MKS

Use Cases: Background, Best Practices, and Benefits

An MKS White Paper
By Dennis Elenburg
Application Engineer

MKS

Abstract

Many books and papers have been written on the topic of use cases. In fact, the body of knowledge surrounding use cases is so large that it can be intimidating to the uninitiated. One of the barriers to successful adoption of use cases is navigating this abundance of information. This introductory paper discusses the background and benefits of use cases as well as two best practices for deploying use cases as a requirements management technique in your organization.

What are Use Cases?

Use cases are a powerful technique for capturing and communicating functional requirements for software development. Prior to the advent of use cases, functional requirements were typically captured by a long list of declarative statements. A typical Software Requirements Specification (SRS) used language such as "The system shall <insert requirement>…" in a long list of discrete requirements. The SRS was written from the perspective of the system. Functional requirements written in this manner lack the context and perspective for what the user actually wants to do with the system.

Use cases are written from the perspective of the user as a flow of events. The user is called an "actor" and the narrative of the flow of events between this actor and the system is called the "use case." Use cases are literally the specific "cases" for which the actor wants to "use" the system. Use cases differ from declarative statements in two primary ways. They describe functional requirements from the user perspective rather than the system perspective, and they provide a coherent goal focused flow of events rather than a set of discrete declarative statements. A fully described use case will have a main or basic flow as well as alternate flows. The alternate flows describe regular variants to the main flow as well as error handling or unusual situations.

Two Methods of Writing Functional Requirements

Use Cases	Declarative Statements
Broad Perspective	Narrow Perspective
User Orientation	System Orientation
Goal Focused Flow of Events	Many Discrete Items
Example: 1. *The Student enters a student ID and password and the system validates the student.* 2. *The system displays the functions available to the student: create, modify, delete. The student chooses create.* 3. *The system presents a list of course offerings. The student chooses up to four…* 4. *The System validates the courses selected and displays a confirmation number…*	*Example:* ▪ *The system shall provide a list of class offerings for the current semester* ▪ *The system shall only allow registration for courses where the prerequisites are fulfilled.* ▪ *The systems shall provide a secure login.* ▪ *The system shall provide a confirmation number when the schedule is confirmed*

Figure 1: Comparison of Declarative Statements and Use Cases

Today, use cases are commonly used throughout the requirements management community for capturing functional requirements for software development. Their popularity has even expanded into other domains such as packaged software implementations and business modeling. A companion paper to this introductory treatment of use cases will describe the benefits of use cases in deploying the Application Lifecycle Management tools used in a software development project. Similarly, use cases are valuable in deploying other types of commercial software packages that require customization such as Peoplesoft, Oracle, or SAP.

A Brief History of Use Cases

Use cases originated as a requirements modeling technique within the object-oriented (OO) software development community many years ago. By the mid 1990s, use case modeling was formalized as part of the Unified Modeling Language (UML) specification, which is managed by a not-for-profit open consortium known as the Object Management Group (OMG). The UML is the OMG's most-used specification and the defacto standard for modeling application structure, behavior, and architecture within the OO paradigm of software development. Use case diagrams are one of the twelve types of diagrams defined within the UML specification. Class diagrams, sequence diagrams, and activity diagrams are some of the other well-known UML diagrams.

Unfortunately, the UML specification is highly technical, terse, and very difficult to understand. According to the UML 2.0, a use case is "the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system." Don't let this definition intimidate you! The good news is there is no need to dig into the technical details of UML diagramming to obtain the many benefits of use cases. The diagrams themselves are relatively unimportant. The value of use cases is in the technique of writing the textual descriptions, the basic and alternate flows.

Use case diagrams are intentionally very simple. They are basically stick figures (actors) and ovals (use cases) with lines connecting them. They are great for white board discussions, and anyone can learn the basics of use case diagramming with a few minutes of instruction. However, use case diagrams are only one part of a complete use case model.

