the Rational edge
e-zine for the rational community

# Why Use Cases Are Not "Functions"

by **Kurt Bittner**
General Manager
Rational Unified Process Business Unit

*Most people go astray right from the start with use cases. Perhaps it is the similarity between use case diagrams and dataflow diagrams which leads people to define use cases that are simply functions or menu items. Whatever the reason may be, it is notably the most prevalent mistake that novices make.*

What's wrong with this picture? In simplest terms, I like to regard a use case as *a story about some way of using a system to do something useful.* Using this definition, are all of these "use cases" independently useful?

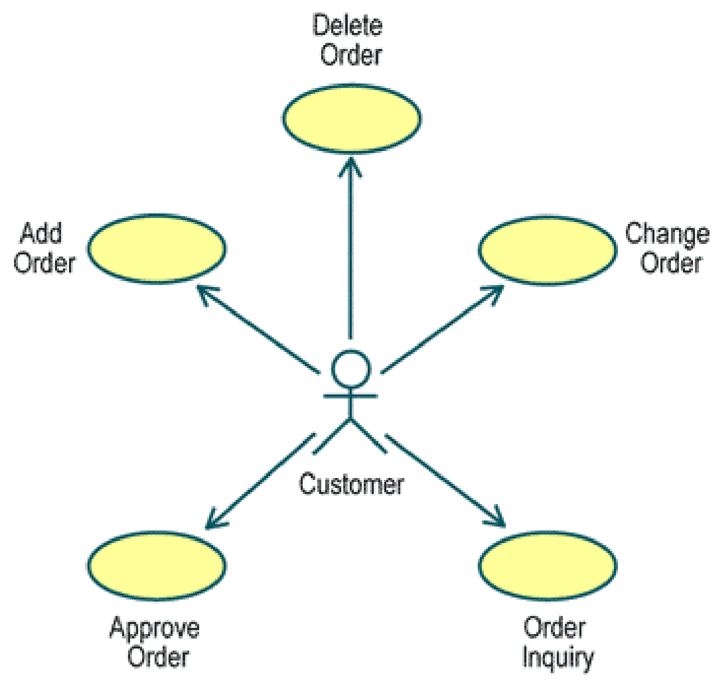

**Figure 1: The wrong way: use cases as menu options or functions**

The answer, of course, is no. In this example, the use case denotes all things that the system needs to do, but it also represents the one single thing that the customer wants to do on the system: place an order. All of the remaining elements are alternate flows in this one use case. They are steps that may be taken when placing an order. Where there is only one useful thing being done, there should only be one use case. Figure 1 is an example of functional decomposition, or (as one colleague puts it) an example of the "circled wagons" formation -- one actor at the center of a circle of use cases.

This problem is a common one. Why do people fall into this trap? We have an intrinsic need for order, and where none exists we will impose it if necessary. In the case of functional decomposition, we have a natural tendency to try to break the problem down into smaller and smaller chunks. There is a naive belief that by breaking the use cases into smaller and smaller units, we have simplified the problem. This perception is dead wrong; when we decompose the use cases, we actually compound the problem.

## Here's Why

The purpose of a use case is to describe how someone or some thing will use the system to do something that is useful to them. It describes what the system does at a conceptual level so that we can understand enough about the system to decide if the system will do the right thing or not. It enables us to form a conceptual model of the system.

Again, refer back to Figure 1. Now ask yourself, would I want to use this system to inquire into the status of an order if I had never placed an order? It's not very likely. Or would I need to change an order if I had never placed an order? No, probably not. Individually, these things are useful to me only if I have placed an order; all of them are necessary, however, to the system's ability to allow me to place an order.

Decomposing the system into smaller use cases actually *obscures* the real purpose of the system; at the extreme, we end up with lots of isolated odd bits of behavior. As a result, we can't tell *what* the system does. It's just like looking at a car that's been taken apart -- maybe you can tell that it's a car, and you know that the parts must be useful somehow, but you really can't tell how they fit together.

When working with use cases, remember that use cases are a way to think of the overall system and organize it into manageable chunks of functionality -- chunks that do something useful. To get the right set of use cases, ask yourself this question: "What are the actors really trying to do with this system?"

In case you're wondering what the improved version of Figure 1 would look like, the figure below presents the improved version:

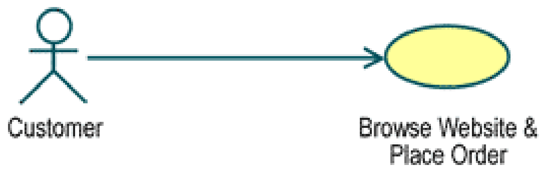

**Figure 2: A better, simpler approach: combine functions to reflect the real value to the actor**

This one use case encompasses all the "functions" that the earlier diagram split out as use cases. You may ask why this is better. The answer is simple. It focuses on the value that the customer wants from the system, not on how we subdivide and structure the functionality within the system. If you split all these functions into separate use cases, you force *your* customer (the one paying for the system) to reassemble the decomposed use cases into something meaningful to them in order to understand whether the system described is what they want (and are willing to pay for).

## Focus on Value

Lots of small use cases are a common problem, especially among teams with a strong background in (or covert sympathies for) functional decomposition. Their use case names read like a list of functions that the system will perform: "Enter Order," "Review Order," "Cancel Order," "Fulfill Order." These may not sound so bad at first, but there are more. For even a small order entry system, use case lists can run well into the hundreds. If one stays on this path, they are soon drowning in a sea of use cases, especially if it is a "really big" system. In this case, you would end up with many hundreds, maybe thousands, of use cases.

## So What's So Wrong With This?

The value of these use cases would be lost. *A use case's sole purpose is to result in some sort of value to the actor*, and at one level being able to enter an order is something of value. But if the order could never be fulfilled, would it still have value? Probably not.

Or what about entering an order and modifying the order, or perhaps canceling the order -- all of these things are related to the real thing a customer wants to do, which is to receive the goods being ordered. These actions are also all necessary to what the company wants, which is to receive payment for the goods shipped.

Another problem with a set of functions that appear to be disconnected, without any apparent relationship, is that they result in a hard-to-use system. Too many systems are like this -- they are just jumbles of features. Remember, use cases help us focus on what is really important -- the things that have real value -- and enable us to define a system around those elements. Use cases do *not* present a functionally decomposed picture of the system.

**Example**

Consider an e-commerce system that you have used on the Web. When you go to the site, your goal may be to find information about products, select products to buy, and arrange payment and shipping terms for those products. In the course of doing those things, you may change your mind, enter incorrect information and have to change it, change your mailing or shipping address, and a number of other things. If the site does not allow you to find products and order them in an appealing way, you probably won't even complete your order, let alone return to the site again.

When building systems, always refer back to the core definition of a use case: a story about some way of using the system to do something useful. If you can implement this definition to display the value that users expect to obtain from the system, and then create use cases that reflect these values, your system will better meet user expectations.

***For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!***