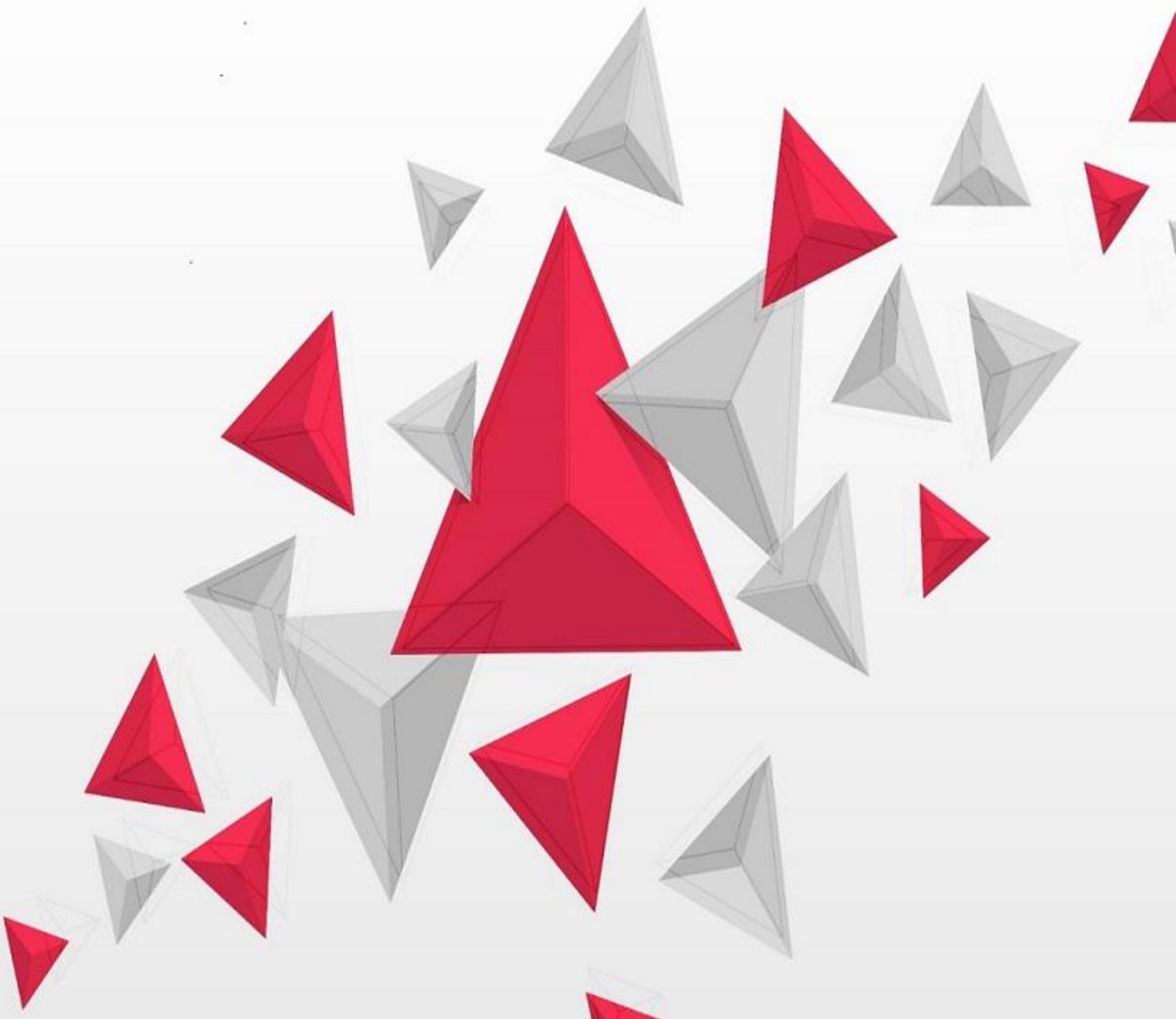


Use Case Guide Book





USE CASE – INTRODUCTION

I have often seen this when I ask business analysts about use cases; most of them refer to the use case diagrams. The fact of the matter is that they are very different from each other – 'Use case diagrams' are a '*modeling technique*' and the 'Use cases' are 'requirement specification document'. Although they complement each other well, they are not supplements of each other!

