Requirement engineering

Week 3 lectures keywords:

Requirements, Functional requirements; non-functional requirements; Use case diagram;

actor; use case; include and extend relationship;

Requirements in software engineering

System requirements are a set of specifications that define the functions, features, and characteristics that a system or software application must possess to meet the needs of its users and stakeholders. These requirements typically include the hardware and software specifications, performance standards, security requirements, compatibility requirements, and other criteria that define the capabilities of the system or application.

In software engineering, requirements refer to the functional and non-functional needs and constraints that a software system or application must meet to satisfy the stakeholders.

Functional requirements describe what the software should do or accomplish, including specific features, behaviors, and functionality.

- Statements of services the system should provide;
- How the system should react to particular inputs;
- How the system should behave in particular situations;
- May also state what the system should NOT do.

Non-functional requirements, on the other hand, define how the software should perform, including factors such as reliability, usability, security, and performance.

Requirements gathering is a critical phase of software development, and it involves gathering, analyzing, and documenting the requirements to ensure that they are clear,

accurate, and complete. The requirements serve as a basis for design, development, testing, and maintenance of the software.

(how the functional requirements are realised)

- Constrains on the services or functions offered by system;
- Often apply to whole system, not just individual features.

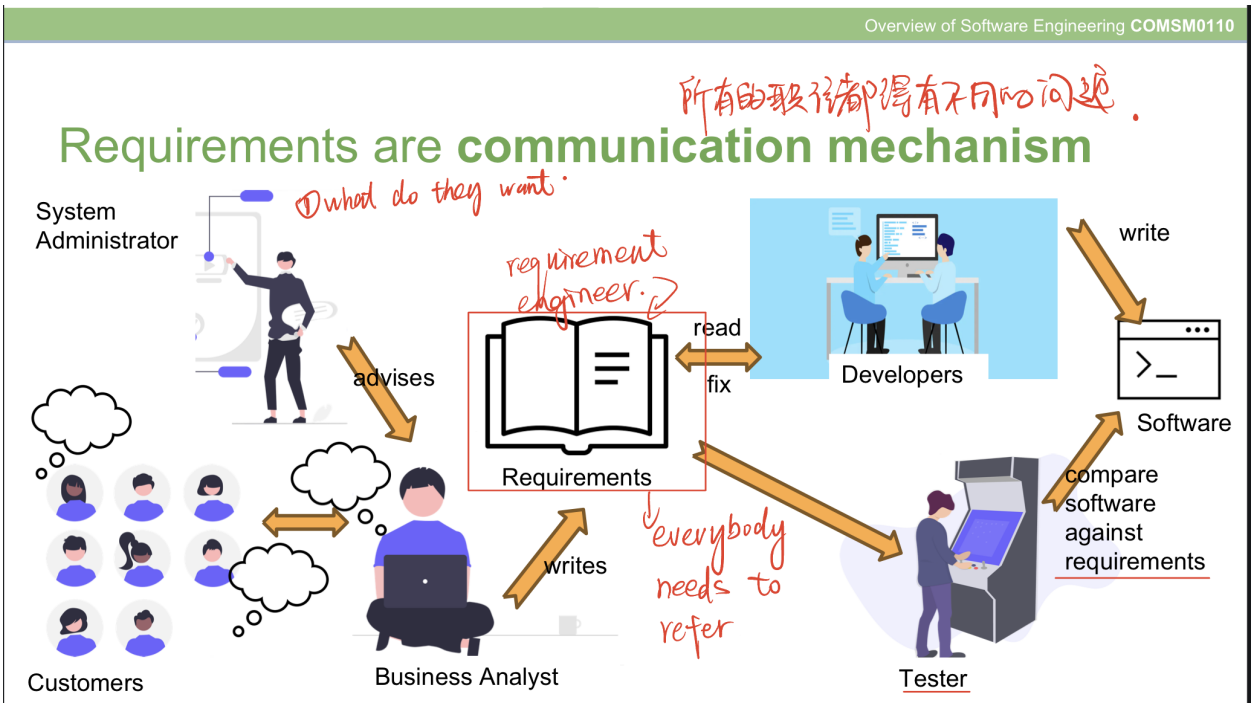
Why do we need requirements Engineering?

Requirements engineering is a critical process in software engineering that involves eliciting(激发) , analyzing（分析）, specifying（指定）, validating（认证）, and managing（管理） the requirements of a software system. There are several reasons why we need requirements engineering:

1. To understand the problem: Requirements engineering helps in understanding the problem that needs to be solved by the software system. By eliciting and analyzing the requirements, software engineers can identify the stakeholders, their needs, and the scope of the problem.
2. To ensure customer satisfaction: The requirements engineering process helps in capturing the requirements of the customers and other stakeholders. By involving the customers in the process, software engineers can ensure that the software system meets the needs and expectations of the customers.
3. To avoid costly errors: Requirements engineering helps in identifying and resolving the issues and conflicts that may arise during the development process. By defining the requirements clearly, software engineers can avoid costly errors and rework in the later stages of development.
4. To facilitate communication: Requirements engineering involves a collaborative process that involves the customers, developers, and other stakeholders. By facilitating communication and collaboration, software engineers can ensure that everyone is on the same page and working towards the same goal.
5. To manage project scope: Requirements engineering helps in managing the scope of the project. By defining the requirements and setting priorities, software

engineers can ensure that the project stays on track and delivers the desired outcomes within the given time and budget constraints.

Requirements are communication mechanism（通信机制)



Requirements serve as a communication mechanism between stakeholders, such as customers, users, developers, and testers. They define the scope and purpose of the software to be developed, and provide a common understanding of the problem to be solved. Requirements also help to identify potential conflicts or issues early on in the development process, and ensure that the final product meets the needs and expectations of the stakeholders. By having clear and well-defined requirements, all stakeholders can work towards a common goal and reduce the risk of misunderstandings or misinterpretations.

Requirements are acceptance criteria (验收标准)

Requirements, refer to a broader set of criteria or specifications that describe what the product or feature should do, how it should perform, and what features or functionality it should have. In other words, acceptance criteria are a subset of requirements that focus on the end-user's needs and expectations. Requirements engineering helps to elicit, analyze, and manage these requirements throughout the software development lifecycle to ensure that the end product meets the stakeholders' needs and expectations.

To be able to fairly assess whether the team have produces something that matched what you asked for, the thing that you asked for must be:

- Unambiguous/ Precise
- Complete
- Understandable / clear

Analysing requirements:

Analyzing requirements is an important step in the requirements engineering process. It involves a deep **understanding of the customer's needs, wants**, and **expectations**, and **translating them into a detailed and comprehensive set of requirements** that can be used to design and develop a software system. Here are some steps to follow when analyzing requirements:

1. Understand the business context: It is important to understand the business context in which the software system will be used. This includes understanding the customer's business goals, objectives, and strategies.
2. Identify stakeholders (分析甲方): Identify all the stakeholders who will be impacted by the software system. This includes end-users, business owners, system administrators, and others.
3. Elicit requirements（引起需求）: Elicitation is the process of gathering requirements from stakeholders. This can be done through various techniques such as interviews, questionnaires, workshops, and surveys.

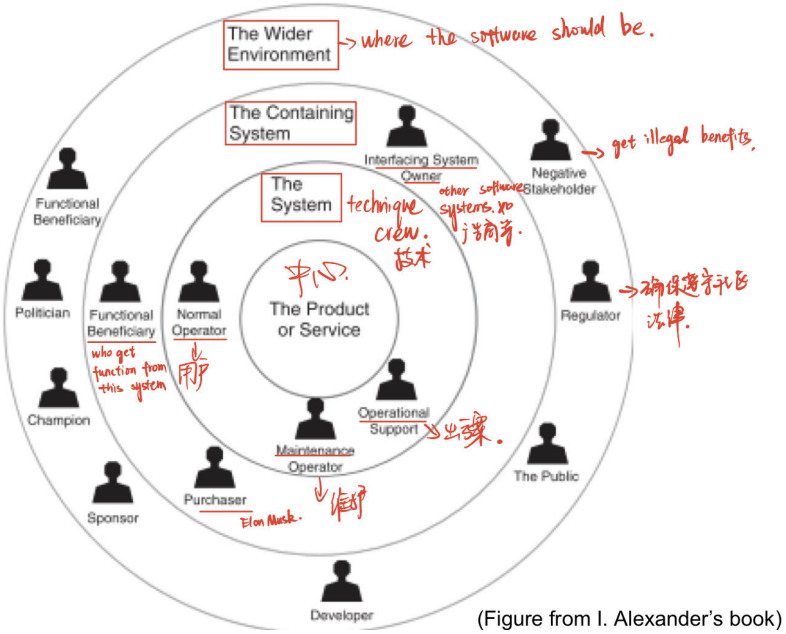
4. Document requirements: Once requirements have been elicited, they must be documented in a clear and concise manner. This includes capturing requirements in a format that is easy to understand, such as use cases or user stories.
5. Analyze requirements/identify top-level user needs: Analyzing requirements involves identifying any inconsistencies or conflicts between requirements. It also involves prioritizing requirements based on their importance to the customer. (e.g., as NFRs or “User stories”)
6. Validate requirements/ break down stories into individual steps: Validate the requirements by ensuring that they meet the customer's needs and that they are complete, accurate, and consistent.
7. Manage requirements/ specify atomic requirements : Requirements must be managed throughout the software development lifecycle. This includes tracking changes to requirements, ensuring that they are traceable, and ensuring that they are prioritized appropriately.

