# 16 Running with Use Case Lite

The goal of this chapter is to introduce the Use Case Lite practice and its elements, and to demonstrate its application within the TravelEssence journey. In this chapter, the reader will be shown

- the differences between use cases and user stories;
- the elements of the use case practice, including relationships between elements, activity flows, and relationships with kernel elements;
- the concept of use-case slices and their benefit when use cases are used together with Scrum;
- the importance of monitoring progress and health of use cases and use-case slices;
- a simplified version of the use case practice (called Use Case Lite) in a real endeavor, together with the obstacles and challenges that might arise; and
- the coverage of kernel solution activity spaces by the Use Case Lite practice.

In this chapter, we will see why and how Smith's team started applying use cases in their development. The use case practice is a requirements analysis technique that has been widely used in modern software engineering since its introduction by Ivar Jacobson in 1987 [Jacobson 1987]. In fact, Ivar Jacobson introduced a larger practice, Use-Case Driven Development (UCDD) that extends the use case practice beyond requirements analysis to driving design, implementation, and testing. Since its introduction, the use case practice has been widely adopted in basically all industries and inspired the user story practice we presented in the previous chapter. In fact, the use case idea has become so widespread that the term "use case" has become a normal English expression used to understand the usages of virtually anything. The use case practice has evolved since 1987 and in its turn

has become inspired by the lightness of the user story practice, making it practical to use in all kinds of endeavors and in particular in agile endeavors [Jacobson et al. 2011, 2016]. Similar to what we did with Scrum and user stories, in this chapter we will describe a simplified version of the use case practice, which we refer to as Use Case Lite. It is designed for this book only and to be exchangeable with the user story practice; the better way to handle use cases is provided by the UCDD practice.

Smith's team had already been working with product backlog items as part of their Scrum explicit practice, as previously described. These product backlog items frequently had relationships between them that were not evident from the simple product backlog list. For example, some product backlog items provided basic functionality (meaning high-level requirements that do not include details), while other product backlog items provided additional extended functionality. Use cases can help teams understand the bigger picture and how product backlog items are related.

## 16.1 Use Cases Explained

*A use case is all the ways of using a system to achieve a particular goal for a particular user.* Use cases [Bittner and Spence 2003] help teams understand how user needs and requirements affect the behavior of the system. Often, at the start of a software endeavor, development teams are given a requirements specification which is essentially a narrative that supposedly captures the requirements for the system to be built. As shown in Chapter 15, user stories can help flesh out missing requirements by encouraging informal discussion between developers and users. However, often these informal discussions lead to user stories that are fragmented—there are too many of them, it is not clear how they make up something more complete—and they lack structure. This can become a significant problem especially for large complex systems. One approach that has been used to help address this problem is developing larger stories first, which are referred to as epics (see Section 15.1). This is an area, though, where use cases may be more helpful to teams. Use cases provide a systematic way to organize requirements. This structure makes it easier for teams to conduct analysis and orchestrate facets such as user interface (UI) design, service design, implementation, tests, and so on.

In the Unified Modeling Language [Booch et al. 2005], the relationships between users and use cases are represented in what is referred to as a use-case model—a model of the use cases within a system. Since users are not always human but can

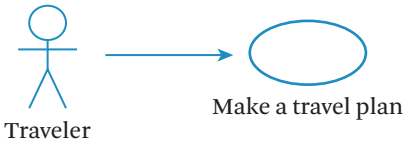



**Figure 16.1** Use-Case Model example: "Make a Travel Plan."

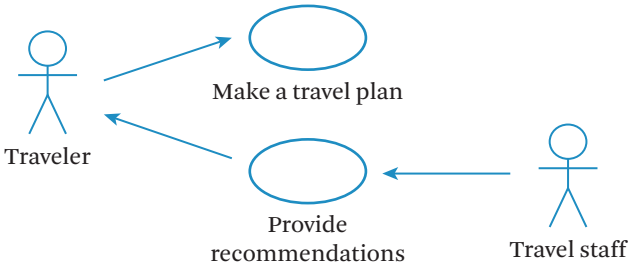



**Figure 16.2** Use-Case Model for TravelEssence providing travel recommendations.

also be other systems, we use a more general term than users and speak instead about actors.

As you will recall, TravelEssence was a leading travel service provider that targeted both leisure and business travelers. One of their customer-facing systems, a travel portal, had the use case shown in Figure 16.1 (among others, but right now we will pay our attention to only this one).

The stick figure is the UML symbol for an actor and the oval is the UML symbol for a use case. A use-case model provides a visual representation of the software system, which is a very useful way for brainstorming its overall scope. What we show in Figure 16.1 is a use-case model with only one actor and one use case. In real systems, there are many actors and use cases. Figure 16.2 shows the actors and use cases for the development endeavor our TravelEssence team had just embarked upon with the following requirement items implemented:

**Req-Item #1.** System generates recommendations for a traveler

**Req-Item #2.** Mobile plug-in displays recommendations

**Req-Item #3.** System handles user's selection to view or discard recommendations

**Req-Item #4.** System tracks recommendation success rate

**Basic Flow:**

1. Traveler provides travel details (travel dates and destination).
2. System searches and displays hotels for the travel dates and destination.
3. Traveler selects a hotel and room type.
4. System computes and displays available rooms and prices.
5. System makes a tentative reservation for the traveler at the selected hotel.

**Alternate Flows:**

A1. Travel plan with multiple destinations.

A2. Travel plan with a single destination but non-consecutive dates.

A3. Travel plan with non-consecutive dates and multiple destinations.

**Figure 16.3** Use-Case Narrative example: "Make a Travel Plan."

The reader should not jump to the conclusion that use cases are just about diagrams. Use cases include the actual functionality and behavior of the system. Each use case is described in a use-case narrative. A use-case narrative provides a textual description of the sequence of interactions between the actor and the system. It also describes what the system does as a response to each message from the actor. This response includes internal actions as well as the sending of messages back to the actor or to other actors (which could be other systems). Figure 16.3 shows an example use-case narrative for the "Make a travel plan" use case. The use-case narrative is usually separated into two parts, referred to as the basic flow and the alternate flows. The basic flow describes a normal use of the described use case, often called the *happy day scenario*. In our TravelEssence case, this would be making a travel plan. The basic flow is worded in a sequence of steps you would expect to encounter when using or testing the system.

The alternate flows (there may be a single or multiple alternate flows) are variations of the basic flow to deal with more specific cases. These variations can be enhancements, special cases, etc. In this case, we have three alternative paths A1–A3 (see Figure 16.3). Their steps should be listed, but for brevity, we did not show them in the referenced figure.

As another example, the use-case narrative for "Provide travel recommendations" is depicted in Figure 16.4. The basic flow enumerates the sequence of interactions between the software system and the user to provide travel recommendations to the user (i.e., the traveler).

**Basic Flow:**

1. Traveler verifies travel details (travel dates and destination).
2. Traveler requests recommendations.
3. System provides list of recommendations.
4. Traveler browses recommendations.
5. Traveler selects and views a recommendation.

**Alternate Flows:**

A1. Recommendations of different entities
- a. Hotel
- b. Place of Interest

A2. Recommendations
- a. Recommendations based on popularity rating
- b. Recommendations based on pricing
- c. Weighting function (preference indicator) for the above parameters

A3. Recommendation request trigger
- a. User initiated
- b. System triggered

A4. Sorting of recommendations
- a. Sorting based on prices

**Figure 16.4** Use-Case Narrative: "Provide Travel Recommendations."

When trying to understand requirements, or anything, we start by understanding the heart of the matter, before diving into details. Use cases are structured to help with this kind of thinking. This same idea works when you are implementing a system; you start building the skeleton before fleshing out the details. The basic flow acts as the skeleton. If you compare the use case approach to capturing requirements with that of user stories in Chapter 15, it should be clear that the use case approach provides more structure through the separation of basic and alternate flows, in addition to other features we will describe. This structure also makes the requirements easier to understand, especially for endeavors that are large and complex.

Keep in mind that what you find in this chapter is just a brief introduction to the use case approach. In Section 16.9, we have made a more complete comparison. All of the practices we present in this book are example practices. We want the reader to understand how they can be represented in an essentialized form and what value essentialization brings to comparing practices helping you make the best decision

given your own situation. If you are interested in learning more, we recommend you study this approach in detail [Jacobson et al. 2011].