Use cases are widely accepted as the cornerstone of requirement documentation in waterfall-based projects and accurately describe how a user or actor will interact with the system being developed to achieve a specific goal. Use cases are always written from the 'perspective of a user', and that's why the name *Use* cases. Let's have the definition:

A use case is a methodology used in system analysis to identify, define, and organize the system requirements that come under the project's scope.

Simply put, a use case is a method to define the project requirements in a structured format.



ASPECTS OF A USE CASE

Let us see the salient features of a use case document.

- **Records scenarios in which a user will interact with the system**

A use case is created from the perspective of a user and records all the scenarios in which a user will interact with the system.

While writing a use case, these scenarios are written in the form of steps that the user takes to achieve an objective. These steps are categorized as:

- '*normal flow*' which is the basic flow for attaining that objective and
- '*alternative flow*', which is taken when any exceptions are encountered in the normal flow

- **Organizes the functional requirements**

Use case usually stresses the *functional* requirements where a specific set of project functionalities are organized into a sequential fashion within the document.

As a refresher - Functional Requirements describe the operations, capabilities, and activities, a system must perform, *and* they specify the overall behavior of the system to be developed.

- **Defines other aspects like conditions, UI elements, business rules, etc...**

Use cases not only determine the functional requirements but also identify the preconditions of functionality and its post-conditions.

Additionally, it lists the UI guidelines and associated design notes, and finally, it categorizes the *business rules* that will control how functionality should function.

- **Iterative in nature and updated throughout the project lifecycle**

Since requirements are dynamic and will change based on the business priorities, so does the document that elaborates them. Use cases are iterative and are continuously updated throughout the complete project lifecycle. Document revision is controlled by versioning, and each new version is supplemented with proper versioning comments. Versioning highlights how a requirement has evolved over time and keeps track of the changes that have happened to the projects since their inception.

- **Facilitates testing and quality assurance efforts**

Use cases provide a valuable groundwork for testing documentation, and testers can design their test cases based on the scenarios already identified in the use case document. The quality assurance team can have a seamless experience while testing the functional as well as non-functional requirements by merely going through the use case document.



BENEFITS OF A USE CASE

For the use cases, let us talk about the benefits of using them rather than their audience.

Reason? Almost all the stakeholders associated with the project will refer to at least one of the use case documents throughout the project's lifetime.

There are intricate, multi-moduled projects with even 100 plus use cases, but putting in so much effort has its own advantages. So, let's see some of the benefits of creating thorough use cases:

1. It fits in any development methodology

Whatever the development methodology, use cases fit in as the perfect requirement documentation artifact – they are pretty much the standard in waterfall project, and owing to their comprehensive nature, they are now being used in agile projects.

They are written in a simple language that any business or technical stakeholder quickly understands, and that's why receiving feedback is easy.

2. Splits complex requirements into manageable scenarios

Use cases are '*user*' as well as '*feature-driven*'.

You can't discuss a lot of features in a use case, but you can always discuss a lot of scenarios for a specific function. This way, the use case documents are more elaborate, focused, and manageable. In case of any requirement change, you don't have to go and modify the complete project documentation; you will simply open that use case containing that particular functionality and edit the same.

3. Drives the complete development efforts

Use cases act as a base not only for the tester but also for the dev team.

After an initial round of requirement explanation by the BA, the developers can always go back to the use case document for any clarification regarding the functional and non-functional flows. Thus, saving everybody from the numerous 'ask and explain' sessions.

4. Acts as an excellent communication tool

Experienced business analysts use the use cases as a communication cum clarification tool.

They first elicit the requirements with the stakeholders, then based on the discussion, they detail out the same in the use cases, which are finally shared with the client for consensus and feedback.