By following these steps, you can ensure that the requirements are analyzed thoroughly and accurately, and that the resulting software system meets the needs of the customer.

Identify stakeholders

The onion model

The Onion model



(Figure from I. Alexander's book)

1. Product or Service: This is the core layer, representing the product or service that is being developed or delivered. It includes the features, functions, and characteristics of the product or service.
2. System (technical crew): The second layer represents the system that supports the product or service. It includes the hardware, software, and other technology components required to deliver the product or service.

- normal opetator(系统维护，图中标错了);
- maintenance operator;
- and operational support;

3. Containing System: The third layer represents the containing system, which is the broader system or context in which the product or service is used. It includes the business processes, organizational structures, and other systems that the product or service interacts with.

- Functional beneficiary: the one who get fuction from this system;
- purchasers: the one who buy the system;
- interfacing system owner: other software system 如广告商等

4. Wider Environment: The outermost layer represents the wider environment in which the product or service operates. This includes factors such as economic, political, and social factors, as well as regulatory and legal considerations. —Where the software should be.

- Regulatory bodies that govern the use of the system, such as data protection laws or industry-specific regulations
- External systems or services that the system must interact with, such as payment gateways or third-party APIs
- End users or customers of the system who may have different needs or requirements
- Competing systems or products that may impact the system's success or adoption
- Social, economic, or political factors that may influence the system or its use, such as cultural norms or economic trends.
- Negative stakeholders are individuals, groups, or organizations who have a vested interest in the project but do not support its goals, objectives, or outcomes. They may be affected by the project in some way and could try to sabotage or oppose it for their own benefit or reasons. Examples of negative stakeholders include competitors, regulatory bodies, special interest groups, and individuals or groups with opposing views or agendas. Negative stakeholders can create significant challenges for a project team, and it is important to identify them early and develop strategies to manage their influence and impact on the project.

Stakeholders: (who will use it)

- clients;
- documentation, eg. organisation chart
- Templates
- Similar projects
- Analysing the context of the project

Surrogate stakeholders (代理利益相关者)

Surrogate stakeholders are individuals or groups who are not direct beneficiaries or users of the product or service, but who have an interest in the project or are affected by its outcome. They may include regulatory bodies, industry groups, government agencies, or other organizations with a stake in the project. Surrogate stakeholders are important to identify and involve in the requirements engineering process to ensure that all relevant perspectives are considered and addressed. e.g., legal, unavailable at present, mass product users.

Negative stakeholders (一些阻力)

Individuals or groups who may be impacted negatively by the project or product being developed. These stakeholders may have opposing views or interests and may be resistant to change, which can result in project delays, increased costs, or reduced quality. It is important to identify and manage negative stakeholders early in the project to minimize their impact and ensure project success. Strategies for managing negative stakeholders may include engaging them early in the project, addressing their concerns and issues, and finding common ground to work together towards project goals.

e.g.

1. Competitors who are trying to undermine the project or product
2. Customers who are dissatisfied with the product or service
3. Government or regulatory agencies that impose additional constraints or regulations
4. Unions or other employee groups who are resistant to changes
5. Suppliers who are unreliable or provide poor quality products
6. Activist groups who may protest against the project or product
7. Local communities who are impacted by the project or product and may oppose it
8. Internal groups or individuals who have conflicting goals or interests
9. Shareholders who may be unhappy with the direction or performance of the organization

10. Media or journalists who may report negative news or reviews about the project or product.

Identify top level needs/ concerns

1. Stakeholder Interviews: Conducting interviews with stakeholders to understand their requirements, concerns, and expectations.
2. Surveys: Using surveys to collect data on the needs and concerns of different stakeholders.
3. Focus Groups: Bringing together a group of stakeholders to discuss their needs and concerns in a facilitated discussion.
4. Observations: Observing stakeholders as they interact with the product or service to identify their needs and concerns.
5. Document Analysis: Reviewing documents such as business plans, strategic plans, and user manuals to identify key needs and concerns.
6. Brainstorming: Conducting brainstorming sessions with stakeholders to identify their needs and concerns.
7. Market Research: Conducting market research to understand the needs and concerns of potential customers and competitors.

Identify “User Stories”

A popular way to record user need:

Formula:

As a <type of user>, I want to <some goal> so that <some reason>.

1. Brainstorm: Gather all stakeholders and development team members together to brainstorm and identify potential user stories. This can involve using techniques such as mind mapping, affinity grouping, or card sorting.

2. Prioritize: Once a list of potential user stories has been identified, prioritize them according to business value or customer needs. This can involve using techniques such as MoSCoW prioritization or impact mapping.
3. Write user stories: After prioritizing the potential user stories, the development team should write them in a consistent format, such as "As a [user], I want [action], so that [benefit]."
4. Refine: Once user stories have been written, they should be refined and broken down into smaller, more manageable pieces. This can involve techniques such as slicing or splitting stories, or defining acceptance criteria.
5. Validate: Finally, user stories should be validated with stakeholders and end-users to ensure they accurately capture the intended requirements and deliver value. This can involve techniques such as user acceptance testing or usability testing.

System behaviour 🌟🌟

(how a system acts and reacts)

-it comprises the actions and activities of a system.

(System behavior is captures in use cases)

-Use cases describe the interactions between the system and(Parts of) it environment.

Refers to the way a system behaves and responds to different inputs or stimuli**刺激**. It includes the **various functions**, **operations, and activities that a system can perform to meet its intended purpose or objectives**. System behavior can be analyzed and described using different modeling techniques, such as use case diagrams, activity diagrams, state diagrams, or sequence diagrams, depending on the level of detail and complexity required. Understanding system behavior is important in software engineering to ensure that the system meets the needs and requirements of its users and stakeholders.

Use-case model

- The use-case model is a technique used in requirements engineering to describe the functional requirements of a system.
- It consists of actors, use cases, and relationships between them.
- Actors are users or other systems that interact with the system being developed.
- Use cases can be described in a narrative form or using a visual notation, such as UML.
- Use cases can be used to identify requirements, validate design decisions, and communicate system behavior to stakeholders.
- links stakeholder needs to software requirements
- serves as a planning tool
- Use cases can be prioritized and organized into a use-case diagram to provide an overview of the system's functionality.

