# PART II

# WRITING AND REVIEWING USE-CASE DESCRIPTIONS

Part I, Getting Started with Use-Case Modeling, introduced the basic concepts of use-case modeling, including defining the basic concepts and understanding how to use these concepts to define the vision, find actors and use cases, and to define the basic concepts the system will use. If we go no further, we have an overview of what the system will do, an understanding of the stakeholders of the system, and an understanding of the ways the system provides value to those stakeholders. What we do not have, if we stop at this point, is an understanding of exactly what the system does. In short, we lack the details needed to actually develop and test the system.

Some people, having only come this far, wonder what use-case modeling is all about and question its value. If one only comes this far with use-case modeling, we are forced to agree; the real value of use-case modeling comes from the descriptions of the interactions of the actors and the system, and from the descriptions of what the system does in response to the actions of the actors. Surprisingly, and disappointingly, many teams stop after developing little more than simple outlines for their use cases and consider themselves done. These same teams encounter problems because their use cases are vague and lack detail, so they blame the use-case approach for having let them down. The failing in these cases is not with the approach, but with its application.

The following chapters describe how to write use-case descriptions, how to manage detail, and how to structure the model to make it easier to understand. We also discuss how to review use cases, including how to organize

and staff the reviews. The intent of these chapters is to reveal how the use-case descriptions unfold from the basic modeling effort and how the structure of the use-case model emerges from the contents of the use-case descriptions.

The goal of Part II is to equip you with the knowledge needed to write good use-case descriptions, managing detail appropriately and avoiding the pitfalls of too much or too little structure. Part II also represents a transition from a "group" style of working to a more solitary style. While it is best to identify actors and use cases as a group, it is impractical to write use-case descriptions as a group; writing is almost always principally an activity performed by one person, with reviews of the material conducted as a group. Finally, we conclude Part II with a discussion of how and when to review use cases.

So let's continue on our journey into the world of use cases.

Chapter 6

# The Life Cycle of a Use Case

So far, we have seen the basic concepts behind the use-case modeling approach to eliciting and capturing software requirements and looked at how to get started in applying them. Before we look at the mechanics of authoring full use-case descriptions, we need to have a better understanding of the life cycle of a use case and how well-formed, good quality use cases can drive and facilitate the other, downstream software development activities. We also need to put what we have learned into a broader perspective with regard to software development and team working.

Use cases have a complex life cycle—they undergo a series of transformations as they mature through a number of development stages, from discovery to implementation and eventually to user acceptance. One way that this life cycle manifests itself is in the style and form adopted for the use-case descriptions. To speak of a single way of representing a use case is to miss the point—there are different presentation approaches and styles that are useful at different points in the use case's evolution. There is no one single form that is "better" in the absolute sense; they all play a role. This is why you will often see use cases expressed in different formats by different authors in different use-case texts.

Use cases also play a broader role, outside of the requirements space, in driving the analysis, design, implementation, and testing of the system. This is why you will also read about use cases being realized in design and tested by testers. Sometimes the use cases are so embedded in the design process of the system that the impression is given that the use cases are a development artifact rather than a requirements one. This misconception often leads to

developers trying to manipulate the use-case model in a misguided attempt to design the system using use cases.

To fully understand the role and purpose of use cases, and consequently the most appropriate form to use, we need to look at the life cycle of a use case from a number of different but complementary perspectives:

- **Software development:** how the use case is reflected throughout the full software development life cycle
- **Use-case authoring:** how the use case and its description evolves through the authoring process
- **Team working:** the activities involved in creating a use case model and how these impact on team and individual working practices

## THE SOFTWARE DEVELOPMENT LIFE CYCLE

As well as facilitating the elicitation, organization, and documentation of requirements, use cases can play a more central and significant role in the software development life cycle. This is especially true for many of the object-oriented and iterative development processes for which use cases are recommended.

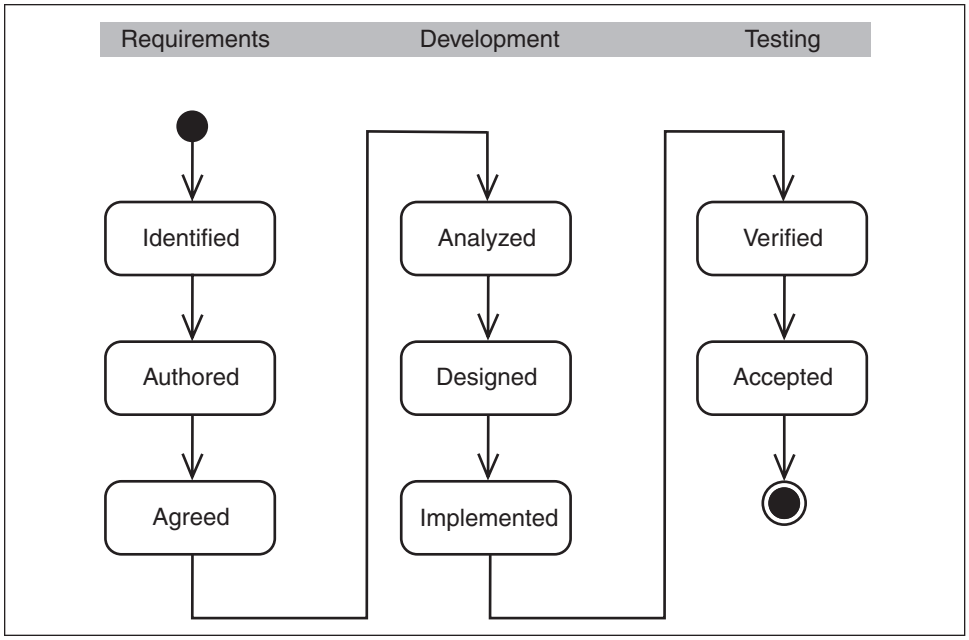
From a traditional object-oriented system model, it's often difficult to tell how a system does what it's supposed to do. This difficulty stems from the lack of a "red thread" through the system when it performs certain tasks.[1] Use cases can provide that thread because they define the behavior performed by a system. Use cases are not part of traditional object orientation, but over time their importance to object-oriented methods has become ever more apparent. This is further emphasized by the fact that use cases are part of the Unified Modeling Language.

