Week 5 – Requirements Engineering

CSC 317 / INF 307 – SOFTWARE ENGINEERING
2021/2022 Academic Year
Semester 1

Recommended Reading

- **Chapter 8 & 9** of “Software Engineering: a practitioner’s approach” by Roger S. Pressman & Bruce R. Maxim

Scenario

A client approaches you to build a system for them:

- Where do you start?
- What do you do first?
- What questions would you ask them?

Scenario

A client approaches you to build a system for them:

- Where do you start?
- What do you do first?
- What questions would you ask them?

It all comes down to first identifying the requirements of the system

Today

- Introduction
  - What are requirements?
  - Types of Requirements
  - Importance of requirements
- Requirement Engineering Process
- Use Case Modeling

Software Requirements

- According to Sommerville (2011), *"requirements are the description of what the system should do - the services that it provides and the constraints on its operation"*.
  - A software system has to offer some services i.e., the things or functionalities it must perform in order to be successful.
  - There are also certain conditions (constraints) that must be satisfied before the software can be considered successful

Software Requirements

- Software requirements are very important
  - They help us to better understand the client's problems and also ensure that we are solving the right problem for the client.
  - A poor understanding of the goals of the system can lead to an incorrectly implemented system
  - Having a mutual understanding of requirements allows easier planning and estimation of the timelines and budgets for the project.

Levels of Requirements

Business requirements

- They are the highest level of requirements
- They describe the reason for creating the system from the perspective of the entire business
- These requirements outline the match between the goals of the business and the goals of the system

Levels of Requirements

Business requirements

- Scenario: Consider a university that wants to obtain a software for online registration and fee payment
  - What are the goals of the organization?
  - How are these goals met by the intended system?

Levels of Requirements

Business requirements

- Scenario: Consider a university that wants to obtain a software for online registration and fee payment
  - *"The system will allow the university to automatically process a high number of student registration requests in order to save time."*
  - *"The system will allow the university to keep efficient records of received fee payments in order to have effective and transparent audits of our financial transactions."*

Levels of Requirements

User requirements

- These are from the perspective of the users of the system
- There can be different types of users with similar, overlapping, and different requirements.

Levels of Requirements

User requirements

- Using the same scenario of the online registration and fee payment system for a University
  - What are the different categories of users?
  - In what ways would these user categories use the intended system?

Levels of Requirements

User requirements

- *A common format for stating user requirements*
  - *"The [user type] shall [interact with the software] in order to [meet a business goal or achieve a result]."*
  - *"The registrar shall indicate the courses available in a given semester in order to maintain a current and complete list for registration by students."*
- Can you provide a user requirement for a student?

Levels of Requirements

System requirements

- These are the more detailed descriptions of the functions, services, and constraints that the system would provide to the user and the organization.
- Further classified into:
  - Functional requirements
  - Non-functional requirements

Functional Requirements

- Functional requirements describe what a software system should do and is some cases what the system should not do.
- Varies from one system to another
  - Very dependent on the type of system, users, and goals of the business

Functional Requirements

- Common format
  - *"The system shall [functionality or service to be provided]"*

- *Examples:*
  - *"The system shall generate each semester, for each student, a set of courses that the student is eligible to register."*
  - *"The system shall generate a unique receipt number for each fee payment received."*

Functional Requirements

- Functional requirements should be:
  - Specific and unambiguous
  - Measurable and observable
  - Testable

- What are some functional requirements for:
  - An online quiz system
  - Music player

Non-functional Requirements

- Non-functional requirements are not about the actual functionalities of the system
- They are concerned with the constraints and properties of the system.
- The common non-functional requirements are related to aspects of the system such as performance, security, scalability, availability, and usability.