A use-case model is comprises of:

use-case diagrams (visual representation)

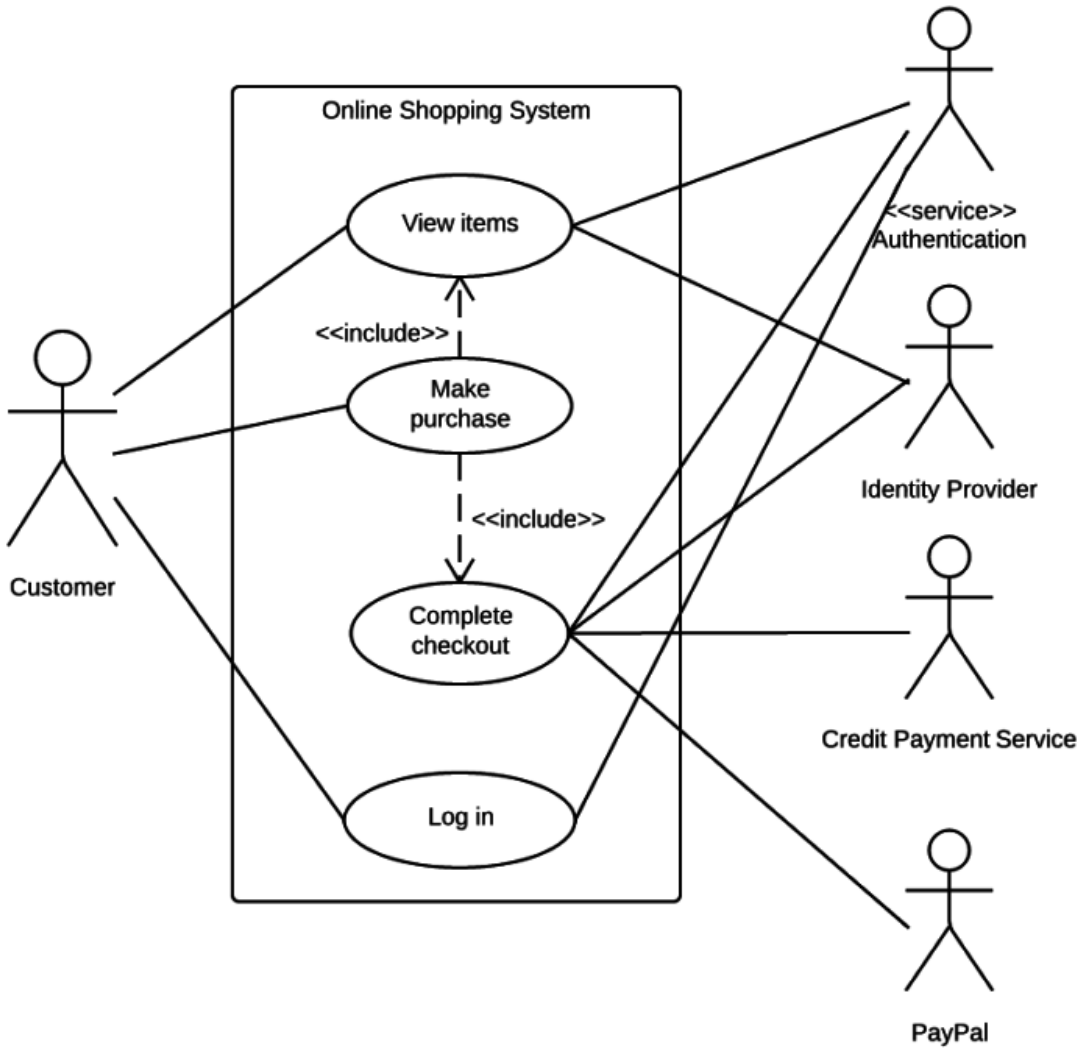
use-case specifications (text representation)

Use-case diagrams

A use case diagram is a UML (Unified Modeling Language) diagram that represents the behavior of a system by illustrating how actors (users or other systems) interact with that system. Use case diagrams are used to identify, clarify, and organize system requirements by demonstrating how users will interact with the system. Here are some key points about use case diagrams:

- A use case diagram displays the system's functions and how users or external systems interact with those functions. identifies who or what interacts with the system.
- The actors are represented by stick figures, while the use cases are represented by ovals.

- support requirements elicitation and capture by identifying the system's functional requirements.
- demonstrate the system's functional architecture and to model user scenarios and workflows.
- They help stakeholders understand the functional requirements of the system and the roles of different actors in the system.
- help with testing and validation by providing a visual representation of the system's behavior.
- Define clear boundaries of a system.
- Summarizes the behaviour of the system



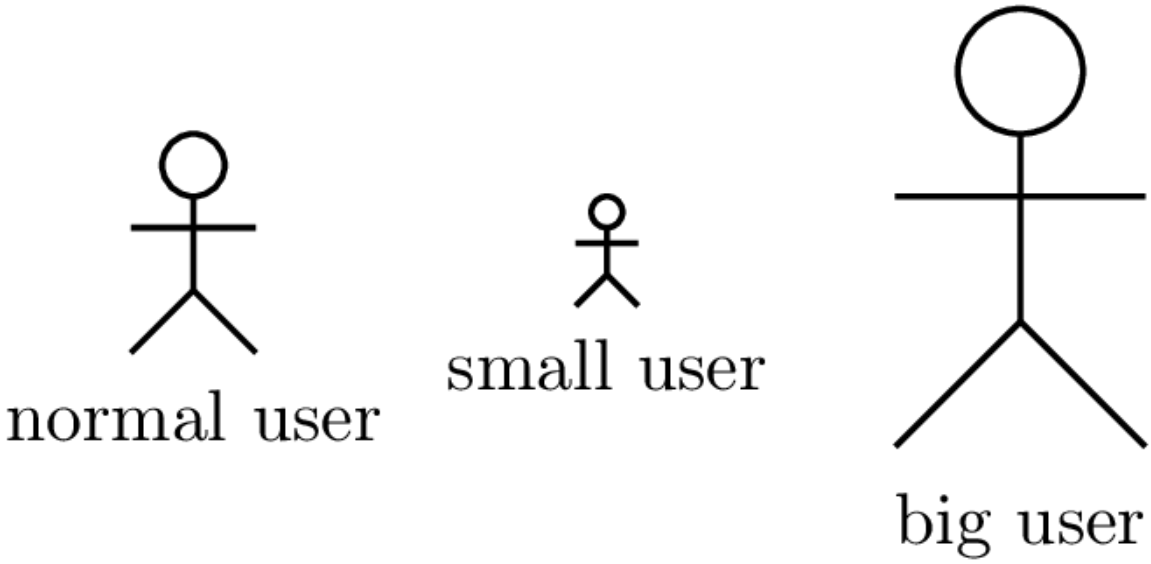
Benefits of a Use-case model

- Communication
- Identification
- Testing

1. Improved communication: Use-cases provide a clear and concise way of communicating software requirements to stakeholders, developers, and testers.
2. Better understanding of user needs: Use-cases help identify user goals and requirements and ensure that software meets those needs.
3. Early detection of problems: Use-cases can help identify potential problems or issues early in the development process, reducing the risk of costly rework later on.
4. Improved testing: Use-cases can serve as a basis for testing, ensuring that software meets user requirements and functions correctly.
5. Better project planning: Use-cases can help define project scope, prioritize requirements, and estimate project timelines and costs.
6. Enhanced user experience: Use-cases can help ensure that the software is designed with the user's needs and expectations in mind, leading to a better user experience.