In fact, many software development processes, including the Rational Unified Process, describe themselves as "use-case driven."[2] When a process employs a "use-case driven approach" it means that the use cases defined for a system are the basis for the entire development process. In these cases the life cycle of the use case continues beyond its authoring to cover activities such as analysis, design, implementation, and testing. This life cycle is shown,

[1] Ivar Jacobson introduced the notion that use cases can tie together the activities in the software development life cycle; see *Object-Oriented Software Engineering, A Use-Case Driven Approach*, 1992, ACM Press.

[2] See, for example, Philippe Kruchten's *The Rational Unified Process: An Introduction* or Jacobson et al., *The Unified Software Development Process*.

**Figure 6-1** The software development life cycle*




* This life cycle diagram is not intended to imply that analysis cannot be started until all the use cases have been agreed on or even until any use cases have been agreed on. The diagram is just saying that you cannot consider the analysis of a use case to be completed before the use case authoring has been completed and the use case itself agreed on.

in simplified form, in Figure 6-1. Figure 6-1 is arranged to emphasize the three main applications for the use cases:

- **Requirements:** the identification, authoring and agreement of the use cases and their descriptions for use as a requirement specification. This is the focus of this book.
- **Development:** the analysis, design, and implementation of a system based on the use cases. This topic is outside the scope of this book.[3]
- **Testing:** the use-case-based verification and acceptance of the system produced. Again, the details of how to undertake use-case-based testing is outside the scope of this book.

[3] For more information on using use cases to drive the analysis and design of software systems, we would recommend Doug Rosenberg and Kendall Scott's *Use Case Driven Object Modeling with UML: Practical Approach* and Craig Larman's *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*.

It is this ability of use cases to unify the development activities that makes them such a powerful tool for the planning and tracking of software development projects.[4]

To fully understand the power of use cases, it is worth considering this life cycle in a little more detail. Use cases can play a part in the majority of the disciplines directly associated with software development.

- **Requirements:** The use-case model is the result of the requirements discipline. Requirements work matures the use cases through the first three states, from Identified to Agreed. It also evolves the glossary, or domain model, that defines the terminology used by the use cases and the Supplementary Specification that contains the systemwide requirements not captured by the use-case model.
- **Analysis and Design:** Use cases are realized in analysis and design models. Use-case realizations are created that describe how the use cases are performed in terms of interacting objects in the model. This model describes, in terms of subsystems and objects, the different parts of the implemented system and how the parts need to interact to perform the use cases. Analysis and design of the use cases matures them through the states of Analyzed and Designed. These states do not change the description of the use cases, but indicate that the use cases have been realized in the analysis and design of the system.
- **Implementation** (also known as code and unit test or code and build): During implementation, the design model is the implementation specification. Because use cases are the basis for the design model, they are implemented in terms of design classes. Once the code has been written to enable a use case to be executed, it can be considered to be in the Implemented state.
- **Testing:** During testing, the use cases constitute the basis for identifying test cases and test procedures; that is, the system is verified by performing each use case. When the tests related to a use case have been successfully passed by the system, the use case can be considered to be in the Verified state. The Accepted state is reached when a version of the system that implements the use case passes independent user-acceptance testing. Note: If the system is being developed in an incremental fashion, the use cases need to be verified for each release that implements them.

[4] If a project manager's perspective on use cases is desired, we recommend Walker Royce's *Software Project Management: A Unified Framework*.

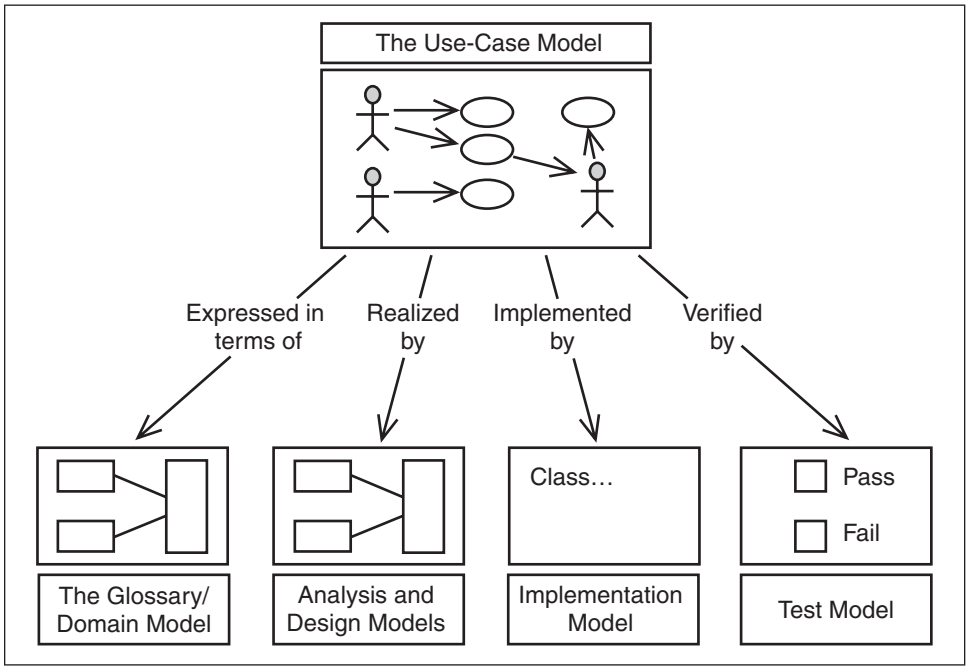
These relationships are directly reflected in the life cycle of the use case just described and are illustrated in Figure 6-2.

Use cases can also help with the supporting disciplines, although these do not impact upon the life cycle of the use cases themselves:

- **Project Management:** In the project management discipline, use cases are used as a basis for planning and tracking the progress of the development project. This is particularly true for iterative development where use cases are often the primary planning mechanism.
- **Deployment:** In the deployment discipline, use cases are the foundation for what is described in user's manuals. Use cases can also be used to define how to order units of the product. For example, a customer could order a system configured with a particular mix of use cases.

Although primarily a requirement-capture technique, use cases have a significant role to play in the ongoing planning, control, development, and testing of the system. It is this unification of the software development process that makes use cases such a powerful technique. To get the full benefit of

**Figure 6-2** The use-case model and its relationship to the other software development models



using use cases, they should be placed at the heart of all the software development and project planning activities.[5]

## THE AUTHORING LIFE CYCLE

Of more direct relevance to the people involved in the writing of use cases is having a clear understanding how the use case and its description evolves through the authoring process. We have seen the following use-case formats in use in various different projects and texts:

- Use cases that look like just brief descriptions—a short paragraph that describes something that the system does
- Use cases that look like outlines—a numbered or bulleted list of events and responses
- Use cases presented in the form of a table of actor actions and system responses
- Use cases that present a purely "black box" view of the system, focusing on the actions taken by the actor and the system's response
- Use cases presented as structured English, using sequential paragraphs of text and a more expansive, narrative form, like many of the examples presented in this book

There are also many different popular styles of use case, such as *essential use cases*[6] and *conversational style*[7] use cases.

