Writing Effective Use Cases

**Tip 1. When creating use cases, be productive without perfection**

**Tip 2. Define your use case actors**

**Tip 3. Define your "Sunny Day" Use Cases (Primary Use Cases)**

**Tip 4. Identify reuse opportunity for use cases**

**Tip 5. Create a use case index**

**Tip 6. Identify the key components of your use case**

**Tip 7. Name and briefly describe your use case**

**Tip 8. Create the use case basic flow**

**Tip 9. Create the use case alternate flows**

**Tip 10. Produce your use case document**

**Tip 11. Generate a Use Case Model Diagram**

**Tip 12. Do you need User Stories?**

**Tip 1. To write effective use cases, be productive without perfection**

When it comes to writing effective use cases, you don't need to be a perfectionist and concern yourself with getting it right the first time. Developing use cases should be looked at as an iterative process where you work and refine. You can always refine it later, so again, don't go for perfection from the get-go. Loosen up and have some fun while you're doing it.

**Tip 2. Define your use case actors**

There are possibly over a dozen actors that interact with Ebay, from buyers and sellers, down to suppliers, wholesalers, auditors, and customer service. But we're going for grass-roots, so who are the basic users of Ebay? BUYERS and SELLERS. So let's put them down as our first actors. (The visual notation in the figures below is based on UML -- Unified Markup Language for Use Cases)

Buyer  Seller

Do you notice how the actors aren't John and Sue which would be people? While John may be a seller and Sue may be a buyer, an actor is a Role. And a role in this case would be that of a buyer and that of a seller. Now that things are clicking, let's throw some more actors on your paper just so we can try and identify...
more possible users.

Now we have a bunch of actors. Wait a minute? Paypal? That's not a person. An actor can be a system, because a system plays another role in the context of your new system and has goals and interacts with other actors as you will see later.

**Tip 3. Define your Sunny Day Use Cases**

For those of you who haven't heard the expression, "Sunny Day" use cases, it is in reference to the use cases that are most likely going to occur when all goes well. These are sometimes referred to as your primary use cases. You always want to focus on the sunny day scenarios first because you can then pivot off these and figure out your "rainy day" scenarios (or edge cases) later.

Use the 80/20 rule -- if you write an exhaustive list of all possible use cases, typically 20% of the use cases will account for 80% of the activity. The other 80% of the use cases would support 20% of the activity.

In my experience in various offices, the perfectionists will say, "well what about this? isn't that possible?" referring to an edge case. The product manager should be able to discern a common use case from the edge case and prioritize accordingly. So, once you are done with your sunny-day use cases, distribute it among your project team and get consensus that you have covered them all.

**Now Collect your Rainy Day Use Cases** After you have a well-defined list of your primary use cases, you'll want to collect the list of edge cases (rainy-day) and with the help of the product manager or stakeholder, prioritize them in terms of likelihood. It should be a business question as far as how much development costs do you want to spend on something that is not likely to happen.
Tip 4. Identify reuse opportunity for use cases

In this step, you are going to cross the bridge into object modeling. Don't get overly concerned about terms like generalization, inheritance and extends. The goal of this Ebay use case example is to keep it understandable so we will explain this concept in terms of the example.

What does the word general mean? Something is broad and not as detailed. Generalization is when you "inherit" from something general and then add more detail. A "person" is very general. A "man" is still general, but not as general as a "person". You can say that a "man" inherits behavior and attributes of a "person".

Look at the requirements management use case diagram above and you will see there is duplicate behavior in both the buyer and seller which includes "create an account" and "search listings". Rather than have all of this duplication, we will have a more general user that has this behavior and then the actors will "inherit" this behavior from the new user.
The above use case example diagram illustrates that a generic user creates accounts and search listings and that a buyer and a seller have their own behavior but also have the behavior of the generic user. The benefits of generalization are that you eliminate duplicate behavior and attributes that will ultimately make the system more understandable and flexible. We will see in later steps that this inheritance applies both to use cases and to the actors.

**Tip 5. Create a use case index**

After producing your initial visual list of use case actors and goals, we can take this list and create an initial use case grid which provides the basis for the use case index. Every use case will have various attributes relating both to the use case itself and to the project. At the project level, these attributes include scope, complexity, status and priority.

