, calendars, contacts, etc., giving them control over their data away from big tech providers. The idea is to develop a new "Household Hub" app for Nextcloud that brings together several household management features into one dashboard. This could include a shared shopping list, a family budget tracker, a meal planner, and a chore scheduler.

This project fits into **Group 2**, where the objective is clear (build a household management hub) but the solution is little-known. Because of this, the **iterative and incremental life cycle** makes the most sense.

- We can start with a basic version of the app, maybe just the meal planner, and release it to a small group of users (like a pilot family) for feedback.
- This lets us get early feedback and incorporate changes quickly in the next iteration.
- It also helps us tackle technical risks one at a time, like figuring out how to best integrate with Nextcloud's calendar and authentication systems, before committing to the full project.

Why Other Methodologies Are Not as Suitable

- The **classic or cascade life cycle** wouldn't work because it assumes all requirements are known upfront. For a new app like this, features and user needs will definitely evolve once people start using it.
- **Lean and agile development** (Group 4) is unlikely. We do have a clear project goal, unlike a pure R&D project. We're not trying to find a market; we're building a specific tool for an existing platform. However, the project would probably use agile practices (like sprints) within the overall iterative framework.º

Module 2 Questions

Question 4

1. The `Publication` class is abstract. Which attributes of `Publication` are visible from the class `Book`? And from the `Reader` class? Explain why.

- **From the `Book` class:** Both the `title` and `publisher` attributes are visible. As a subclass of `Publication`, `Book` has special access. It can see `public` attributes (`title`) and also `protected` attributes (`publisher`). However, it cannot see the `code` attribute because that is `private` to `Publication`.
- **From the `Reader` class:** Only the `title` attribute is visible. Because `Reader` is just a separate, unrelated class, it can only access `public` members. The `protected` (`publisher`) and `private` (`code`) attributes are hidden from it.

2. If a `Reader` can borrow multiple publications, could a `Reader` directly query the `code` for all the publications they have borrowed?

No, a `Reader` could not do that. The `code` attribute is `private` within the `Publication` class, which means only code inside of the `Publication` class itself is allowed to access it. For this to be possible, the `Publication` class would need to offer a `public` method, like a `getCode()` function, that returns the private `code` value. Without that, the data is inaccessible to outside classes like `Reader`.

3. a) According to this model, do books have the attribute `publisher`? b) And the publications, do they have the `author` attribute?

- a) **Yes, books do have a `publisher` attribute.** Because `Book` is a subclass of `Publication`, it inherits all of `Publication`'s public and protected attributes. So, every `Book` object automatically gets the `publisher` attribute from its parent class.
- b) **No, publications do not have an `author` attribute.** The `author` attribute is defined in the `Book` subclass, so it only exists for `Book` objects (and any of its future subclasses). The `Publication` superclass doesn't gain attributes from its children, so a generic `Publication` object wouldn't have an author.

4. a) Propose an association between the class `Reader` and any of the other classes. Also indicate the multiplicity of the association between both classes. b) Is this a polymorphic association?

- a) I would propose a "borrows" association between `Reader` and `Publication`.
  - **Multiplicity:** A `Reader` can borrow zero or more `Publications` (**0..***), and any single `Publication` can be borrowed by at most one `Reader` at a time (**0..1**).
- b) **Yes, this is a polymorphic association.** It's polymorphic because the `Reader` class is associated with the abstract `Publication` class, not directly with `Book` or `Magazine`. This is powerful because it allows a `Reader` to borrow a `Book`, a `Magazine`, or any other future type of `Publication`, and the system can handle them all uniformly through their common superclass interface in the context of the "borrows" relationship.

5. Which object-oriented mechanism (inheritance or association) would you use to model the relationship between each of these new classes and existing ones? Briefly justify your answer for each case.

