25-26

# SophiaTech Eats – V3

SI4 - CASE STUDY FOR MODULE OBJECT ORIENTED DESIGN

MIREILLE BLAY-FORNARINO*

** With the help of Anne-Marie, Seb, Philippe, Jean-Mi, … see disclaimer*

## Disclaimer

This document was written with the help of ChatGPT.

This document draws inspiration from various sources:

- The book [1] provides the structure.
- Sébastien Mosser's subject at McMaster University influenced its format and original concept, tailored to our campus context.
- Jean-Michel Bruel (Toulouse) and Anne-Marie Pinna-Dery contributed to the requirements section.
- Philippe Collet's SI5 project proposal inspired the project's spirit.

# Note de lecture (à lire!)

Ce document est structuré de manière à reproduire une étude de cas telle qu'on pourrait la trouver dans le formalisme présenté dans le livre [1] et qui est utilisé par des industriels.

- ❑ Les **chapitres G, E et S** présentent les éléments de contexte et de spécification.
  - ⇨ La **partie G5** est centrale : elle définit les principales exigences que vous devrez modéliser et implémenter. Les TDs vous guideront pas à pas pour ce travail.
    - Vous devez étudier et implémenter toutes les exigences en **bleu**.
    - Pour les exigences en **vert**, les attentes précises vous seront données dans le TD2.
    - **Vous n'aurez pas à implémenter les autres exigences.**
- ❑ Le **chapitre P** porte sur la gestion de projet. Il situe cette étude de cas dans une perspective plus large de développement informatique. L'**annexe** complète ce chapitre en l'adaptant à notre contexte pédagogique. Vous devez « simplement » appliquer tout ce que vous avez vu en gestion de projets en PS5 et PS6.

Vos développements suivront deux étapes principales :

1. **Jusqu'aux vacances (octobre)** : implémentation du backend, en respectant une décomposition par domaines (comme illustré ci-après).
2. **Après les vacances** : mise en œuvre de services connectés à une interface graphique simple, dont l'objectif est uniquement de vous initier à la conception d'une architecture de bout en bout.

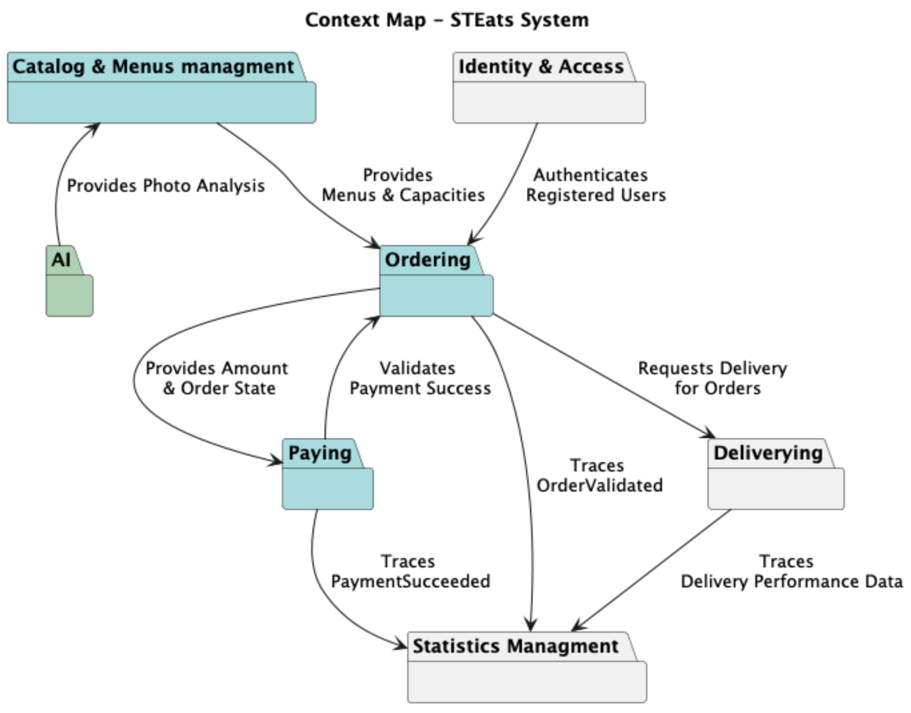
Context Map – STEats System
Catalog & Menus managment
Identity & Access
Provides Photo Analysis
Provides Menus & Capacities
Authenticates Registered Users
AI
Ordering
Provides Amount & Order State
Validates Payment Success
Requests Delivery for Orders
Paying
Traces OrderValidated
Delivering
Traces PaymentSucceeded
Traces Delivery Performance Data
Statistics Managment

