

2020

Software Engineering

Use Case Diagrams

- 📄 What is a Use Case Diagram?
- 📄 Use Case Components
- 📄 Use Case Diagrams Guideline
- 📄 How to create a use case diagram
- 📄 When to use an Use Case Diagram
- 📄 Benefits of Use Case

Note by Sriyantha Deepal



What is a Use Case Diagram?

- A diagram that shows a set of use cases and actors and their relationships
- A formal way of representing how a business system interacts with its environment
- Illustrates the activities that are performed by the users of the system
- Use cases represent system functionality, the requirements of the system from the user's perspective. The emphasis is on what a system does rather than how
- Use case diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with the system - A use case can also said as a summary of scenarios for a single task or goal.

A Use Case consists of use cases, persons, or various things that are invoking the features called as actors and the elements that are responsible for implementing the use cases. Use case diagrams capture the dynamic behaviour of a live system. It models how an external entity interacts with the system to make it work. Use case diagrams are responsible for visualizing the external things that interact with the part of the system.

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

To model a system, the most important aspect is to capture the dynamic behavior. Dynamic behavior means the behavior of the system when it is running/operating. Only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. Use case diagrams consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system. Hence to model the entire system, a number of use case diagrams are used.

• Importance of Use Case Diagrams

As mentioned before use case diagrams are used to gather a usage requirement of a system. Depending on your requirement you can use that data in different ways. Below are few ways to use them.

- **To identify functions and how roles interact with them** – The primary purpose of use case diagrams.
- **For a high-level view of the system** – Especially useful when presenting to managers or stakeholders. You can highlight the roles that interact with the system and the functionality provided by the system without going deep into inner workings of the system.
- **To identify internal and external factors** – This might sound simple but in large complex projects a system can be identified as an external role in another use case.

Use case diagrams are typically developed in the early stage of development and people often apply use case modeling for the following purposes:

- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases
- Developed by analysts together with domain experts

The purpose of use case diagram is to capture the dynamic aspect of a system. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and State chart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view.

In brief, the purposes of use case diagrams can be said to be as follows –

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Shows the interaction among the requirements is actors.

- **Scenario**

- A scenario is a specific sequence of actions and interactions between actors and the system. Also called as a use case instance.
- The Main Success Scenario or “happy path” is the expected primary use of the system, without problems or exceptions.
- Alternative Scenarios or Extensions are used to document other common paths through the system and error handling or exceptions.
- Examples
 - The scenario of successfully purchasing items with cash
 - The scenario of failing to purchase items because of a credit payment denial.

Use Case Components

The use case has three components.

- The actor(s) who trigger the use case to activate.
- Task referred to as the use case that represent. The use case to a feature needed in a software system.
- The communication/relationship line to show how the actors communicate with the use case.

Actor

- An actor is a person, organization, or external system that plays a role in one or more interactions with the system; they are among the stakeholders of a system.
- An actor is a role, not a specific user.
- An actor is outside or external to the system.
- An actor can be a:
 - A Person/Human
 - Peripheral device (hardware)
 - External system or subsystem
 - Time or time-based event
- Whether human or not, represented by stick figure
- Labelled using a descriptive noun or phrase

It is used inside use case diagrams. The actor is an entity that interacts with the system. A user is the best example of an actor. An actor is an entity that initiates the use case from outside the scope of a use case. It can be any element that can trigger an interaction with the use case. One actor can be associated with multiple use cases in the system. The actor notation in UML is given below. Actor in a use case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below.



- Someone interacts with use case (system function).
- Named by noun.
- Actor plays a role in the business
- Similar to the concept of user, but a user can play different roles
- For example:
 - A prof. can be instructor and also researcher
 - plays 2 roles with two systems
- Actor triggers use case(s).
- Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).

Use Case

- A use case is a type of complete interaction between and product and its environment. It describes a sequence of actions that provide something of measurable value to an actor
- The collection of use cases characterizes all externally observable behavior.
- Use case drawn as a horizontal ellipse
- Each use case must have a name

Use cases are used to represent high-level functionalities and how the user will handle the system. A use case represents a distinct functionality of a system, a component, a package, or a class. It is denoted by an oval shape with the name of a use case written inside the oval shape. The notation of a use case in UML is given below:



- System function (process - automated or manual)
- Named by verb + Noun (or Noun Phrase).
- i.e. Do something
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.

