Integration of Use Cases and User Stories

A report from the Reframing Use Cases working group

Introduction

This document provides guidance to people who wish to integrate the usage of **use cases** and **user stories**.

The advice shared in this document was previously discussed in the *"Use Cases and User Stories: Perfect Partners" webinar* in the summer of 2024 – attended by working group members: **Mike Cohn**, **Ian Spence**, **Ivar Jacobson** and **Keith de Mendonca**. **Gunnar Overgaard** is also a member of this working group.

If you *only* use user stories today in your work, we hope this document will encourage you to investigate how and when use cases could help you communicate the "big picture" to your stakeholders - and describe your intended system behavior with more clarity.

You can **watch the whole webinar here**, and you will also find links to relevant sections of that webinar throughout this document.

Use Cases and User Stories

Use Cases are an established and well-known technique to describe the behavior of systems in a structured manner.

"A use case is all the ways of using a system to achieve a goal of a particular user."
(Ivar Jacobson & Alistair Cockburn)

Use cases help users, developers, customers and other stakeholders communicate easily with each other, see the big picture, agree on the scope of a system, and plan together how changes to that system will be delivered as incremental releases. If you are new to use cases, take a look at these two papers written recently by Ivar Jacobson and Alistair Cockburn **[1][2]**.

User Stories are a popular technique to describe small work items from the perspective of the user who wants the system to do something for them.

"*User stories are short simple statements telling us something a user wants*" (Mike Cohn).

If you are new to user stories, Mike Cohn has written an excellent book on this topic **[3]** and also has many resources available on his Mountain Goat website.

The development team (should!) treat user stories as "*a promise for a conversation*" with the user (or user representative) to learn more detail about that expected behavior. User stories are prioritized against each other in a backlog to plan how the work will be sequenced, implemented and delivered to the customer by the development team.

One benefit of user stories is that they introduce the concept of defining something a user needs - which can be delivered by a team within a specific timebox such as a sprint, or iteration. In contrast, requirement statements (“*the system shall…*”) or use cases normally have no explicit relationship to time: some may require only a small amount of work to implement them, while others may require a huge implementation effort.

Use cases and user stories are both used widely in organisations. But our experience has been that software developers prefer to work with user stories and they do not always understand the potential limitations of that technique. For example: authors may be tempted to capture too much information in a user story - which undermines the purpose of user stories, and mistakenly treat them as a specification rather than the placeholder for a conversation.

Business analysts (BAs) like use cases because they provide a structure to their analysis and they provide a simple language to communicate with their stakeholders. However, these use cases are often not shared with the development teams, so a rich source of information does not get communicated to the people who build the system. Sometimes BAs are asked by developers to write the user stories. But BAs may be unsure how to leverage their pre-existing work on use cases, and how to retain traceability between the two sets of work items.

Use Cases and User Stories - together

Use cases describe the desired behavior of a system in a clear structured way that helps to identify the user’s needs, and they are excellent at conveying the “big picture” of the system as a whole. But use cases do not make good backlog items as each use case may take a completely different amount of time to implement.

User stories are simple statements of one of the users’ needs and are focused on identifying what to do in the next iteration. They represent the absence from the current system and consequently struggle to convey the big picture.

Both techniques are based on the idea that telling stories is an effective way for people to communicate ideas and user needs to each other. Both techniques focus on identifying the desired behaviors but work at different levels of abstraction. The two techniques are actually complementary and there are clear benefits for people to use them together.

But how exactly can they be used together? This document describes a set of simple patterns for using use cases and user stories together. The first three patterns focus on creating user stories from use cases and seamlessly bridging the gap between the business analysts, the developers and the other stakeholders. We also have a pattern for situations where you have built a system using user stories - but have lost track of the big picture of when and how it will be used. Finally, we present another pattern where you have a single epic, then create a use case to benefit from its structure, and finally create a number of user stories afterwards to drive the implementation work.

	Pattern Name	**Useful when...**
1	Brainstorm user stories directly from a use case	You do not have a use-case narrative for a use case. You do have a use-case model, or you have someone who can describe the requirements.
2	Follow the extensions	You have the use-case narrative documented for a use case.
3	Slice my use case	You are using Use-Case 3.0 concepts and practices to identify and implement use cases in an agile manner.
4	My use-case model emerges	You need to create a use-case model to describe a system that already exists, using existing user stories, test cases and documentation.
5	Understand your epic	New idea is captured as a user story or an epic, but you also wish to describe your requirements with use cases.

