Click to prove you're human

Use cases definition

Try Innerview(psst... for free!)Transform user interviews into actionable takeaways & faster decisions Analyzing or designing the various features and functions of a software system can be daunting, especially when there are multiple actors and other interfacing systems involved. Thankfully, analysts can turn to use cases to make this process much easier. At the very minimum, an effective use case should: define how stakeholders interact with a system define how a system interacts with other systems provide a common understanding of both stakeholder and system requirements This article introduces the different types of use cases you may encounter and identifies some of the ways use cases are represented. Identifying a Use Case Since Ivar Jacobson first formulated textual and visual modeling techniques for use cases in 1986, quite a bit has changed. However, certain key factors have remained effectively the same; particularly the methods employed to identify use cases. The first and probably most critical step to identify a use case is to first identify the actors because the use case will be written from their points of view. Next, the actor’s functions must be identified as well as the actors’ actions within the system. Last but not least, it is important to identify all external events that can influence the actors and system as the actors perform their functions within the system. Below is an illustration of the identification process. Use Case Example So how do you identify a use case? Well, suppose that ABC Corp has customers that must register on ABC’s online portal in order to receive services. Using the Use Case Identification method, we can determine that the actor is the customer. The actor’s function is to provide personal information to complete his/her profile. The actor’s actions in the system include accessing it to provide personal information. Next, we identify external events. These may come from multiple sources and can also include other use cases. In this example, it will be login credentials provided to the customer by a system administrator (a poor user provisioning method, but that’s a conversation for another day/blog article). In order to provide the credentials to the customer, the system administrator will need to first log into the system themselves with their own credentials. This process has nothing to do with the customer and as a result, “Provide Credentials” is considered an external influence and can be considered its own use case even though it’s performed in the same system (this concept is known as use case partitioning). Formulating a Use Case Formulating a use case after having identified the use case is quite simple. Typically, use cases are part of a larger documentation effort and should be named and numbered for easier identification and to facilitate referencing by another use case. Another thing to note is that the primary courses defined for a use case may vary from analyst to analyst. It all depends on how critical each stated step is, as perceived by the analyst. In addition, there are alternative courses to the primary courses and these are typically used to achieve the use case goal using other means that are not used by the primary course. Alternative courses typically handle situations when a primary course fails or deviates due to some specific factor. Using the ABC Corp example, a formulated use case would look like this: Use Case Name Complete Customer Profile Goal Input personal information into ABC Corp online portal Use Case ID UC001 Actors Customer, System Admin PreCondition Customer has login credentials provided by ABC Corp System Admin Primary Course Customer accesses ABC Corp’s portal Customer logs in using credentials provided by System Admin System verifies customer and displays personal details entry form Customer enters personal details into form and clicks save System presents a confirmation screen informing the Customer that the information has been saved Alternative Course 1a. System is down 1a1. Customer emails personal information to System Admin 1a2. Proceed with Alternative Course 1b. 1b. System Admin enters customer information 1b1. System Admin accesses “customer information” functionality 1b2. System Admin enters personal information emailed by Customer 1b3. End Use Case 3a. System Cannot verify Customer 3a1. Customer emails System Admin for credentials reset 3a2. Resume at step 3 Types of Use Cases and their Presentation Methods There are basically two types of use cases analysts can draw from: Business Use Cases and System Use Cases. Business Use Cases are more about what a user expects from a system while System Use Cases are more about what the system does. Both use case types can be represented by diagrams or text. Diagrammatically, both types of use cases are denoted differently. In our example, we have used the System Use Case. Textual Use Cases When representing the use cases in textual form, both use case types can be presented as either “informal” use cases or “formal” use cases. “Informal” use cases are quite brief with just enough information to get the point across. “Formal” use cases are meant to be more detailed. ABC Corp’s use case UC001 is a good example of a “formal” use case. Below is an “informal” example of use case UC001. Use Case Name Complete Customer Profile Use Case ID UC001 Primary Course 1. Customer enters provided credentials 2. System verifies customer and provides personal details entry form 3. Customer enters personal details 4. System presents a confirmation screen informing the customer that the information has been saved 6. End Use Case Even though the examples listed above are in tabular form, they are still considered examples of the text-based method of presenting use Cases. Use Case Diagrams