Table of Contents

# (G) Goals

*Goals are "needs of the target organization, which the system will address". While the development team is the principal user of the other books, the Goals book addresses a wider audience: essentially, all stakeholders [1]*

*It must contain enough information to provide — if read just by itself — a general sketch of the entire project. To this effect, chapter G.3 presents a short overview of the system and G.1 will typically include some key properties of the environment. As it addresses a wide readership, it should be clear and minimize the use of specialized technical terms. Together, G.1, G.2 and G.3 describe the rationale for the project. It is important to state these justifications explicitly. Typically, they are well understood at the start of the project, but management and priorities can change [1]*

## (G.1) Context and Overall Objectives

*High-level view of the project: organizational context and reason for building a system. It explains why the project is needed, recalls the business context, and presents the general business objectives. [1]*

The project aims to develop an online food ordering system for a campus environment. It addresses the need for convenient food delivery services for students and campus community members. The system enables users to place orders from various restaurants by selecting delivery locations on campus. The project aims to streamline food orders and provide efficient delivery options. It also facilitates restaurant registration and management, promotes campus restaurants, and enhances overall dining experiences. Key objectives include easy access to diverse meal options without the need for lengthy travels, adaptation to student context (pricing, format), offering multiple restaurant choices, and optimizing food delivery on campus (grouping orders for larger groups, adhering to schedules for shorter eating times, etc.).

## (G.2) Current situation

*Current state of processes to be addressed by the project and the resulting system. It describes the current situation, upon which the system is expected to improve [1]*

Currently, there is no system on campus to manage on-site food orders, which makes it difficult to diversify dining options and often leads to significant wait times. This statement highlights the need for a campus-specific solution that can streamline the ordering process while considering the constraints of campus members and restaurateurs. The developed solution is also expected to improve delivery efficiency, optimizing the use of designated space on campus and ultimately improving the overall campus living experience.

## (G.3) Expected Benefits

*New processes, or improvement to existing processes, made possible by the project's results. It presents the business benefits expected from the successful execution of the project.* ***This chapter is the core of the Goals book****, describing what the organization expects from the system. It ensures that the project remains focused: if* *at some stage it gets pushed in different directions, with "creeping featurism" threatening its integrity, a reminder about the original business goals stated in those chapters will help. [1]*

**Campus users** will benefit from a streamlined ordering process with reduced waiting times and more reliable delivery schedules. They can browse dishes, filter restaurants by criteria such as availability, cuisine type, or dietary preferences, and prepare their orders in advance. Special offers like "Campus Specials" and the student credit mechanism will provide flexibility and affordability adapted to student life.

**For restaurant owners,** the system offers a practical platform to showcase their food and beverage offerings, register dishes and options efficiently, and keep their catalog up to date. By reducing the workload of dish management and enabling targeted promotions, the system helps restaurants reach a wider student audience and potentially increase revenue.

**Campus administrators** gain enhanced oversight of partnerships and delivery services. Access to statistical data such as order volumes, popular items, and delivery efficiency provides valuable insights, enabling better decision-making and continuous optimization of campus food services.

## (G.4) Functionality overview

*Overview of the functions (behavior) of the system. Principal properties only (details are in the System book). It is a short overview of the functions of the future system, a kind of capsule version of book S, skipping details but enabling readers to get a quick grasp of what the system will do. [1]*

The system will encompass the following functionalities. All these features require refinement.

**Ordering:** The system will allow registered users to place **orders at registered restaurants**. Access to the system will be open without restrictions, with the sole requirements being user registration and providing a pre-registered delivery location.

**External Financial Transactions:** The system will integrate with conventional external payment systems like PayPal and Google Pay. Restaurants will receive direct payment upon order placement. You don't have to manage these financial aspects. In the event of an order cancellation, the system will initiate a refund for the user.

**Real-time updates:** Campus restaurants can update their offers, operating hours, and production capacity.

**Order cancellation:** An order can only be canceled if the expected delivery time is more than half an hour away. Under these conditions, a user can cancel their order if it still needs to be prepared. A restaurant may cancel an order if it was placed outside its opening hours or if less than half an hour has passed since the user set the order. There is no scenario for the refusal of delivery by a delivery person.

**Statistical Insights:** The statistical data will offer valuable insights for decision-making by restaurateurs, administrators, and even users who may place advance orders.

## (G.5) High-level usage scenarios

*Fundamental usage paths through the system. It presents the main scenarios (use cases) that the system should cover. The scenarios chosen for appearing here, in the Goals book, should only be the main usage patterns, without details such as special and erroneous cases; they should be stated in user terms only, independently of the system's structure. Detailed usage scenarios, taking into account system details and special cases, will appear in the System book (S.4)[1]*

### *Customer-side*

[C1] Any internet user can browse dishes from different campus restaurants.