Relationships in Use Case Diagrams

- Represent communication between actor and use case
- Types of relationship
 - Association between an actor and a use case
 - Extend relationship between two use cases
 - Include relationship between two use cases

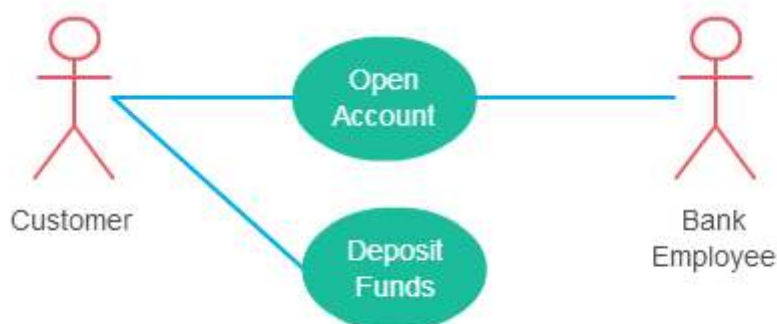
Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.

• Association Relationship

- Exist between actor and a use case
- Often referred to as a communicate association
- Navigation direction of an association represents who is initiating the communication
 - Use an arrowhead where arrowhead denotes the direction of communication
- The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.
- Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.

This one is straightforward and present in every use case diagram. Few things to note.

- An actor must be associated with at least one use case.
- An actor can be associated with multiple use cases.
- Multiple actors can be associated with a single use case.



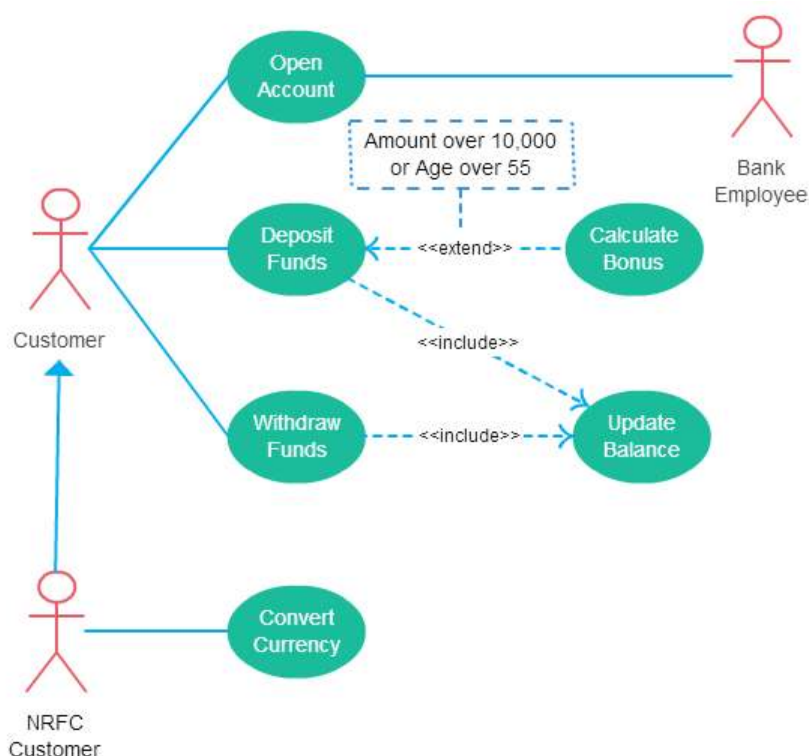
Different ways association relationship appears in use case diagrams

• Include Relationship -- <<include>> -->

- Represents the inclusion of the functionality of one use case within another
- Depicted by arrowhead with the word "<<include>>" above it
- Arrow is drawn from the base use case to the used use case

Include relationship show that the behavior of the included use case is part of the including (base) use case. The main reason for this is to reuse common actions across multiple use cases. In some situations, this is done to simplify complex behaviors. Few things to consider when using the <<include>> relationship.