- **a) The class `Author`:**
  - This should be an **association**. The relationship is that an `Author` "writes" a `Book`. An `Author` *is not a* type of `Book` (and vice-versa), so inheritance doesn't make sense. They are distinct objects that are linked together. This "has-a" or "writes-a" relationship is exactly what associations are for.
- **b) The class `Novel`:**
  - This should use **inheritance**. A `Novel` *is a* specific kind of `Book`. It would share all of a `Book`'s attributes (like title and author) and just add more specific ones (like `literaryGenre`). This "is-a" relationship is the classic use case for creating a `Novel` subclass that inherits from the `Book` class.

Question 5

1. Visibility of `name` in `Character`: I'd set the visibility of `name` to `protected`. This way, subclasses like `Warrior` and `Archer` can access it, but it remains hidden from unrelated classes like `Shield`.

2. Visibility of `protection` in `Shield`: The `protection` attribute isn't directly visible to `Character` or its subclasses because it's an internal part of the `Shield` class. For a `Character` to know a shield's protection value, you first need an association linking them (e.g., a "wields" relationship). Then, the `protection` attribute would need to be made `public` so the `Character` could access the attribute from the `Shield` object it is wielding.

3. Visibility of `accuracy` in `Archer`: a. To make the `accuracy` attribute accessible *only* within the `Archer` class, it should be set to `private`. b. This wouldn't change if a new `Mage` subclass was added. Because `accuracy` is private to `Archer`, no other class can access it directly, including other subclasses of `Character`.

4. Association between `Shield` and `Character`: I would model this with a "wields" association between `Character` and `Shield`. The multiplicity should be `0..1` on both ends, as a character might not have a shield, and a shield could be unequipped. This is a polymorphic association because the link is to the abstract `Character` class, meaning any of its subclasses (`Warrior`, `Archer`, etc.) can participate in the relationship and wield a `Shield`.

5. Modeling `Bow` and `MagicShield` Relationships: a. For a `Bow` and `Archer`, an association is the right choice. A bow is something an archer *uses*, not a core part of what an archer *is*. This makes sense because an archer could change bows.

b. With `MagicShield` and `Shield`, inheritance is a better fit. A `MagicShield` *is a* specific kind of `Shield`. It would inherit all the regular properties of a shield but could add new magical ones. This is a good use case for creating a specialized subclass.

Question 6

Based on the scenario here is a detailed model for the basketball league app:

a) List of Classes and Attributes

Class: `TeamMember`

- **Type:** Abstract. I made this abstract since you can't have a generic "TeamMember"; they have to be either a `Player` or a `Coach`.
- **Superclass:** N/A
- **Attributes:**
  - `id`
  - `firstName`
  - `lastName`

Class: `Player`

- **Type:** Concrete
- **Superclass:** `TeamMember`
- **Attributes:**
  - `jerseyNumber`
  - `position` (multivalued): A player might be able to play more than one position.

Class: `Coach`

- **Type:** Concrete
- **Superclass:** `TeamMember`
- **Attributes:**
  - `federationLicenseNumber`

Class: `Team`

- **Type:** Concrete
- **Superclass:** N/A
- **Attributes:**
  - `name`
  - `city`

b) List of Associations

i) belongsTo / hasRoster: Associates each **Team** with **one or more TeamMember**s, and each **TeamMember** belongs to **one and only one Team**.

- I'd call this a **polymorphic association**. Because **Team** is linked to the abstract **TeamMember** class, its roster can be filled with both **Player** and **Coach** objects without the **Team** class needing to worry about the specifics.

ii) isCoachedBy: Associates each **Team** with **one and only one Coach**, and each **Coach** is linked to **one and only one Team**.

- This one is **not polymorphic**. It's just a direct link between two specific classes, **Team** and **Coach**. There's no inheritance involved where different types of coaches could be swapped in.

iii) hasSynergyWith: Associates a **TeamMember** with **any number of** other **TeamMember**s.

- This one is also **polymorphic**, but it's a bit different because it's recursive—it connects **TeamMember** to itself. This is useful because it means any team member, whether a **Player** or a **Coach**, can have a synergy relationship with any other member.

Assumptions Made:

- A team needs at least one member to be valid (so a roster can't be empty).
- The `isCoachedBy` relationship adds a rule on top of the main roster: exactly one of the members on the team must be a **Coach**.