### 16.1.1 Slicing Use Cases

Use cases can help solve one of the key problems with incremental development that many teams face. That is the fact that the functionality of the entire software system can become fragmented into product backlog items scattered across all the iterations throughout the evolution of the software system. As such, it is often challenging to understand what the system can do at any point in time, or answer questions about the impact of a new product backlog item on the current software system. Use cases provide an approach for putting all these product backlog items into context from the user's point of view. A use case often contains too much functionality to be developed in a single iteration, such as a single sprint when using Scrum. That is why a use case is split up into a number of smaller parts that are referred to as *use-case slices*. These use-case slices taken together represent the whole use case. Each use-case slice should provide some value to its users, so they should typically include a sequence of activities and not just represent a user interface part or a business rule.

For example, as explained, the "Make a travel plan" use case has a basic flow and three alternate flows. So, you might implement the basic flow first as one use-case slice, before implementing the alternate flows as subsequent use-case slices. Recall that in Chapter 15, we discussed an example for splitting user stories. With use cases, the way a use-case narrative is expressed (partitioned into basic and alternate flows) facilitates splitting up the use case into smaller use-case slices. These can then be placed into the backlog for development (see Jacobson et al. 2011, 2016).

*A use-case slice is a slice through a use case that is meaningful to describe, design, implement, and test in one go.* It doesn't need to by itself give value to a user, but together with all other slices of the same use case, the value is achieved. For example, the basic flow of a use case is a good candidate to become an early use-case slice. Additional slices can then be added to complete the whole use case later. The slicing mechanism is very flexible, enabling you to create slices as big or small as you need to drive your development. The use-case slices include more than just the requirements. They also slice through all the other aspects of the system—e.g., user experience (user interface), architecture, design, code, test—and its documentation. Thinking about and developing a software system through slices in this way helps you create the right system. The slices provide a way to link the requirements to the parts of the system that implement them, the tests used to verify that the requirements have been imple-

mented successfully, and the release and endeavor plans that direct the development work.

The Use Case Lite practice that we discuss below provides a scalable, agile practice that utilizes use cases to capture the functionality of a software system and test them to ensure the system fulfills them.

## 16.2 Making the Use Case Lite Practice Explicit Using Essence

We have just introduced the concepts and the purpose of use cases and use-case slices, which are essential parts of our Use Case Lite practice. We will now look at how to describe this practice using Essence. The first questions we always ask when essentializing a practice are as follows:

- What are the things you need to work with?
- What are the activities you do?

Figure 16.5 expresses the elements of our Use Case Lite practice using the Essence language.

The Use Case Lite practice decomposes Requirements into Use Cases, which in turn are broken down into Use-Case Slices. Requirements, Use Cases, and Use-Case Slices are all important things we need to work with and progress. They are alphas in our essentialized representation of the Use Case Lite practice.

The Requirements are described using a Use-Case Model. A Use-Case Model is a tangible description of the Requirements, and therefore it is a work product. Each Use Case within our Use-Case Lite practice has one related work product, a Use-Case Narrative, and each Use-Case Slice has one related work product, a Use-Case Slice Test Case.

The bottom of Figure 16.5 shows four activities in our Use Case Lite practice, namely:

1. Find Actors and Use Cases to gain an overall understanding of what the system is about;
2. Slice the Use Cases to break them up into a number of intelligently selected Use-Case Slices that each fit within a single sprint;
3. Prepare a Use-Case Slice by enhancing the Narrative and Test Cases to clearly define what it means to successfully implement the slice; and
4. Test a Use-Case Slice to verify it is done and ready for inclusion in a release.

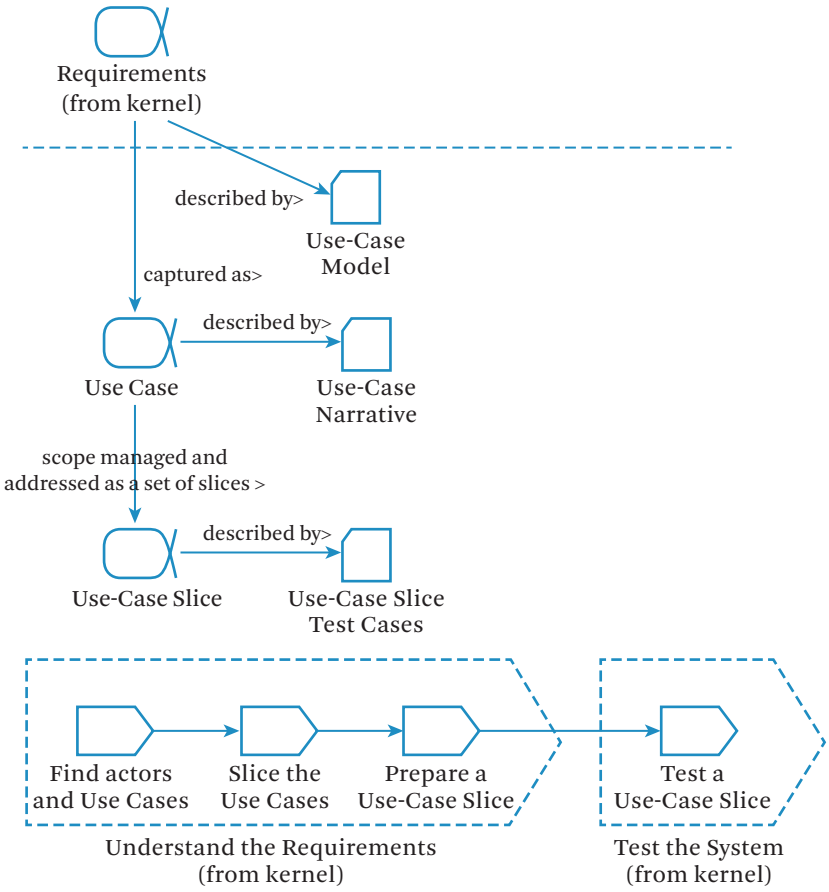



**Figure 16.5** Use Case Lite practice expressed in the Essence language.

The first three activities reside in the Understand the Requirements activity space and the fourth resides in the Test the System activity space.

Use-Case Slices are identified by working through the use case to identify paths, scenarios, or—as we say—the stories that build up the use case. Typically, a *story* is any path that you may want to follow going through the use case: its basic flow or its alternative flows. Concrete examples of stories in the use case "Make a travel plan" are

1. the basic flow;
2. the basic flow complemented with alternative flow 1; and
3. the basic flow complemented with alternative flow 2.

The story idea is similar to that in the User Story Lite practice and is a very important step to find good slices. A use-case slice typically includes one or more stories.

Note that this practice does not provide any patterns. This illustrates that it is up to the author of a practice to dictate its scope. Normally, the more there is in a practice, the more specific it is. The less complexity it has, the more generic it is, and others can supply more specific information, such as patterns, when necessary.

The elements of the Use Case Lite practice are summarized in Table 16.1.

**Table 16.1** Elements of the Use Case Lite practice

| Element | Type | Description |
|---|---|---|
| Use Case | Alpha | All the ways of using a system to achieve a particular goal for a particular user. |
| Use-Case Narrative | Work Product | The story of how the system and its actors work together to achieve a particular goal. |
| Use-Case Slice | Alpha | One or more stories selected from a use case to form a work item that is of clear value to the customer. |
| Use-Case Model | Work Product | A model that captures and visualizes all of the practical ways to use a system. |
| Use-Case Slice Test Case | Work Product | Defined inputs and expected results to help evaluate whether a system works correctly.<br>There can be one or more test cases to verify each use-case slice. |
| Find Actors and Use Cases | Activity | Agree on the goals and value of the system by identifying ways of using and testing it. |
| Slice the Use Cases | Activity | Break each use case up into a number of intelligently selected smaller parts for development. |
| Prepare a Use-Case Slice | Activity | Enhance the narrative and test cases to clearly define what it means to successfully implement the slice. |
| Test a Use-Case Slice | Activity | Verify the slice is done and ready for inclusion in a release. |

This practice is primarily in the solution area of concern because it focuses on the Requirements and the Software System kernel alphas. Like the User Story Lite practice, it provides no explicit guidance in the endeavor area of concern, and is therefore well complemented by the Scrum Lite practice.