What are all these use cases, and how do they relate to one another?

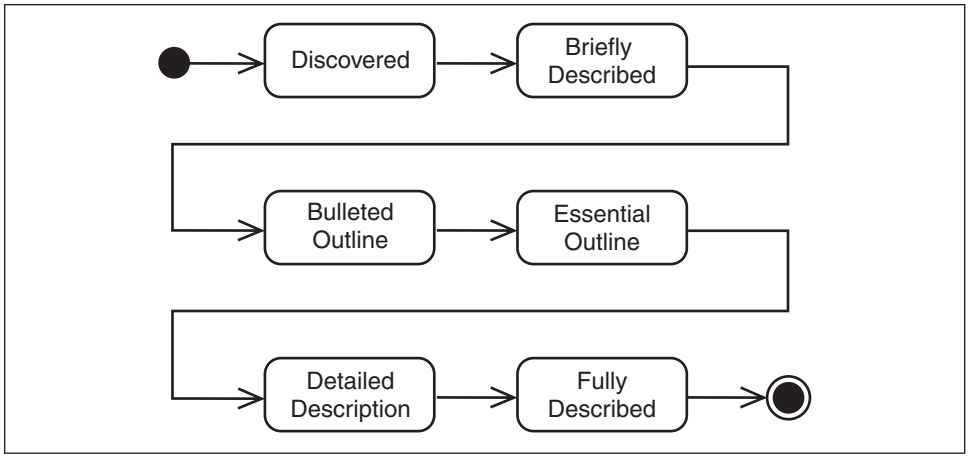
It is our contention that these are all just states in the evolution of a use case. Figure 6-3 provides a visual summary of the states of a use case during its evolution from its initial discovery to the production of its fully detailed and cross-referenced description. Each of these different forms is appropriate at different points in the evolution of a use-case model. Different use cases will evolve at different rates. It is not uncommon for an early version of the

[5] For more information on how use cases can shape and drive the entire software development process, we would recommend the following texts:
- Philippe Kruchten, *The Rational Unified Process: An Introduction*
- Jacobson, Booch, and Rumbaugh, *The Unified Software Development Process*
- Jacobson, Christerson, Jonsson, and Overgaard, *Object Oriented Software Engineering: A Use Case Driven Approach*, the original books that popularized use cases.

[6] Larry Constantine is most often associated with this formulation of use cases; see L. Constantine, "The Case for Essential Use Cases," *Object Magazine,* May 1997. SIGS Publications.

[7] Rebecca Wirfs-Brock has notably promoted this technique; see R. Wirfs-Brock, "Designing Scenarios: Making the Case for a Use Case Framework," *Smalltalk Report,* Nov-Dec 1993.

**Figure 6-3** The authoring life cycle*



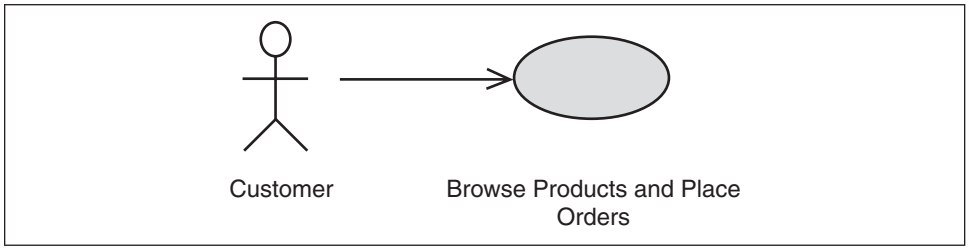

* The states shown in the authoring life cycle can be considered to be substates of Identified and Authored states in the software development life cycle shown in Figure 6-1. Discovered and Briefly Described are substates of Identified; the others are substates of Authored.

use-case model to contain a number of key use cases that are fully described and other, less important use cases that are still in the briefly described state or even awaiting discovery. It is worth taking a detailed look at each of these states, how they are manifested in the use-case description, and the role that they play in the evolution of the use case.

### *State 1: Discovered*

A use case will begin as just a name (for example, *Browse Products and Place Orders*), perhaps on a diagram with an associate actor (for example, Customer), as in Figure 6-4. This name is a placeholder for what is to come, but if

**Figure 6-4** A newly discovered use case.



this is as far as the description goes, it is not very useful. The use-case diagrams produced at this stage really act as no more than a visual index, providing a context for the use-case descriptions that are to come.

### State 2: Briefly Described

Almost immediately, usually while the name is being discussed, people will start briefly describing the use case; typically, they can't help it. Even as a name is being proposed, people will start to elaborate on the name (for example: *This use case enables the customer to see the products we have to offer and, we hope, to buy them. While browsing, they may use a number of techniques to find products, including direct navigation and using a search facility.*) These discussions should be captured more formally as the brief description of the use case.

**Example**

**Brief description for the use case *Browse Products and Place Orders* in an on-line ordering system**

This use case describes how a Customer uses the system to view and purchase the products on sale. Products can be found by various methods, including browsing by product type, browsing by manufacturer, or keyword searches.

This brief description is important, and it may be as far as the use case evolves, especially if the required behavior is simple, easily understood, and can be expressed in the form of a prototype more easily than in words. But if the behavior is more complex, particularly if there is some defined sequence of steps that must be followed, more work is needed.

### State 3: Bulleted Outline

The next stage in the evolution of the use case is to prepare an outline of its steps. The outline captures the simple steps of the use case in short sentences, organized sequentially. Initially, the focus is on the basic flow of the use case—generally this can be summarized in 5–10 simple statements. Then the most significant alternatives and exceptions are identified to indicate the scale and complexity of the use case. This process was discussed in detail in Chapter 4, Finding Actors and Use Cases, as it is an integral part of establishing the initial shape and scope of the use-case model.

**Example**

**Outline for the use case *Browse Products and Place Orders***

*Basic Flow*

1. Browse Products
2. Select Products
3. Identify Payment Method
4. Identify Shipping Method
5. Confirm Purchase

*Alternative Flows*

A1 Keyword Search

A2 No Product Selected

A3 Product Out of Stock

A4 Payment Method Rejected

A5 Shipping Method Rejected

A6 Product Explicitly Identified

A7 Order Deferred

A8 Ship to Alternative Address

A9 Purchase Not Confirmed

A10 Confirmation Fails

etc….