Non-functional Requirements

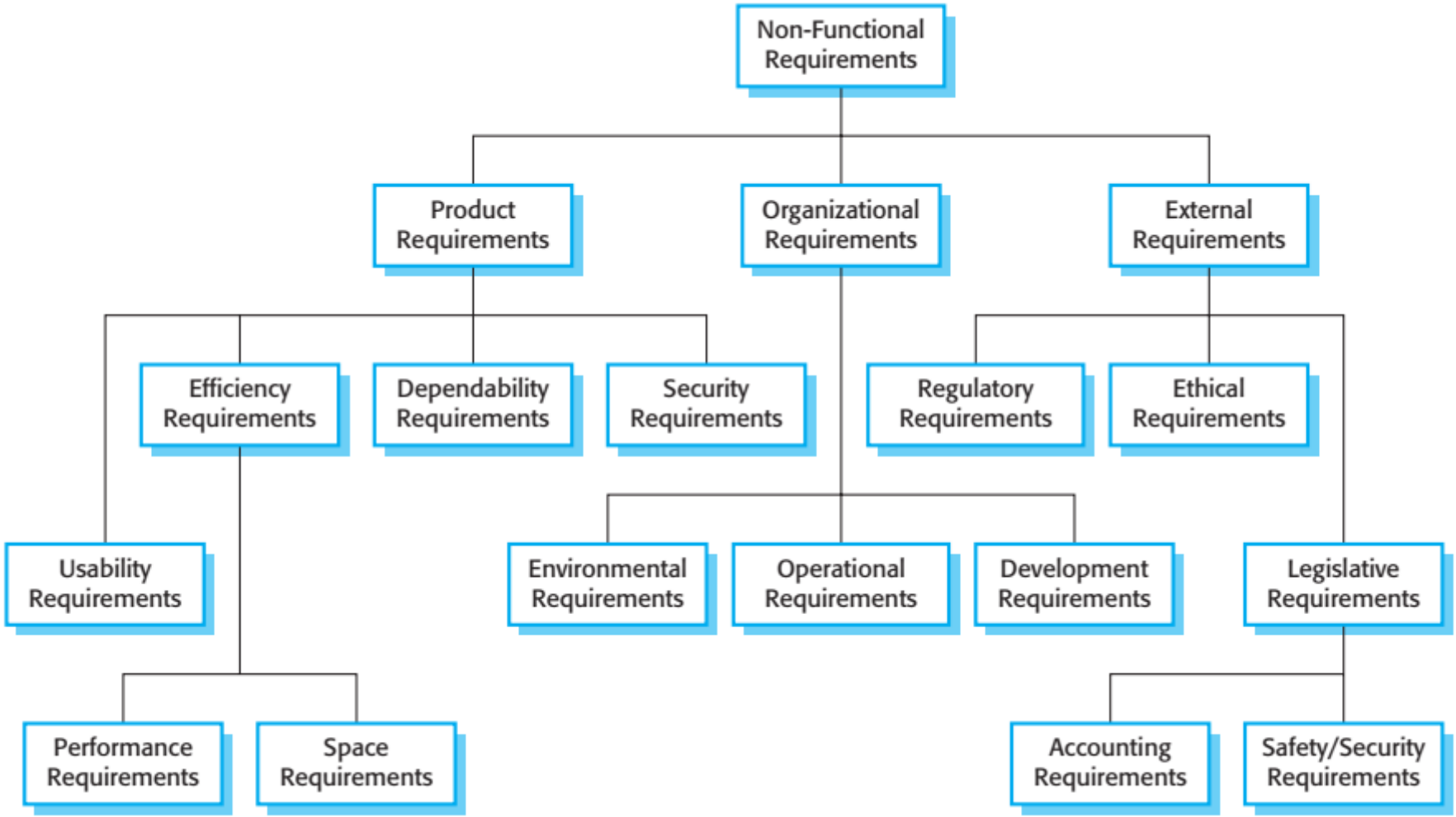


Figure source: Sommerville (2011, p.88)

Functional vs. Non-functional Requirements

- NFRs are targeted to the system as a whole and not to some particular functionality.
  - E.g., the performance and security of a system is not a functionality provided to specific users but a characteristic of the system as a whole.
- A single NFR can have several related functional requirements.
  - For example, a requirement such as “the system shall be accessed by only users with authenticated usernames and passwords” is a functional requirement that is related to the security NFR.