- The base use case is incomplete without the included use case.
 - The included use case is mandatory and not optional.
- When a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship.
 - A use case includes the functionality described in another use case as a part of its business process flow.
 - A uses relationship from base use case to child use case indicates that an instance of the base use case will include the behavior as specified in the child use case.
 - An include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the child use case and the parent use case connected at the base of the arrow.
 - The stereotype "<<include>>" identifies the relationship as an include relationship.

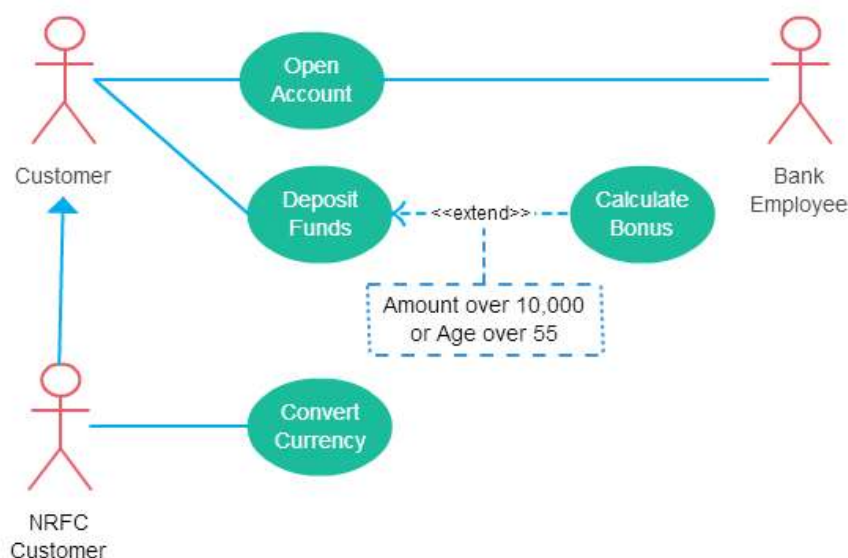


- **Extend Relationship** --> <<extend>>

- Used to show optional behavior (behavior that is run only under certain circumstances)
- Specifies that the target use case extends the behaviour of the source
- Represents the extension of the use case to include optional functionality
- Extend notation is a dotted line, labeled <<extend>>, and with an arrow towards the base case.

Many people confuse the extend relationship in use cases. As the name implies it extends the base use case and adds more functionality to the system. Here are a few things to consider when using the <<extend>> relationship.

- **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.
- Indicates that an **"Invalid Password"** use case may include (subject to specified in the extension) the behavior specified by base use case **"Login Account"**.
 - Depict with a directed arrow having a dotted line. The tip of arrowhead points to the base use case and the child use case is connected at the base of the arrow.
 - The stereotype "<<extends>>" identifies as an extend relationship



Extend relationship in use case diagrams

System Boundary

- Indicates the scope of the system. It defines the scope/limits of the system
- A system cannot have infinite functionality.
- Anything within the box represents functionality that is in scope and anything outside the box is not
- Actors in the system are outside the system boundary.
- The system boundary is shown as a rectangle spanning all the use cases in the system.
- The system boundary is potentially the entire system, but for large and complex systems, each of the modules may be the system boundary. Ex. ERP System

The system is used to **define the scope of the use case** and drawn as a rectangle. This optional element but useful when you're visualizing large systems. For example, you can create all the use cases and then use the system object to define the scope covered by your project. Or you can even use it to show the different areas covered in different releases.

- The system boundary is potentially the entire system as defined in the requirements document.
- For large and complex systems, each module may be the system boundary.
- For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, etc.
- Can form a system boundary for use cases specific to each of these business functions.
- The entire system can span all of these modules depicting the overall system boundary