Bulleted outlines of this form are good for getting an understanding of the size and complexity of the use case, assessing the use case's architectural significance, verifying the scope of the use case, and validating that the use-case model is well formed. They also provide a good basis for exploratory prototyping aimed at revealing requirement and technology-related risks.

If the use cases are to act as the specification of the system and provide a basis for more formal analysis, design, and testing, then more detail is required.

## *State 4: Essential Outline*

So-called *essential* use cases are at another point in the use case's evolutionary timeline. Essential use cases focus on only the most important behavior of the system and leave much of the detail out (even omitting the mention of a PIN when describing the ATM's Withdraw Cash use case, for instance) in order to

focus on getting right what the system must do. This is important early in the use-case identification process, when it is easy to get mired in details that will become important later but are not essential to defining the system as a whole.

The defining characteristic of this format is that it presents a pure, external, "black-box" view of the system, intentionally focusing on its usability. The strength of this approach is that it places usability "front and center" and in so doing ensures that the needs of the user are placed first. This format helps describe user intent and actions, along with the observable response of the system, but it does not elicit details about what is happening inside the system. It also ignores the specifics of the user-interface (because this information is better and more easily presented in prototypes and user interface mock-ups). The description is often presented in a two-column format:

**Example**

**The essential form of the use case *Browse Products and Place Orders***

| *User Action* | *System Response* |
|---|---|
| 1. Browse product offerings | Display product offerings |
| 2. Select items for purchase | Record selected items and quantities |
| 3. Provide payment instructions | Record payment instructions |
| 4. Provide shipping instructions | Record shipping instructions |
| 5. Complete transaction | Record transaction and provide receipt |

The mistake made with essential use cases is forgetting that they will continue to evolve, adding detail and increasing in both scope and number, as the project progresses. Not every use case will pass through the Essential Outline state. Many use cases will progress straight from the bulleted outline to the more detailed formats, if they evolve beyond the bulleted outline form at all. Typically, the essential use-case form is used to provide an early embryonic description of the most important use cases in the system. The descriptions will then continue to evolve. You do not develop a set of essential use cases, then move on to a separate set of conversational use cases, and then move on to a another, different set of more detailed use cases. They are the same things at different points in their evolution.

Essential use cases are very effective for facilitating user-interface and user-experience analysis and design, especially where a system's visual metaphor needs to be established, typically early in the project's life cycle. Too much detail in the use cases often limits and constrains the creativity of the user-interface designers. The stripped-down essential outlines capture the essence of the required dialog without forcing the designers into any particular technology or mode of interaction. This allows them to start to explore the presentation

options for the system, which, once defined, may impact in turn on the style and level of detail adopted in the final-form, fully detailed use-case descriptions.

Some people recommend that use-case authoring stop at the essential outline state, but if the use cases are to be used to drive the other aspects of systems design, act as the basis for formal integration and system testing, or be used as the basis for contractual relationships, more detail is required.

## *State 5: Detailed Description*

The next step in the authoring life cycle is to start adding to the outline the detail required to complete the specification of the system. In this state, the use case is evolving, as more and more detail is added to flesh out the outline. If the use case expresses a strong sense of a dialog between an actor and the system, then the description may be in the *conversational form*; otherwise, it will be in the *narrative form* and simply list the steps in order.

### The Conversational Form

The conversational form of use-case description is most useful when the system and actor engage in a well-defined dialog in which the actor does something and the system does something in response.

**Example**

**The conversational form of the use case *Browse Products and Place Orders***

| *User Action* | *System Response* |
|---|---|
| 1. Browse product offerings | Display product offerings, showing categories selected by the user |
| 2. Select items for purchase | For each selected item in stock, record selected items and quantities, reserving them in inventory. |
| 3. Provide payment instructions | Record payment instructions, capturing payment terms and credit card type, number, and expiration date using a secure protocol. |
| 4. Provide shipping instructions | Record shipping instructions, capturing billing address, shipping address, shipper preferences, and delivery options. |
| 5. Complete transaction | Record transaction and provide receipt containing a list of the products ordered, their quantity and prices, as well as the billing and shipping addresses and the payment terms. The credit card information should be partially omitted, displaying only the last 4 digits of the credit card number. |

This *conversational* format is excellent for a number of situations: where there is only one actor and where the system and actor engage in an interactive dialog. It can be expanded to include a considerable amount of detail but will often become a liability. It is difficult to use when there is more than one actor (as often happens in real business systems) or when there is a simple actor action (like pressing on the brake pedal) with a complex response (such as controlling the antilock braking system).

### The Narrative Form

The most common format for a detailed use-case description is the *narrative form*. In this form, the outline is again expanded by adding detail but the tabular format is replaced by a more narrative description.

**Example**

**The narrative form of the use case *Browse Products and Place Orders***

1. The use case starts when the Customer selects to browse the catalogue of product offerings. The system displays the product offerings showing the categories selected by the Customer.
2. The Customer selects the items to be purchased. For each selected item that is in stock the system records the items and quantity required, reserving them in inventory.
3. The system prompts the Customer to enter payment instructions. Once entered, the system records payment instructions, capturing payment terms and credit card type, number, and expiration date using a secure protocol.
4. The system prompts the Customer to enter shipping instructions. Once entered, the system records the shipping instructions, capturing billing address, shipping address, shipper preferences, and delivery options.
5. The system prompts the Customer to confirm the transaction. Once confirmed, the system records the transaction details and provides a receipt containing a list of the products ordered, their quantity and prices, as well as the billing address, shipping address, and payment terms. Credit card information is partially omitted, displaying only the last 4 digits of the credit card number.

This format is more flexible, allowing the system to initiate actions and supporting the interaction with multiple actors if required. This is the format that we prefer, as it more readily supports the ongoing evolution of the use case into its final form and the use of subflows to further structure the text.

### Using the Detailed Description

Regardless of the form chosen for the detailed description, it is a state that the majority of use cases will pass through as they evolve toward the fully detailed description. In fact, this is the state that most allegedly "completed" use cases are left in as the use-case modeling efforts run out of steam. Unfortunately, it is dangerous to evolve the use cases to this state only and not to complete their evolution. The detailed description loses the benefits of brevity and succinctness offered by the bulleted and essential outline formats and lacks the detail required of a fully featured requirements specification. We do not recommend stopping work on the use cases when they have reached this state. If it is not necessary to evolve a use case to its full description, then stop at the outline format and don't waste time adding incomplete and ambiguous detail just for the sake of it.