## 16.3 Use Case Lite Alphas

The Use Case Lite practice involves the following alphas (shown in Figure 16.6): Requirements (from the kernel), Use Case, and Use-Case Slice. The cards show the states of each alpha.

### 16.3.1 Progressing Use Cases

The Use Case Lite practice provides an effective way to progress requirements. Again, *a use case is all the ways of using a system to achieve a particular goal for a particular user.* So, the use case first starts off with a goal, and along with it go all the ways—or what we call stories—for achieving that goal. Through the use-case narrative, the stories are organized with a structure that is understandable by both the customer representative and the development team. A team then fulfills the simplest story and proceeds to incrementally progress through other stories until all of them are fulfilled.

Thus, the Use Case alpha has the following states of progression and health (see Figure 16.7).

**Goal Established.** The scope of a use case is defined by the goal of the use case: what the actor wants to achieve.

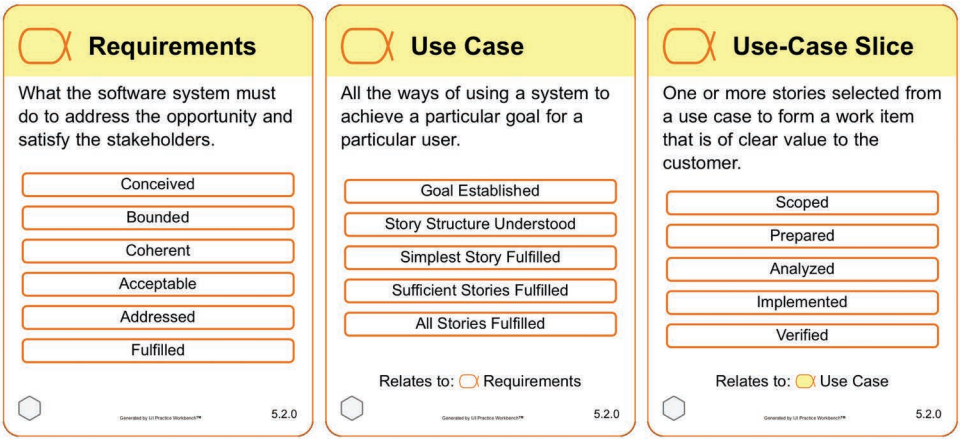



**Figure 16.6** Things to work with (alphas).

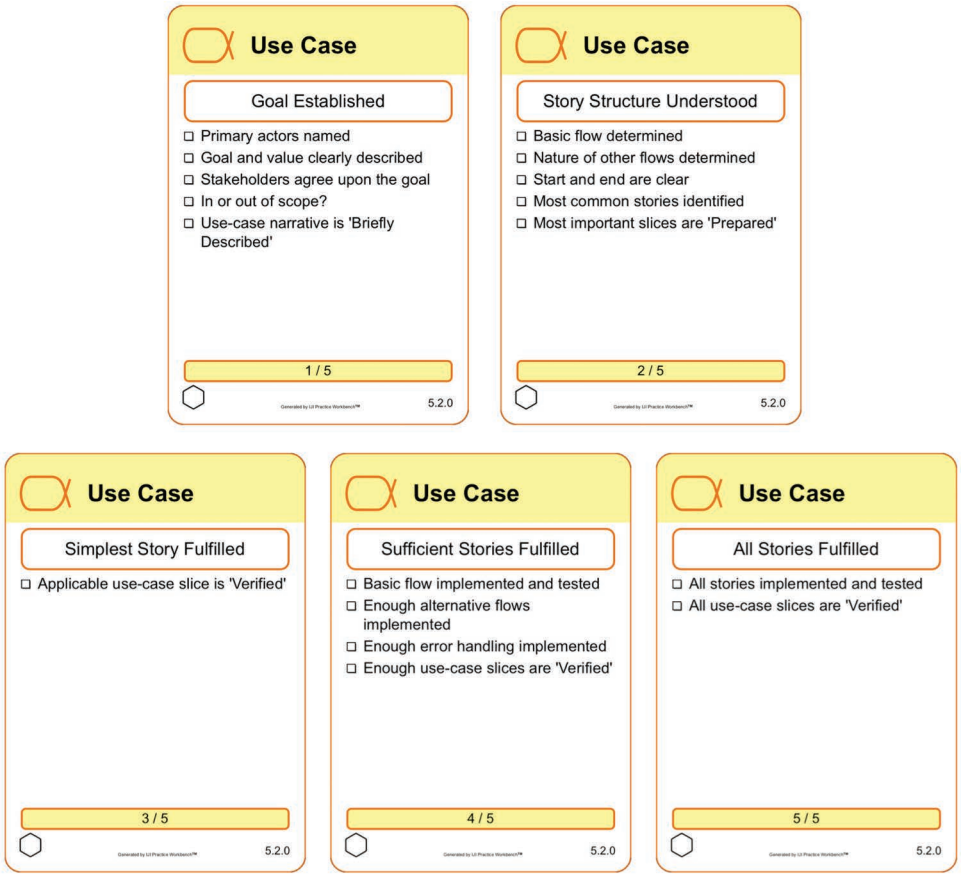



**Figure 16.7** Use Case alpha state cards.

**Story Structure Understood.** One of the key benefits of use cases is that it provides a structure. Rather than a heap of requirement items or user stories, use cases provide a structure. In this state, that structure is defined and understood.

**Simplest Story Fulfilled.** The simplest or the most basic flow through the use case drives the code skeleton. Once this code skeleton is formed and stabilized, it becomes easy to implement the rest of the stories.

**Sufficient Stories Fulfilled.** Once sufficient stories are fulfilled, the use case can be evaluated as to whether it achieves its goal well.

**All Stories Fulfilled.** Finally, the entire use case is completed.

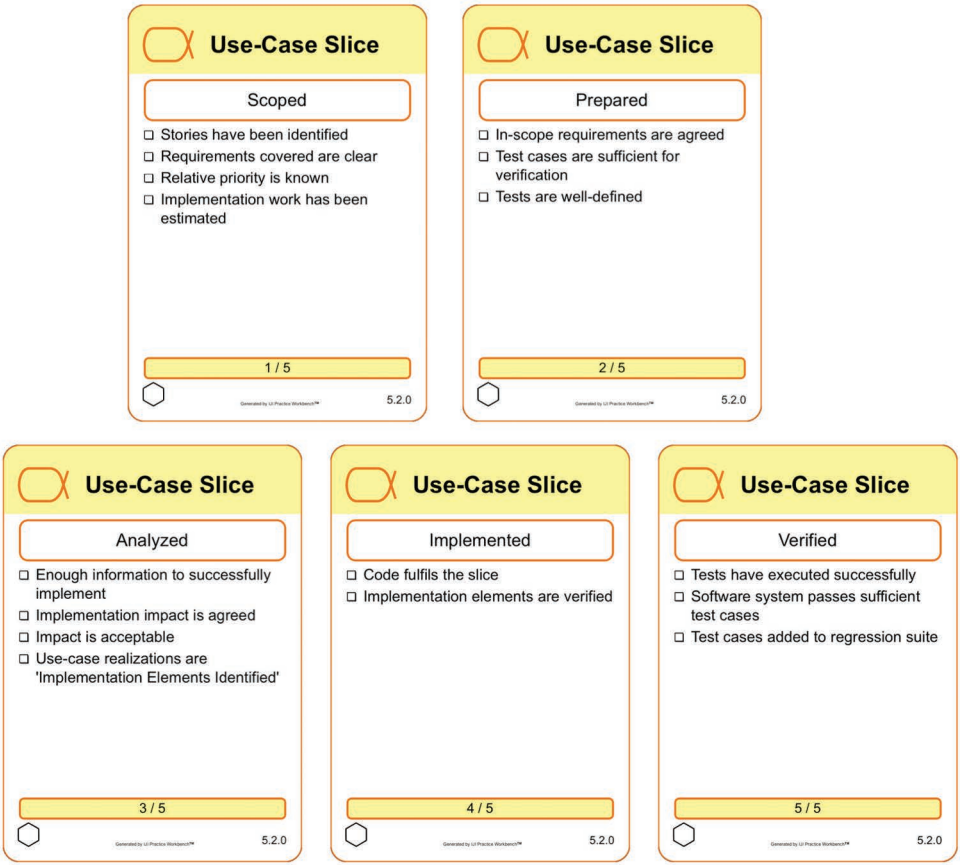



**Figure 16.8** Use-Case Slice alpha state cards.

### 16.3.2 Progressing Use-Case Slices

It will take several sprints, or even releases, to fulfill all the stories in a use case. In each sprint, teams fulfill a portion of a use case, which (as you know) we call a use-case slice.