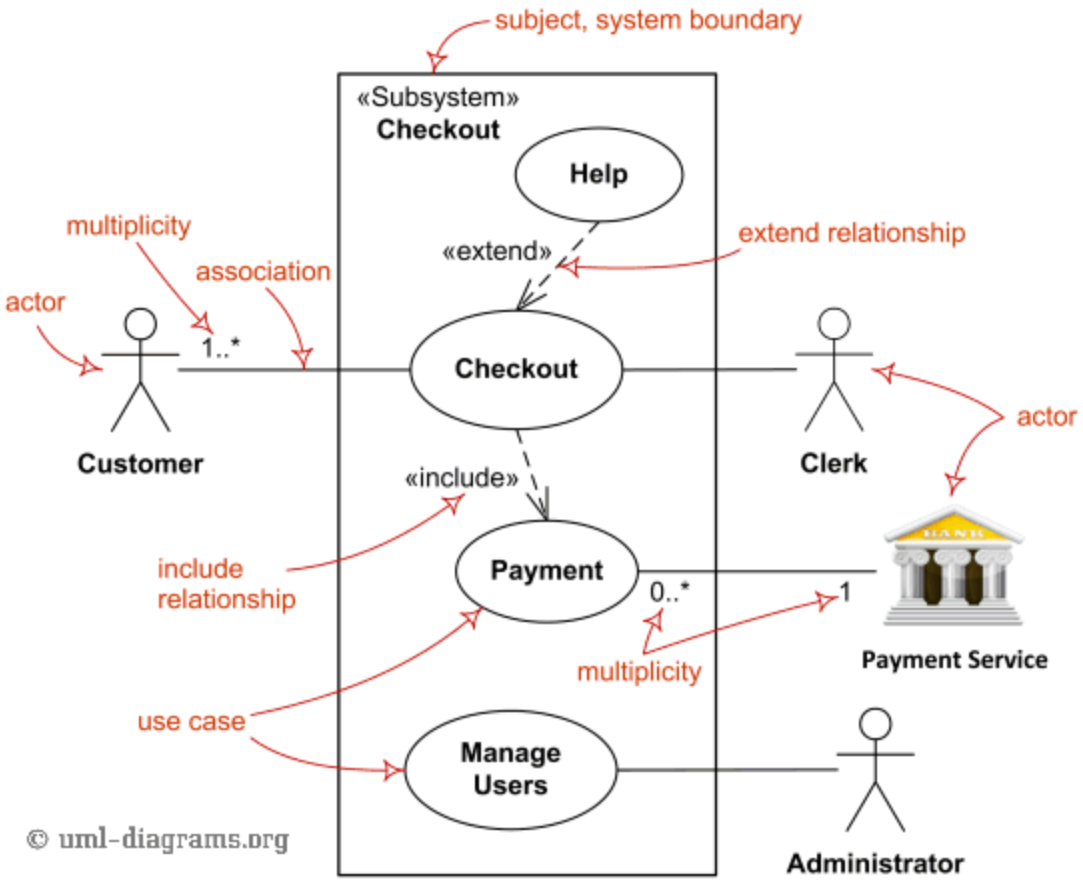
Major concepts in Use-case modeling

1. Actors: An actor is any person, system or external entity that interacts with the system and performs an action on the system. Actors are usually represented as stick figures in use-case diagrams.



2. Use-cases: A use-case is a sequence of interactions between an actor and the system that produces a result of value to the actor. Use-cases capture the functional requirements of the system and are usually represented as ovals in use-case diagrams.
3. Relationships: There are three types of relationships in use-case modeling:

- Generalization: A generalization relationship represents an "is-a" relationship between two use-cases. For example, "Login" and "Logout" use-cases are both specialized forms of the "Authenticate" use-case.
- Include: An include relationship represents a "use" relationship between two use-cases. For example, the "Place Order" use-case may include the "View Order History" use-case.
- Extend: An extend relationship represents an optional behavior that can be added to a use-case. For example, the "Cancel Order" use-case may extend the "Place Order" use-case.



4. System boundary: The system boundary represents the boundary between the system being modeled and its environment. Actors interact with the system by sending and receiving messages across the system boundary. The system boundary is usually represented as a rectangle in use-case diagrams.
5. Use-case diagram: A use-case diagram is a visual representation of the actors, use-cases, relationships and system boundary of a system. Use-case diagrams help to communicate the requirements of the system to stakeholders in a clear and concise way.

What is an actor?

- An actor is an entity (e.g. a person, system, organization, or external system/they are not part of the system) that interacts with the system being developed.

- Actors are identified as users or other systems that carry out activities (use cases) within the system and may trigger one or more use cases.
- Actors are represented in use case diagrams as stick figures.
- Actors can be a passive recipient of information

What is a Use Case?

Description of a specific task or goal that a system or software application must perform to meet the needs of its users or stakeholders. It is a textual description or a diagrammatic representation of interactions between the system and its users. Use cases are typically written from the perspective of an end-user or a stakeholder and describe the sequence of steps or events that occur during the execution of a task or goal. Use cases help to clarify the system requirements, identify potential errors or omissions, and ensure that the system meets the needs of its intended users.

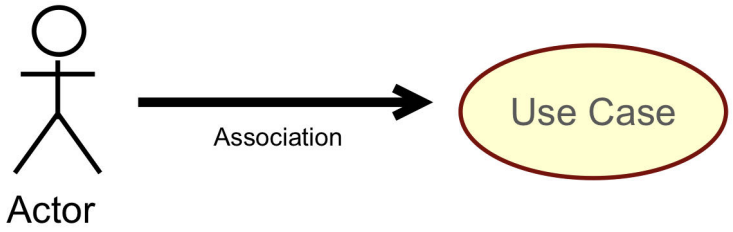
• A use case models a dialogue between one or more actors and the system

• A use case describes the actions the system takes to deliver something of value to the actor

Use case and actors

Use Cases and Actors

- A use case models a dialog between actors and the system.
- A use case is initiated by an actor to invoke a certain functionality in the system.



An actor is a user or a role that interacts with the system.

A use case, is a set of actions or steps that describe the behavior of the system when an actor interacts with it to achieve a particular goal or objective.

The relationship between use case and actors is that an actor initiates a use case by interacting with the system to accomplish a specific task. In other words, the actor is an ***external entity that triggers a use case***, and the use case describes how the system responds to the actor's actions.

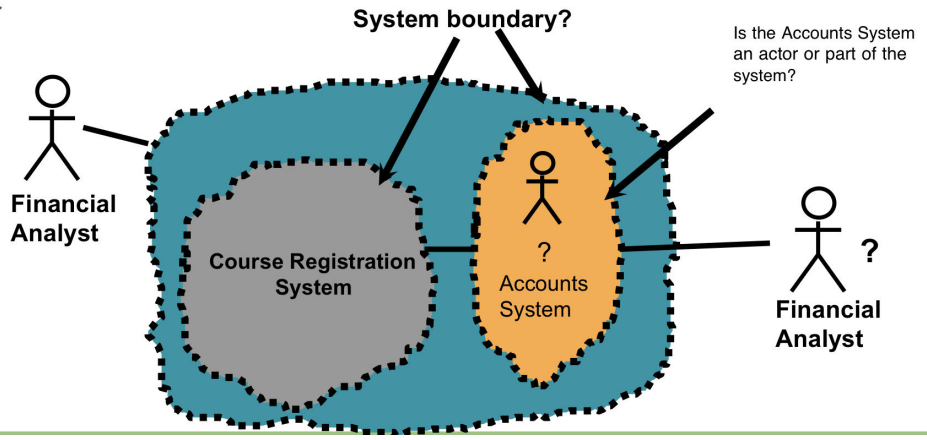
Therefore, use cases and actors are closely related and are used together to model the system's behavior and the various ways in which it interacts with its users or external entities.

Actors and the system boundary

Actors are the external entities that interact with the system, while the system boundary defines the scope of the system under consideration.

Actors and the system boundary

- Determine what the system boundary is
- Everything beyond the boundary that interacts with the system is an instance of an actor



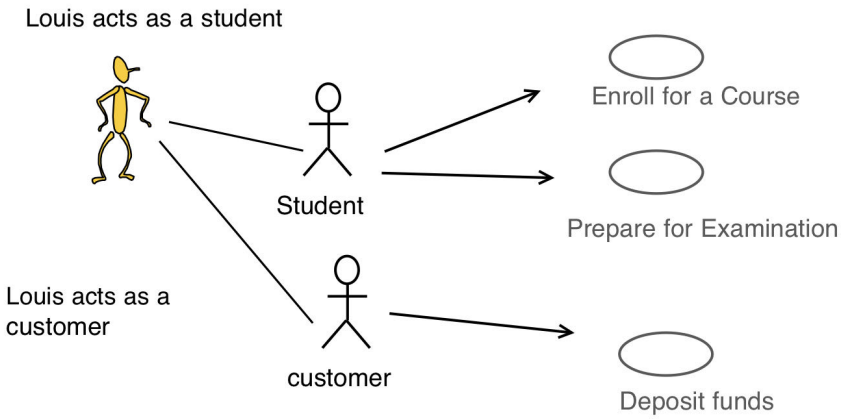
Together, actors and the system boundary help to identify the boundaries and responsibilities of the system, as well as the interactions between the system and external entities. Actors can help to identify the different types of users or stakeholders who will interact with the system, and the system boundary helps to define the limits of the system's functionality and behavior.

Actors and roles

• An actor represents a role that a human, hardware device, or another system can plan in relation to the system.