## *State 6: Fully Described*

The final state in the evolution of a use case is Fully Described. This is the state in which the use case has a complete flow of events, has all of its terminology fully defined in the supporting glossary, and unambiguously defines all of the inputs and outputs involved in the flow of events.

Fully described use cases are

- **Testable:** There is sufficient information in the use case to enable the system to be tested.
- **Understandable:** The use case can be understood by all of the stakeholders.
- **Unambiguous:** The use case and the requirements that it contains have only one interpretation.
- **Correct:** All of the information contained within the use case is actually requirements information.
- **Complete:** There is nothing missing from the use cases. All the terminology used is defined. The flow of events and all of the other use-case properties are defined.
- **Attainable:** The system described by the use case can actually be created.

Fully described use cases support many of the other software development activities, including analysis, design, and testing. One of the best checks of whether the use-case description is finished is to ask yourself if you could use the use case to derive system tests. The best way to tell if the use cases fit the purpose is to pass them along to the analysis and design team for analysis and the test team for test design. If these teams are satisfied that they can use the use cases to support their activities, then they contain sufficient levels of detail.

**Example**

**An extract from the fully described use case *Browse Products and Place Orders***

*Basic Flow*

1. The use case starts when the actor Customer selects to browse the **catalogue of product offerings**.

**{Display Product Catalogue}**

2. The system displays the **product offerings** highlighting the **product categories** associated with the Customer's **profile**.

**{Select Products}**

3. The Customer selects a **product** to be purchased entering the number of items required.
4. For each selected item that is in stock the system records the **product identifier** and the number of items required, reserving them in inventory and adding them to the Customer's **shopping cart**.

**{Out of Stock}**

5. Steps 3 and 4 are repeated until the Customer selects to order the **products**.

**{Process the Order}**

6. The system prompts the Customer to enter **payment instructions**.
7. The Customer enters the **payment instructions**.
8. The system captures the **payment instructions** using a **secure protocol**.
9. Perform ***Subflow Validate Payment Instructions***

. . .

Note that this fully described use case uses the narrative format. If the use case has only one actor and the system and actor engage in an interactive dialog, then the conversational style could also be used.

As you can see, there is much more to be said about the formatting and authoring of fully described use-case descriptions. This is the subject of Chapter 7, The Structure and Contents of a Use Case; Chapter 8, Writing Use-Case Descriptions: An Overview; and Chapter 9, Writing Use-Case Descriptions: Revisited.

## TEAM WORKING

Another interesting perspective on the life cycle of a use case is that related to team working and the activities that are undertaken to produce the use-case model. We have seen that use cases have an important role to play in the

software development life cycle and also have an authoring life cycle of their own. In Chapters 3, 4, and 5, we also looked at how the use-case model starts to emerge from the vision of the system via a series of workshops and other group-focused activities. In this section, we will look at the use-case modeling process and how this impacts on individual and team working.

You may wonder why we have saved this more formal look at the use-case modeling process for the second part of the book rather than presenting it earlier. Well, basically, we wanted you to have a good understanding of the concepts before we started to talk about all of the activities involved in creating a use-case model. So treat this section as part recap of what you have already learned and part teaser for what you will learn in Part II.