As mentioned above, use cases can also be represented using diagrams. The UML notation is the most widely used standard. Within UML notation, use case diagrams are classified as behavioral diagrams. Using ABC Corp’s example above, a typical system use case diagram looks like this: Use case diagrams can be particularly helpful when multiple actors overlap and perform the same function in the same system. Being able to represent all actors and their interactions with the system in one diagram gives a better picture of the dynamic system behavior. Although this is a “by the book” standard description of Use Cases, sometime the objective to achieve in your business analysis efforts is just to simply inventory use cases, rather than launching into a fully dressed illustration of each use case’s steps. If so, then a great way to do this fast is with the System Context diagram instead. It can be created rapidly, and also be the basis for more detailed use cases as described here. Insights Background Topics Use Cases: Diagram & Examples (Updated 2024) Use Cases: Diagram & Examples (Updated 2024) by Adam Sandman on Monday, April 7, 2025 Developing an initial set of functional requirements during the Requirements Gathering phase should provide a good understanding of the intended behavior of the system. However, one shortcoming of this list of requirements is that it’s static and doesn’t adapt to the different business processes that need to be supported by new features. So what is a use case? A use case is a description of the different ways that a user can interact with an application or product. They define the various external entities that exist outside the system, as well as the specific interactions they have with the system. This can come in the form of success scenarios, alternate paths, and more. Use cases can also be formatted in a text-based medium like a list/table, or a visual medium like a diagram. The major benefits that use cases provide are in the planning stage of development. Steps like requirements gathering, defining scope, and roadmap creation are all improved through the clarity that these use cases bring. They also allow the team to identify the best possible outcome scenario, accurately depicting the intended design and use of the system. When it comes to less-than-ideal use cases, the brainstorming of these alternative possible outcomes helps developers identify potential problems before they happen. Writing a use case can be started with three basic steps or questions: Who is going to use the product? What will it be used for? How are they going to use it? These questions form the basis of any successful use case, and should be used as an initial framework when writing your own. Beyond that, use cases can be as high-level or detailed as they need to be for the audience and system. The key to an effective and helpful use case is to carefully consider the flow of events for a typical user, and use those to form the use case. As you’re writing your use case, there are several components that should typically be included. These range from the identifier of the use case to who’s involved and alternative scenarios: Use case number: Assigning a number to each use case helps organize your records and can be sorted in chronological order or by other criteria depending on how the developers decide to label them. Use case name, description, and goal: Giving a clear and concise name, description, and goal to each use case also helps organize documentation while preventing scope creep. Actor: This is someone or something that performs a behavior or action (can be a person or an object). If we take an e-commerce site as an example, actors might include buyers, credit card companies, shipping companies, and more. Primary actor: This is someone or something whose goals are fulfilled by the system in question, and while they don’t always start the use case, it’s fairly common for them to do so. Stakeholders: Anyone with interests in the functionality and success of the system are called stakeholders, and are often indirectly involved (not users, but those that benefit from how the system functions). Pre-conditions: The statements about what needs to happen before the use case starts. Triggers: Used to start a use case, triggers are events that initiate the first steps of a scenario. Post-conditions: The statements about the possible states that the system can be in after the use case ends. Basic flow: Also called the main success scenario, this is a use case path that works perfectly and as intended with no exceptions (this is often used as a base to create alternative paths). Alternative path (or flow): A variation of the main success scenario, these usually show what happens when there’s an error or unexpected event in a use case. Description: An educational technology company wants to develop a feature that allows students to take quizzes and receive instant feedback Goals: Take a quiz, view quiz results Stakeholders: Educational technology company, students, educators, school administration, investors Pre-conditions: Student must be logged in, student must have access to the quiz Post-conditions: Student can take the quiz and receive instant feedback on their performance System presents the student with the quiz questions Student answers the questions and submits the quiz System evaluates the student's answers and provides instant feedback on their performance System records the quiz results and makes them available for the student, educator, and administration to view Student views the feedback and can review their answers if they desire Student logs into the educational technology platform and selects the option to view previous quiz results System presents the student with a list of previous quizzes they have taken, along with their results Student