A complete use case model consists of both use case diagrams and textual descriptions. There is no way to avoid writing down in detail what the system is supposed to do. Use case diagrams are not a substitute for well written basic and alternate flows. In fact, the vast majority of the effort goes into the use case descriptions, not the diagram. Interestingly, the UML does not specify how the use case descriptions should be structured, organized, or written. The UML specifies the use case *diagram* details, but leaves the *textual description* style up to the use case author.

The ambiguity over how to write use case descriptions is one of the pitfalls organizations encounter when adopting use cases as part of their requirements management efforts. There are many opinions on the best way to write a good use case description including many books and papers on the topic.[i] However, the value of use cases across an organization is greatly enhanced by adopting a common style for writing the descriptions. A common style for writing use case descriptions enhances communication and improves productivity.

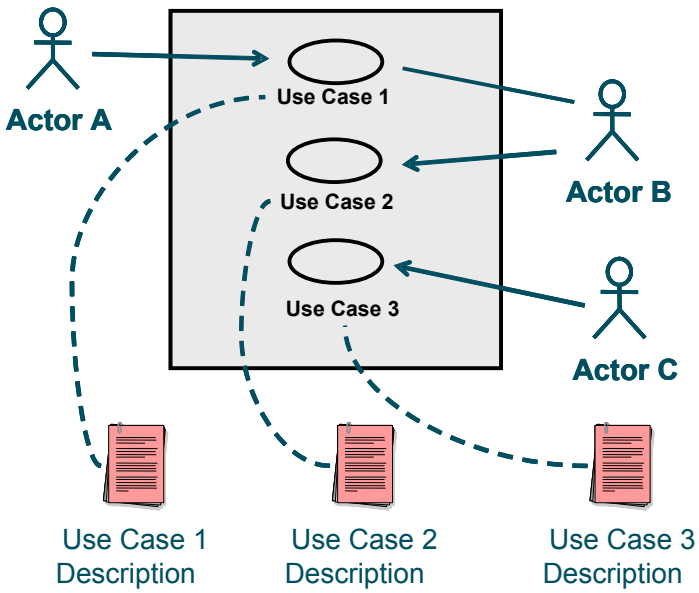

Figure 2: A complete use case model includes a diagram and textual descriptions

Another pitfall when adopting use cases is ambiguity over the relationship between use cases and the rest of the requirements for the system. The pitfalls of ambiguity over writing use case descriptions and understanding how use cases fit into the overall requirements process can be overcome by developing a Requirements Management Plan and a Use Case Style guide that clearly explains these relationships and how your organization intends to employ use cases.

Best Practice: A Requirements Management Plan

Use cases are an effective technique for capturing and communicating *functional* requirements, but *functional* requirements are only one of many requirement types. Other types of requirements are involved in developing software such as business rules, legal and regulatory requirements, usability, reliability, and performance requirements among others.

Since requirements come in many shapes and sizes, it is important to clearly identify, define, and document the types of requirements used by your organization. It is also essential to document the relationships between these types of requirement. This leads us to a best practice of requirements management, which is having a well-documented Requirements Management Plan. An organization's Requirements Management Plan is the definitive guide for eliciting, capturing, and managing of all types of requirements, not just the functional requirements in the use cases. Organizations that successfully manage requirements with use cases understand that use cases are focused on *functional* requirements, and they know how the other types of requirements relate to one another.