[C2] Any internet user can **filter restaurants** according to:

- Availability (can the restaurant prepare a meal at a given time).
- Presence of specific dishes (vegetarian, vegan, gluten-free, organic, …).
- Price range.
- Cuisine type (French, Mediterranean, Italian, …).
- Type of establishment (CROUS, restaurant, food truck, …).

**[C3]** Only registered users can place orders.

**[C4]** Registered campus users can access their previous food orders.

[C5] To create an order, the registered user selects a restaurant from among pre-recorded restaurants. The user selects a pre-recorded delivery location. The user can visualize the delivery dates within the restaurant's preparation capabilities.

[C6] Each time a new item (or set of items) is added to an order, the possible delivery dates change according to restaurant capacity and other validated orders.

[C7] Before proceeding to payment, the user must validate the order and select an available delivery time.

**[C8]** The user cannot modify the delivery details once an order is validated.

### *Restaurant-side*

**[R1]** Restaurant Staff can consult orders that are awaiting preparation and validate those that are ready for pickup.

[R2] A restaurant must be able to register and update its dishes.
Each dish must have a name and a description.
Each dish must belong to a general category (e.g., Starter, Main Course, Dessert, Drink).
Each dish may optionally be associated with a more specific type (e.g., Pasta, Meat, Pizza, Ice Cream, Cake).
Each dish may offer additional paid toppings (e.g., extra ingredients, garnishes).
Each dish may be tagged with dietary or composition information (e.g., gluten-free, contains frozen products, may contain traces of peanuts).

**[R3]** Restaurants can also define **extra options** (e.g., portion of fries, salad, fruit compote).

**[R4*]** To reduce the workload of restaurant owners, the system must provide intelligent assistance when entering dishes, while ensuring that the information remains accurate.
The system should support different strategies of assistance, for example:

- Keyword-based suggestions (e.g., typing “Salad” proposes “Starter → Salad → Vegetarian”),
- History-based suggestions (e.g., reusing information from previously entered dishes),
- Default templates (e.g., a basic template for a dessert or a drink).

**[R5]** Restaurant managers can update their opening hours and their capacities in terms of number of orders by half an hour.

- For example, Jeanne specifies
  - from 11:00 AM to 11:30 AM → a capacity of 5 orders
  - from 11:30 AM to 12:00 PM → a capacity of 10 orders
  - between12:00 PM and 2:00 PM → a capacity of 30 orders per half hours

**[R6]** Restaurants can define “Campus Specials” (special offers available only for students or staff, with time or quantity limits). The system must immediately apply the reduction to the price.

**[R7*]** Restaurants can define discounts such as: orders of more than *n* items benefit from a discount of *r*%. The system applies this discount as a credit for future orders in the same restaurant.

## *Payment Requirements*

**[P1]** A user must complete the payment step to validate an order. For standard payments, the system redirects the user to an external payment service for validation.

**[P2]** A successful payment results in order registration in the user's account.

**[P3]** The system must provide a **student credit mechanism**: each student receives an allowance (e.g., €20) that can be used as a payment method.

**[P4]** The student credit is considered a **loan**: it must be reimbursed within one month.

- *Example*: If Bob uses €12 on 28/09/2025, repayment is due no later than 28/10/2025. After this date, a warning is sent and the student is no longer allowed to use the credit until repayment is completed. (Other sanctions are out of scope for this system.)

**[P5]** The system must **block further use** of the student credit if the previous month has not been reimbursed.

- *Example*: After 28/10/2025, Bob cannot use the remaining €8 until he has reimbursed the €12 loan.

**[P6]** The system must clearly record whether a payment was made with student credit or with a normal method. This information is visible to the student but not exposed to restaurants, which only see that the order has been paid.

**[P8]** If a credit due to a previous discount is available for the current user and current restaurant, it is automatically deducted from the price of the order.

### *Administration-side*

**[A1]** Campus Administrators manage restaurant partnerships and delivery services. This includes adding or removing restaurants, ensuring compliance with agreements, and managing delivery staff. The project also uses statistics for effective partnership management.

**[A2*]** The system collects statistical data to give administrators and stakeholders insights for decision-making. This dataset includes order volumes, delivery efficiency, popular dishes items, and user behavior

### *Delivery-side*

**[D1]** Each delivery is limited to a single order, but the system may optimize delivery assignments by grouping orders with the same delivery location or time slot, while still ensuring that each order comes from a single restaurant. For example, if two students order from the same restaurant for delivery to the same campus residence, the system may assign the same delivery person to handle both orders.