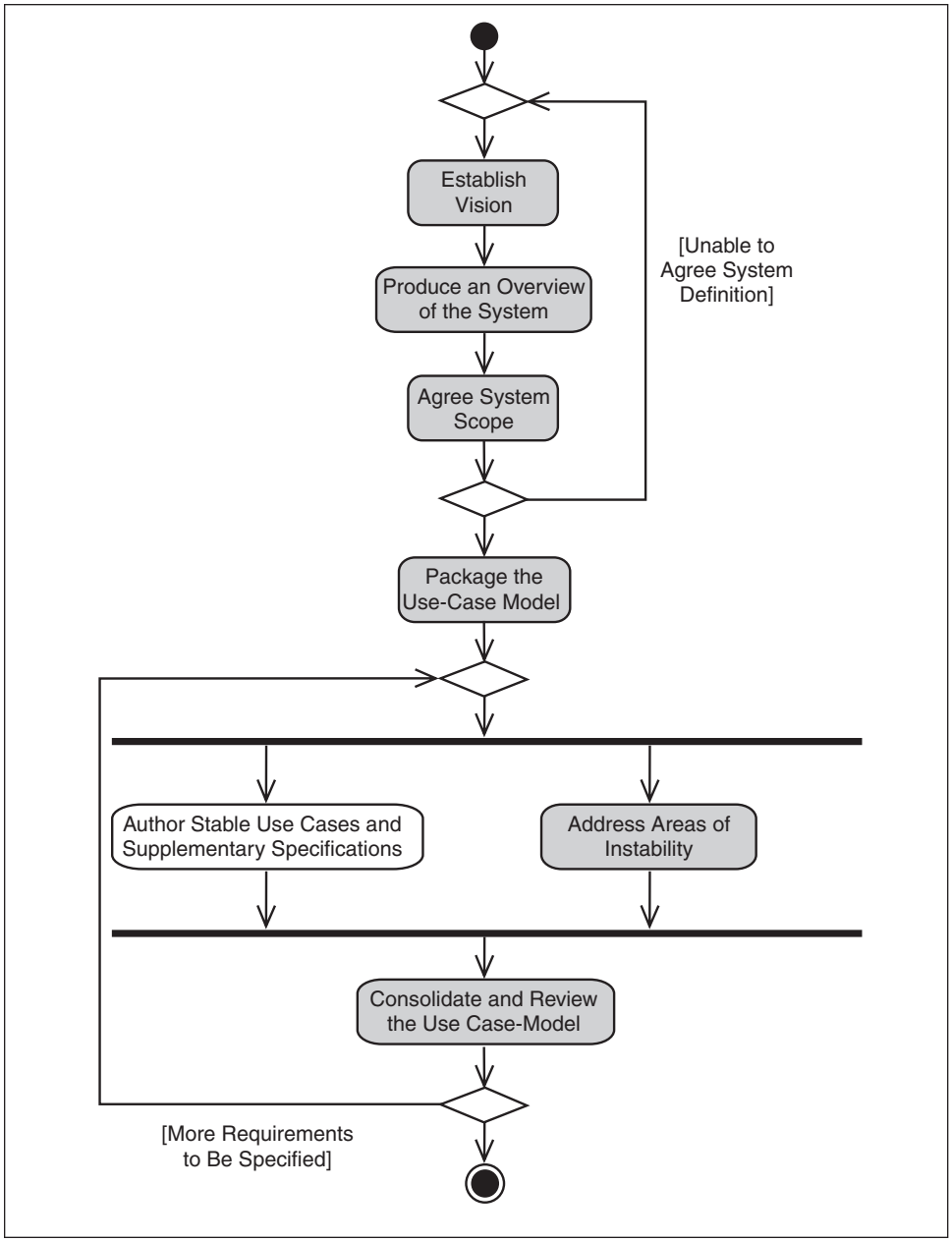
## The Use-Case Modeling Process

Figure 6-5 illustrates the activities involved in the development of a use-case model. This is a simplified subset of a full requirements process[8] and emphasizes the major activities involved in the evolution of the use-case model, which is being used as the primary requirements artifact. It is interesting to look at this workflow from the perspective of group and individual activities. In Figure 6-5, the group activities are shown in gray and are focused on preparing the groundwork for the evolution of the use-case model and its supporting Supplementary Specification by establishing the vision, scoping the system, addressing areas of uncertainty and instability in the requirements definition, and consolidating and reviewing the use-case model as a whole. The diagram can give the wrong impression that the majority of the effort in use-case modeling is related to group activities and that the model can be accomplished by simply holding a series of workshops and brainstorming sessions with the user and stakeholder representatives.

In fact, more time is typically spent on the individual use case and Supplementary Specification authoring activities than is spent on all of the group activities put together. Figure 6-6 shows the relative amounts of effort expended on group and individual activities across the life of a project, which would typically iterate through the process many times. Note that the figure shows the relative amounts of effort and is not intended to be indicative of the total amount of effort required at any point in the project. The graph illustrates where healthy projects spend their time and should not be taken as a definitive statement. The amount of time the group activities will take is dependent on the ability of the group to focus and reach decisions. If all the

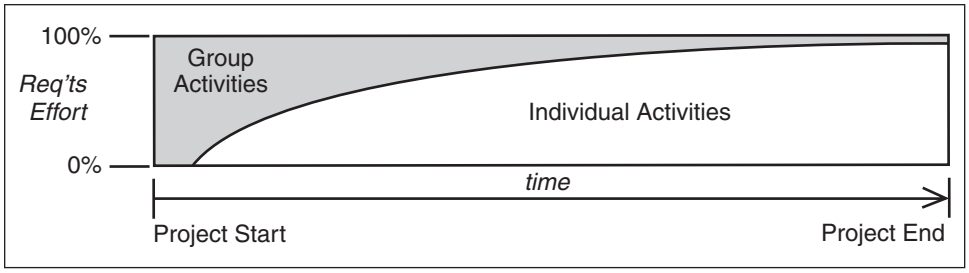
[8] For a fully documented Requirements Life Cycle that is seamlessly integrated with all of the other software development disciplines, see the Rational Unified Process.

**Figure 6-5** The use-case modeling process*




* Note: The use-case modeling process is not as waterfall / linear as this figure may imply. If applying the process iteratively, then you only need agreement that a single use case is in scope and its purpose is stable before you start to author it; there is no need to have a full scope definition in place. This process can in fact be applied in every iteration, with just enough envisioning and scoping of the system to select the use cases to be worked on in the iteration.

**Figure 6-6** Ratio of group and individual activities for a typical project




stakeholder representatives disagree with each other and spend all of their time fighting and arguing, the project may never achieve enough stability for it to be worth undertaking the authoring of the use cases. These issues were addressed in Part I: Getting Started with Use-Case Modeling. The amount of time that the individual authoring activities will take is dependent on the complexity of the solution and the capabilities of the individuals involved. These issues are addressed in more detail in Chapter 8, Writing Use-Case Descriptions: An Overview.

It is worth taking a detailed look at each of the activities shown in Figure 6-5 and the roles that use cases and the use-case model play in undertaking them.

## *Establish the Vision*

Establishing the vision is a group activity aimed at getting all of the stakeholders to agree about the purpose and objectives for both the project and the system to be built. The best way to achieve this is to use traditional requirements-management techniques to produce a high-level system definition and to ensure that there is agreement on the problem to be solved. Typically, this is done via a series of workshops involving the project's major stakeholder representatives. This topic was covered in detail in Chapter 3, Establishing the Vision.

The use-case model can help in establishing the vision by defining the system boundary and providing a brief overview of the system's behavior, but it is really no substitute for a vision document. If this stage is skipped, then no real attempt is made to analyze the problem before starting on the definition of the solution. This is really only applicable for small-scale, informal, low-accountability projects with a very small set of stakeholders and where the developers and the users work very closely together. Without undertaking any problem analysis, it can be difficult to know when the use-case model itself describes a suitable solution.

### Produce an Overview of the System

The initial use-case model, containing the key actors and use cases with brief descriptions and outlines, provides a very good overview of the functionality of a system. This should be complemented with an initial draft of the key Supplementary Specifications and an outline glossary or domain model. At this stage, there is no need to fully detail any of the use cases, although it is a good idea to have identified the majority of the significant alternative flows for each of them. We are just looking for enough information to allow the scoping of the system with regard to the current project. This activity is best done as a group activity in a series of use-case modeling workshops, as described in Chapter 5, Getting Started with a Use-Case Modeling Workshop, and using the techniques described in Chapter 4, Finding Actors and Use Cases.

### Reach Agreement on System Scope

The next activity is to reach agreement on the scope of the system. To do this, the proposed use-case model needs to be examined in light of the vision and any other high-level requirements documentation produced as part of the project.

Use cases are a very powerful aid when attempting to manage the scope or the system. Use cases lend themselves to prioritization. This prioritization should be undertaken from three perspectives:

1. **Customer Priority:** What is the value placed on each of the use cases from a stakeholder perspective? This will identify any use cases that are not required by the stakeholders and allow the others to be ranked in order of customer priority.
2. **Architectural Significance:** Which of the use cases are going to stress and drive the definition of the architecture? The architect should examine the use cases and identify those use cases that are of architectural significance.
3. **Initial Operational Capability:** What set of use cases would provide enough functionality to enable the system to be used? Are all of the use cases needed to provide a useful system?

By considering these three perspectives it should be possible to arrive at a definition of system scope, and order of work, that satisfies all parties involved in the project.