System



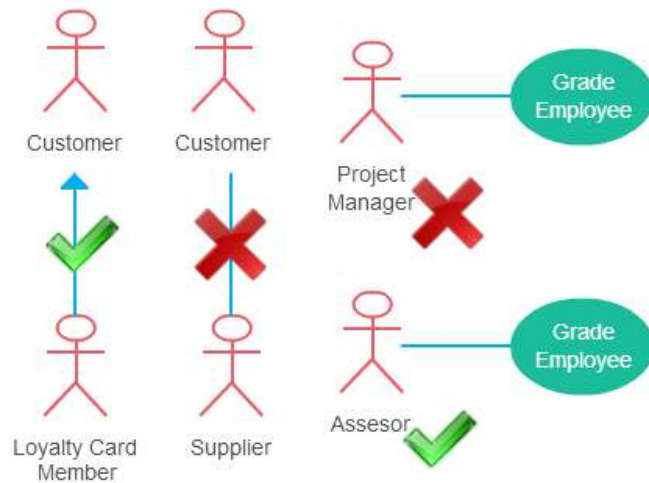
Use Case Guidelines

- Use cases should ideally begin with a verb (verb+noun) – i.e generate report.
- Use cases should not be open ended – i.e Register (instead should be named as Register New User)
- Avoid showing communication between actors
- Actors should be named as singular. i.e. student and NOT students. No names should be used – i.e Vishan, Chaminda, etc.
- Do not show behaviour in a use case diagram; instead depict only system functionality.
- Use case diagram does not show sequence – unlike DFDs.

When it comes to analyzing the requirement of a system use case diagrams are second to none. They are visual and usually very easy to understand. The following use case diagram guidelines will help you to create better use cases that are appreciated by your clients and peers alike. A use case diagram mainly consists of actors, use cases and relationships. More complex larger diagrams can include systems and boundaries. We'll discuss the use case diagram guidelines based on objects. It is important to remember that these are use case diagram guidelines and not rules. It's alright to deviate if it improves the overall quality of the diagram.

Actors

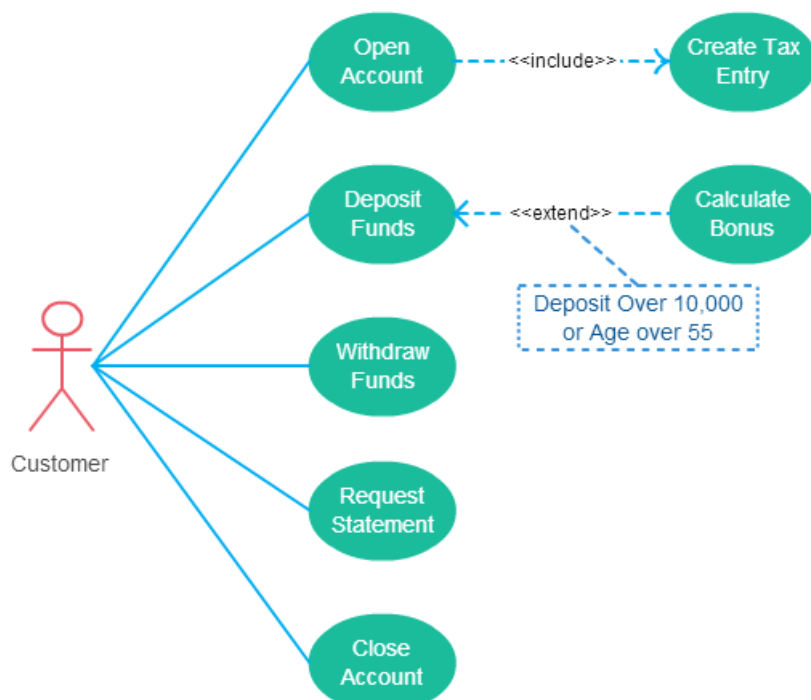
- **Give meaningful business relevant names for actors** – For example, if your use case interacts with an outside organization it's much better to name it with the function rather than the organization name. (Eg: Airline Company is better than PanAir)
- **Primary actors should be to the left side of the diagram** – This enables you to quickly highlight the important roles in the system.
- **Actors model roles (not positions)** – In a hotel both the front office executive and shift manager can make reservations. So something like "Reservation Agent" should be used for actor name to highlight the role.
- **External systems are actors** – If your use case is send-email and if interacts with the email management software then the software is an actor to that particular use case.
- **Actors don't interact with other actors** – In case actors interact within a system you need to create a new use case diagram with the system in the previous diagram represented as an actor.
- **Place inheriting actors below the parent actor** – This is to make it more readable and to quickly highlight the use cases specific for that actor.