Pattern 1 – Brainstorm user stories directly from a use case

This is the most basic approach to producing user stories from use cases. This approach requires access to the users and other stakeholders who want the system to be created. It can be used in situations where a use case has been identified and described as part of a use-case diagram or model, but where the use-case narrative (i.e. a description of the basic scenario, extensions, and other related information) does not exist.

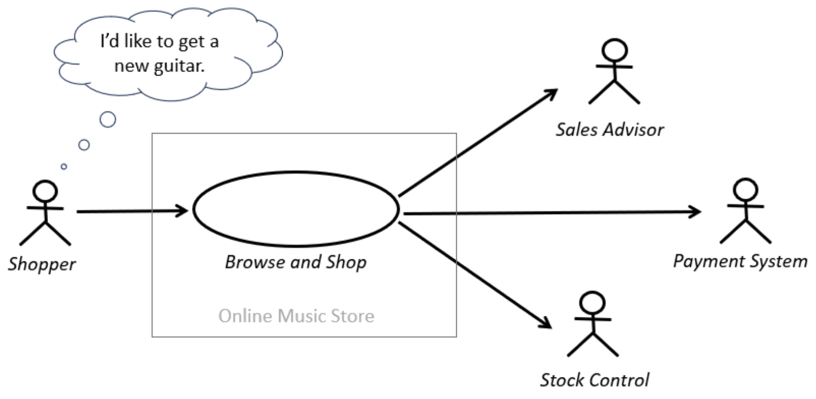


Figure 1: Browse and Shop use case

Step 1: Stakeholders (i.e. users and people representing the customer together with members of the development team) discuss and agree which use case(s) will be implemented. They must agree on:

- What is the name of the use case?
- What is the system of interest?
- Who is the primary actor, and what is their goal?
- Are there any supporting actors?

If you already have a use-case model, then this work has already been done - you just need to select which use case you would like to implement from the model. If not, then draw the use-case diagram during the discussion.

Step 2: Stakeholders use their shared knowledge of the use case to brainstorm the user stories required for the user to achieve their goal. Don't forget to also consider any errors the user may encounter whilst trying to reach their goal – and capture those user stories as well.

Step 3: Review the generated user stories to verify they are clear, and each story provides some value to the user.

Step 4: Agree which set of user stories will be in-scope for the planned work.

Step 5: Split each selected user story into a set of smaller user stories, if required, to ensure each user story is small enough for the team to complete in one iteration. (e.g. using the SPIDR splitting technique).

Benefits of applying this pattern are:

- The brainstorming session is quick to set up and run.
- Stakeholders do not need to have a detailed knowledge of use case concepts to take part.
- It does not rely on a description of the basic scenario and extensions to exist for the use case to be discussed.

By starting with an individual use case to provide context for the user stories they are generating, stakeholders will gain more clarity on which system is being described and agree what is the high level goal of the user. This may seem an unnecessary or irrelevant step, but experience tells us that when a disparate group of people come together (developers, customers, users, etc) from different domains, and with different contexts - it Is all too easy to lose sight of the big picture during the discussion, or to believe you are all aligned when in fact each person leaves the discussion with a completely different interpretation of the user's goal in their mind.

Watch Ian Spence describe how to apply this pattern in the webinar **here.**

Pattern 2 – Follow the extensions

This approach can be applied if the use-case narrative has been described. Figure 2 shows how the basic scenario of a use case can be described as a series of numbered steps, and how the names of extensions (covering optional and exceptional behavior related to the basic scenario) can each be described and given a unique ID.

This simple and effective pattern identifies an initial set of user stories. The development team can then agree with the stakeholders how they should prioritize work on these stories during the development of the system.