**[D2]** Users receive order notifications with ID, delivery date, and location.

**[D3]** When a delivery is ready, the system notifies a delivery person with route details, pickup time, restaurant, delivery location, and username. The user receives the delivery person's ID and phone number.

**[D4]** Upon delivery, the user acknowledges receipt on their phone. The delivery person is notified and delivers the order. He, in turn, validates the delivery. The delivery person may also provide an order without user confirmation, which requires a signature and identity verification.

### ***AI Considerations (exploratory)***

**[AI1*]** The system must include an AI-based photo analysis service that can generate a short description of a dish, suggest a suitable category and type (e.g., Starter, Main Course, Pizza, Pasta, Kebbab), and propose dietary or allergen tags when relevant.
**[AI2*]** The system must ensure that inaccurate or misleading information generated by the AI is not directly exposed to end users. Since such errors could have legal and safety implications, the design should include mechanisms that guarantee reliability and accountability.

## (G.6) Limitations and Exclusions

*Aspects that the system does not need to address. It states what the system will not do. This chapter addresses a key quality attribute of good requirements: the requirements must be delimited (or "scoped"). G.6 is not, however, the place for an analysis of risks and obstacles, which pertain to the project rather than the goals and correspondingly appears in chapter P.6. [1]*

The project delineates specific areas that lie outside its scope:

- Optimization of deliveries based on traffic conditions for restaurants located off-campus.
- Comprehensive payment management: the system will focus solely on payment validation.
- Addressing scenarios where a restaurant is unable to fulfill orders once accepted, which is considered beyond the teaching module's scope.
- **The management of access rights is not considered in this modeling**. It will be managed by a suitable software system such as studying in the Software Security module.

## (G.7) Stakeholders and requirements sources

*Groups of people who can affect the project or be affected by it, and other places to consider for information about the project and system. It lists stakeholders and other requirements sources. It should define stakeholders as categories of people, not individuals, even if such individuals are known at the time of writing. The main goal of chapter G.7 is to avoid forgetting any category of people whose input is relevant to the project. It also lists documents and other information that the project, aside from soliciting input from stakeholders, can consult for requirements information. [1]*

Stakeholders:

- Campus Users (Students, Staff, Faculty)
- Restaurant Managers and Staff
- Delivery Personnel
- Administrators

**Requirements Sources:** (not provided as part of the teaching module)

- Existing Campus Policies and Guidelines
- RGPD (General Data Protection Regulation)
- API of external payment systems

# (E) Environment

*The Environment book describes the application domain and external context, physical or virtual (or a mix), in which the system will operate. [1]*

## (E.1) Glossary

*Clear and precise definitions of all the vocabulary specific to the application domain, including technical terms, words from ordinary language used in a special meaning, and acronyms. It introduces the terminology of the project; not just of the environment in the strict sense, but of all its parts. [1]*

**Campus:** corresponds to an area centered on PNS, which is a 10-minute walk from the center. This includes PNS, the IUTs, INRIA, Algorithms, and extends to Luciole and Saint-Philippe.
**Campus Users:** Refers to individuals within the campus community, including students, staff, and faculty members. Only registered campus users can access the SophiaTech Eats System to place an order.
**Restaurant:** Refers to restaurant staff who interact with the system.
**Delivery Person:** Refers to delivery personnel responsible for delivering food orders to designated locations on campus.
**Campus Administrators:** Represent individuals in administrative roles responsible for overseeing and managing campus activities. For instance, they may collaborate to distribute course hours across campus formation to avoid congestion at the University Restaurant.
**RGPD (General Data Protection Regulation):** Refers to the European Union regulation governing the protection of personal data and privacy.
**API (Application Programming Interface**): A set of protocols and tools that allows different software applications to communicate and interact with each other.
**External Payment Systems:** Refers to third-party payment platforms, such as PayPal and Google Pay, used for processing financial transactions.
**Order:** A request made by a campus user for specific food items from one restaurant.
**Delivery:** The process of transporting ordered food from restaurants to designate campus locations.
**User interface:** Visual and interactive components that allow users to interact with the system. *However, no UI implementation is requested in this project until the last 15 days.*
**Module:** Designates the educational unit within which this project is located (COO).

This glossary provides clear definitions for the specific vocabulary and terms used within the context of the project. It ensures a shared understanding of key concepts and terminology among stakeholders and team members.

## (E.2) Components

*List of elements of the environment that may affect or be affected by the system and project. It includes other systems to which the system must be interfaced. These components may include existing systems, particularly software systems, with which the system will interact — by using their APIs (program interfaces), or by providing APIs to them, or both. These are interfaces provided to the system from the outside world. They are distinct from both: interfaces provided by the system to the outside world (S.3); and technology elements that the system's development will require (P.5). [1]*

The following elements within the environment have been identified as components likely to influence or be influenced by the project.