Some things to consider when creating the actor element in use cases

Use Cases

- **Names begin with a verb** – A use case models an action so the name should begin with a verb.
- **Make the name descriptive** – This is to give more information for others who are looking at the diagram. For example “Print Invoice” is better than “Print”.
- **Highlight the logical order** – For example, if you’re analyzing a bank customer typical use cases include open account, deposit and withdraw. Showing them in the logical order makes more sense.
- **Place included use cases to the right of the invoking use case** – This is done to improve readability and add clarity.
- **Place inheriting use case below parent use case** – Again this is done to improve the readability of the diagram.



Relationships

- Arrow points to the base use case when using <<extend>>
- <<extend>> can have optional extension conditions
- Arrow points to the included use case when using <<include>>
- Both <<extend>> and <<include>> are shown as dashed arrows.
- Actor and use case relationship don't show arrows.

Systems / Packages

- Use them sparingly and only when necessary
- Give meaningful and descriptive names to these objects

How to Create Use Case Diagram

To draw a use case diagram in UML first one need to analyse the entire system carefully. You have to find out every single function that is provided by the system. After all the functionalities of a system are found out, then these functionalities are converted into various use cases which will be used in the use case diagram.

A use case is nothing but a core functionality of any working system. After organizing the use cases, we have to enlist the various actors or things that are going to interact with the system. These actors are responsible for invoking the functionality of a system. Actors can be a person or a thing. It can also be a private entity of a system. These actors must be relevant to the functionality or a system they are interacting with.

After the actors and use cases are enlisted, then you have to explore the relationship of a particular actor with the use case or a system. One must identify the total number of ways an actor could interact with the system. A single actor can interact with multiple use cases at the same time, or it can interact with numerous use cases simultaneously.

Following rules must be followed while drawing use-case for any system:

1. The name of an actor or a use case must be meaningful and relevant to the system.
2. Interaction of an actor with the use case must be defined clearly and in an understandable way.
3. Annotations must be used wherever they are required.
4. If a use case or an actor has multiple relationships, then only significant interactions must be displayed.

There are four steps create a use case diagram

1. Identify actors
2. Identify use case
3. Look for common functionality to use include <<include>> 3
4. Look for use of <<extend>>

• Identifying Actors

Actors are external entities that interact with your system. It can be a person, another system or an organization. In a banking system, the most obvious actor is the customer. Other actors can be bank employee or cashier depending on the role you're trying to show in the use case.

An example of an external organization can be the tax authority or the central bank. The loan processor is a good example of an external system associated as an actor.

Often, people find it easiest to start the requirements elicitation process by identifying the actors. The following questions can help you identify the actors of your system (Schneider and Winters - 1998):

- Who uses the system?
- Who installs the system?
- Who starts up the system?
- Who maintains the system?
- Who shuts down the system?
- What other systems use this system?
- Who gets information from this system?
- Who provides information to the system?
- Does anything happen automatically at a present time?

Identifying Use Cases

A good way to do this is to identify what the actors need from the system. In a banking system, a customer will need to open accounts, deposit and withdraw funds, request check books and similar functions. So all of these can be considered as use cases.

Top level use cases should always provide a complete function required by an actor. You can extend or include use cases depending on the complexity of the system. Once you identify the actors and the top level use case you have a basic idea of the system. Now you can fine tune it and add extra layers of detail to it.

Identifying the Use Cases, and then the scenario-based elicitation process carries on by asking what externally visible, observable value that each actor desires. The following questions can be asked to identify use cases, once your actors have been identified (Schneider and Winters - 1998):

- What functions will the actor want from the system?
- Does the system store information? What actors will create, read, update or delete this information?
- Does the system need to notify an actor about changes in the internal state?
- Are there any external events the system must know about? What actor informs the system of those events?