Primary Actor: Shopper	Help the shopper to find the most suitable product to meet their needs and help them to purchase it.
Basic Scenario The use case starts when a Shopper indicates they'd like to find a product 1. Browse Products 2. Select Products for Purchase 3. Provide Payment Details 4. Provide Delivery Details 5. Confirm Purchase The use case ends.	Extensions Ext 1 – Keyword search for products Ext 2 – No products selected Ext 3 – Invalid payment details Ext 4 – Payment system unavailable Ext 5 – Retrieve stored payment and delivery details Ext 6 – Invalid delivery details Ext 7 – Product out of stock Ext 8 – Stock control system unavailable Ext 9 – No purchase confirmation
Constraints (pre-conditions and post-conditions): 1. Shopper can only purchase products which are in-stock.	Ext 10 – Quit shopping with no purchase Ext 11 – Shopper stops responding Ext 12 – Shopper needs expert advice

Figure 2: Example showing the basic scenario and an initial set of extensions for the "Browse and shop" use case.

Step 1: Create a user story to implement the basic scenario for the user to achieve their goal.

If that user story is too large to implement in one iteration, here are two alternate strategies you can use to identify a set of smaller stories. One strategy is to create a user story for each step in the basic scenario (i.e. five user stories for our example in Figure 2). Another strategy is to create more specific user stories that flesh out the basic scenario. For example create a story: "As a Shopper I want to buy a single product with my credit card and have it delivered to the address on the card so that …" and then create additional stories to handle multiple products, different addresses etc.

The first team goal for this use case is to deliver the behavior described by the basic scenario.

Note, whilst doing this you will probably identify more candidate use-case extensions that can be added to the list for future consideration.

Step 2: For each use-case extension, consider it as a candidate user story. For example, "Ext 5 – Retrieve stored payment and delivery details" would be represented by one user story, *As a shopper, I want to be able to retrieve my delivery details and my payment details, so I save time and avoid accidentally entering incorrect information whenever I return to the shop to purchase more items*.

Again, split the user story into a set of smaller user stories, if required.

Step 3: Consider any constraints (e.g. only positive numbers will be handled by the system, negative numbers are an illegal input). First relax all constraints – to simplify the initial implementation and testing. During testing, inputs which do not confirm to the use-case constraints are likely to cause a failure of the system, and we should expect this to happen.

Step 4: For each constraint that was previously relaxed, now create a user story to enforce that constraint. With our example: "only accept positive numbers", implement a user story which checks the input data and handles gracefully the error if negative numbers are received.

Step 5: Agree which set of user stories will be in-scope for the planned work.

This pattern delivers a number of benefits:

- It is simple to apply.
- It directly leverages the use-case narrative information (i.e. basic scenario and extensions) we have already created for the use case.
- We can trace each user story back to an element of the use case to ensure that use case will be fully implemented and tested.
- The basic scenario and its extensions provide well-defined iteration goals (i.e. to implement one or more of them in the next iteration) - as the stakeholders have already agreed that each extension describes needed system behavior.

Watch Mike Cohn describe this pattern in the webinar.

Pattern 3 - Slice my use case

Use Case 3.0 defines the concept of 'slicing' a use case as follows:

"Use-case slices are used to stage the delivery of the use case."

"A use-case slice is a slice of a use case that provides clear value to the development stakeholders. It includes a behavioral slice specifying the interactions between the system and its actor(s) and the actions taken by the system, and a test slice verifying the identified behaviors."

Slices **[4]** allow a use case to be delivered incrementally as a series of small development tasks. Each slice delivers *"some of the ways of using a system to achieve a goal of a particular user."* The basic scenario will be implemented in the first slice(s), extensions to the basic scenario will be delivered in later slices. Slices can be identified and refined incrementally during development – you should not specify all slices before the implementation begins.