**External Payment Systems:** Third-party platforms such as PayPal and Google Pay, which are utilized for processing financial transactions.

**User Devices:** Devices used by campus users to access the system, including smartphones, computers, or tablets. Although they will not be directly addressed within the framework of this module, they can still influence your choice of scenarios.

## (E.3) Constraints

*Obligations and limits imposed on the project and system by the environment. This chapter defines non-negotiable restrictions coming from the environment (business rules, physical laws, engineering decisions), which the development will have to take into account. [1]*

Within the scope of this educational endeavor, it's imperative to adhere to the following constraints:

1. **Scalability**: The system's architecture should be designed to gracefully handle potential increases in user volume, restaurant partnerships, and order frequency, all while maintaining optimal performance.
2. **Evolvability:** As the project progresses, it's expected that new features and functionalities will be introduced. The application's design should be flexible enough to accommodate such enhancements seamlessly.
3. **Robust Error Handling:** The system should be equipped to handle unexpected errors and exceptions gracefully, preventing disruptions in user experience. Effective error messages and recovery mechanisms should be in place to guide users through any unforeseen scenarios.
4. **Comprehensive Testing:** Rigorous testing is a cornerstone of this project. The system must undergo thorough testing at various levels to ensure its functionality and performance. This includes unit testing, integration testing, and end-to-end testing of critical scenarios.

## (E.4) Assumptions

*Properties of the environment that may be assumed, with the goal of facilitating the project and simplifying the system. It defines properties that are not imposed by the environment (like those in E.3) but assumed to hold, as an explicit decision meant to facilitate the system's construction. [1]*

To streamline the project and simplify system development, the following assumptions were made:

- **Stable Internet Connectivity:** It is assumed that users will have stable and reliable Internet connectivity when accessing the system.
- **Availability of delivery personnel**: The availability of delivery personnel is assumed to be sufficient to meet delivery requests. Although the module does not deal directly with the hiring of delivery personnel, it is assumed that the necessary labor is available.
- **Consistent API integration**: Integration with external payment systems, such as PayPal and Google Pay, is assumed to be simple and consistent and always follow the same principle.
- **Cooperative Restaurant Managers:** It is assumed that restaurant managers will actively engage in the system and accurately maintain their offerings, including removing dishes that are no longer available.
- **Precise delivery locations:** Users must provide pre-identified delivery locations on campus. This assumption ensures that deliveries are made to the correct designated areas.

## (E.5) Effects

*Elements and properties of the environment that the system will affect. It defines effects of the system's operations on properties of the environment. Where the previous two categories (E.3, E.4) defined influences of the environment on the system, effects are influences in the reverse direction. [1]*

Out of the scope of the project

## (E.6) Invariants

*Properties of the environment that the system's operation must preserve, i.e., properties of the environment that operations of the system may assume to hold when they start, and must maintain [1]*

Out of the scope of the project

# (S) System

*the System book refines the Goal one by focusing on more detailed requirements.*

This part corresponds more or less to the learning objectives targeted in this module. Therefore, we present the expected major sections here without providing their content.

## (S.1) Components

*Overall structure expressed by the list of major software and, if applicable, hardware parts. [1]*

This part will be addressed when we focus on the overall architecture of the application and determine the components that make it up.

## (S.2) Functionality

***This is the bulk of the System book, describing elements of functionality (behaviors)**. This chapter corresponds to the traditional view of requirements as defining "**what the system does**". It is organized as one section, S.2.n, for each of the components identified in S.1, describing the corresponding behaviors (functional and non-functional properties). [1]*

## (S.3) Interfaces

*How the system makes the functionality of S.2 available to the rest of the world, particularly user interfaces and program interfaces (APIs). It specifies how that functionality will be made available to the rest of the world, including people (users) and other systems. These are interfaces provided by the system to the outside; the other way around, interfaces from other systems, which the system may use, are specified in E.2. [1]*

We will address this question throughout the project, initially by defining the domain model to characterize the domain elements being utilized, and then by refining component interfaces in terms of exposed (offered) behaviors and required behaviors.

## (S.4) Detailed usage scenarios

*Examples of interaction between the environment (or human users) and the system, expressed as user stories. Such scenarios are not by themselves a substitute for precise descriptions of functionality (S.3) but provide an important complement by specifying cases that these behavior descriptions must support; they also serve as a basis for developing test cases. The scenarios most relevant for stakeholders are given in chapter G.5 in the Goals book, at a general level, as use cases; in contrast, S.4 can refer to system components and functionality (from other chapters of the System book) as well as special and erroneous cases and introduce more specific scenarios. [1]*

We will define user stories and their quality in connection with validation tests (S.2).