Clients are always delighted to find a detailed requirement document and eagerly give reviews, often as comments in the use case document itself. The BA then incorporates those comments, edits the use case, and sends it back to the client for a final review and approval. Post-approval, this document then becomes a baseline use case document for all stakeholders and minimizes any chances of project scope creep.



HOW TO CREATE A USE CASE

I want to call your attention as we are deep diving into creating a use case document, and carefully authoring it is the primary responsibility of a Business Analyst. Let's start with looking at the sections of a use case and what goes into each one of them:

- ***Use Case Name***

This heading consists of the title or a short name of the use case. This title should depict the use case's goal and should be using an active verb like 'Create User'.

- ***Use Case ID***

The use case ID is a unique numeric identifier for the Use Case and is usually created by following the terminology of the project name followed by a sequence number. E.g., UC-UMT-1.05

- ***Brief Description***

Here we will briefly describe the 'reason for' and 'outcome of' this use case. This description should give the user a context of what will be contained in the use case and should be a logical expansion of what you entered in the 'Use Case Name' field. For our 'Create User' use case, this could read like:

This use case describes the actions and activities that are carried on the Create User page. On this page, we shall describe the necessary details, specify the Account Security, provide Additional Information, and define the User Management Capabilities for the user being created.

- *Trigger*

A trigger identifies the event that initiates the use case - this could be an external business event or system event that causes the use case to begin, or it could be the first step in the normal flow of the use case. For e.g.

The user clicks on the 'Create User' icon on the application header.

- *Preconditions*

Preconditions list any activities that must take place or any conditions that must be true before the use case can be initiated. They describe the state the system should be in so that the normal or exception flow of the use case could be started. E.g., for the same create user scenario, the precondition could be:

1. The logged-in user should be a registered user of the application
2. User must have the user creation permission
3. The user must be at the 'Create User' page

- *Post-conditions*

Describe the system's state after the use case execution and include the conditions for both the positive (normal) flow and the exception flow. E.g.:

1. A user is successfully created with the assigned permissions
2. The user gets a confirmation email on his registered email ID

- *Actor(s)*

An actor is a person or an entity external to the system which interacts with the system and performs actions to accomplish a task(s). Different actors often correspond to different user classes or roles.

In some cases, there could be two types of actors in the use case:

- Primary actor - someone who will be initiating this use case
- Secondary actor - someone who will participate in completing the use case.

Along with the actor, their type and role are also specified in the use case document.

- *Main Flow*

The main flow is sometimes also called a 'positive or normal' flow, and this flow provides a detailed description of the user actions and system responses that will take

place during the execution of the use case under normal, expected conditions. Take a look at the example:

The user arrives at the 'create user' page, and the system presents him a form containing fields to create a user.

1. **First Name*:** This field is a text box and allows the user to enter the 'First Name' of the user being created.
2. **Last Name*:** This field is a text box and allows the user to enter the 'Last Name' of the user being created.
3. **Email*:** This field is a text box and allows the user to enter the 'Email ID' of the user being created. The email ID will be different for every user and will be the login ID of the user being created.
4. **Phone:** This field is a text box and allows the user to enter the 'Phone Number' of the user being created.
5. **Password:** This field will allow the user to define the password of the user being created.
- .
- .
- .
- .

And the use case continues like this.

- **Alternative Flows**

The Alternative Flows describes the legitimate branches from the main flow to handle special conditions (also known as extensions). For each alternative flow, we should reference the branching step number of the normal flow and the condition, which must be valid for this extension to be executed.

For e.g., If a user is already created in the application with the email ID entered, the following are the events:

1. The system prompts an alert pop-up reading, 'This email ID is already in Use. Please enter a different email ID.'
2. The user clicks on the 'Ok' button on the alert pop-up
3. Use case resumes at step 4, i.e., the user is sent back to the Email field, and the email ID field is set to blank.

- ***Exception Flows***

The exception flow describes any anticipated error conditions that could occur during the use case execution and defines how the system responds to those conditions.