Functional vs. Non-functional Requirements

- Often harder to deal with
  - They cut across the entire system (i.e. not limited to one functionality)
  - They concern the general properties of the system
  - Difficult to specify their success criteria
- A problem with an NFR may require a total redo of the system
  - A system that can perform all its functionalities perfectly can still be rendered useless if it takes a very long time for a single functionality to be completed.

Understanding Requirements is HARD

Expectation

- The customer should know what is required.
- End users should have an understanding of the important features of a system.

Reality

- Unfortunately, this is not always the case
- Customer and end-user needs change throughout the project

Today

- Introduction
  - What are requirements?
  - Types of Requirements
  - Importance of requirements
- Requirement Engineering Process
- Use Case Modeling

Understanding Requirements is HARD

- SE engineers struggle to elicit requirements
  - The acquired information is difficult to understand
  - Requirements are often acquired in a disorganized manner
  - Difficult to verify recorded information
  - Lack of mechanisms
- **Requirements Engineering** provides a spectrum of tasks and techniques to make the understanding of requirements much easier

Requirements Engineering

"Requirements engineering is the branch of software engineering **concerned with the real-world goals** for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors **to precise specifications of software behavior**, and to their **evolution over time** and across software families"
- Zave, P. (1997)

Requirement Engineering Tasks

1. Inception
2. Elicitation
3. Elaboration
4. Negotiation
5. Specification
6. Validation
7. Management

N.B. These tasks usually occur in parallel

Inception

- Aimed at providing an initial understanding of the problem being addressed by the client or organization
- An initial description of the project scope is determined through a feasibility analysis
  - What is the problem?
  - Who requires a solution?
  - Nature of the solution?

Inception – the Five Steps

1. Gain agreement on the problem definition.
2. Understand the root causes (not symptoms) — the problem behind the problem.
3. Identify the stakeholders, especially users.
4. Define the solution system boundary.
5. Identify the constraints to be imposed on the solution (e.g., technology, economics, political, schedule/resources etc.)

Elicitation

- AKA capturing the requirements
- The purpose is to capture business goals from the stakeholders
  - What are the objectives of the system?
  - How does the system fit into the business' needs?
  - How will the system be used in a day-to-day basis?

Elicitation - Major problems faced

- Ill-defined system boundaries (scope)
- Customers unsure of needs
- Lack of full understanding of problem domain
- Customers unable to clearly communicate needs
- Omitted information
- Ambiguous/conflicting requirements

Elicitation - Techniques

- Storyboards
  - Uses sketches, pictures etc. to provide an animated/pictorial view of the system behavior
  - Example can be found here
- Interviews and questionnaires
- Requirements workshop
  - Gathers all stakeholders for a short but intense focused period
  - Involves brainstorming/Idea reduction sessions

Elaboration

- The information gathered from the elicitation task is expanded and refined
- During this elaboration process, we may identify conflicting requirements or requirements that exceed the specified constraints.

Negotiation

- This task involves reconciling (eliminating, combining, modifying) conflicting requirements
  - Requirements are prioritized
  - Costs and risks are analysed

Specification

- This task involves documenting the requirements.
- Usually a combination of textual descriptions and graphical models of the requirements
- The output is the Requirement Specification document
- A template can be found here

Validation

- Ensures that all documented requirements are unambiguous: having no errors, omissions or inconsistencies.
- Done by a review team consisting of customers, users, software engineers and other business stakeholders
- The Requirement Specification becomes official **only after** it has been validated

Validation

- Some important questions that must be asked of each documented requirement are:
  - Is the requirement consistent with the overall objectives for the system or product?
  - Is the requirement really necessary?
  - Is the requirement bounded and unambiguous?
  - Is the source of the requirement indicated?
  - Is there a conflict with any other requirements?
  - Is the requirement implementable?
  - Can the requirement be tested once implemented?