Working with a use-case slice is quite similar to working with a user story. The use-case slice alpha has the following states (see Figure 16.8).

**Scoped.** When this state is complete, the use-case slice has been identified and its scope clarified.

**Prepared.** At this state, the information the development team needs to implement the use-case slice is available, including priorities relative to other

slices, estimates of cost to implement, dependencies on other use-case slices, etc.

**Analyzed.** At this state, the development team has a common agreement on how the use-case slice will be implemented. This includes agreeing on aspects such as user interfaces (i.e., how information should be presented on the screen, how the user would interact with the system), persistence (e.g., updating the database), and so on.

**Implemented.** At this state, the use-case slice is implemented. This involves writing actual code.

**Verified.** At this state, product owners verify the use-case slice does what is expected.

## 16.4 Use Case Lite Work Products

The work products in this practice are depicted in Figure 16.9. There are cards whose content provides guidance on the levels of detail for each of the work products (Use-Case Model, Use-Case Narrative, Use-Case Slice Test Case). The following quote is taken from the Use Case 2.0 ebook [Jacobson et al. 2011]:

> *All of the work products are defined with a number of levels of detail. The first level of detail defines the bare essentials, the minimal amount of information that is required for the practice to work. Further levels of detail help the team cope with any special circumstances they might encounter. For example, this allows small, collaborative teams to have very lightweight use-case narratives defined on simple index cards and large distributed teams to have more detailed use-case narratives presented as documents. The teams can then grow the narratives as needed to help with communication, or thoroughly define the important or safety critical requirements. It is up to the team to decide whether or not they need to go beyond the essentials, adding detail in a natural fashion as they encounter problems that the bare essentials cannot cope with.*

The lightest level of detail is shown at the top of each work product card. The amount of detail increases as you go down the card. For example, in Figure 16.9, Value Established is the lightest level of detail for the Use-Case Model, while Structured provides the greatest detail. For the Use-Case Narrative, Briefly Described is the lightest level of detail, whereas Fully Described is the deepest level. A team always starts with the lightest level, and can then decide how much more detail they need, based on their own situation. These levels of details will be described in the next section, when we elaborate each work product.

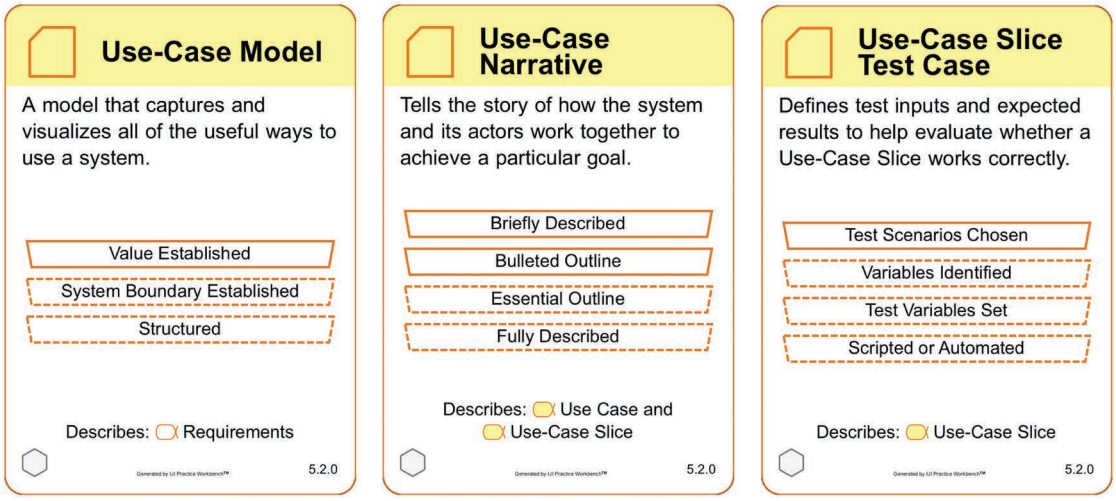



**Figure 16.9** Things to work with (work products).

It is to be expected that each use case will evolve over several iterations/sprints by coming back to the activity slice the use cases. More slices may be added to the use case, and each use case is improved as we learn more. As such, the alpha states that show progress of a use case could be based on the evolution of the slices. This is a similar idea to what we saw earlier in Part II, where the progress of the Requirement alpha was based on the progress of Requirements Items. This will be more apparent later when we provide a concrete example.

### 16.4.1 Use-Case Model

A use-case model captures and visually presents all the useful ways to work with a system (see Figure 16.10). Figure 16.2 gave an example of a simple use-case model.

A use-case model describes not just one but several use cases and how together they provide value to its users (i.e., actors). These use cases need to have clearly defined scope. The use-case model has the following levels of detail.

**Value Established.** At this level of detail, the value of the use cases and hence the use-case model is established. Readers of the use-case model have a good understanding of what the use cases are about, what they do, and how actors benefit from them.

**System Boundary Established.** At this level, the scope and boundaries of the requirements are described. The team and stakeholders have a clear understanding of what is within its scope and what is not.

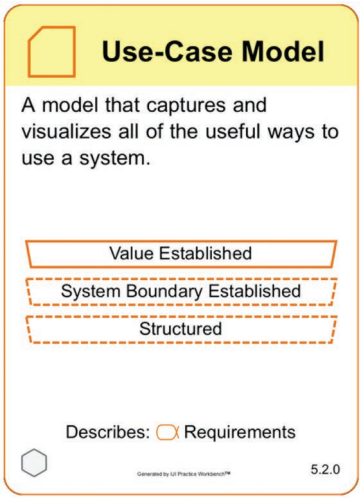



**Figure 16.10** Use-Case Model work product.

**Structured.** At this level of detail, the Use-Case Model is well structured. There is little or no overlap between Use Cases. The dependencies and relationships between Use Cases are described clearly.

### 16.4.2 Use-Case Narrative

A Use-Case Narrative describes the story (i.e., sequence of steps) of how the system and the actors work together to achieve a particular goal. (The term "story" here is not the "user story" we presented in the previous chapter. Rather, the term "story" is just the normal English word.)

A use-case narrative can be described at several levels of detail (see also Figure 16.11).

**Briefly Described.** At this level of detail, the use-case narrative only has a brief description of the use-case goal and what it is about.

**Bulleted Outline.** At this level of detail, the story of how the system and actors will work together is available. The examples provided in Figures 16.3 and 16.4 are at this level of detail. Of course, the lists can also be numbered.

**Essential Outline.** At this level of detail, the story is full blown. In the context of requirements of the software system, the various alternative usages and exceptions to be handled are clearly described.

**Fully Described.** This is a very detailed description of the use case. All conversations are clearly spelled out.

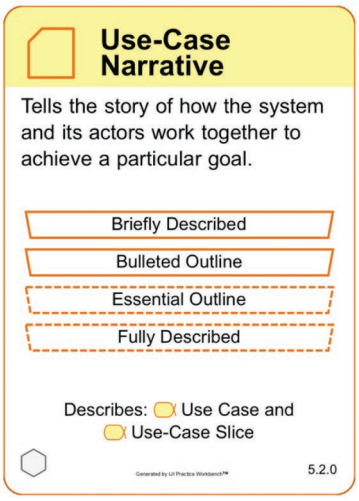



**Figure 16.11** Use-Case Narrative work product.

### 16.4.3 Use-Case Slice Test Case

The use-case slice test case work product defines the inputs and expected outputs to help evaluate whether a use-case slice is implemented correctly.

Figure 16.12 depicts the level of details in a use-case slice test case.