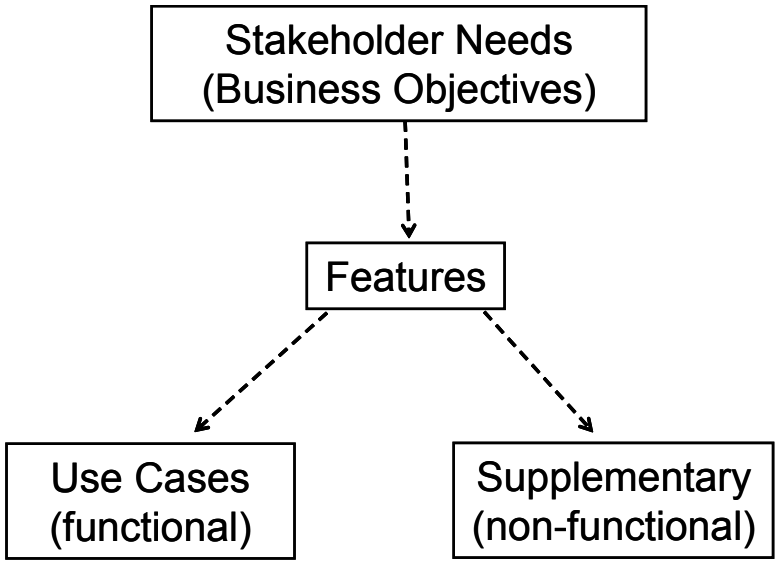

Figure 3: Relationships between types of requirements

A common mistake in adopting use cases is not clearly identifying the relationship between the functional requirements captured in the use cases and the other types of requirements managed by the organization. One symptom of this requirement type confusion is putting all types of requirements into the use case descriptions. Another symptom is neglecting or overlooking the non-functional requirements due to over emphasis on use cases being the primary means of capturing requirements.

Use cases are much more effective once your organization understands the focus and context of use cases within the larger scope of the overall requirements management process. The next step toward writing effective use cases is having a well defined Use Case Style Guide that works within the context of the Requirements Management Plan for your organization.

Best Practice: Use Case Style Guide

Use cases are not just diagrams of stick figures and ovals. Perhaps ninety percent or more of the effort goes into the textual descriptions. And, as stated above, the UML gives no guidance on writing use case descriptions. How you write a use case directly impacts how easily it facilitates design and testing. Style and technique really do matter!

A Use Case Style Guide documents your organization's approach to use case modeling. The specificity of the Style Guide will depend on the size and complexity of your organization and the variety of systems you develop. There is no "one right way" to write a use case, but obviously some styles and techniques are better than others depending on the context in which the use cases are being consumed. The objective of a Style Guide is to improve the consistency and clarity of your use cases. The Style Guide should become your repository of corporate knowledge on what works best in the context of your organization. It should include examples and a template for use case descriptions.

There is a rich body of knowledge from which you can gather ideas for your style guide, but at a minimum a good use case description should fulfill the following four objectives in order to rightfully be called a use case[ii]:

(1) The use case must provide value to a stakeholder. One of the key differences between use cases and declarative statements is the goal orientation of use cases. The use case should provide some value back to the actor or some stakeholder in order to be a valid use case.

(2) The use case must be a complete narrative describing how the value is provided. Use cases describe a "large chunk" of the system. They are the antithesis of functional decomposition. The use case should have both a main flow (sometimes called the basic flow or "happy day" scenario) and alternate flows. Any use case that doesn't have at least a few alternate flows should be suspect. It is probably too granular to rightfully be a good use case if you cannot come up with a few alternate flows to accompany the basic flow.

(3) The use case must stand alone. Use cases should not be sequenced or have dependencies. Within the UML there are some structuring constructs that allow relationships between use cases, but these should be used sparingly and generally not early in the evolution of the use case model. A use case should be something that could be delivered in a release of the software as a self contained capability that provides value back to the actor or some stakeholder.

(4) The use case must not describe design. Use cases describe "what" the system does not "how" it does it. They are a "black box" description of the system from the perspective of the actor. Your use case description template can certainly provide additional sections for special requirements such as design constraints or business rules, but the basic and alternate flows should focus on the "what" not "how."

For a given project (or organization) it is vital to choose and be consistent with a use case style for consistency, readability, and usability by the development team and all the other consumers of the use case. The four objectives above are a common denominator for all good use cases no matter what style you adopt in your Use Case Style Guide. However, this is only the start of the questions a Style Guide should answer. Here are some examples of the types of questions a Style Guide should answer in order to eliminate ambiguity over how to write a use case:

(a) Does the main flow reference other flows or not?
(b) Do steps in the flows have numbers or titles or both?
(c) Do alternative flows have numbers or titles or both?
(d) How do you reference one part of a use case from another?
(e) Can flows have embedded flows?
(f) How do alternative flows refer back to the main flow when they are done?