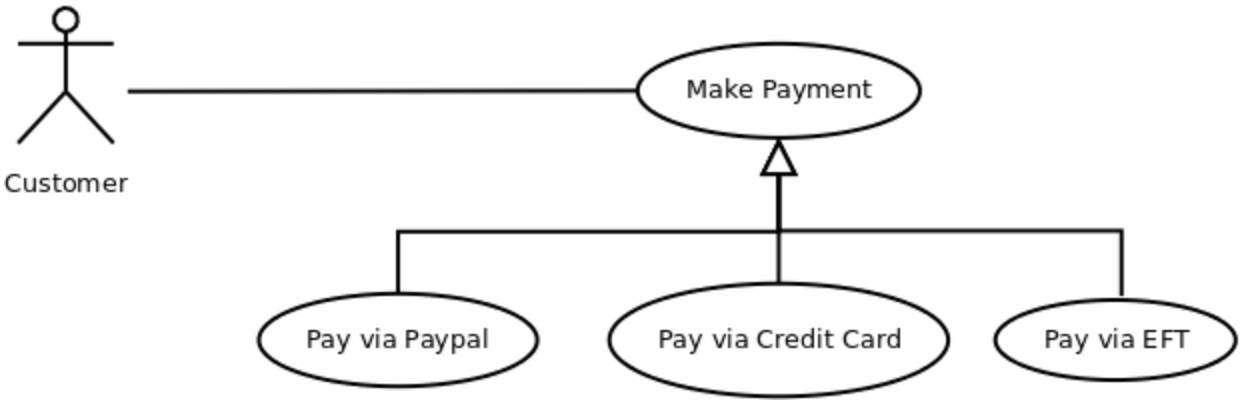
• Actor can play different roles;

Actors



Modeling with use case

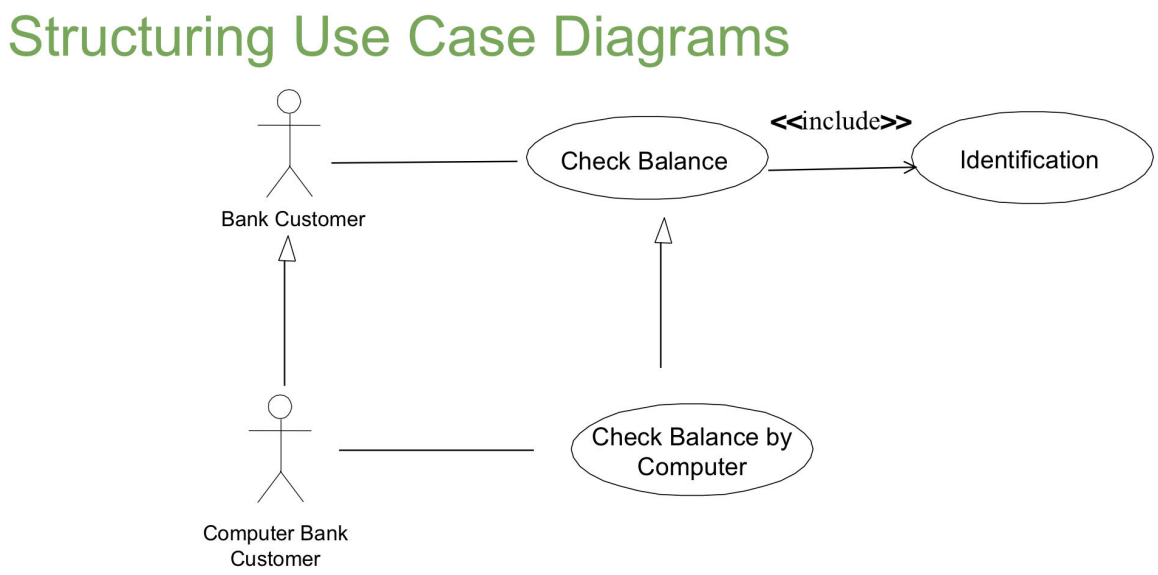
Generalization is a relationship between two use cases in which one use case is a more specific version of another use case. The more specific use case inherits the behavior and structure of the more general use case, but may also introduce additional behavior and structure specific to its own context. This relationship is denoted using an arrow with an open triangle pointing from the specific use case to the general use case. For example, a use case for "Login as a User" may be a more specific version of a use case for "Authenticate User".



• child is more specific than parent

• the child inherits all attributes and associations of the parent, but may add new features.

Example of structuring Use case diagram



Extend relationship & include relationship

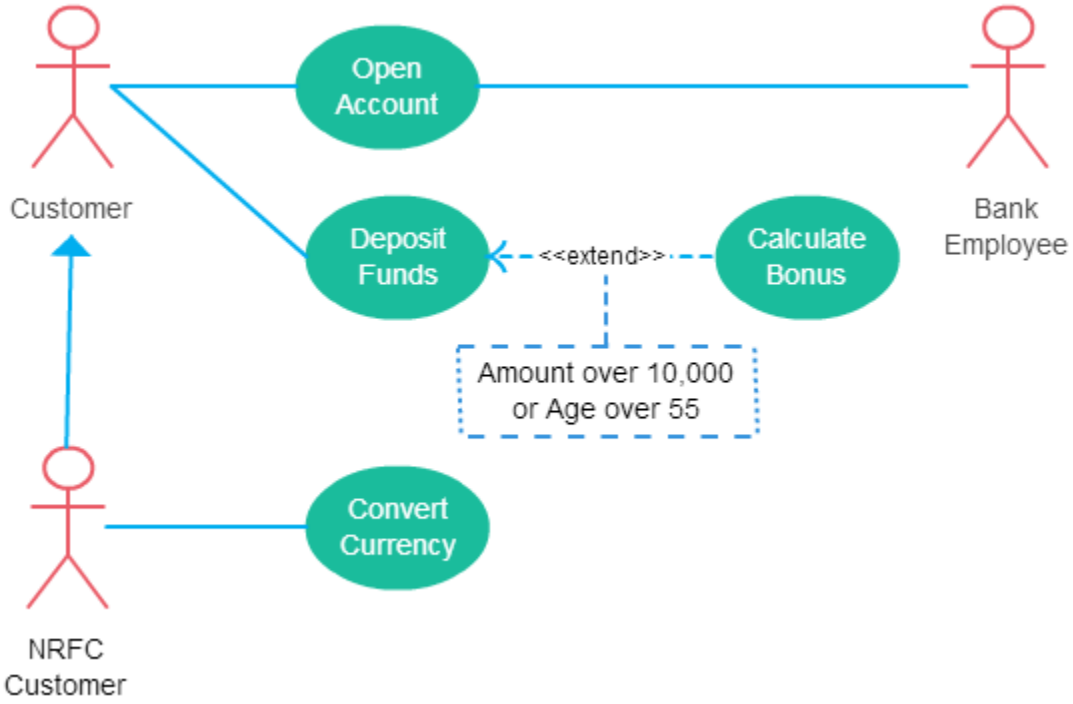
"Extend" is a relationship where a use case can optionally extend the behavior of another use case under certain conditions. The extended use case is not required for the base use case to be completed, but it can add additional functionality if the extension conditions are met.

一个use case 可以在某些条件下选择性地扩展另一个use case的行为。扩展的use case不需要基础use case的完成，但如果扩展条件得到满足，它可以增加额外的功能。

- **The extending use case is dependent on the extended (base) use case**. In the below diagram the “Calculate Bonus” use case doesn’t make much sense without

the “Deposit Funds” use case.

- **The extending use case is usually optional** and can be triggered conditionally. In the diagram, you can see that the extending use case is triggered only for deposits over 10,000 or when the age is over 55.
- **The extended (base) use case must be meaningful on its own**. This means it should be independent and must not rely on the behavior of the extending use case.