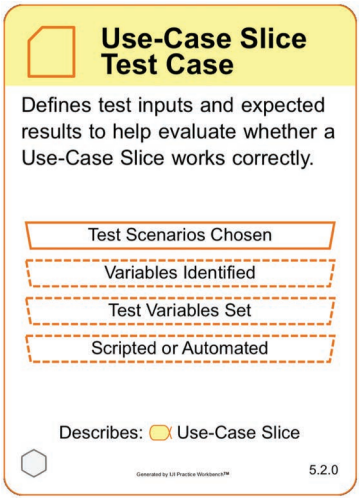



**Figure 16.12** Use-Case Slice Test Case work product.

**Scenario Chosen.** At this level of detail, the different scenarios required to test the use-case slice are described. This includes the normal way of using the use-case slice and other variations (alternative usages and exception cases). The example in Section 16.6.2, Table 16.3, is at this level of detail.

**Variables Identified.** At this level, the different variables are listed. For example, the variables for testing the "Make a travel plan" use case include traveler identification and destinations (see Figure 16.3).

**Variables Set.** At this level, the actual variables are defined. To continue the same example, the traveler might be Sam, whose identification is 12345678. The destination is Singapore. The popularity rating of the Singapore Zoological Gardens and Shangri-La Hotel are set.

**Scripted or Automated.** At this level of detail, the test cases are clearly described such that a person can run the test case by following a step-by-step procedure without misinterpretation, or a software tool can execute it repeatedly with pass/fail results clearly defined.

## 16.5 Kick-Starting Use Cases Lite to Solve a Problem Our Team Is Facing

One day at TravelEssence, Tom raised a question: "Our endeavor is getting more complex and it is difficult to see the big picture when looking at our product backlog. I have heard that the Use Case Lite practive could help us with this problem. But if we migrate to use cases, do we need to rewrite all our old product backlog items into use cases?"

Smith replied, "Kick-starting Use Case Lite for an endeavor starts with identifying users and use cases to produce a skeletal use-case model, and skeletal use-case narratives. By skeletal, I just mean it is not necessary to have detailed descriptions of all use cases of the system. So, no, we don't need to rewrite all our old product backlog items. We just need to make a use-case model and map the backlog items that are already done to use-case slices of the use cases in that model. The new backlog items will of course be the use-case slices that are not yet implemented."

The activities to apply Use Case Lite are Find Actors and Use Cases, Slice the Use Cases, Prepare a Use-Case Slice, and Test a Use-Case Slice, which we will discuss in subsequent subsections.

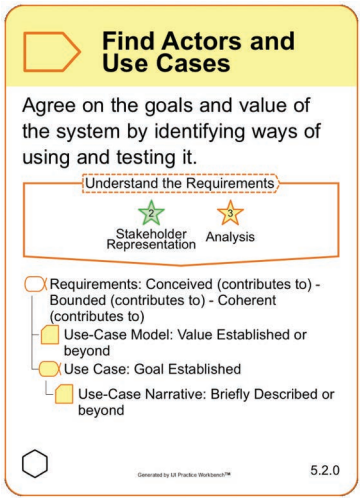



**Figure 16.13** Find Actors and Use Cases work product.

### 16.5.1 Find Actors and Use Cases

The Find Actors and Use Cases activity is about agreeing on the goals and value of the software system by identifying the different ways of using it. As a corollary to this, we also find the different ways of testing it (see Figure 16.13).

On the Find Actors and Use Cases card, we see that the Stakeholder Representation and Analysis competencies are needed. We also see that this activity contributes to achieving the Requirements alpha's Conceived, Bounded, and Coherent states.

The card also indicates that the use-case model needs at a minimum to achieve the Value Established level of detail and that the use-case narrative at a minimum must be Briefly Described. The use case alpha needs to achieve the Goal Established state.

At TravelEssence, Smith next worked with his team to create a skeletal use-case model for the system as developed so far (see Figure 16.2). This can be said to represent release 1 of the TravelEssence system.

Smith then went on to create a new use-case model for the new release of the system—release 2—that they were now working on (see Figure 16.14). He added a new actor, Digital Officer, responsible for overseeing all new product releases. He also added two new use cases: "Manage release" and "Analyze recommendation results." The "Manage release" use case provided features to review upcoming planned releases, and the "Analyze recommendation results" use case contributed

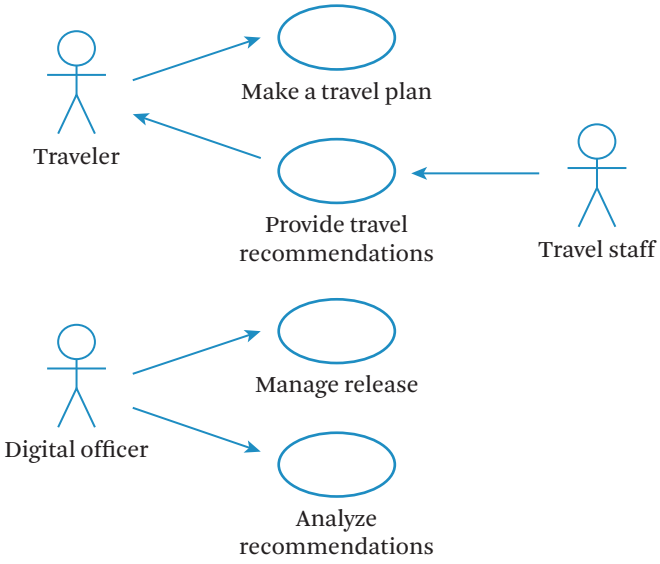



**Figure 16.14** Use-Case Model for the next release of TravelEssence.

features to examine recommendations generated by the system and provide feedback.

Moreover, after a successful rollout to internal users, Angela had collected a number of feedback comments, which Smith also wanted to act upon for the next release of TravelEssence. Some of the feedback comments related to usability enhancements, while others related to new functionality requests, such as the following:

**Recommendations by Advertisements.** "We have revenue from advertisers. If these advertisers are within the vicinity of the traveler's destination, they should be in the recommendations. However, we need to come up with a fair approach for prioritizing recommendations."

**Sorting by Vicinity.** "The list of recommendations is rather long; it should be sorted according to how close the advertisers are to the traveler."

**Handling Favorites.** "Sometimes, the traveler might want to remember the recommendation for future trips through some kind of 'favorites.' Favorites should appear in future recommendations."

From the above list of suggested improvements, Smith made the following updates to the "Provide travel recommendation" use-case narrative shown in Figure 16.15. Updated and new lines are labeled in the figure.

**Basic Flow:**

1. Traveler provides travel details (travel dates and destination).
2. Traveler requests recommendations.
3. System provides list of recommendations.
4. Traveler browses recommendations.
5. Traveler selects and views a recommendation.

**Alternate Flows:**

A.1 Recommendations of different entities

- a. Hotel
- b. Places of Interest

A.2 Recommendation computation

- a. Recommendations based on popularity rating
- b. Recommendations based on pricing
- c. **#New** Recommendations based on advertisements
- d. **#New** Recommendations based on favorites
- e. **#Updated** Weighting function for the above parameters (popularity, pricing, etc.)

A.3 Recommendation request trigger

- a. User initiated
- b. System triggered

A.4 Sorting of recommendations

- a. Sorting based on prices
- b. **#New** Sorting based on vicinity

A.5 **#New** Recommendation actions

- a. **#New** Add selected recommendations to favorites.

**Figure 16.15** Use-Case Narrative: Provide Travel Recommendations (Updated in the next release).

## 16.6 Working with Use Cases and Use-Case Slices

Working with Use Cases Lite is about iteratively updating the use cases and use-case slices. It provides a high-level guide to everyone on the team, not only to developers and testers, but also to product owners like Angela. As a software system evolves, the use-case model, with the use-case narratives, continuously provides an easy-to-understand overview of what the system does.

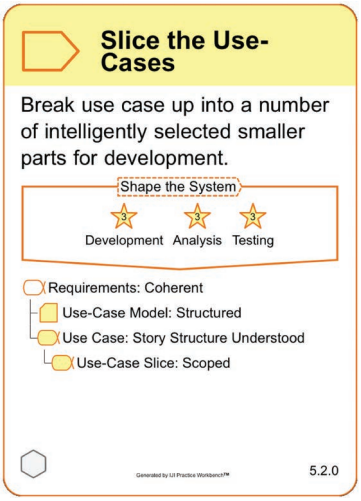



**Figure 16.16** Slice the Use Cases activity card.

### 16.6.1 Slice the Use Cases

In the Slice the Use Cases activity, each use case gets broken up into smaller parts to facilitate development (see Figure 16.16).