Management

- Requirements change throughout the lifetime of a system
- Requirement management involves identifying and tracking changes to requirements over time.

TLDR

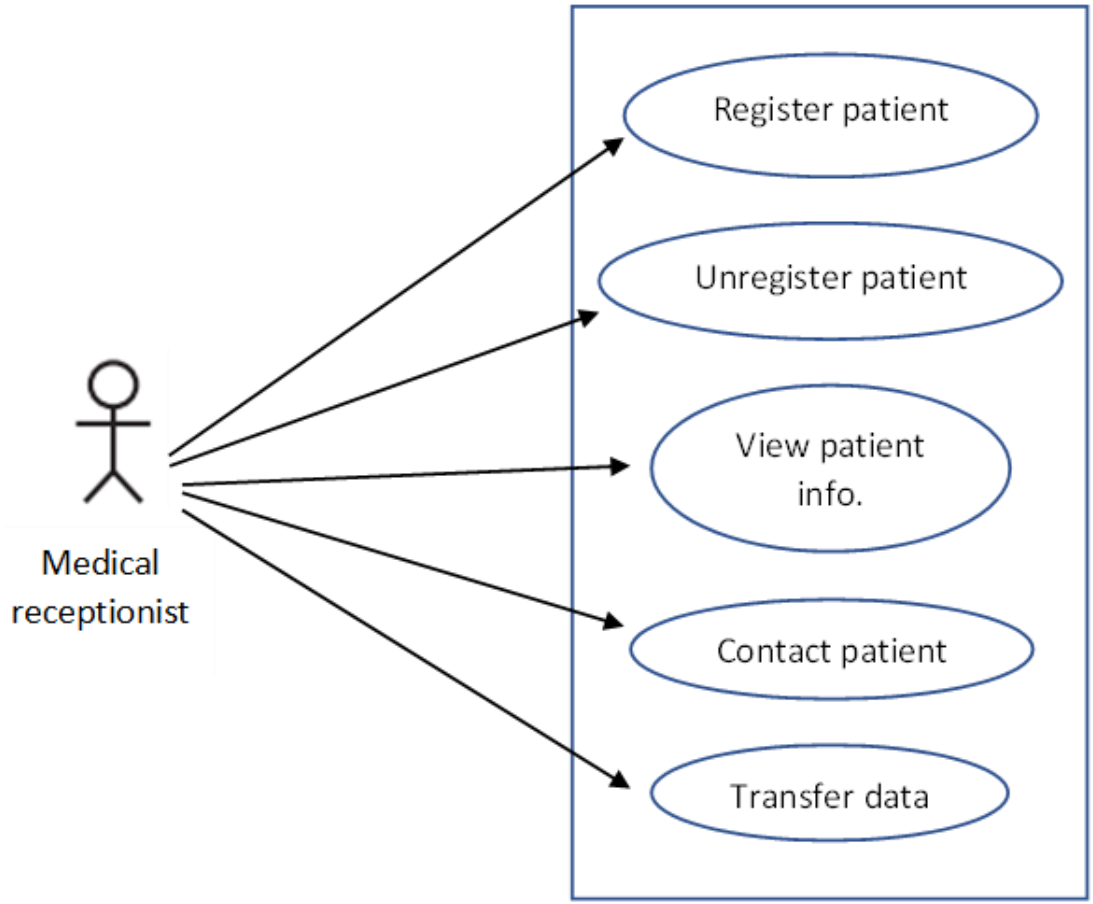
- Make sure you are solving the right problem.
- Do not get caught up in the shiny tools, frameworks, technologies, etc.
- Understand the customer's problem
- Importantly, make sure the customer understands their problem

Today

- Introduction
  - What are requirements?
  - Types of Requirements
  - Importance of requirements
- Requirement Engineering Process
- Use Case Modeling

Use Case Modeling

- What do you make of this diagram?