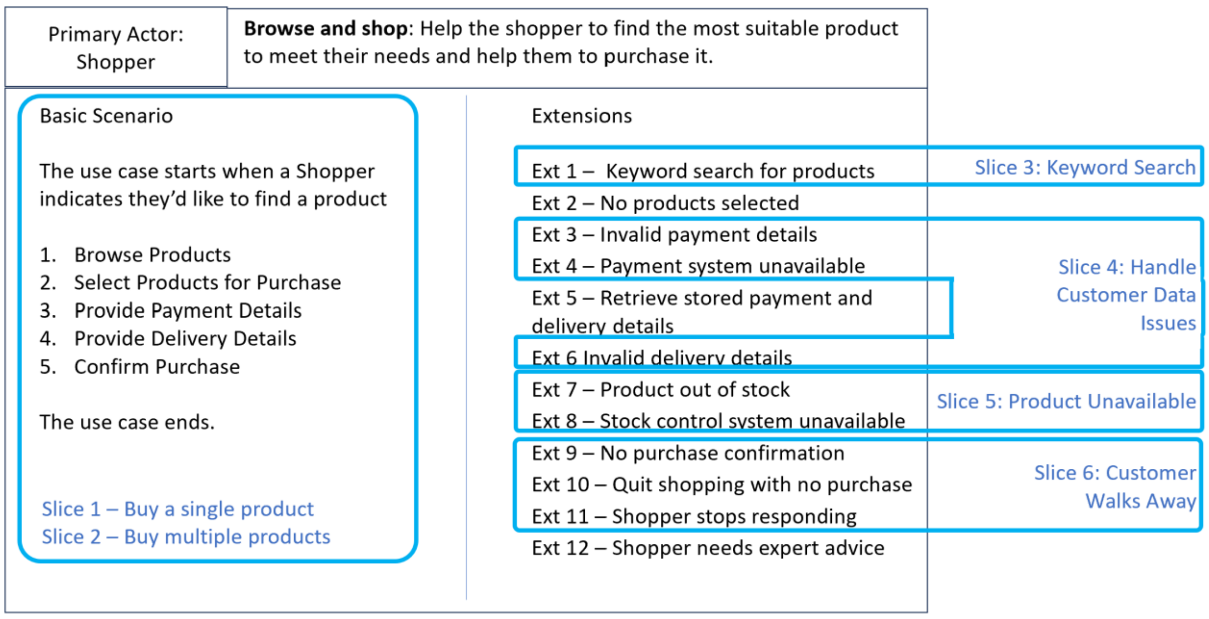
Each use-case slice will be built and tested by the development team before they begin development work on the next slice. If a slice delivers some new value to the user (such as new functionality, or an improved experience), it should be delivered to the user to get feedback on the new work, and to support the user in their daily work. The user, or other stakeholders may decide not to request the developers to build all use-case extensions identified by the use case.

Note that this pattern is similar to "follow the extensions", but with the added benefit that slices can group together sets of extensions into more meaningful chunks (for example in the example above it might make sense to tackle all the Payment System extensions (Ext 3 – 5) in one go. This is very empowering for the stakeholders when considering release planning and incremental delivery without prematurely creating lots of user stories or getting caught up in the details of use cases and use-case narratives.

The slicing mechanism can also help when addressing a use case with a long and complex basic scenario. In this case the basic mapping of step to user story can lead to lots of development work before it is demonstrated that the basic scenario can be implemented end-to-end. Instead, the first slice may deliver a "walking skeleton" simplest level of functionality to get early feedback from users; subsequent slices could flesh out the steps of the basic scenario, or make performance improvements.

The use of use-case slices can also help with end-to-end testing. When working with a use-case slice the first activity (even before identifying the user stories) is to identify a set of test cases that will be used to verify the slice as done. The test case(s) identified for each use-case slice are a critical output of the slicing activity. They will be independent of the approach taken to the identification and sizing of the user stories - ensuring that the real user goals don't get lost as the user stories are split into smaller and smaller pieces. As the test cases are identified before any implementation starts, use-case slicing supports behavior-driven and test-driven development.

Please refer to **[4]** if you want to learn more about use-case slicing.



Primary Actor: Shopper

Browse and shop: Help the shopper to find the most suitable product to meet their needs and help them to purchase it.

Basic Scenario

The use case starts when a Shopper indicates they'd like to find a product

1. Browse Products
2. Select Products for Purchase
3. Provide Payment Details
4. Provide Delivery Details
5. Confirm Purchase

The use case ends.

Slice 1 – Buy a single product
Slice 2 – Buy multiple products

Extensions

Ext 1 – Keyword search for products
Ext 2 – No products selected
Ext 3 – Invalid payment details
Ext 4 – Payment system unavailable
Ext 5 – Retrieve stored payment and delivery details
Ext 6 Invalid delivery details
Ext 7 – Product out of stock
Ext 8 – Stock control system unavailable
Ext 9 – No purchase confirmation
Ext 10 – Quit shopping with no purchase
Ext 11 – Shopper stops responding
Ext 12 – Shopper needs expert advice

Slice 3: Keyword Search
Slice 4: Handle Customer Data Issues
Slice 5: Product Unavailable
Slice 6: Customer Walks Away

Figure 3: Slicing up our "Browse and shop" use case.