selects a previous quiz to view System displays the student's answers and the correct answers for each question Student reviews their answers and receives feedback on their performance Description: A social media company wants to develop a feature that allows users to live stream to their followers Primary Actor: Social media users Goals: Start a live stream, end a live stream, view live stream Stakeholders: Social media company, social media users, followers, advertisers, investors Pre-conditions: The user must have a stable internet connection Post-conditions: The user's followers can view the live stream, user can end the live stream at any time User logs into the social media platform and selects the option to start a live stream System accesses the user's camera and microphone and displays a preview of the live stream to the user User starts the live stream, which is broadcast to their followers in real time Followers can view the live stream, engage with the user through comments or other interactions, and can share the live stream to their own followers User logs into the social media platform and selects the option to schedule a live stream System prompts the user to enter the date, time, and title of the scheduled live stream. System sends a notification to the user's followers, letting them know when the scheduled live stream is starting At the scheduled time, the user starts the live stream, which is broadcast to their followers in real-time Followers can view the live stream, engage with the user through comments or other interactions, and can share the live stream with their own followers Use cases can be described either at an abstract level (known as a business use case) or at an implementation-specific level (known as a system use case): Business Use Case: Also known as an "Abstract-Level Use Case", these are written in a technology-agnostic manner, simply referring to the high-level business process being described (e.g. "book return") and the various external actors that take part in the process (e.g. "borrower", "librarian", etc.). The business use case will define the sequence of actions that the business needs to perform to give a meaningful, observable result to the external entity. System Use Case: Also known as an "Implementation Use Case", these have a more granular level of detail than the business use case and refer to specific processes that will be carried out by different parts of the system. For example, a system use case might be "return book when overdue". It would then describe the interactions of the various actors and the system to carry out the end-to-end process. Typically, you will start by defining the high-level business use cases. As the system requirements get defined, they will be drilled down into one or more lower-level system use cases. One related artifact is the "business scenario" or user story. These are similar to use cases in terms of what they seek to accomplish — a description of how the system will carry out a specified business process to fulfill the stated requirements. However, unlike a use case (which is a step-by-step enumeration of the tasks carried out during a process), a scenario is much more free-form. A user story is typically a narrative "short story" that describes the tasks that the users carry out, what information they see, and how they interact with the system. User stories have become more popular with the rise of Agile Methodologies that emphasize customer collaboration, user interaction, and simplicity. In addition to use cases and user stories, diagrams can be used to more clearly illustrate the set of use cases that are provided by the functionality in a system. These contain both the actors that will be using the system and the discrete use cases (or goals) that the users will be carrying out. These diagrams are typically represented in the UML modeling language, though other forms do exist. They help the business analyst convey the relationships between the actors and their business goals and how the design of the system needs to support their different objectives with integrated business processes. While UML use-case diagrams depict the different actors and goals, you can also use process flow diagrams to display the steps that will take place in each process: In the simplest form, it can be a linear flow from start to finish. In more complex use cases, you may have multiple branches and decision points. In this case, it would become a fully-fledged flowchart. Use cases are often used as a means of discovering and representing functional and system requirements because they define the interactions and tasks necessary for carrying out specific business objectives. However, they are typically not the best way to define non-functional requirements such as technical requirements or system qualities. A requirements traceability matrix is used to ensure completeness — namely that all functional requirements are covered by at least one business use case, and that all system requirements are covered by at least one system use case. Typically, you’ll need to ensure that there is complete requirements test coverage for a successful quality assurance program. Use cases provide a good starting point for the design of test cases that will be used to test that the system meets the specified requirements. Once the requirements engineering activities have been completed and the business analysts are happy with the requirements definition, the team can create test cases based on the system use cases. This usually involves adding more detailed pre-conditions and post-conditions and writing different test case "variants" of the same use case to cover different scenarios. SpiraTeam manages your project's requirements, use cases, tests, tasks, source code, and issues in one integrated environment, with