Requirements Specification

Imane Fouad, UM6P

1 Objectives

- Understand how to extract and organize system requirements.
- Learn the basics of identifying actors and use cases.
- Practice modeling requirements using a use case diagram.

Exercise (Warming up)

Consider the following software and systems, estimated in millions of lines of code (for readability, Google services are not shown, they account for **2,000 million lines**).

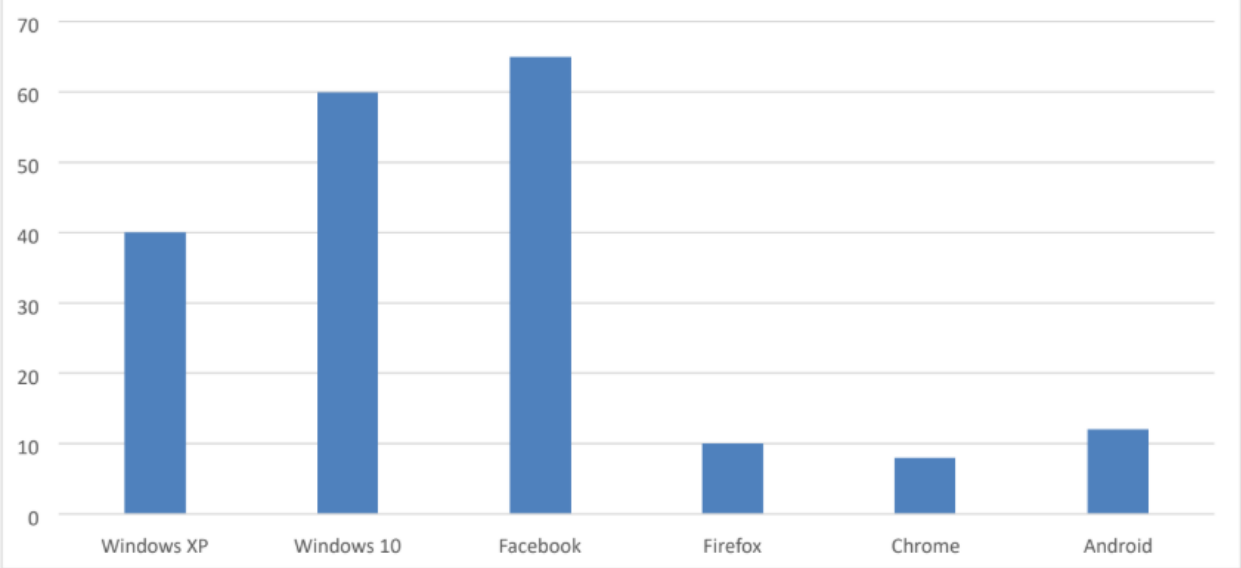

Figure 1: System sizes (millions of lines of code)

1. Suppose your productivity is **100 lines of code per day**. How long would it take you, working alone, to "implement Firefox", How long for Google? Express your answer in days, years.
2. If you got question 1 right, you should be amazed! In your opinion, how did these companies succeed despite such enormous amounts of code?
3. Lines of code are often cited as an indicator of productivity.
   - Imagine two developers: one writes 500 lines to solve a problem, another only 50. Who is more productive? Why?
   - Propose better ways to evaluate developer or team productivity.

4. Imagine a future where AI generates most of the code. What would remain as the unique contribution of human developers?

Exercise 2

Based on the following description, create a use case diagram for a ticket distributor system for trains:

- **Actors:**
  - Traveler: A traveler who purchases different types of tickets.
  - Central Computer System: Maintains the reference database for the system.
- **Use Cases:**
  - BuyOneWayTicket: Represents the purchase of a one-way ticket.
  - BuyWeeklyCard: Represents the purchase of a weekly card.
  - BuyMonthlyCard: Represents the purchase of a monthly card.
  - UpdateTariff: Represents updating tariff details.

Exercice 3

A recruitment company wants to develop an innovative web system that relieves hiring companies of most of the routine recruitment tasks. The principle is that when a job applicant reaches the interview stage, they have the maximum chance of obtaining the position.