## (S.6) Verification and acceptance criteria

*Specification of the conditions under which an implementation will be deemed satisfactory. Here, "verification" as shorthand for what is more explicitly called "Verification & Validation" (V&V), covering several levels of testing — module testing, integration testing, system testing, user acceptance testing — as well as other techniques such as static analysis and, when applicable, program proving. [1]*

The acceptance criteria are distributed across user stories. The expected tests include unit tests (Junit 5), validation tests (Gherkin), integration tests using mocks, and, of course, all of them should be executable in continuous integration. You will use static analysis, at least SonarLint, to check the quality of your code, test coverage, code review to verify the general philosophy within the team.

# (P) Project

*Goals are "needs of the target organization, which the system will address". While the development team is the principal user of the other books, the Goals book addresses a wider audience: essentially, all stakeholders [1]*

## (P.1) Roles and personnel

Main responsibilities in the project; required project staff and their needed qualifications. It defines the roles (as a human responsibility) involved in the project. [1]

Each student in the project primarily assumes the role of a Software Developer.
• **Software Developer (SD)**: Implements and codes system components. SD should have programming experience, follow best practices, collaborate effectively, and consider how AI models are integrated in their code, including respecting model card constraints and monitoring system behavior.

Additionally, the following responsibilities are distributed:
• **Business Analyst and Product Owner (PO):** Guides the team to meet functional objectives and aligns AI usage with user needs, fairness, and safety requirements.
The PO takes responsibility for gathering analysis and design documents produced by the team. The PO should also consider the AI features from a product perspective: defining requirements for AI behavior, fairness, and explainability, and ensuring that AI integration aligns with user needs.

• **Software Architect (SA):** Designs the overall system, including AI integration points, data flows, and monitoring strategies, ensuring scalability and maintainability.
The SA consistently can explain the current architecture of the project and the choices made for future stages.
The SA is additionally responsible for designing how AI components fit into the architecture, defining interfaces, data flows, and monitoring points, and ensuring the system can accommodate updates or replacements of AI models.

• **Quality Assurance Engineer (QA):** Ensures the project's quality in terms of testing for functionality, performance, and code quality.
The QA can explain the strengths and weaknesses of the project's quality throughout the project and outline the upcoming steps. The QA role now includes testing AI behavior, verifying compliance with ethical guidelines or model card specifications, and designing test cases that specifically cover AI outputs and reliability.

• **Continuous Integration and Repository Manager (Ops):** Oversees the continuous integration of code, manages the source code repository, tracks commits, and pull requests. Facilitates seamless coordination among developers and ensures that code is properly integrated and continuously tested. The Ops should always be able to justify the state of the repository and the envisaged steps. Ops is also responsible for ensuring that AI components are properly versioned, integrated into CI pipelines, and that deployment procedures consider model updates and monitoring requirements.

**Collective AI Responsibility:** All team members actively consider AI-related aspects, including model limitations, ethical implications, and integration challenges. Decisions about AI design, usage, and monitoring are made collaboratively, so that the whole team understands and mitigates potential risks.

## (P.2) Imposed technical choices

*Any a priori choices binding the project to specific tools, hardware, languages or other technical parameters. Not all technical choices in projects derive from a pure technical analysis; some result from company policies. While some project members may dislike non-strictly-technical decisions, they are a fact of project life and must be documented, in particular for the benefit of one of the quality factors for requirements: "requirements must be justified". [1]*

**Programming Language**: The project is built on the Java programming language, using Java 1.21 or higher. This choice is motivated by the widespread use of Java, and the knowledge of the students as well as the existence of advanced frameworks.

**Test Frameworks:** For testing, the project uses the Cucumber with Gherkin syntax to create and run behavior-based tests. This approach allows us to write tests in a natural language style that can be understood by technical and non-technical stakeholders. It improves collaboration and ensures that requirements are well defined and verified.

**Unit tests:** Unit testing is performed using JUnit 5, a testing framework widely adopted in the Java community. JUnit 5 provides a full suite of testing features, allowing us to ensure the correctness of individual code components and their behavior.

**Integration testing with Mockito:** The project uses Mockito for integration testing. This approach allows to ensure that the different parts of the system work well together and that interactions go as intended. You use Mockito to write tests that don't depend on external components.

**Version control:** Git is chosen as the version control system for its popularity, powerful branching and merging capabilities, and transparent collaboration features. GitHub is used as the platform to host the repository, allowing transparent and distributed development among team members.