Use Case Modeling

- Is a way to specify functional requirements
- Is notated using a **use case specification**
- Is not part of the Unified Modeling Language (UML), but is many times used in conjunction with it

Use Case Modeling - Notations

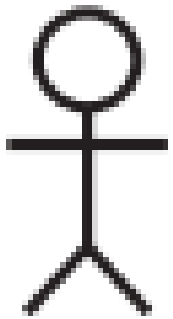
Use Case

- A use case is a single system function with which a user can interact.
  - For example, ‘register patient’

Use Case Modeling - Notations

Actor

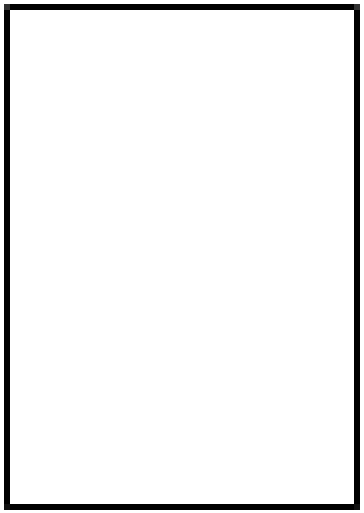
- An actor is a role of an interactor of the system,
  - can be either an individual, organization or another system.
  - **Primary actor**: They initiate an interaction with the system to achieve a goal.
  - **Secondary actor**: Provide a service to the system. Is almost never a person



Use Case Modeling - Notations

System Boundary

- All functionality that are provided by the system must be contained within a system boundary.



Use Case Modeling - Notations

Connections

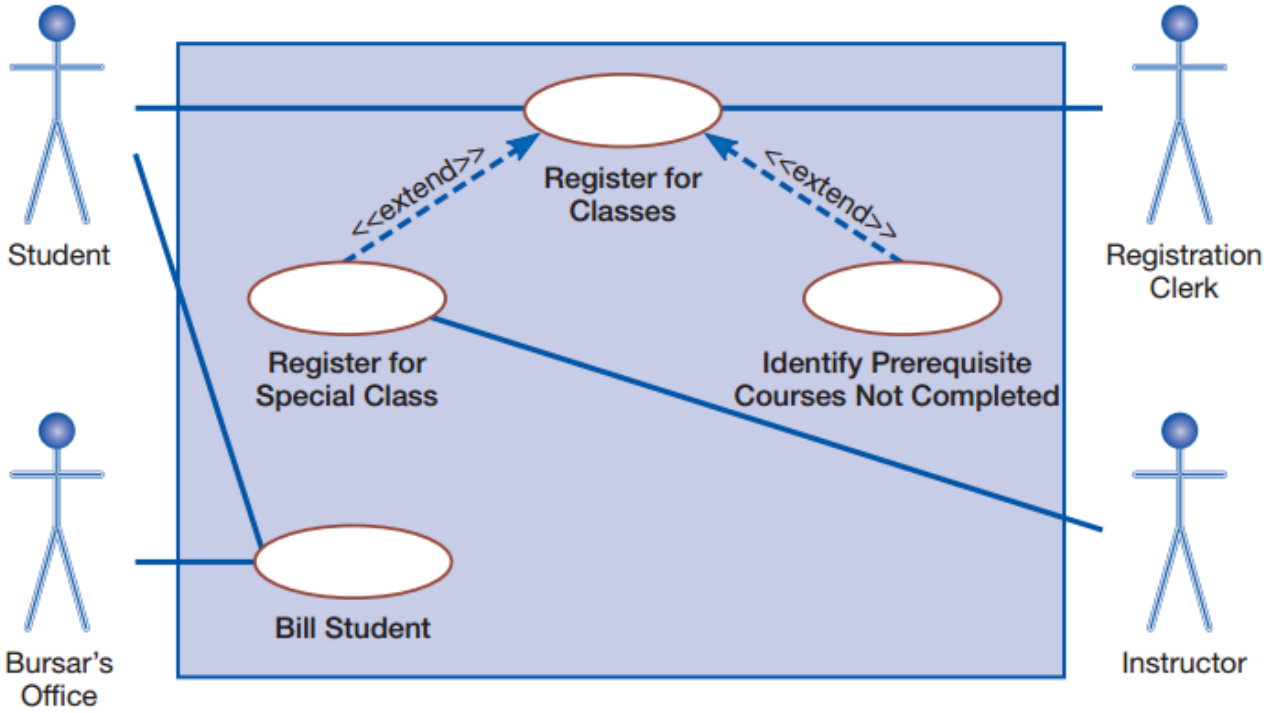
- used to indicate the relationship between actors and use cases, and also between two use cases
  - Simple connectors 
  - Includes 