The system is composed of several modules. The first module, "CV Builder", allows job applicants to create their CVs.

Once the CV is validated by the applicant, it is reviewed by a career expert through the "Orientation" module. If the CV is well-prepared, the career expert adds it to a shortlist corresponding to a specific job position and then submits this shortlist to the client manager. If the CV has gaps, the career expert formalizes their recommendations and sends them to the applicant, who then addresses them and resubmits the CV.

The client manager receives the shortlists in the "Tests" module and creates automated tests, which they invite all shortlisted candidates to take. Once the automated evaluations are completed, the client manager sends the list of candidates who passed the tests to the recruiter and informs unsuccessful candidates of the skill areas they need to improve.

Weekly, based on profile analysis and ongoing job postings, the system sends job announcements to users for positions that match their preferences.

- Give **5 functional requirements** of the system. For each requirement, indicate priority, and Estimated effort
- Draw the **use case diagram** of the system, keeping the modules as described above. Highlight each module using system boundaries.

Exercise 4

A start-up wants to develop a hotel management software system that can be used via the web or a mobile application.

A hotel client can make a reservation either traditionally by phone, or via the web or mobile app. The client can make an online payment using a CIB card. The payment is validated by the client's bank server, and communication with the bank server must use SSL.

When the client arrives at the hotel, the receptionist uses the application to find an available room matching the client's criteria. The receptionist can also verify the client's reservation. Once the room is found, the receptionist performs a check-in, changing the room status from "available" to "occupied," and a stay is created for the chosen duration, with an associated invoice.

During the stay, the client may add consumptions (breakfast, lunch, or dinner), and cashiers add these consumptions to the invoice using the application's Billing module.

Upon checkout, using the application, the receptionist closes the client's stay and invoice and receives the payment. Once freed, the room status changes to "dirty available."

Cleaning staff clean all rooms with the status "dirty available" and mark them as "clean waiting" in the Cleaning module of the application. The floor manager verifies that the room is properly cleaned and marks its status as "available."

If the room has noticeable damages (air conditioner failure, leaks, etc.), the floor manager sets the room status to "unavailable," and the room becomes available again only after maintenance intervention.

For the system development, the team chose Node.js for the backend, and Android for the mobile applications.

1. Based on the text, identify the **actors** of the system.
2. Identify the **actions that occur outside the system**.
3. If we wanted to decompose the system into boundaries, what would these boundaries be?
4. Draw the **use case diagram** of the system with functional boundaries (modules).

Exercise - Reverse engineering

Reverse engineering consists of studying a system in order to reproduce it entirely or partially, possibly with improvements. Sometimes reverse engineering stops at the analysis of the existing system.

The objective of this exercise is to perform reverse engineering on an existing system in order to provide a requirements model.

1.1 Step 1: Group Formation

Each group consists of two (2) students. Groups are formed based on seating: each student will work with the person sitting directly in front of or behind them.

1.2 Step 2: Choice of System to Study

Each group will be randomly assigned one of the following web applications/software:

Facebook	LinkedIn	YouTube
Google Doc	Google Maps	Twitter

1.3 Step 3: System Analysis

- Write **30–60 precise, non-redundant formal requirements** for the selected system.
  - At least 5 requirements should address exceptional cases (e.g., errors, failures, or unusual behavior).
  - At least 5 requirements should address non-functional aspects (e.g., security, performance, usability, etc.).
- Create and draw the use case diagrams for the system.

Clearly document any assumptions you make about the system that are not explicitly stated.

Note

To succeed in this exercise:

1. **List all core functions** of the system (e.g., sign-up, login, post content, comment, send messages).
2. **Explore each function** carefully and write corresponding functional requirements.
3. **Push the limits** by testing wrong inputs or unusual behavior (e.g., wrong password, invalid file, ...). From this, write exceptional requirements.
4. **Check non-functional aspects** such as speed, security, reliability, and usability. Translate these into non-functional requirements.
5. **Organize** all requirements into a clear list (30–60 items total).
6. **Go beyond surface-level features.** Think critically about hidden rules, exceptional behaviors, and quality constraints of the system.
7. **Check** official system documentation and cite your sources properly in your report.