**Continuous integration:** GitHub Actions is integrated into the workflow for continuous integration. This ensures that code changes are automatically tested and integrated into the main code base with every push or pull request. This proactive approach aids in identifying and resolving integration issues early in the development process.

## (P.3) Schedule and milestones

*List of tasks to be carried out and their scheduling. It defines the project's key dates. [1]*

Planning the rest of the development at the end of the prototyping phase can be valuable in ensuring a smooth and well-organized software development process.

## (P.4) Tasks and deliverables

***This is the core of the Project book**. It details the individual tasks listed under P.3 and their expected outcomes. It defines the project's main activities and the results they must produce, associated with the milestone dates defined in P.3. [1]*

Planning the rest of the development at the end of the prototyping phase can be valuable in ensuring a smooth and well-organized software development process.

Out of the scope of the project

## (P.5) Required technology elements

*External systems, hardware and software, expected to be necessary for building the system. It lists external technology elements, such as program libraries and hardware devices, that the project is expected to require. Although the actual use of such products belongs to design and implementation rather than requirements, it is part of the requirements task to identify elements whose availability is critical to the success of the project — an important element of risk analysis (P.6). [1]*

Out of the scope of the project

## (P.6) Risk and mitigation analysis

*Potential obstacles to meeting the schedule of P.4, and measures for adapting the plan if they do arise. It is essential to be on the lookout for events that could derail the project, and devise mitigation strategies. It can include a SWOT analysis (Strengths, Weaknesses, Opportunities, Threats) for the project. [1]*

Pour en savoir plus sur ce sujet : ici

Comme précédemment, ce qui serait intéressant est d'évaluer le risque sur la poursuite de ce projet avant d'aller au-delà d'un prototypage pédagogique. Cela reste en dehors de l'étude dans ce module.

## (P.7) Requirements process and report

*Initially, description of what the requirements process will be; later, report on its steps. It starts out as a plan for conducting the requirements elicitation process, but is meant to be updated as part of that process so that it includes the key lessons of elicitation.* [1]

Comme précédemment, il est indispensable de réfléchir aux moyens à mettre en œuvre pour mieux spécifier notre étude, notamment par des workshops avec les parties prenantes, des sondages, etc.
C'est une étape requise pour envisager de réaliser un nouveau produit. Elle reste en dehors de l'étude dans ce module.

# Annexe : Étapes de développement logiciel et Livrables

Ce document est principalement axé sur une première analyse des besoins de notre projet. **Nous allons la compléter ensemble.**

L'étape d'élicitation des exigences est difficile. Elle fait l'objet de cours dédiés sur l'élicitation des exigences par exemple à McMaster. Ici elle n'est pas notre objectif premier. Nous nous positionnons davantage en conception et visons à nous assurer de la cohérence des spécifications avec lesquelles nous allons travailler.

Ce projet englobe pratiquement toutes les étapes du développement d'un logiciel. Pour vous donner un aperçu complet de ce que nous allons couvrir, nous allons décomposer le projet en plusieurs étapes clés, chacune accompagnée de ses livrables correspondants. Pour cela nous décomposons notre analyse en suivant les étapes décrites dans l'article d'IBM : Steps in the software development process[6] . Notez bien que comme dans ce document, il ne s'agit pas d'une décomposition temporelle de construction du projet.

Nous verrons également comment ces étapes sont liées au présent document. Cela vous aidera à comprendre comment cette analyse des besoins s'intègre dans le processus de développement global du logiciel que nous entreprenons.

## Responsabilités et Rôles

En accord avec les rôles définis en P1, les responsabilités associées aux livrables sont susceptibles d'évoluer en fonction de l'avancement du projet et des contenus qui seront effectivement demandés. Leur explicitation vise uniquement à mieux circonscrire la portée des livrables.

**Nous rappelons néanmoins que toute l'équipe est responsable, et fière, de l'ensemble des résultats produits et en a donc aussi la « culpabilité ».**

C'est donc bien toute l'équipe qui est évaluée. Cependant, cette évaluation sera pondérée en fonction des responsabilités individuelles. Par exemple, un projet avec des performances globales modérées mais une excellente conception et des objectifs bien justifiés, mais avec une intégration continue non fonctionnelle, pourrait entraîner un bonus pour le Product Owner (PO) et une pénalité pour le Repository Manager(Ops). Toute l'équipe est donc bien impliquée dans toutes les tâches, mais chacun, selon son rôle, doit plus particulièrement veiller à ce que les choses soient bien faites et à guider ses camarades.