Step 1: Using the use-case narrative description of the use case, identify the simplest path through the basic scenario that allows the user to achieve the goal of the use case and / or address any major development risks. A slice based on the basic scenario is guaranteed to travel through the entire use case as it will be the most straightforward way for the user to achieve their goal.

Step 2: Identify one or more test cases necessary to confirm the functionality of this basic scenario. We will call the scope of the functionality, tests and the code required to implement the functionality the first 'slice' of the use case. In our Browse and shop example, the simplest version of the basic scenario we could plan to build and test could be buy one product which is in stock.

Step 3: Brainstorm one or more user stories to implement the functionality represented by the use-case slice. Use the test cases to identify the acceptance criteria of those user stories.

Step 4: Identify any additional work necessary to implement the basic scenario – including any non-functional requirements. Identify test cases required to confirm this enhanced functionality. Confirm with the stakeholders how much scope should be included in our slice, and then brainstorm what user stories are required to implement the scope of this slice.

Step 5: Repeat the process, identifying additional slices, until the full scope of the basic scenario has been delivered and validated by the user. Continuing with our Browser and Shop example, a second slice of the basic scenario might be Browse and shop for more than one product.

Step 6: Identify the next slice to deliver one or a small set of related use-case extensions. Identify test cases required to confirm the functionality, and then brainstorm what user stories are needed to implement the scope of this slice. For example: try to purchase zero products in shopping cart, product is out of stock, etc.

Step 7: Repeat this process with other use-case extensions to identify and deliver each incremental slice – until the user can achieve their goal, and stakeholders confirm they have received sufficient value from the system to satisfy their needs.

Benefits:

- Slices can be used to plan releases with stakeholders: e.g. agree which sets of use-case extensions should be delivered in what order.
- A complex basic scenario of a use case can be sliced into a number of smaller incremental, but valuable steps to deliver that system behavior.
- Use-case slicing supports behavior-driven development, and test-driven development. Tests for each slice are identified before user story definition or backlog refinement occurs – so test cases won't get lost if/when the user stories are split, discarded or rewritten.
- Not all slices need to be identified before development starts. Newly identified use-case extensions can be added to the slice we are currently working on, or we can create a new slice to prepare and complete that work later on.

Watch Ian Spence and Ivar Jacobson describe use-case slicing.

Pattern 4 – My use-case model emerges

A use-case model provides a concise high-level view of the system of interest. It helps you answer the question: “Have I missed anything?” It prevents you from getting “lost in the weeds”, and it communicates the “bigger picture” scope with stakeholders - to ensure that all the important behaviors have been identified and that the resulting system is actually useful. This context is very hard to determine when looking at a large set of user stories on their own.

A use-case model can also be used to display progress during implementation, by indicating on each use case what percentage has been completed.

In some situations, a system may have been created without the owners having a permanent record to describe the scope and purpose of that system. In other situations, the functionality and the usage of a system may have emerged over time – or may have pivoted from its original intended purpose. In any of these circumstances, it may be useful to reverse-engineer the use cases from an existing system.

Step 1: Examine the software, its tests, any original documentation, and user stories – to familiarise yourself with the scope of the system. Be aware that some documentation and user stories may be out of date.

Step 2: Work with the product experts to identify a set of use cases that represent why and how the system is actually used.

Step 3: Present the resulting use-case model to your various groups of stakeholders to 1) verify that the system is fit for purpose, 2) identify any missing behaviors, and 3) identify where to go next.

Step 4: Consider how well each of the use cases has been implemented. Are there any missing steps of extensions? Is there anything that has been implemented that shouldn’t have?

Step 5: Validate that each existing ‘system test’ can be mapped to one or more use cases. If there are system tests that don’t map to the use cases, consider adding the use cases they cover to the model. If there are use cases with no system tests, consider adding additional tests to close the gap.

Step 6: Use the resulting model to decide where to go next.

Benefits:

- Build an accessible, common understanding of a system’s purpose and its usage.
- Organize and manage your system tests (to complement the user story generated developer tests).
- Refocus your development efforts on the true goals of the system’s users and other stakeholders.
- Establish an accessible model to act as your permanent record.