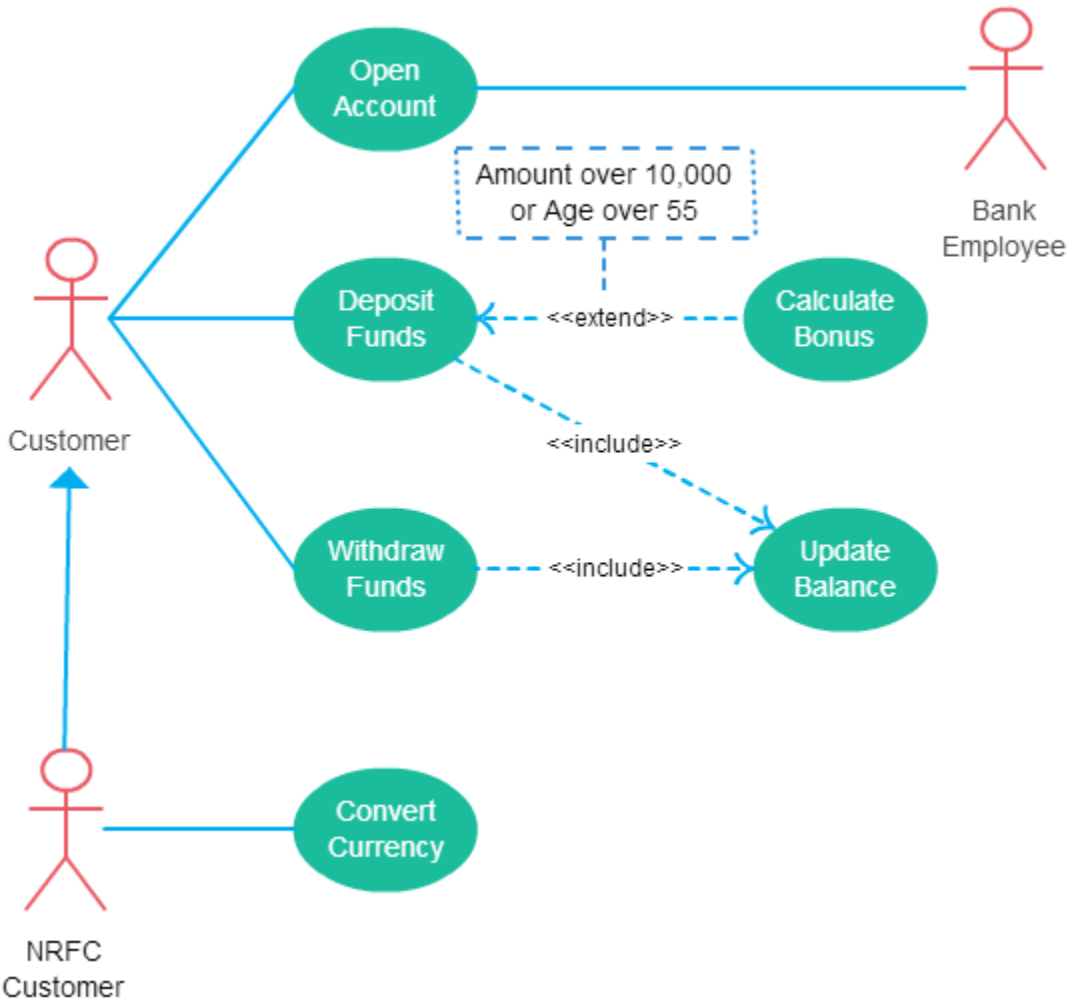


Include

"Include" is a relationship where a use case includes the behavior of another use case as part of its own behavior. The included use case is required for the base use case to be completed, and it can be reused in other use cases as well.

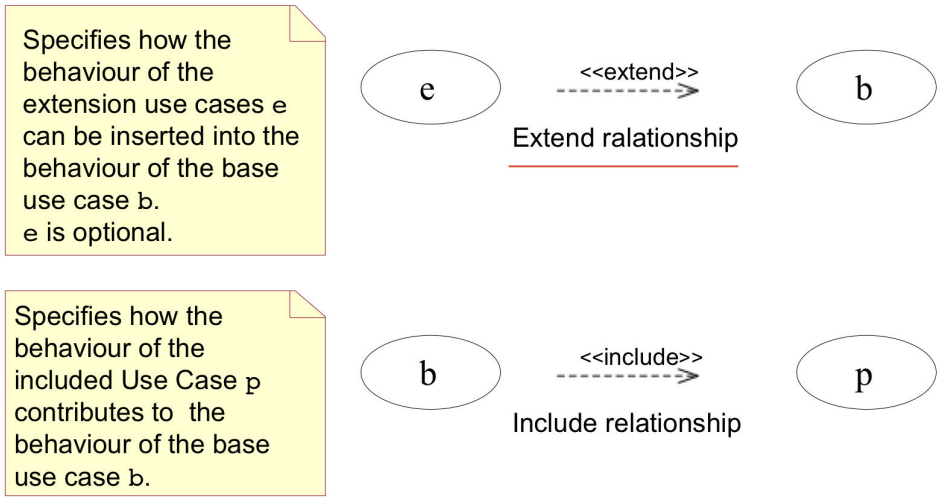
use case将另一个use case的行为作为其自身行为的一部分。所包含的use case对于基本use case的完成是必需的，而且它也可以在其他use case中重复使用

- The base use case is incomplete without the included use case.
- The included use case is mandatory and not optional.



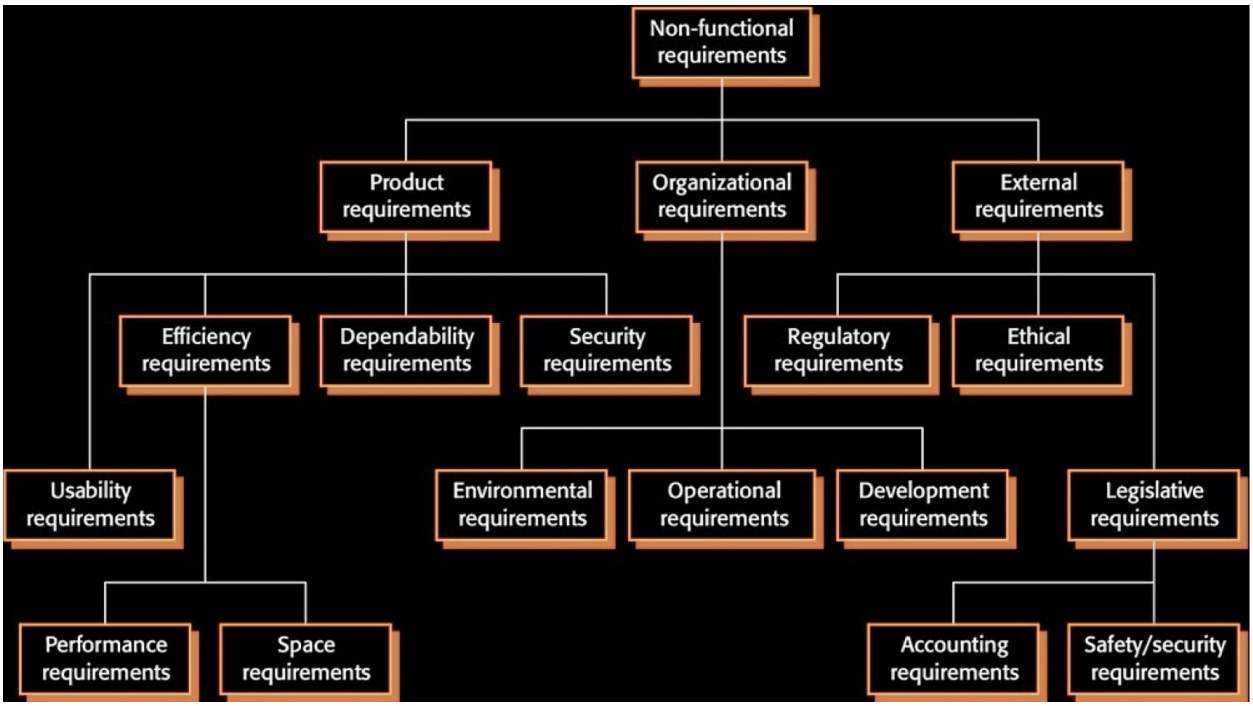
Explanation in class

Use Cases



Non-functional Requirements

Non-functional requirements (NFRs) are the qualities or attributes that describe how well a software system performs its functions, rather than describing the functions themselves. NFRs are also referred to as "quality attributes" or "system qualities." Some common categories of non-functional requirements include:



1. Performance: describes how quickly the system responds to user input, processes data, and handles load or stress.
2. Scalability: describes the ability of the system to handle increasing amounts of data, traffic, or users over time.
3. Reliability: describes the system's ability to perform consistently over time and under varying conditions, including handling errors and recovering from failures.
4. Availability: describes how often the system is available for use, including downtime for maintenance, upgrades, and other factors.
5. Security: describes how the system protects against unauthorized access, data breaches, and other security threats.
6. Maintainability: describes how easily the system can be updated, modified, or repaired over time.
7. Usability: describes how easy the system is to learn, use, and understand by users.
8. Compatibility: describes the ability of the system to work with other software or hardware components, such as operating systems, browsers, or databases.