Identifying the use-case slices for simple situations is extremely easy, because we don't need to think about all possible conditions. Of course, we eventually have to include alternative paths to cover each possible situation, but by handling each separately we keep things simple. The use-case narrative itself is structured in a form amenable for this purpose.

The use-case slices for the subsequent iterations of the TravelEssence team's endeavor are shown in Table 16.2. The three use-case slices in the table correspond to the three requests listed earlier. The key difference is that the use-case slices now have explicit reference to modular and testable changes to the use-case narrative.

After Smith and Angela had discussed and gained agreement on the modifications to the use-case narratives, they started conversing with the team about the changes for the next iteration.

After reviewing the narratives Smith had developed, Tom exclaimed, "Wow! Now I see how everything fits. This organization of the narratives gives a very good structure to help me understand how new requirement items will impact our system, the requirements, and the tests. After we implement our chosen slices for each sprint, we will need to verify that each one is done and is ready for inclusion in our next release."

**Table 16.2** New Use-Case Slices for the Use Case "Provide Travel Recommendations"

| Use-Case Slice Name | Use-Case Slice Description |
|---|---|
| Recommendation by advertisements | **#New** Recommendations based on advertisements<br>**#Updated** Weighting function for the above parameters |
| Sorting by vicinity | **#New** Sorting based on vicinity |
| Handle favorites | **#New** Add selected recommendations to favorites<br>**#New** Recommendations based on favorites<br>**#Updated** Weighting function for the above parameters |

As you can see from Figure 16.15, the use-case narrative is organized such that each of the alternate flows provides a way to group related requirements. For example, recommendations on places of interest to visit or hotels to stay at (alternate flow 1) are grouped together, and recommendations based on attributes such as pricing and popularity are grouped together (alternate flow 2). This kind of organization can help developers structure their code and test cases in a way that eases the long-term maintainability of the system.

### 16.6.2 Prepare a Use-Case Slice

The Prepare a Use-Case Slice activity enhances the use-case narrative and the use-case slice test cases to clearly define what it means to successfully implement the use-case slice.

For brevity, we will only focus on one use-case slice: "Handle favorites." Favorites are just a list, which the application stores for the user. If a user determines that a recommendation is useful, she might want to store this recommendation in her/his favorites list. This favorite list also acts as an input to the recommendation engine. So, the recommendation will behave differently for a new user without any favorites than for an old user that has some favorites.

It is clear from Table 16.2 that there are three distinct and separate slices:

1. **#New** Add selected recommendations to favorites
2. **#New** Recommendations based on favorites
3. **#Updated** Weighting function for the above parameters

Another important work product output of this activity is the use-case slice test cases. Smith and his team brainstormed the use-case slice test cases for each of these slices. The resulting outlines are summarized in Table 16.3.

The level of detail for the use-case slice test cases was Scenario Chosen.

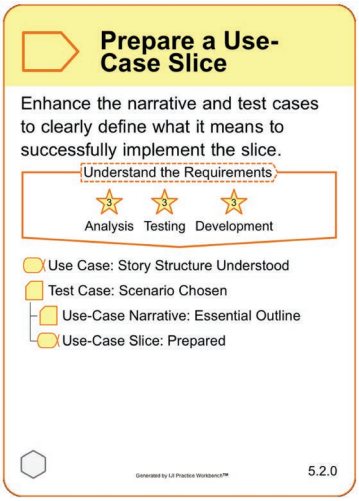



**Figure 16.17** Prepare a Use-Case Slice activity card.

**Table 16.3** Test Cases for "Handle Favorites"

| Use-Case Slice | Use-Case Test Cases |
|---|---|
| **#New** Add selected recommendations to favorites | 1. New favorite<br>2. Favorite already exists<br>3. Maximum number of favorites |
| **#New** Recommendations based on favorites | 1. No favorites<br>2. One favorite within vicinity of traveler destination<br>3. One favorite outside vicinity of traveler destination |
| **#Updated** Weighting function for the above parameters | 1. Weightage of favorites set to 0<br>2. Weightage of favorites set to 0.5 |

### 16.6.3 Test a Use-Case Slice

The goal of the Test a Use-Case Slice activity is to verify that the slice is done and ready for inclusion in a release (see Figure 16.18).

The use-case slice test cases chosen in Section 16.6.2 are of course an important input to this activity. During testing, these test cases are refined further with additional details to make sure that they are repeatable.


**Figure 16.18** Test a Use-Case Slice activity card.

## 16.7 Visualizing the Impact of Using Use Cases for the Team

In our story, as the complexity of the endeavor grew, team members recognized that their approach to capturing requirements as a collection of user stories was insufficient. By migrating to use cases, the team found they could see the big picture of the system better through their use-case model and use-case narratives.

This is made visible by looking at Use Case Lite's coverage of the solution activity spaces in Figure 16.19. Compare this with Figure 15.13 in Section 15.7.1, and you will see that the activity space Shape the System is covered by the Find Actors and Use Cases activity. This activity has the dual purpose of understanding requirements and shaping the system. Both the use-case model and use-case narratives (basic and alternate flows) are tools that help teams organize/structure/shape requirements. They help teams see the big picture.

As new requirements came to the team, they could easily see where these would fit in the overall structure of use cases, use-case slices, and associated work products such as use-case narratives and use-case slice test cases. The team members agreed that the overall maintainability of the system had been much improved by their move to use cases. This was for multiple reasons: first, because of the structure provided by the organization of the narrative separating basic and multiple alternative flows; second, because of the big picture provided by the use-case model.

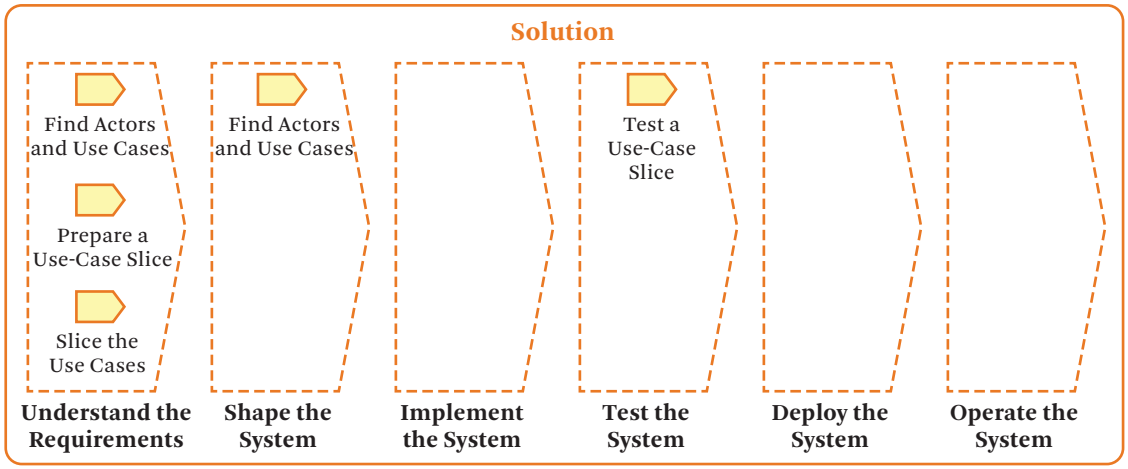



**Figure 16.19** Use Case Lite practice coverage of kernel solution activity spaces.

# 16.8 Progress and Health of Use-Case Slices

You have just learned about how to apply the Use-Case Lite practice for several use-case slices from a single use case, "Provide travel recommendations." In general, teams work with multiple use-case slices from multiple use cases at any given point in time. They complete use-case slices within each sprint (i.e., drive them to the Use-Case Slice: Verified state). While individual use-case slices are completed in each sprint, often it requires multiple sprints to complete a full use case.

Since on most software endeavors there are a number of use cases and use-case slices progressing in parallel, it is important that the team has agreed to a way to monitor their progress and health. One approach that is popular at the time of writing this book is for team members to have the competency to self-monitor their progress and health. The alpha state cards for Use Case and Use-Case Slices (see Figures 16.7 and 16.8) provide a tool for this purpose.

Now, recalling that our TravelEssence team has chosen multiple practices, this means that they have a number of alphas to juggle.

- From the Scrum Lite practice:
  - Sprint—focusing on the sprint goals.
  - Product Backlog Item—a change to be made to the product in a future release.