More information about re-engineering old existing systems can be found in **[6]**.

Watch Ian Spence describe when this pattern may be required.

Pattern 5 – Understand Your epic

The final pattern we present here is a collection of some previous patterns. At the start of this document we suggested that "telling stories" is a good way to convey information – both for use cases and user stories. Many development teams write user stories – but when a new user need is first captured, we do not know how much effort will be required to implement it, and it is almost certainly going to be big.

Large user stories (i.e. those which cannot be delivered by a team within an iteration or sprint) are often called epics; many stories start as epics.

We could simply use a story splitting pattern (e.g. the SPIDR splitting pattern) to break an epic directly into a number of user stories. However, there are some benefits in converting a high-level, vague epic into a use case as an intermediate step in the splitting process:

- If we are using use cases to describe the requirements of the system, we will want to add it to our use-case model.
- Translating the epic into the simple structure of a use case (goal, system of interest, actors) will help stakeholders gain more clarity of that user need.
- If we decide to describe the basic scenario and the full set of extensions, it will give us a clear route to split the use case into user stories - where each user story is guaranteed to describe some specific and clear value delivered to the user.

Step 1: Stakeholders (i.e. users and people representing the customer together with members of the development team) discuss the epic being considered. To turn it into a use case, they must agree on:

- What is the name of the use case?
- What is the system of interest?
- Who is the primary actor, and what is their goal?
- Are there any supporting actors?

Step 2: The same group of stakeholders now agree on the what is the basic scenario, and describe it as a sequence of steps that will result in the user achieving their goal.

Step 3: Stakeholders discuss alternate ways for the user to achieve their goal (e.g. optional paths through the use case), and give each extension a suitable name. Additional information can also be captured in this use-case narrative. They will also discuss what problems the user may face that could prevent them from reaching their goal, and give each of those error situations a suitable name as another extension. We now have a description of the basic scenario and a full set of extensions for our use-case, like the example shown in Figure 2.

We can now apply one of our use case patterns to generate a set of user stories from the use case.

Step 4: Apply the "Follow the extensions" pattern, or the "Slice my use case" pattern to create a number of user stories from the use case.

Watch Mike Cohn and Ivar Jacobson discuss the differences between use cases and user stories in the webinar.

Benefits:

- If we are using use cases to describe the requirements of the system, we will want to add it to our use-case model.
- Translating the epic into the simple structure of a use case (goal, system of interest, actors) will help stakeholders gain more clarity of that user need.
- If we decide to describe the basic scenario and the full set of extensions, it will give us a clear route to split the use case into user stories - where each user story is guaranteed to describe some specific and clear value delivered to the user.

Terminology

In the world of use cases, different terms are used by different authors to describe the same concepts.

In this document, we are using the terminology defined in version 1.1 of the Use Case Foundation **[1].** In the webinar, you may hear the participants use slightly different terminology. Here is a table that shows the relationship between terms used in the webinar and the terms used in this document:

Use-Case concept	**Other names for the same concept**
Basic scenario	Basic flow, main success scenario
Extension	Alternate flow
Basic scenario and set of extensions	Flow of events

References

[1] Use Cases Foundation by Ivar Jacobson and Alistair Cockburn. 2024
A concise paper which defines the common ground that lies between the two most widely practiced ways of working with use cases. Version 1.1

[2] Use Cases are Essential. By Ivar Jacobson and Alistair Cockburn. acmqueue. 2023

[3] User Stories Applied: For Agile Software Development by Mike Cohn. Addison Wesley. 2004

[4] USE-CASE 3.0 The Guide to Succeeding with Use Cases by Ivar Jacobson, Ian Spence, Keith de Mendonca. 2024

[5] ***"Use Cases and User Stories: Perfect Partners"*** *webinar* with Mike Cohn, Ian Spence, Ivar Jacobson and Keith *de Mendonca.*
https://youtu.be/yT3FLPR8OFU?si=SVBQTLwlJhkGRXCQ

[6] "**Re-engineering of old systems to an object-oriented architecture**" by Ivar Jacobson and Fredrik Lindstrőm

Find out more about "Reframing Use Cases" by following our posts on LinkedIn.