  - Extends 

Use Case Modeling - Notations

Connections

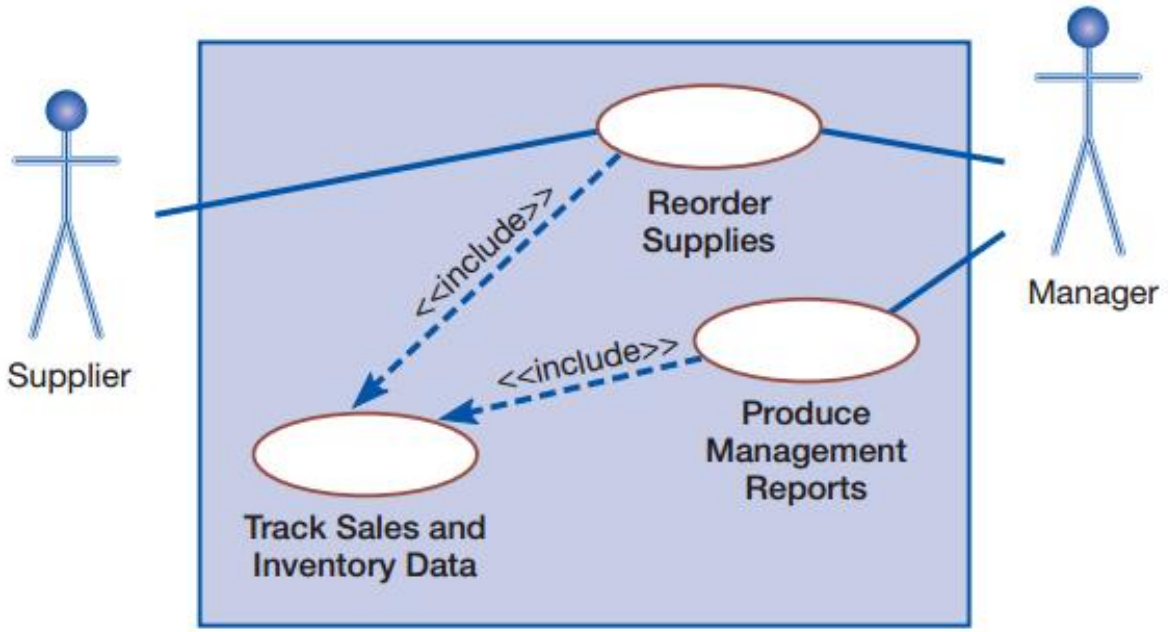
The extend relationship is used to indicate that **one use case is the extension of another**. This is when one use case is created by adding additional behaviour to an existing use case.



Use Case Modeling - Notations

Connections

The include relationship is used to indicate that **one use case is included within another.**



Use Case Specification: Natural Language Example

Use Case 1. Withdraw Money

The system displays the account types available to be withdrawn from and the user indicates the desired type. The system asks for the amount to be withdrawn and the user specifies it. Next, the system debits the user's account and dispenses the money. The user removes the money, the system prints a receipt, and the user removes the receipt. Then the system displays a closing message and dispenses the user's ATM card. After the user removes his card, the system displays the welcome message