full traceability throughout the product development lifecycle. You can also connect your requirements to your other product artifacts for end-to-end traceability, as well as linking requirements to your test cases, issues, tasks, defects, builds, and source code. To learn more about SpiraTeam and how it can be used to improve your requirements and use case management: Take a tour of the available features Read some of the testimonials from satisfied customers Sign up for a 30-day trial version to try it out for yourself Get Started with Spira for Free And if you have any questions, please email or call us at +1 (202) 558-6885 What defines a use case? A use case describes user interactions with a system to achieve specific outcomes, illustrating success and failure scenarios. How do use cases differ from user stories? Use cases offer detailed interactions, while user stories summarize user goals in simple terms. What are the key benefits of use cases? They help manage project scope, establish requirements, visualize architecture, and communicate with stakeholders. What is a use case model? This is a visual representation of actor-system interactions, often using UML, showing processes, preconditions, and triggers. How can Wrike assist with use cases? Wrike provides a prebuilt requirements management template to streamline project tasks and ensure stakeholder alignment. A use case is a concept used in software development, product design, and other fields to describe how a system can be used to achieve specific goals or tasks. It outlines the interactions between users or actors and the system to achieve a specific outcome. In this article, we’ll dive into the details of what use cases are, how they are used in software development, and their benefits. We’ll also explore common types of use cases and provide some tips on how to create effective use cases. Moreover, to help you effectively manage your project’s use cases, we’ll offer a prebuilt requirements management template that can help you gather all the necessary information and ensure all stakeholders are aligned on the project’s goals. Use cases explained A use case is a description of the ways in which a user interacts with a system or product. It may establish the success scenarios, the failure scenarios, and any critical variations or exceptions. A use case can be written or made visual with the help of a use case model tool. Is a use case the same as a user story? Not exactly. While use cases and user stories describe interactions between a user and a system, they’re different tools with different purposes. User stories are simple sentences that describe what a user wants to accomplish. For example, “As a user, I want to log in to my account so I can view my orders.” That’s a simple user story. Business analysts and developers often use both tools together. While the use case digs into the details, user stories keep things simple. Together, they help teams understand how to build successful products. The history of the use caseSwedish computer scientist Ivar Jacobson presented the first article on use cases in 1987, describing how the technique was used at telecommunications company Ericsson to capture system requirements. In 1992, Jacobson co-authored the book “Object-Oriented Software Engineering — A Use Case Driven Approach,” which helped popularize use cases for specifying functional requirements in software development. Jacobson later joined American software engineers Grady Booch and James Rumbaugh to create the Unified Modeling Language (UML), a programming language that introduced a standard way to visualize the design of a system. Since then, the technique has been adapted into use case writing “templates” to streamline the capture of high-level requirements. What is the purpose of a use case? The purpose of a use case is to: Manage scope Establish requirements Outline the ways a user will interact with the system Visualize system architecture Communicate technical requirements to business stakeholders Risk management Types of use cases How do use cases help teams get things done? Here are the most common use cases to help you build successful systems. Business use case A business use case describes the high-level goals and interactions between a business and its users. It focuses on business processes and helps teams understand what the business wants to achieve. Let’s say you’re developing a mobile app for online shopping. A business use case might explain how users browse products, add them to their carts, and make purchases. System use case This type of use case breaks down every step of the interaction between the user and the system, defining exactly what happens behind the scenes. A system use case would explain what happens when the user logs in to an app, browses products, and places an order. It is important, especially for developers, because it shows exactly how the system should function. It includes technical details like how the system handles errors and what steps are needed to ensure a smooth user experience. Test case A test case checks whether the user can accomplish their goal without problems. If the use case involves a customer logging in to their account, the test case will check if the login process works smoothly. Does the system accept the correct username and password? Does it show an error if the password is wrong? This type of use case is important for ensuring the system functions as intended and helps catch any bugs or issues before the product goes live. Why do project managers need to know about use cases? Project managers need to know about use cases because they help communicate strategy to stakeholders and bridge the gap between business justification and technical requirements. PMI also notes that “use cases provide a structure for gathering customer