If these three perspectives do not align (that is, the use cases the customer most wants are not those of architectural significance and do not form a significant part of a minimally functional system), then the project is probably out of balance and likely to hit political and budgetary problems. A lot of

expectation management would be required to bring these three perspectives into alignment and place the project on a healthy footing where the customer and the architectural goals are complementary rather than contradictory.

Beyond the use cases themselves, we can also use the flow-of-events structure for scope management. In most cases, the basic functionality of the majority of the use cases will be needed to provide a working system. The same cannot be said of all of the alternative flows. In the ATM system, is it really necessary to support the withdrawal of nonstandard amounts or the use of the secondary accounts associated with the card? In many use cases, the majority of the alternative flows will be "bells and whistles" that are neither desired by the customer nor necessary to produce a useable system. This will be discussed in more detail in Chapter 7, The Structure and Contents of a Use Case, when we discuss the additive nature of use-case flows.

Once the scope for the project has been agreed on, the use cases that have been selected for initial implementation can be driven through the rest of their life cycle to completion and implementation. If iterative and incremental development is being undertaken, then the use cases can be assigned to particular phases and iterations.

## *Package the Use-Case Model*

As the scope of the system and the structure of the use-case model start to become apparent, it is often a good idea to package up the use cases and actors into a logical, more manageable structure to support team working and scope management. Using the UML, packages can be used to structure the use-case model.

The UML defines the package as

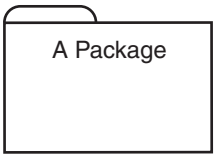
> A general-purpose mechanism for organizing elements into groups.

Graphically, the package is represented using a folder icon, as shown in Figure 6-7. In a use-case model a package will contain a number of actors, use cases, their relationships, use-case diagrams, and other packages; thus, you can have multiple levels of use-case packages (packages within packages), allowing the use of hierarchical structures where appropriate. Often, the use-case model itself will be represented as a package that contains all of the elements that make up the model.

There are many reasons for using use-case packages to partition the use-case model:

- **To manage complexity.** It is not unusual for a system to have many actors and use cases. This can become very confusing and inaccessible to the stakeholder representatives and developers working with the

**Figure 6-7** The graphical representation of a package




model. A model structured into smaller units is easier to understand than a flat model structure (without packages) if the use-case model is relatively large. It is also easier to show relationships among the model's main parts if you can express them in terms of packages.

- **To reflect functional areas.** Often, there are families of use cases all related to the same concepts and areas of functionality (for example, customer service, operations, security, or reporting). Use-case packages can be used to explicitly group these use cases into named groups. This can make the model more accessible and easier to manage and discuss. It also helps to reduce the need for enormous "super" use cases that include massive sets of only loosely-related requirements.
- **To reflect user types.** Many change requests originate from users. Packaging the use-case model in this way can ensure that changes from a particular user type will affect only the parts of the system that correspond to that user type.
- **To support team working.** Allocation of resources and the competence of different development teams may require that the project be divided among different groups at different sites. Use-case packages offer a good opportunity to distribute work and responsibilities among several teams or developers according to their area of competence. This is particularly important when you are building a large system. Each package must have distinct responsibilities if development is to be performed in parallel. Use-case packages should be units having high cohesion so that changing the contents of one package will not affect the others.
- **To illustrate scope.** Use-case packages can be used to reflect configuration or delivery units in the finished system.
- **To ensure confidentiality.** In some applications, certain information should be accessible to only a few people. Use-case packages let you preserve secrecy in areas where it is needed.

The introduction of use-case packages does have a downside. Maintaining the use-case packages means more work for the use-case modeling team, and the use of packaging means that there is yet another notational concept

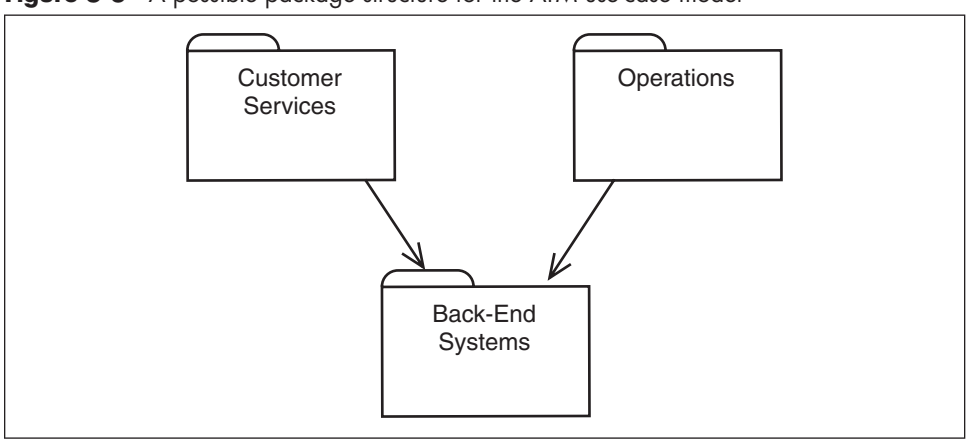
for the developers to learn. As the need for packaging is directly related to the size and complexity of the use-case model, this is an optional activity and may be skipped for smaller models.

If you use this technique, you have to decide how many levels of packages to use. A rule of thumb is that each use-case package should contain approximately 3 to 10 smaller units (use cases, actors, or other packages). The following list gives some suggestions as to how many packages you should use given the number of use cases and actors. The quantities overlap because it is impossible to give exact guidelines.

- 0–15: No use-case packages needed.
- 10–50: Use one level of use-case packages.
- > 25: Use two levels of use-case packages.

Packages are named in the passive, as opposed to the active names used for the use cases themselves, typically representing some area of the system's functionality or some organizational element of the business that is going to use or support the system. For example, the ATM functionality could be split into two packages, Customer Services and Operations, both of which are supported by the back-end banking systems, as shown in Figure 6-8. The dashed arrows are UML dependency relationships, which, in this case, indicate that model elements in the Customer Services and Operations packages access model elements in the Back End Systems package. This allows us to see how independent the packages are from one another, which is essential if the packaging is to support team working and model management. Packages are a

**Figure 6-8** A possible package structure for the ATM use-case model



standard UML model element and are not any different for use-case models than they are for any other UML model.[9]

Once the packaging has been put in place, it is usually difficult to change without causing great disruption to the people working with the model. For this reason, it is not advisable to attempt the packaging too early in the evolution of the use-case model. Packaging the model is again primarily a group activity that is undertaken, with the help of the stakeholder representatives, as part of the final use-case modeling workshop or review.