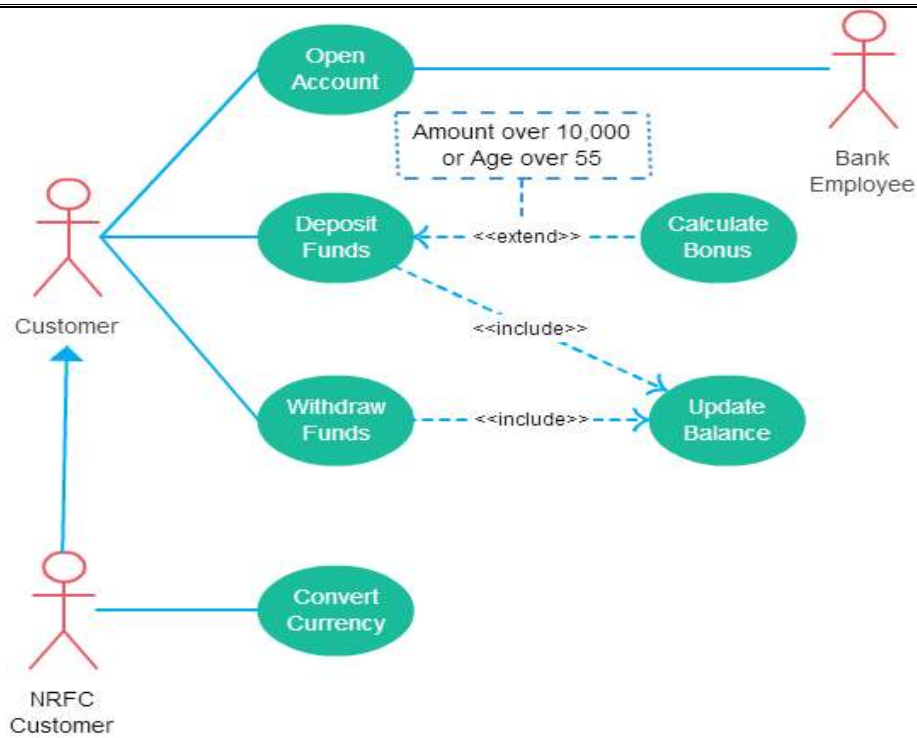
Look for Common Functionality to use Include

Look for common functionality that can be reused across the system. If you find two or more use cases that share common functionality you can extract the common functions and add it to a separate use case. Then you can connect it via the include relationship to show that it's always called when the original use case is executed.

Optional Functions or Additional Functions

There are some functions that are triggered optionally. In such cases, you can use the extend relationship and attach an extension rule to it. In the below banking system example "Calculate Bonus" is optional and only triggers when a certain condition is matched.

Extend doesn't always mean it's optional. Sometimes the use case connected by extending can supplement the base use case. The thing to remember is that the base use case should be able to perform a function on its own even if the extending use case is not called.



Rules of thumb in use case diagramming

- Every use case diagram must have
 - At least one use case
 - At least one actor
 - At least one actor associated with each use case
 - At least one use case associated with each actor
- No association line between actors
- No association line between use cases
- Name every actor and use case
- Not label any association line

When to use an Use Case Diagram

A use case is a unique functionality of a system which is accomplished by a user. A purpose of use case diagram is to capture core functionalities of a system and visualize the interactions of various things called as actors with the use case. This is the general use of a use case diagram.

The use case diagrams represent the core parts of a system and the workflow between them. In use case, implementation details are hidden from the external use only the event flow is represented. With the help of use case diagrams, we can find out pre and post conditions after the interaction with the actor. These conditions can be determined using various test cases.

In general use case diagrams are used for:

1. Analyzing the requirements of a system
2. High-level visual software designing
3. Capturing the functionalities of a system
4. Modeling the basic idea behind the system
5. Forward and reverse engineering of a system using various test cases.

Use cases are intended to convey desired functionality so the exact scope of a use case may vary according to the system and the purpose of creating UML model.

Benefits of Use Case Diagram

- Use cases are the primary vehicle for requirements capture in Unified Modeling Language - UML
- Use cases are described using the language of the customer (language of the domain which is defined in the glossary)
- Use cases provide a contractual delivery process
- Use cases provide an easily-understood communication mechanism
- When requirements are traced, they make it difficult for requirements to fall through the cracks
- Use cases provide a concise summary of what the system should do at an abstract (low modification cost) level.