Refine requirements

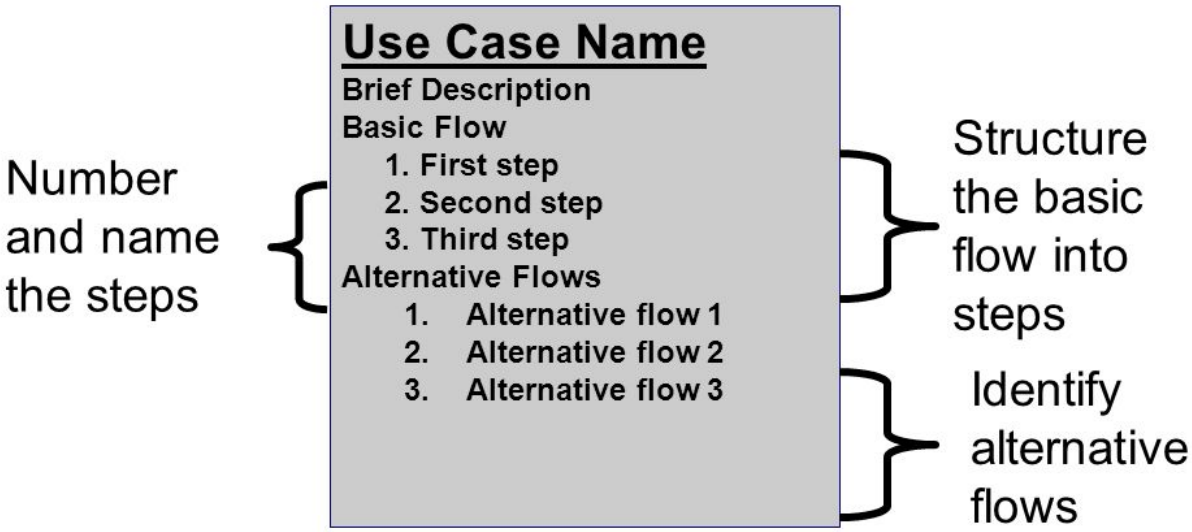
(Break down stories into individual steps)

Use case specification is a document that provides detailed information about the steps, actions, and events that occur during the execution of a use case. It describes the specific behavior of the system in response to a user's actions, as well as any constraints, assumptions, and dependencies associated with the use case. The use case specification typically includes the following sections:

1. Use case name and identification
2. Brief description
3. Actors involved
4. Preconditions
5. Basic flow of events
6. key scenarios
7. sub-flows
8. Alternative flows
9. Postconditions
10. Business rules
11. Special requirements
12. Assumptions and dependencies

Outline each use case

- An outline captures use case steps in short sentences, organized sequentially



Flow of events

Flows of events describe the steps or sequences of actions that occur during the execution of a use case. It represents a narrative description of how a user interacts with the system to accomplish a specific goal or task.

The flow of events typically includes the following components:

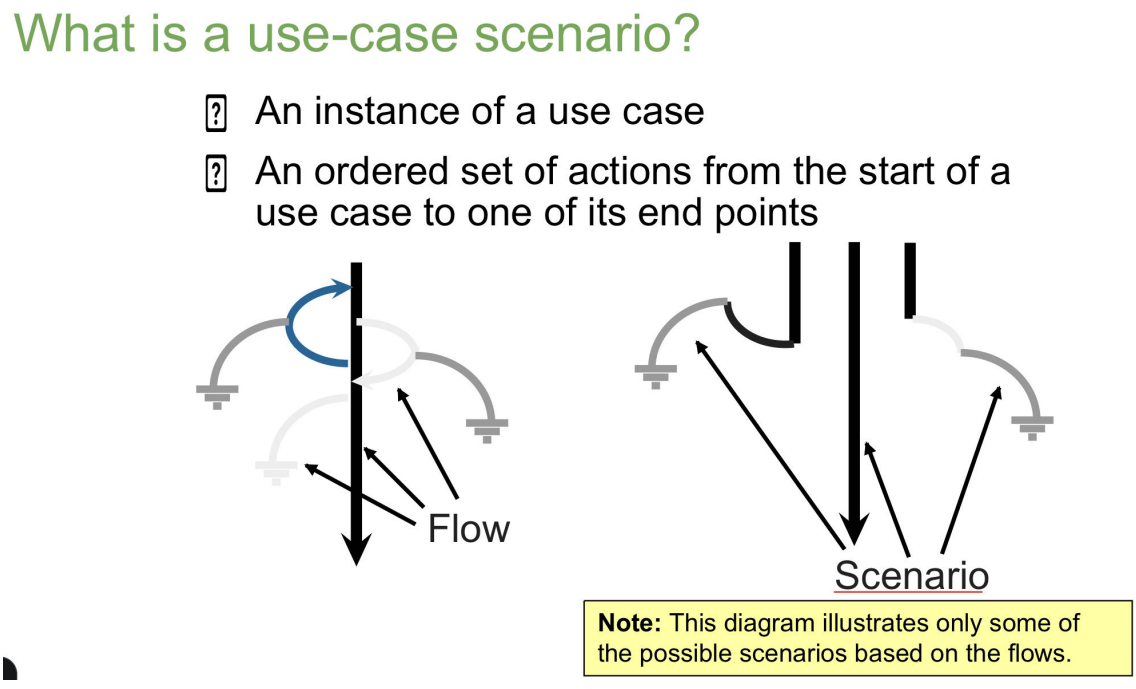
1. Basic flow: The main, most common path that a user follows to accomplish the goal or task of the use case. (successful scenario from start to finish)
2. Alternative flows: The alternative paths that a user may follow in case of errors, exceptions, or alternate conditions. (Regular variants/ odd cases/ exceptional(error) flows)
3. Pre-conditions: The conditions that must be true before the use case can begin.

4. Post-conditions: The conditions that must be true after the use case is successfully completed.
5. Exceptional flows: The possible paths that a user may follow in case of exceptions or errors.

The flows of events serve as a detailed description of the use case and are used as a basis for development and testing. It is important to capture all the relevant details and ensure that the flows are complete and accurate.

What is a use-case scenario?

A use-case scenario is a specific instance of a use case that describes a sequence of events that occur when a user interacts with a system to achieve a specific goal. It describes a particular path through the use case that shows how the system responds to user inputs and what outputs or results are produced. A use-case scenario typically includes a description of the user's goal, a sequence of steps that the user follows to achieve the goal, and any possible variations or alternate paths. It is a detailed narrative of a single use case that helps to clarify the requirements and design of the system.



Checkpoints for use cases:

• Each use case is independent of the others

• No use cases have very similar behaviour or flows of events

• No part of the flow of events has already been modeled as another use case.

Specify atomic requirements

Atomic requirements are specific and discrete statements that describe a single action or function that a software system must perform. They should be clear, concise, and unambiguous to avoid confusion and ensure that they can be easily tested.

Some examples of atomic requirements include:

- The system must allow users to log in with a valid username and password.
- The system must display an error message if the user enters an invalid username or password.
- The system must allow users to add items to their shopping cart.
- The system must calculate the total cost of items in the shopping cart and apply any discounts or promotions.

When specifying atomic requirements, it is important to consider the context in which they will be used, such as the user's needs and the system's constraints. Additionally, requirements should be prioritized based on their importance and feasibility to ensure that they are implemented in the correct order.

Quality attributes of requirements:

Quality attributes of requirements refer to characteristics or properties that requirements should possess to ensure they are of high quality. Here are some examples of quality attributes of requirements:

1. Correctness: Requirements should be accurate and free from errors.
2. Completeness: Requirements should be comprehensive and include all necessary information.

3. Consistency: Requirements should not conflict with each other and should be compatible with other related documents.
4. Clarity/comprehend-ability : Requirements should be clear and unambiguous to ensure that they can be easily understood by stakeholders. Can requirement be understood?
5. Feasibility: Requirements should be realistic and feasible within the constraints of the project.
6. Traceability: Requirements should be traceable to other related requirements, design elements, and test cases.
7. Testability: Requirements should be testable, meaning that they can be validated and verified through testing.
8. Usability
9. Prioritization: Requirements should be prioritized based on their importance to the project and their impact on project goals.
10. Realism: can the requirements be implemented given available resources and technology?
11. Modifiability: Requirements should be flexible and easily modifiable to accommodate changes or new information.
12. Verifiability**可验证性**: Requirements should be verifiable, meaning that they can be checked for accuracy, completeness, and consistency./ can requirements be “ticked off”?

Verifiability of requirements

- requirements can be tested or verified.
- A requirement has been implemented correctly and meets the intended purpose.
- Verifiable requirements should be specific, measurable, and testable.
- Be free from ambiguity and open to objective evaluation.

Unverifiable requirement example

-"the system should be user-friendly".

The term "user-friendly" is subjective and can be interpreted differently by different stakeholders. There is no objective way to measure or verify whether the system is actually user-friendly or not.

Testable example

"Upon clicking the 'Submit' button on the registration form, the system shall validate all required fields and display an error message if any field is left blank."