For e.g., If the user enters alphabets instead of digits in the phone field

1. The application prompts an alert pop-up reading, 'Please enter numerical only.'
2. The user clicks on the 'Ok' button on the alert pop-up
3. Use case resumes at step 4, i.e., the user is sent back to the Phone field

- ***Related Use Cases***

This header list any other use cases that are functionally related to the scenarios or steps mentioned in the current use case.

- ***Business Rules: Use Case***

Business rules are specific rules or conditions imposed by the business and are considered business requirements that must be met by the use case being created. It is always a good practice to include these business rules in the use case document to make sure the conditions are explicitly stated are not missed.

- ***UI Notes***

Sometimes, particularly in product and application development projects, the UI notes, wireframes, or sketches are also included so that the business users and the development team get a fair amount of idea as to how the actual screen should look like post-development.

- ***Special Requirements***

This section identifies any additional requirements, such as non-functional requirements that may need to be addressed during the design or implementation phases. These may include performance requirements or any other quality attributes.

Well, we have discussed a lot of fields above, and by now, you should have a fair amount of ideas as to what all goes into creating a use case.

Several tips and best practices should be followed for creating use cases. Wouldn't it be great if you could get your hands on them? So let's dive in.



USE CASE - BEST PRACTICES

1. Develop use case iteratively

It's always advisable to create a lightweight use case while the requirements are still being discussed as it gets the documentation ball rolling and avoids *analysis paralysis**.

Additionally, these use cases could always be elaborated, and a newer version could be created in the event of new scenarios and flows being added to the primary requirement.

**A situation in which requirements are over-analyzed, but a decision couldn't be taken around them, thus hindering and paralyzing any outcome.*

2. Consider use cases as a basis to capture the non-functional requirements

It's always challenging to elicit the non-functional requirements (also called as *NFRs*) for any project as neither the client nor the analysts can fully wrap their mind around the complete set of non-functional requirements for all the functionalities of the project. Thus, they are limited to phrases like – *low response time, intuitive user interface, and high security*.

However, analyzing and collecting the NFRs at the time of extracting the functional requirements of the use case gives a better understanding of the functionality as a whole, along with a reduced effort to collect them at a later stage. Also, this way, the NFRs are more exact and functionality appropriate. E.g., *The device performance reports should be rendered within 4 seconds, while all the other user-related reports should have a response time of 5-7 seconds.*

3. Always have a single main flow and multiple alternate flows

The main flow, sometimes called the positive flow, contains the flow of events that will be followed when everything goes right.

4. Strike the right balance between size and complexity of a use case

I have seen this time and again that some BA's keep their user cases very short, just one scenario per use case. In the end, their complete project amasses 150 use cases – *which is* a daunting situation for both the reviewers of the use case and the developers, not to forget the testers.

And, there is another class of BA's whose use cases are 40+ pages long with a puzzle of alternate and exception flow references. Such use cases might land an end-user with more confusion than comprehension.

So what's the right balance, you ask?

As a thumb rule, your use case should have a single flow, not more than 5 alternate flows, and at most 3 exception flows. Additionally, it spans more than 20+ pages of content; you should be alarmed as you are probably clubbing two use cases.

5. Use cases are written for the end-user and not the system developer

While creating a use case, we often forget that it is created from an end user's perspective and not a developer's. So, please think twice before adding all the technical details you have mentioned there, or at least keep them in a separate section or heading in the use case.

6. Ask - Have I really verified my use case document?

There are several questions you can ask yourself when verifying a use case. Like:

- Is the use case within the project scope?
- Is this use case testable?
- Has the use case diagram been updated to support references made in the use case?
- Are all the business rules 'accounted for' in this use case?
- Are all the exceptions and alternate flows considered, or am I missing something?

After asking similar questions, I assure you that you will come up with at least two points you have missed mentioning in the use case document. Follow this

best practice, consider yourself as a reviewer verifying somebody else's use case, and you will see that you will get way fewer review comments when peers and project managers review your use cases.