requirements and setting the project scope.” But what does that mean in practical terms? Let’s say that you are a project manager for an education tech firm. Your company’s latest product idea is an app for students where they can receive live tuition for a monthly subscription fee. Creating a use case for this application can tell stakeholders and the project team who the customer is, how the customer will interact with the product, and what the scope gap meaning and requirements of the project will be. How to write a use case for a project When presented in written form, a use case can be a helpful piece of project documentation. Use cases are a common requirements artifact, and they can smoothen communication across technical and business stakeholders. Depending on the intended audience and system under discussion, the use case can be as detailed or basic as needed. A use case document should establish and identify a few key components — these are: System: A system is the product, service, or software under discussion. Objective: This is the goal the use case aims to achieve. Preconditions: These are conditions that must be true before the use case begins. Actors: An actor is a user or anything else that exhibits behavior when interacting with the system. The actor could be another system, a piece of hardware, or an entire organization. There are four types of actors: a system under discussion, an internal actor, a primary actor, and a secondary actor. The most commonly referred to are the latter two systems. A primary actor initiates the interaction with the system, while a secondary actor may provide a service to the system. Basic flow: This is the ideal sequence of actions where everything works as expected. It’s the main, happy path of how a process should unfold. Scenario: In “Applying UML and Patterns,” Larman notes that “a scenario is a specific sequence of actions and interactions between actors and the system under discussion; it is also called a use case instance.” Use case: A use case outlines the success and failure scenarios that can occur when the actor(s) interact with the system. In this section, you’d establish the main success scenario, i.e., the most desirable outcome between the actor and the system. You would also establish the alternate flow, which explains what happens in the event of failure or error. Postconditions: This is the state of the system and actors after the use case has been completed. Instead of writing a use case from scratch, you can use Wrike’s requirements management template to organize every detail of your project. Wrike’s template makes it easy to assign tasks, track progress, and collaborate with your team. Everyone can see updates in real time and stay aligned on the project’s objective. Simple use case example Use case for meal delivery application: Individuals can use an app to place food orders directly to restaurants. When the user places an order, they are prompted to pay through the app or pay when the food arrives. Once that’s confirmed, the restaurant will receive a request through its system. The food will then be prepared, packaged, and delivered to the individual. In this case, the app must be able to receive orders, process payments, and communicate with the restaurant electronically. System: Food delivery application Primary actor: Customer ordering a meal Scenario: The user browses restaurant options. Once the preferred restaurant is selected, they place an order through the application. The user pays online or verifies they will pay in person. The order is sent from the app to the restaurant’s internal system. The restaurant worker receives and processes the electronic order. This use case illustrates how both the customer and restaurant employee (the actors) interact with the food delivery application (the system) and the expected outcome of each interaction. This helps sketch a framework for what is expected in the development stage. The app must be able to process payments, for example. What is a use case model? A use case model is a visual representation of the interactions between an actor and a system. As PMI also notes, use case models depict processes, which helps to further express preconditions and triggers. A use case model is commonly expressed using UML. In these visualizations, there are three main components: the system, the actors, and the use case. The system is represented by a rectangle or “boundary box.” Actors are shown as stick people outside of the boundary box, while the use cases are presented as text in ovals within the box. Solid and dashed lines represent the association between the actors and the system’s use cases. What’s the difference between a use case model and a use case diagram? A use case diagram is simply a type of use case model. A use case model diagram uses text and shapes to represent the relationship between a user and a system. Primarily, use case model diagrams are used to: Visualize the flow and behavior of the system Illustrate the functionality of the system Represent key system user interactions Depending on the system, a use case model diagram can vary in complexity, showing basic associations or expanding to show multiple exceptions. Use case model diagram example Build a use case in Wrike Skip the effort of creating a use case from the group up. Building use cases with Wrike can streamline your product development process and help ensure your software meets the needs of its users. With Wrike’s requirements management template, you can track all of your use case requirements in one place. When it’s time to plan and execute your project, Wrike’s project scheduling template can help you create a clear, actionable plan that keeps your team on track. Try Wrike today and see how easy it is to incorporate use cases into your product development process.