### *Address Areas of Instability and Author Stable Use Cases and Supplementary Specifications*

Once the scope of the system has been established and the use-case model structured to facilitate the further development of the use cases, we are faced with two parallel activities:

1. The detailed authoring of the requirements for those areas of the model where there is stability. This is an individual activity and is the subject of Chapter 8, Writing Use-Case Descriptions: An Overview, and Chapter 9, Writing Use-Case Descriptions: Revisited. It is in the authoring of the detail that most of the effort related to use cases is expended.
2. Continuing to run additional workshops to address those areas where there is still instability in the use-case model. This entails running use-case modeling workshops (as described in Chapter 5, Getting Started with a Use-Case Modeling Workshop) with more detailed objectives and a more specialized selection of stakeholder representatives.

Typically, when the use-case model is being constructed initially, there will be some areas of the model with which everybody agrees and others where consensus is harder to reach during the early project brainstorming and use-case modeling workshops. There is no need to wait for agreement on every area of the use-case model before proceeding to the authoring of detailed use-case descriptions. Once agreement has been reached that a use case is required, it can be driven through the authoring process to produce the fully detailed description and through the software development process to facilitate the design and implementation of the software. It is counterproductive to start doing detail work for use cases whose scope, purpose, and intention are still under debate. To evolve these beyond the essential outline stage

[9] For more information on packages and package relationships, we would recommend the *Unified Modeling Language User Guide* by Booch, Rumbaugh, and Jacobson.

is likely to cause large amounts of scrap and rework. The level of detail provided by the outlines should be sufficient to allow scoping and other decisions to be made.

The first use cases to stabilize and then proceed through the authoring process should be those of architectural significance, those that explicitly help to attack project risk, and those essential to the initial release. Once the authoring of any of these use cases is complete, they should be passed over to the designers so that they can progress through the rest of the software development life cycle. In the same way that there is no need for all the use cases to have been identified and outlined before detailed authoring starts, there is no need for all the use cases to have been authored before analysis, design, and the other downstream activities start. It is our recommendation that use cases be passed on to the other project teams as soon as they become available. This allows the downstream activities to start as soon as possible and will provide the use-case authors with the immediate feedback on their work that they can use to improve the quality of the use-case model as a whole.

### *Consolidate and Review the Use-Case Model*

As the use cases, the Supplementary Specifications, and the use-case model evolve, it is worth taking some time to consolidate and review the team's work as a whole. This should be a group activity and should focus on achieving consistency and completeness across the whole of the requirements space. This is also the time when you may want to do some more detailed structuring of the use cases themselves. These topics are covered in more detail in Chapter 10, Here There Be Dragons, and Chapter 11, Reviewing Use Cases. It is also worthwhile to check the detailed requirements work against the vision for the system to make sure that they have not diverged as the use-case model has evolved.

These suggestions are not intended to imply that all of the use cases are to be reviewed in one go at the end of the process. Walkthroughs and reviews are an essential part of the authoring process, as we shall see in Chapter 11, Reviewing Use Cases. Here we are talking about looking at the model as a whole rather than at the individual use cases.

## SUMMARY

There is a common misconception that use cases have one form or can be stated in only one way. Practitioners are therefore confused when they see use cases stated in different ways. Many of the differences between use cases stem

from the fact that a use case has a life cycle, and it will take different forms at different points in that life cycle.

The life cycle of a use case can be considered from many perspectives. It is important that people working with use cases understand the life cycle from the broader team working and software development perspectives as well as the use-case authoring perspective.

For the purposes of this book, the most important life cycle is use-case authoring. Initially, use cases begin as drawings that show the use cases and the actors who interact with the system during the use case. The use cases are little more than "ovals" and very terse names. This is sufficient for identification, but not much more. Very quickly, however, they evolve into brief descriptions, short paragraphs that summarize the things that the use case accomplishes. This brief description is sufficient for clarification, but more is still needed. The brief descriptions quickly give rise to outlines of the flows of events. Initially, these are just bulleted lists illustrating the basic flow and identifying the significant alternative flows. These bulleted outlines give an indication of the size and complexity of the use cases and are very useful for initial prototyping aimed at revealing requirements and technology-related risks.

For user-interface-intensive systems, the flows are often elaborated to cover the important things the user sees and does when interacting with the system. These "essential" use-case outlines are the primary drivers of the user interface's design. This level of description, while more than sufficient for users and interface designers, is greatly lacking for software developers and testers.

Additional evolution adds more information about the internal interactions, about testable conditions, and about what the system does, providing a more complete picture of the behavior of the system. These complete descriptions drive the development and testing of the system.

It's important to keep in mind that these are not "different" use cases, but the same use case from different perspectives and at different points in time. This "unified" view makes understanding and employing use cases easier.

The key to deciding how detailed to make your use cases is to consider two factors:

1. How unknown the area of functionality covered by the use case is. The more unknown, misunderstood, and risky the functionality described by the use case, the more detail is required.
2. What use is to be made of the description. It is very difficult to know when the use-case descriptions are complete if the downstream activities that the use cases are going to support are not also understood.

The following table summarizes the purpose, risks addressed, and downstream activities for each of the use-case authoring states:

| Authoring State | Primary Purpose | Risks Addressed | Downstream Activities |
|---|---|---|---|
| Discovered | Identify the use case | • Not knowing the boundary of the system | • Scope management |
| Briefly Described | Summarize the purpose of the use case | • Ambiguity in the model definition | • Scope management |
| Bulleted Outline | Summarize the shape and extent of the use case | • Not knowing the extent, scale or complexity of the system<br>• Not knowing which use cases are required | • Scope management<br>• Low-fidelity estimation.<br>• Prototyping aimed at addressing requirements and technological risks. |
| Essential Outline | Summarize the essence of the use case | • Ease of use | • User interface design<br>• Prototyping aimed at addressing requirements and technological risks |
| Detailed Description | To allow the detail to be added incrementally | • None—it is not recommended that use cases in this state be used outside of the authoring team | • None—this is purely an intermediate step. |
| Fully Described | Provide a full requirements specification for the behavior encapsulated by the use case | • Not knowing exactly what the system is supposed to do<br>• Not having a shared requirements specification | • Analysis and design<br>• Implementation<br>• Integration testing<br>• System testing<br>• User documentation<br>• High-fidelity estimation |