SearchSoftwareQuality.com

How to document use cases

Kevlin Henney

A use case represents a case of use of a system, ideally one that captures a functional requirement in terms of an identifiable and testable goal. So, what is the best way to document a use case? Approaches to content range from diagrammatic to textual, formal to free form, expansive and detailed to brief and abstract. The approaches to tool usage and authoring are just as varied. Here are some suggestions for a simple and streamlined, yet reasoned and thorough, approach to use case documentation.

First things first
In his classic paper, "Structuring Use Cases with Goals," Alistair Cockburn made the following observation:

> Use cases are wonderful but confusing. People, when asked to write them, do not know what to include or how to structure them.

Perhaps the first thing to clarify is what we mean by the term *use case*. A simple and literal reading is that a use case is simply a case of use: someone or something uses a system in a particular way. Here is how use cases were introduced in the book that originally helped to popularize them (*Object-Oriented Software Engineering* by Ivar Jacobson, Magnus Christerson, Patrik Jonsson and Gunnar Övergaard, Addison-Wesley, 1992):

> When a user uses a system, she or he will perform a behaviorally related sequence of transactions in a dialogue with the system. We call such a special sequence a use case.

This description neatly captures the essence of use cases. A set of use cases can be used to capture the functional requirements of a system by slicing it up into its externally driven uses. Instead of identifying requirements as a disconnected and potentially incoherent laundry list of features, utility and usage are the principal drivers. However, framing a system's functional requirements in terms of a sequence of actions raises a number of questions that affect how, when and by whom a use case is documented:

- How should use cases be partitioned?
- Who should be interested in and able to read use cases?
- Who should write use cases?
- Beyond requirements, what role can use cases play?

Without some answers to those questions concerning purpose and audience, we would be guilty of placing the cart before the horse if we tried to nail down how best to document a use case. Put another way, what are the requirements for this requirements approach?

The sometime fashion for plastering use case documentation with UML diagrams -- especially sequence diagrams -- is

How should use cases be partitioned?
In spite of the strong narrative feel of a *sequence of transactions* and a *dialog with a system*, and the implication of related concepts such as *scenarios* and *user stories*, documentation that focuses on use cases as scripts is tedious for both the reader and the use case's author. There is more value in thinking about use cases with respect to goals rather than narratives: a use case is a usage of a system that is intended to achieve a particular outcome. The goal lends a clear focus to what is in (and not in) a given use case.

The emphasis on the goal begins with the name: A use case should have a definite name that clearly reflects its goal, and this should take the form of an imperative. For example, *Open Account* tells you clearly and actively what the goal is, whereas

deeply misguided.

'

Account Opening is too passive and describes an ongoing state, *Accounts* describes a general heading rather than a single goal, *Open/Close/Suspend/Modify Account* describes four quite distinct goals, and *Manage Accounts* covers an indistinct number of goals, in spite of the apparent directness of its name.

Beyond a use case's name, the goal can be elaborated in more detail, but this is not an excuse to break into essay-writing mode. A short paragraph is all that is needed. Beyond this, we can identify other features that add detail and precision to a use case, ideally following an inverted-pyramid style of presentation. But to better understand what we should include and exclude, we really need to know our audience.

Who should be interested in and able to read use cases?
Who cares about what a system should do? Well, put like that, it might be easier to identify who should not be interested! A use case offers common ground for technical and non-technical people to meet and agree on what a system should do. This includes developers, project managers, customers, and so on — in other words, pretty much all of the roles that have some kind of stake or direct involvement in a system's development. What follows from this is that the documentation for a use case should be comprehensible to any of the parties involved; it should be no more technical than the domain it describes.

The sometime fashion for plastering use case documentation with UML diagrams -- especially sequence diagrams -- is deeply misguided. Such documentation has little bearing on the system's externally visible behavior, and it alienates a significant part of the potential readership. It is a waste of everyone's time. Use case descriptions should be concise, precise, readable and goal focused. They should not include parts that are optionally readable by different audiences -- these just add clutter.

Who should write use cases?
This is an interesting question and one that ties back to the audience. In principle, anyone who can read a use case should be able to write a use case. The stakeholders who want the system developed conceptually own the requirements, and it is in their interest to be involved in formulating. On the other hand, project managers, developers, business analysts and other domain experts each hold different perspectives on a system and are likely to raise different questions and make different connections. In other words, it is not simply that anyone who cares about reading a use case should be able to write it; the combined perspectives and the mixture of skills of all these parties has the potential to offer a clearer understanding of a system's requirements than the perspective of any one individual.

More information on use cases and requirements gathering

Use cases, scenarios and user goals

Meeting use case preconditions and postconditions

Five use case traps to avoid

There is, in fact, a hidden question lurking in here: How should use cases be written? If all these different roles can contribute, then what is the most appropriate vehicle for doing so? Let's first consider one of the least effective: The analyst writes all the use cases in a single document and emails this to the other parties for comments. This is typically an exercise in shared indifference that can stretch out over weeks and months with minimal involvement and correspondingly little feedback.

If you want to get people involved, then they need to be involved. If use cases offer common ground for technical and non-technical people to meet and agree on what a system should do, then they should meet and agree. A representative selection of individuals in a meeting room, armed with index cards, whiteboards and a shared goal will cover far more ground and to greater depth. How the use cases are recorded for posterity is a separate matter -- a traditional document, a wiki, the index cards they were first scribbled on -- but the simple act of collaborating is likely to make the single greatest difference to the quality of the resulting use cases.

Beyond requirements, what role can use cases play?
If you are thinking about use cases from the perspective of a sequential development life cycle, such as

the waterfall model, you are likely to restrict their role to an early phase of development. They are likely to be treated as an output of requirements gathering and an input to a more comprehensive analysis, the output of which is fed into a design phase, and so on down the line. That approach misses one of the original goals and key strengths of a use-case-driven approach: incremental development.

In observing that a use case represents a slice of a system's behavior, this is not simply a metaphor; it is a guide to how the system can be developed: a use case at a time. Use cases help to define scope and they help to define tests, especially if use cases are documented with respect to the preconditions and post-conditions that define goal fulfillment.

Not all requirements are created equally, which means that different use cases have different priorities, technical challenges and levels of risk. In looking to grow a system incrementally, it is those different aspects that need to be balanced over time. Incremental development by use case ensures that a system is developed in functionally complete slices, offering stability, certainty and visible signs of progress -- or, to be realistic and evenhanded, early signs of trouble if there are problems.

Developing by use case also allows the development to respond to new needs and discoveries as the system unfolds. Importantly, trying to document all of the use cases up front is not only unnecessary, but it is likely to be harmful. Start with a kernel of key use cases and grow from there.

About the author: Kevlin Henney is an independent consultant and trainer based in the UK. His work focuses on software architecture, patterns, development process and programming languages. He is a coauthor of *A Pattern Language for Distributed Computing* and *On Patterns and Pattern Languages*, two recent volumes in the Pattern-Oriented Software Architecture series. You may contact him at kevlin@curbralan.com.