Use Case Specification Template

Number	*Unique use case number*	
Name	*Brief noun-verb phrase*	
Summary	*Brief summary of use case major actions*	
Priority	*1-5 (1 = lowest priority, 5 = highest priority)*	
Preconditions	*What needs to be true before use case "executes"*	
Postconditions	*What will be true after the use case successfully "executes"*	
Primary Actor(s)	*Primary actor name(s)*	
Secondary Actor(s)	*Secondary actor name(s)*	
Trigger	*The action that causes this use case to begin*	
Main Scenario	**Step**	**Action**
	Step #	*This is the "main success scenario" or "happy path."*
	…	*Description of steps in successful use case* "execution"
	…	*This should be in a "system-user-system, etc." format.*
Extensions	**Step**	**Branching Action**
	Step #	*Alternative paths that the use case may take*
Open Issues	*Issue #*	*Issues regarding the use case that need resolution*

- Adapted from A. Cockburn, "Basic Use Case Template"

Use Case Specification Template

Number	*Unique use case number*	
Name	*Brief noun-verb phrase*	
Summary	*Brief summary of use case major actions*	
Priority	*1-5 (1 = lowest priority, 5 = highest priority)*	
Preconditions	*What needs to be true before use case "executes"*	
Postconditions	*What will be true after the use case successfully "executes"*	
Primary Actor(s)	*Primary actor name(s)*	
Secondary Actor(s)	*Secondary actor name(s)*	
Trigger	*The action that causes th*	
Main Scenario	**Step**	**Action**
	Step #	*This is the "ma*
	…	*Description of*
	…	*This should be in a "system-user-system, etc." format.*
Extensions	**Step**	**Branching Action**
	Step #	*Alternative paths that the use case may take*
Open Issues	*Issue #*	*Issues regarding the use case that need resolution*

Extension

- **Could be an optional path(s)**
- **Could be an error path(s)**
- **Denoted in use case diagrams (UML) by <<extend>>**

- Adapted from A. Cockburn, "Basic Use Case Template"

Use Case Specification Template Example

Number	1
Name	Withdraw Money
Summary	User withdraws money from one of his/her accounts
Priority	5
Preconditions	User has logged into ATM
Postconditions	User has withdrawn money and received a receipt
Primary Actor(s)	Bank Customer
Secondary Actor(s)	Customer Accounts Database

Continued…

Use Case Specification Template Example

Trigger	User has chosen to withdraw money	
Main Scenario	**Step**	**Action**
	1	System displays account types
	2	User chooses account type
	3	System asks for amount to withdraw
	4	User enters amount
	5	System debits user's account and dispenses money
	6	User removes money
	7	System prints and dispenses receipt
	8	User removes receipt
	9	System displays closing message and dispenses user's ATM card
	11	User removes card
	10	System displays welcome message
Extensions	**Step**	**Branching Action**
	5a	System notifies user that account funds are insufficient
	5b	System gives current account balance
	5c	System exits option
Open Issues	1	Should the system ask if the user wants to see the balance?

In-class practice

Consider the problem statement for an "Online Auction System" to be developed:

New users can register to the system through an online process. By registering a user agrees to abide by different predefined terms and conditions as specified by the system. Any registered user can access the different features of the system authorized to him/her, after he authenticates himself through the login screen. An authenticated user can put items in the system for auction. Authenticated users can place bid for an item. Once the auction is over, the item will be sold to the user placing the maximum bid. Payments are to be made by third party payment services, which, of course, is guaranteed to be secure. The user selling the item will be responsible for its shipping. If the seller thinks he's getting a good price, he can, however, sell the item at any point of time to the maximum bidder available.

1. Identify a list of functional requirements
2. Identify a list of non-functional requirements
3. Draw a suitable use case diagram for the above system

Some additional resources

- UML Design Tools
  - VioletUML
  - StarUML
  - Visio
  - ArgoUML

References

- Roger S. Pressman & Bruce R. Maxim (2014). Software Engineering: a practitioner's approach, 8th edition: McGraw-Hill
- Larman, C., (2012). Applying UML and patterns: an introduction to object oriented analysis and design and interative development. Pearson Education India
- Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. ACM Computing Surveys, 29(4): 315-321