## Développement et Livrables

**Gathering requirements** to understand and document what is required by users and other stakeholders.

- Vous complétez le présent document (E1) en posant le vocabulaire partagé dans l'équipe et avec toutes les parties prenantes
    - *Deliverables: D1*
        - Responsabilité(s) : PO **(**toute l'équipe doit utiliser les mêmes termes)
- Vous complétez le présent document (**G5**) par une formalisation des exigences par différents diagrammes UML.
    - *Deliverables: D1 (Use case diagram [3]* ***)***
        - Responsabilité(s) : PO
    - *Deliverables: D1 (Sequence diagrams [2])*
        - Responsabilité(s) : PO(fond), SA(interactions)
    - *Deliverables: D1 (Class diagram at domain level [4] )*
        - *Responsabilité(s) :* PO(compréhension du domaine), SA(séparation)
- Vous complétez le présent document en précisant les exigences
    - *Deliverables: D0*
        - Responsabilité(s) : SD

**Choosing or building an architecture** as the underlying structure within which the software will operate.

- Vous complétez le présent document **(S1)** en explicitant l'architecture choisie. Ce même diagramme sera utilisé dans la partie **modélisation** ci-après. Dans notre contexte d'étude ces deux étapes sont fusionnées en un seul livrable.
    - *Deliverable: D2 (Architecture)*
        - Responsabilité(s) : SA
- Vous complétez le présent document **(S3)** en explicitant les interfaces d'accès à vos composants.
    - *Deliverable: D2*
        - *Responsabilité(s) :* SA, PO (couverture fonctionnelle).

**Developing a design** around solutions to the problems presented by requirements, often involving process models and storyboards.

- Vous complétez le présent document **(S2)** en explicitant les features que vous allez tester. Pour cela vous définissez des cas de tests. Ces mêmes tests seront utilisés dans la partie **Testing** ci-après. Dans notre contexte d'étude ces deux étapes sont fusionnées en un seul livrable.
  - *Deliverable: F1*
    - Responsabilité(s) : SD, PO, SA, Ops
- Vous complétez le présent document (**S4**) en explicitant les histoires utilisateurs
  - *Deliverable F1 (User stories and Tasks)*
    - Responsabilité(s) : Ops(dépôt), PO (contenu)

**Building a model** with a modeling tool that uses a modeling language like SysML or UML to conduct early validation, prototyping and simulation of the design.

- Cette section a déjà été couverte partiellement
  - *Voir livrable D1 essentiellement*
- Verification and acceptance criteria (**S6**) vont vous permettre de renforcer vos modèles
  - *voir livrable D1 et F1.*
- Vous complétez le présent document (**S1**) par votre structuration du projet en utilisant les design patterns.
  - *Deliverable: D1 et F1: Project quality assessment (part Design Patterns)*
    - Responsabilité(s) : SA, QA, SD

**Constructing code** in the appropriate programming language. Involves peer and team review to eliminate problems early and produce quality software faster.

- Ce document ne fait pas vraiment référence aux codes. Il aborde ce point uniquement au travers de l'organisation en P3 et P4. Notez cependant que votre code sera évalué non seulement en tant que tel mais également au travers des modèles rendus, notamment dans D2, qui sont l'abstraction de vos codes.
  - *Deliverable: F1(Codes)*
    - Responsabilité(s) : SD

**Testing** with pre-planned scenarios as part of software design and coding — and conducting performance testing to simulate load testing on the application.

- Voir livrable F1

**Managing configuration and defects** to understand all the software artifacts (requirements, design, code, test) and build distinct versions of the software. Establish quality assurance priorities and release criteria to address and track defects.

- Vous complétez le présent document (**E3**) par un retour sur les exigences non fonctionnelles (Contraintes) associées au projet
  - *Deliverable: D0 et D1*
    - Responsabilité(s) : SA, QA

# References

[1] Bertrand Meyer. Handbook of Requirements and Business Analysis. Springer. 2022.

[2] Sequence diagrams, IBM, https://www.ibm.com/docs/en/rsm/7.5.0?topic=uml-sequence-diagrams

[3] Use-case diagrams, IBM, https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-use-case

[4] Booch, Grady & Rumbaugh, James & Jacobson, Ivar. (1999). Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series). J. Database Manag.. 10. https://www.researchgate.net/publication/234785986_Unified_Modeling_Language_User_Guide_The_2nd_Edition_Addison-Wesley_Object_Technology_Series

[5] UML 2 en action : de l'analyse des besoins à la conception / Pascal Roques, Franck Vallée Roques, Pascal; Vallée Franck, Paris : Eyrolles; DL 2007

[6] IBM, “Steps in the software development process.” https://www.ibm.com/topics/software-development, consultee le 1/9/23