- From the Use Case Lite practice:
  - Use-Case Slices—certain ones need to be Verified by the end of the sprint.
  - Use Cases—they need not be completed for each sprint, but they are useful for determining which Use-Case Slices should be implemented first. As a group, they will take several sprints to be completed (i.e., All Stories Fulfilled). Thus, different use cases will be at different states at the end of each sprint.
- From the Essence kernel:
  - Work—the team needs to maintain the Under Control state as development progresses.
  - Requirements—this alpha progresses towards Addressed or Fulfilled, depending on the goals of the sprint.

Smith's team found that agreeing on these target states helped them achieve focus at the beginning of each sprint. They were also useful for reviewing progress during their Daily Scrum (see Chapter 14).

During one of their sprint retrospectives, Tom stated, "I find the checklists for the Use Case alpha states and Use-Case Slice alpha states more useful to me than the Requirements alpha states."

Smith replied, "Yes, that is reasonable, as you are focusing on progressing specific use-case slices for specific use cases. However, from my perspective, I still need to view the big picture of development progress for the whole endeavor. Thus, I would need to understand the progress and health of it all. That would include not just the Requirements or Work alphas, but also the Opportunity, Stakeholders, Team, Way of Working, and Software System alphas."

## 16.9 User Stories and Use Cases—What Is the Difference?

In the previous chapter, Smith's team applied user stories and in this chapter they applied use cases. This change resulted in much discussion and debate among the team members. In particular, Tom had been reluctant to make the transition. However, Tom agreed that the point should not be about his own preference, but about what was best for the team. The team decided to do their homework, research both approaches (i.e., user stories and use cases), and determine which was most appropriate for them.

The best way to answer the question "What is the difference between User Stories and Use Cases?" is to start by looking at their common properties, the things

that make them both work well as backlog items and enable them both to support popular agile approaches such as Scrum.

Use-case slices and user stories [Cohn 2004] share many common characteristics, such as the following:

- They both define slices of the functionality that teams can accomplish in a Sprint.
- They can both be sliced up if they are too large, resulting in numerous smaller items.
- They can both be written on index cards.
- They both result in test cases that represent the acceptance criteria.
- They are both placeholders for a conversation.
- They can both be estimated in similar ways.

So, given that they share so many things in common, what is it that makes them different?

Use cases and use-case slices provide added value. This is because they:

- provide a big picture to help people understand the extent of the system and the value it provides;
- offer support for simple systems, complex systems, and systems-of-systems; and
- result in easier identification of missing and redundant functionality, thanks to the big picture.

The sweet spot for user stories is when you have easy access to an expert on the subject of the requirements and when the severity of errors is low. use cases and use-case slices are more suitable when there is no easy access to an expert or when error consequences are high. However, because the use case approach can scale down to the same scope as for user stories, you may still want to apply them. If you are confident that the subject system will always be in the best range for user stories, they may be a good choice. If you expect the system to grow outside that area, though, you might consider use cases and use-case slices.

Even though Smith's team had Angela close by, they found that when it came to explaining details of the requirements, she had difficulty expressing herself. That was when the team found that using use-case narratives in the lightweight manner described above became extremely useful. They found that when the requirements

became explicit, communications were simplified. Each person clearly understood what the others meant.

As we mentioned in this chapter's introduction, the term "use case" has been used as a normal English expression since around 2010. Consequently, at the introductory and executive level, many things are presented in terms of their use cases (for instance, within both Dropbox and Industrial Internet). Having this term in common encourages seamless traceability from introductory presentations to their realizations, and simplifies communication between people working with early visioning and people working with development.

Despite his early reluctance to apply use cases, Tom had one meeting with Dave that changed his stance. Dave was talking about the new digital "use cases" for the hotel system where the recommendation engine was part of the digital transformation. Dave used the term "use cases" like a normal English term. Moreover, when Tom was demonstrating the "Provide travel recommendations" use case, he found that he could explain his intent very well by simply walking through its narrative as he stepped through the demo. It was then that Tom was finally convinced about this term and the ideas behind it, from stakeholder conversations early on to realizations by the team (both developers and testers). From then on, the team as a whole had no more qualms living with use cases.

## What Should You Now Be Able to Accomplish?

After studying this chapter, you should be able to

- explain the purpose of the Use Case Lite practice and the problem it solves;
- explain the difference between the basic and alternate flow of a use Case;
- explain use-case slices and their benefits compared to pure use cases;
- list and explain the alphas, work products, and activities of Use Case Lite;
- explain how TravelEssence adopted and applied Use Case Lite and the benefits they achieved, together with the benefits implied by using the Use Case Lite practice in an essentialized form; and
- compare use cases to user stories and describe scenarios in which each of them is more beneficial than the other.

# References


Alpha State Card Games. 2018. https://www.ivarjacobson.com/publications/brochure/alpha-state-card-games. 99, 153

S. Ambler and M. Lines. 2012. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press. 297, 339, 346

K. Beck. 1999. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Longman. 203, 285, 346

K. Beck. 2003. *Test-Driven Development by Example*. Addison Wesley. 346

K. Bittner and I. Spence. 2003. *Use Case Modeling.* Addison-Wesley Professional, 2003. 222, 285

G. Booch, J. Rumbaugh, and I. Jacobson. 2005. *The Unified Modeling Language User Guide.* 2nd edition. Addison-Wesley. 222, 285, 345

F. Brooks. 1975. *The Mythical Man-Month*. Addison Wesley. 342

M. Cohn. 2004. *User Stories Applied: For Agile Software Development.* Addison-Wesley Professional. 204, 247, 285

E. Derby and D. Larsen. 2006. *Agile Retrospectives: Making Good Teams Great.* Pragmatic Bookshelf, Dallas, TX, and Raleigh, NC. 196, 198, 284

E. W. Dijkstra. 1972. "The Humble Programmer." Turing Award Lecture, *CACM* 15 (10): 859–866. DOI: 10.1145/355604.361591. 342

D. Graziotin and P. Abrahamsson. 2013. A web-based modeling tool for the SEMAT Essence theory of software engineering. *Journal of Open Research Software*, 1,1(e4); DOI: 10.5334/jors.ad. 147, 153

ISO/IEC/IEEE 2382. 2015. Information technology–Vocabulary. International Organization/International Electrotechnical Commission, Geneva, Switzerland. https://www.iso.org/obp/ui/#iso:std:iso-iec:2382:ed-1:v1:en. 343

ISO/IEC/IEEE 12207. 2017. https://en.wikipedia.org/wiki/ISO/IEC_12207 345

ISO/IEC/IEEE 15288. 2002, 2008, 2015. Systems and software engineering—System life cycle processes. International Standardization Organization/International Electrotechnical Commission, 1 Rue de Varembe, CH-1211 Geneve 20, Switzerland. 345

ISO/IEC/IEEE 24765. 2017. Systems and software engineering–Vocabulary. International Organization/International Electrotechnical Commission, Geneva, Switzerland. https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:24765:ed-2:v1:en. 343

M. Jackson. 1975. *Principles of Program Design*. Academic Press. 344

I. Jacobson. 1987. Object-oriented software development in an industrial environment. *Conference Proceedings of Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 87)*. DOI: 10.1145/38807.38824. 221, 285

I. Jacobson and H. Lawson, editors. 2015. *Software Engineering in the Systems Context*, Systems Series, Volume 7. College Publications, London. 345

I. Jacobson and E. Seidewitz. 2014. A new software engineering. *Communications of the ACM*, 12(10). DOI: 10.1145/2685690.2693160. 347

I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach*. ACM Press Addison-Wesley. 345

I. Jacobson, I. Spence, and K. Bittner. 2011. Use-Case 2.0: The Guide to Succeeding with Use Cases. https://www.ivarjacobson.com/publications/whitepapers/use-case-ebook. 169, 222, 226, 233, 285

I. Jacobson, P.-W. Ng, P. E. McMahon, I. Spence, and S. Lidman. December 2012. The essence of software engineering: The SEMAT kernel. *Communications of the ACM*, 55(12). http://queue.acm.org/detail.cfm?id=2389616. DOI: 10.1145/2380656.2380670. 30, 95

I. Jacobson, P.-W. Ng, P. E. McMahon, I. Spence, and S. Lidman. 2013a. *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley. xxvi, 30, 90, 95, 336