This requirement is testable because it clearly specifies the expected behavior of the system and can be verified through testing. The test could be performed by intentionally leaving a required field blank and verifying that the system displays an appropriate error message.

Requirement elicitation techniques

1. Interviews: One-on-one meetings with stakeholders to gather information about their needs and expectations.
2. Workshops: Group sessions where stakeholders can discuss requirements and collaborate to come up with solutions.
3. Observation: Observing how users interact with existing systems or processes to identify pain points and opportunities for improvement.
4. Surveys and questionnaires: Collecting information from stakeholders through written surveys or questionnaires.
5. Prototyping: Creating a simplified version of a system or interface to gather feedback and refine requirements.
6. Brainstorming: Encouraging stakeholders to generate ideas and solutions in a free-form, open discussion.
7. Use-case analysis: Analyzing how users interact with a system to identify requirements and use cases.

8. Document analysis: Reviewing existing documents, such as reports, user manuals, and other relevant materials, to identify requirements and constraints.
9. Focus groups: Small group discussions with stakeholders to gather feedback and refine requirements.
10. Joint application development (JAD): Collaborative sessions where stakeholders and developers work together to identify requirements and develop solutions.
11. Similar products and solution

Review:

1️⃣*what are models for?*

Models are simplified representations of complex systems or phenomena that help to better understand, analyze, and communicate about them. They can be physical or abstract, and can be used in various fields such as science, engineering, economics, and social sciences. Models can serve different purposes such as:

1. Prediction: Models can be used to predict future behavior of a system or phenomenon based on current knowledge and assumptions.
2. Explanation: Models can be used to explain how a system or phenomenon works and the relationships between its various components or factors.
3. Evaluation: Models can be used to evaluate the performance or effectiveness of a system or proposed changes to it.
4. Optimization: Models can be used to optimize or improve the performance of a system by identifying the best possible configurations or strategies.
5. Communication: Models can be used to communicate complex information in a clear and understandable way to different stakeholders or audiences.

2️⃣*what is system behavior?*

System behavior refers to the way a system functions and responds to external stimuli or events. It describes the actions and interactions of a system's components and how they work together to achieve the system's objectives. System behavior can be analyzed and modeled to understand the system's operation and identify potential issues or areas for improvement. Examples of system behavior include how a computer system processes data, how an airplane navigates through the air, or how a manufacturing system produces products.

3️⃣ *What is an actor?*

In the context of use case modeling, an actor is an entity (person, organization, or system) that interacts with the system being developed to achieve some goal. Actors are external to the system and provide inputs, receive outputs, or both, from the system under consideration. Actors are represented by stick figures in use case diagrams and are connected to the use cases they interact with using lines.

4️⃣ *What is a use case?*

A use case is a description of how a system interacts with actors (users or other systems) to achieve a particular goal. It describes a sequence of events that occur between the actors and the system, which ultimately leads to the completion of the goal. The use case includes all possible scenarios or paths that the user or system can take to reach the desired outcome. It also specifies any preconditions, postconditions, and variations in the flow of events. Use cases are commonly used in software engineering and requirements analysis to capture the functional requirements of a system.

5️⃣ *what is a role?*

In software engineering, a role refers to a specific set of responsibilities or behaviors that a user or a system component is expected to perform within a given context. A role can be performed by a human user, a machine, or a software component, and it defines a certain level of access and authority to perform actions or access resources within the system. For example, a user might have a "customer" role in an e-commerce system, which would give them the ability to view and purchase products, while a "manager" role

might have additional privileges such as the ability to view sales reports or manage inventory. Roles help to define the permissions and behaviors of different stakeholders within a system and ensure that they can perform their duties effectively and securely.

6️⃣ *How do we know if our requirements are of good quality?*

To determine if requirements are of good quality, we can use the following criteria:

1. Complete: The requirements should address all the necessary aspects of the system and leave no room for ambiguity or misunderstanding.
2. Correct: The requirements should accurately describe the needs and expectations of the stakeholders.
3. Consistent: The requirements should not contradict each other or create conflicts.
4. Feasible: The requirements should be feasible within the constraints of the project, including budget, time, and available resources.
5. Verifiable: The requirements should be testable, and there should be a way to verify that they have been met.
6. Traceable: The requirements should be traceable back to their source and be connected to other requirements and project artifacts.
7. Prioritized: The requirements should be prioritized based on their importance to the stakeholders and the project goals.
8. Unambiguous: The requirements should be clear and concise, without any ambiguity or vagueness.

By ensuring that our requirements meet these criteria, we can increase the likelihood that they will be of good quality and will meet the needs of the stakeholders.

Some quizes:

1. What is the purpose of requirements engineering?
   A) To define the system's architecture
   B) To develop the user interface

C) To identify and document the system's requirements
D) To conduct system testing

2. Which of the following is a benefit of a use case model? (2 answers)
   A) It provides a visual representation of the system's architecture
   B) It facilitates communication between stakeholders
   C) It defines the user interface design
   D) It tests the system's functionality

3. In a use case diagram, what is an actor?
   A) A component of the system
   B) A user of the system
   C) A use case
   D) A system requirement

4. What is the purpose of a use case scenario?
   A) To define the system's architecture
   B) To identify user requirements
   C) To develop the user interface
   D) To conduct system testing

5. What is the difference between a use case and a use case scenario?
   A) A use case defines the system behavior, while a use case scenario outlines a specific interaction with the system
   B) A use case outlines a specific interaction with the system, while a use case scenario defines the system behavior
   C) A use case and a use case scenario are the same thing
   D) A use case and a use case scenario are both used for testing the system

6. What is the purpose of the include relationship in a use case diagram?
   A) To specify a common sequence of events between use cases
   B) To extend the functionality of a use case
   C) To show a relationship between an actor and a use case
   D) To show that one use case is a more general version of another use case

7. Which of the following is an example of a non-functional requirement?
   A) The system must be able to handle 1000 simultaneous users
   B) The system must be able to process credit card payments

C) The system must be easy to use for non-technical users
D) The system must be compatible with Windows 10

8. What is the purpose of requirement elicitation techniques?
   A) To identify and document the system's requirements
   B) To develop the user interface
   C) To conduct system testing
   D) To define the system's architecture

9. What is the purpose of a use case specification?
   A) To define the system's architecture
   B) To identify and document the system's requirements
   C) To develop the user interface
   D) To conduct system testing

10. What is the purpose of a system boundary in a use case diagram?
    A) To show the interactions between actors and the system
    B) To specify the system's architecture
    C) To identify and document the system's requirements
    D) To develop the user interface

11. What is the purpose of a functional requirement?
    A) To describe how the system should perform
    B) To define the system's architecture
    C) To identify the system's users
    D) To specify the system's non-functional requirements

12. Which of the following is an example of a quality attribute for requirements?
    A) Usability
    B) Functionality
    C) Performance
    D) Scalability

13. What is the purpose of a domain model?
    A) To describe the system's behavior
    B) To identify and document the system's requirements
    C) To show the system's architecture
    D) To define the problem domain of the system

Answers:

1. C) To identify and document the system's requirements
2. A & B
3. B) A user of the system
4. B) To identify user requirements: cannot be D because a use case scenario is not used for conducting system testing. It is used to describe a specific sequence of events that occurs when a user interacts with the system in order to achieve a particular goal or task.
5. A) A use case defines the system behavior, while a use case scenario outlines a specific interaction with the system
6. A) To specify a common sequence of events between use cases: The "include" relationship is used to show that one use case includes the functionality of another use case, and that the included use case is always executed when the base use case is executed. Therefore, the correct answer is A, which states that the purpose of the "include" relationship is to specify a common sequence of events between use cases. Answer B, on the other hand, refers to the "extend" relationship, which is used to add additional functionality to a base use case.
7. A) The system must be able to handle 1000 simultaneous users
8. A) To identify and document the system's requirements
9. B) To identify and document the system's requirements
10. A) To show the interactions between actors and the system; the system boundary in a use case diagram is used to define the scope of the system being modeled and to identify the actors involved in the system. It is not used to identify or document the system's requirements. Therefore, option C is not the correct answer, and the correct answer is A, "To show the interactions between actors and the system."
11. A)
12. A) functionality, performance, and scalability are important aspects of a system, they are not quality attributes of requirements. Usability, on the other hand, is a quality attribute that focuses on how easy the system is to use and how well it meets the needs of its users. Therefore, option A is the correct answer.

13. D)