You will certainly find ways to improve your use cases as your organization matures and gains experience writing them, but make a conscious decision about style. Document your style in your Use Case Style Guide, and enforce the style. The Use Case Style Guide is the place to capture the collective wisdom from your organization for writing an effective use case description in the context of your organization.

Benefits of Use Cases

Since use cases came from the OO development paradigm they are generally a good fit for OO development. But, use cases are no longer limited to OO development or even development efforts using the UML. Use cases have been used with success in many types of software development and even in software deployment efforts where very little software is being developed, such as a commercial software package deployment.

Use cases have the most value for systems that have many interfaces or many types of actors. Systems with few interfaces or systems that are dominated by nonfunctional requirements obtain little benefit from use cases (e.g. a system to perform scientific calculations.) The applicability of use cases should be examined for each project based on the guidance in your organization's Requirements Management Plan.

When they do apply, well-written use cases have a variety of benefits. Good use cases improve communication by providing a context rich technique for capturing and describing functional requirements. They should be written in such a way that they are easily understood by all the system stakeholders. This ensures that everyone is in agreement with what the system is supposed to do.

Use case models help avoid scope creep by identifying the system boundaries. Actors are outside the system, and the use cases define what the system does. Although they are represented by stick figures, the "actors" can be other systems or non-human interfaces as well as human users of the system. The actors bound the system and help avoid scope creep by explicitly showing what is outside the system.

Use cases also help avoid premature design by focusing on *what* the system should do, but not *how* to do it. This allows the architects and developers the freedom of making design decisions and implementation technology choices without being unnecessarily constrained by the functional requirements. The use case technique of taking the user's perspective of the system prevents inadvertent design constraints that often sneak into functional requirements written from the system perspective.

Another benefit of use cases is they can be used to develop functional test cases. Use cases can be transformed into test cases by mapping each of the possible scenarios through the basic and alternate flows and providing test data for each of these scenarios. The development team can make sure the functional requirements of the system are reflected in the test plan by deriving functional test cases from the use cases.

Use cases are also a powerful planning instrument for incremental and iterative development. Use cases provide a cohesive handle on the functional requirements so the project manager can deliver useful capability in each release rather than disparate unrelated features. And, as the project manager schedules which use cases will be delivered in which release, the testing team will know ahead of time what test cases will be needed to effectively test that particular iteration.

Some of the Many Benefits of Use Case Modeling

- **Capture and communicate the desired behavior for the system**
- **Provide rich context for functional requirements**
- **Easily understood and facilitate agreement with the stakeholders**
- **Provide a system boundary that prevents scope creep by identifying who or what interacts with the system (actors)**
- **Avoid premature design by specifying what the system should do rather than how to do it.**
- **Lead directly to test cases (coverage and verification)**
- **Become a powerful planning instrument for iterative development**

Figure 4: Benefits of Use Cases

Conclusion

Well-written use case descriptions are a powerful technique for capturing and communicating functional requirements in many software development paradigms. In fact, many software development organizations have adopted use case techniques for their requirements management efforts on projects that are not object oriented or using any other UML constructs. Use cases also have value in deploying commercial software packages when customization and configuration is required.

If you follow the best practices of developing a Requirements Management Plan and following a Use Case Style Guide, you will gain much more value from use cases as your organization embraces a more rigorous and better-defined approach to the requirements discipline within your software development lifecycle.

[i] *Use Case Modeling*, Kurt Bittner/ Ian Spence, 0-201-70913-9
Writing Effective Use Cases, Alistair Cockburn, 0-201-70225-8
A Survey of Approaches For Describing and Formalizing Use Cases, Russell R. Hurlbut,

[ii] Based on the work of Ivar Jacobson, Kurt Bittner and others at IBM Rational Software