I. Jacobson, I. Spence, and P.-W. Ng. (October) 2013b. Agile and SEMAT: Perfect partners. *Communications of the ACM*, 11(9). http://queue.acm.org/detail.cfm?id=2541674. DOI: 10.1145/2524713.2524723. 30, 96

I. Jacobson, I. Spence, and B. Kerr. 2016. Use-Case 2.0: The hub of software development. *Communications of the ACM*, 59(5): 61–69. DOI: 10.1145/2890778. 169, 222, 226, 285

I. Jacobson, I. Spence, and P.-W. Ng. 2017. Is there a single method for the Internet of Things? *Queue*, 15.3: 20. DOI: 10.1145/3106637. 335, 339

P. Johnson and M. Ekstedt. 2016. The Tarpit—A general theory of software engineering. *Information and Software Technology* 70: 181–203. https://www.researchgate.net/profile/Pontus_Johnson/publication/278743539_The_Tarpit_-_A_General_Theory_of_Software_Engineering/links/55b4490008aed621de0114f5/The-Tarpit-A-General-Theory-of-Software-Engineering.pdf. DOI: 10.1016/j.infsof.2015.06.001. 91, 92, 96

P. Johnson, M. Ekstedt, and I. Jacobson. September 2012. Where's the theory for software engineering? *IEEE Software*, 29(5). DOI: 10.1109/MS.2012.127. 84, 87, 96

R. Knaster and D. Leffingwell. 2017. *SAFe 4.0 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering.* Addison-Wesley Professional. 297, 339

P. Kruchten. 2003. *The Rational Unified Process: An Introduction*. 3rd edition. Addison-Wesley. 345

C. Larman and B. Vodde. 2008. *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum.* Pearson Education, Inc. 346

C. Larman and B. Vodde. 2016. *Large-Scale Scrum: More with LeSS.* Addison-Wesley Professional. 297, 339

D. Leffingwell. 2007. *Scaling Software Agility: Best Practices for Large Enterprises.* Addison-Wesley. 346

P. E. McMahon. January/February 2015. A thinking framework to power software development team performance. *Crosstalk, The Journal of Defense Software Engineering*. http://www.crosstalkonline.org/. 96, 154

NATO. 1968. "Software Engineering: Report on a conference sponsored by the NATO Science Committee." P. Naur and B. Randell, editors. Garmisch, Germany, October 7–11. 342

S. Newman. 2015. *Building Microservices.* O'Reilly Media, Inc. 250, 285

P.-W. Ng. 2013. Making software engineering education structured, relevant and engaging through gaming and simulation. *Journal of Communication and Computer* 10: 1365–1373. 99, 153

P.-W. Ng. 2014. Theory based software engineering with the SEMAT kernel: Preliminary investigation and experiences. *Proceedings of the 3rd SEMAT Workshop on General Theories of Software Engineering*. ACM. DOI: 10.1145/2593752.2593756. 30, 96

P.-W. Ng. 2015. Integrating software engineering theory and practice using Essence: A case study. *Science of Computer Programming*, 101: 66–78. DOI: 10.1016/j.scico.2014.11.009. 96, 152, 154

Object Management Group. Essence—Kernel and Language for Software Engineering Methods (Essence). http://www.omg.org/spec/Essence/1.1. 63

OMG Essence Specification. 2014. http://www.omg.org/spec/Essence/Current. 95, 346

D. Ross. 1977. Structured Analysis (SA): A language for communicating ideas. In *IEEE Transactions on Software Engineering*, SE-3(1): 16–34. DOI: 10.1109/TSE.1977.229900. 344

K. Schwaber and J. Sutherland. 2016. "The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game." Scrum.org.

# Index

# Author Biographies

## Ivar Jacobson

**Dr. Ivar Jacobson** received his Ph.D. in computer science from KTH Royal Institute of Technology, was awarded the Gustaf Dalén medal from Chalmers in 2003, and was made an honorary doctor at San Martin de Porres University, Peru, in 2009. Ivar has both an academic and an industry career. He has authored ten books, published more than a hundred papers, and is a frequent keynote speaker at conferences around the world.

Ivar Jacobson is a key founder of components and component architecture, work that was adopted by Ericsson and resulted in the greatest commercial success story ever in the history of Sweden (and it still is). He is the creator of use cases and Objectory—which, after the acquisition of Rational Software around 2000, resulted in the Rational Unified Process, a popular method. He is also one of the three original developers of the Unified Modeling Language. But all this is history. His most recently founded company, Ivar Jacobson International, has been focused since 2004 on using methods and tools in a smart, superlight, and agile way. Ivar is also a founder and leader of a worldwide network, SEMAT, whose mission is to revolutionize software development based on a kernel of software engineering. This kernel has been realized as a formal standard called Essence, which is the key idea described in this book.

## Harold "Bud" Lawson

**Professor Emeritus Dr. Harold "Bud" Lawson** (The Institute of Technology at Linköping University) has been active in the computing and systems arena since 1958 and has broad international experience in private and public organizations as well as academic environments. Bud contributed to several pioneering efforts in hardware and software technologies. He has held professorial appointments at several universities in the USA, Europe, and the Far East. A Fellow of the ACM, IEEE, and INCOSE, he was also head of the Swedish delegation to ISO/IEC JTC1 SC7 WG7 from 1996 to 2004 and the elected architect of the ISO/IEC 15288 standard. In 2000, he received the prestigious IEEE Computer Pioneer Charles Babbage medal award for his 1964 invention of the pointer variable concept for programming languages. He has also been a leader in systems engineering. In 2016, he was recognized as a Systems Engineering Pioneer by INCOSE. He has published several books and was the coordinating editor of the "Systems Series" published by College Publications, UK.

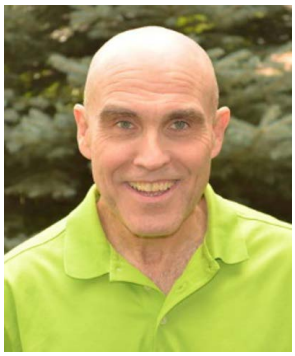
Tragically, Harold Lawson passed away after battling an illness for almost a year, just weeks before the publication of this book.

## Pan-Wei Ng

**Dr. Pan-Wei Ng** has been helping software teams and organizations such as Samsung, Sony, and Huawei since 2000, coaching them in the areas of software development, architecture, agile, lean, DevOps, innovation, digital, Beyond Budgetings, and Agile People. Pan-Wei firmly believes that there is no one-size-fits-all, and helps organizations find a way of working that suits them best. This is why he is so excited about Essence and has been working with it through SEMAT since their inception in 2006, back when Essence was a mere idea. He has contributed several key concepts to the development of Essence.

Pan-Wei coauthored two books with Dr. Ivar Jacobson and frequently shares his views in conferences. He currently works for DBS Singapore, and is also an adjunct lecturer in the National University of Singapore.

## Paul E. McMahon



**Paul E. McMahon** has been active in the software engineering field since 1973 after receiving his master's degree in mathematics from the State University of New York at Binghamton (now Binghamton University). Paul began his career as a software developer, spending the first twenty-five years working in the US Department of Defense modeling and simulation domain. Since 1997, as an independent consultant/coach (http://pemsystems.com), Paul helps organizations and teams using a hands-on practical approach focusing on agility and performance.

Paul has taught software engineering at Binghamton University, conducted workshops on software engineering and management, and has published more than 50 articles and 5 books. Paul is a frequent speaker at industry conferences. He is also a Senior Consulting Partner at Software Quality Center. Paul has been a leader in the SEMAT initiative since its initial meeting in Zurich.

## Michael Goedicke

**Prof. Dr. Michael Goedicke** is head of the working group Specification of Software Systems at the University of Duisburg-Essen. He is vice president of the GI (German National Association for Computer Science), chair of the Technical Assembly of the IFIP (International Federation for Information Processing), and longtime member and steering committee chair of the IEEE/ACM conference series Automated Software Engineering. His research interests include, among others, software engineering methods, technical specification and realization of software systems, and software architecture and modeling. He is also known for his work in views and viewpoints in software engineering and has quite a track record in software architecture. He has been involved in SEMAT activities nearly from the start, and assisted in the standardization process of Essence—especially the language track.