<table>
<thead>
<tr>
<th>Use Case ID</th>
<th>Use Case Name</th>
<th>Primary Actor</th>
<th>Scope</th>
<th>Complexity</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Places a bid</td>
<td>Buyer</td>
<td>In</td>
<td>High</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Purchase an item</td>
<td>Buyer</td>
<td>In</td>
<td>High</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Creates Account</td>
<td>Generic User</td>
<td>In</td>
<td>Med</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Searches listings</td>
<td>Generic User</td>
<td>In</td>
<td>Med</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Provides Feedback</td>
<td>Generic User</td>
<td>In</td>
<td>High</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Creates an auction</td>
<td>Seller</td>
<td>In</td>
<td>High</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Ships an item</td>
<td>Seller</td>
<td>In</td>
<td>High</td>
<td>2</td>
</tr>
</tbody>
</table>

This use case index should be used by the project team to define the use cases against. It will serve as a master inventory to help write effective use cases for the requirements phase of the project.
Tip 6. Identify the key components of your use case

The actual use case is a textual representation illustrating a sequence of events. In our use case example, you will see that there are several components of a use case which we will review. In the mean time, review the table below to get a basic understanding of what is in the use case and then we will review each element as we progress through our use case example.

<table>
<thead>
<tr>
<th>Use Case Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Number</td>
<td>ID to represent your use case</td>
</tr>
<tr>
<td>Application</td>
<td>What system or application does this pertain to</td>
</tr>
<tr>
<td>Use Case Name</td>
<td>The name of your use case, keep it short and sweet</td>
</tr>
<tr>
<td>Use Case Description</td>
<td>Elaborate more on the name, in paragraph form.</td>
</tr>
<tr>
<td>Primary Actor</td>
<td>Who is the main actor that this use case represents</td>
</tr>
<tr>
<td>Precondition</td>
<td>What preconditions must be met before this use case can start</td>
</tr>
<tr>
<td>Trigger</td>
<td>What event triggers this use case</td>
</tr>
<tr>
<td>Basic Flow</td>
<td>The basic flow should be the events of the use case when everything is perfect; there are no errors, no exceptions. This is the &quot;happy day scenario&quot;. The exceptions will be handled in the &quot;Alternate Flows&quot; section.</td>
</tr>
<tr>
<td>Alternate Flows</td>
<td>The most significant alternatives and exceptions</td>
</tr>
</tbody>
</table>

Tip 7. Name and briefly describe your use case

Now that you have a general understanding of what a use case consists of, we are ready to start creating our use case. Typically, while the name of your use case is being discussed, people will start briefly describing the use case. Use plain english and keep it simple. Getting back to our use case example, I will begin with use case #1 from step number four.

Use Case Number: 1
Use Case Name: Buyer Places a Bid
Description: An EBAY buyer has identified an item they wish to buy, so they will place a bid for an item with the intent of winning the auction and paying for the item.

Tip 8. Create the use case basic flow

The basic flow of a use case represents the most important course of events or what happens most of the time, sometimes referred to as the 'Happy Day Scenario' because it is what occurs when everything goes well -- no errors or exceptions. Another reason why the basic flow is so critical is because it's much easier to fully comprehend the exceptions once the norm is understood and if the basic flow represents 70% of the system, the development staff is much more prone to implementing the correct code in the first pass.
For our use case example, the basic flow should be to describe the happy day scenario for your use cases such as "placing a bid". For a consumer to play a successful bid, what is the primary flow when everything goes as planned. An effective use cases needs to have the basic flow before moving forward with writing the alternate flows.

**Tip 9. Create the use case alternate flows**

The basic flow is the key ingredient to your use case and some can argue that you can stop once you're done with the basic flow. It really depends on the level of detail you wish to achieve. However, providing more detail to the consumers of your use case is always a good thing.

The alternate flows providing the following:

- An exception or error flow to any line item in your basic flow
- An additional flow, not necessarily error based, but a flow that COULD happen

A few examples of alternate flows are:

- While a customer places an order, their credit card failed
- While a customer places an order, their user session times out
- While a customer uses an ATM machine, the machine runs out of receipts and needs to warn the customer

**Tip 10. Produce your effective use case document**

Recently at a new project assignment, I introduced a mid level developer to the concept of use cases which was totally foreign to him. Once walking him through the basic concepts and showing him the use case example, the lightbulb went off in his head on how convenient and simple it was to grasp the project.

A few reasons why it's that much easier to learn a system through use cases then a traditional requirements document is probably because with use cases, you are introduced to concepts at a high level, walk through a living scenario and then presented with specifications last.

In several places in this document, I have stated "effective use cases" rather than just "use cases". The purpose of the use cases is for effective knowledge transfer from the domain expert to the software developer -- these use cases will serve as software requirements. If they don't make sense to the person building the software, they are not effective. There are several sources on the web for writing effective use cases including the book by Alistair Cockburn.

**Tip 11. Generate a Use Case Model Diagram**

You can use the Gatherspace.com use case modeling tool to produce a use case model within a few clicks. Once you define your use cases and actors, just go into the reporting section and click on the 'Use Case Model' report and that's it. From the main use case model, you can continue to drill down into the use cases.

To see what this looks like, click the [use case model](#) sample now.

In several places in this document, I have stated "effective use cases" rather than just "use cases". The
The purpose of the use cases is for effective knowledge transfer from the domain expert to the software developer -- these use cases will serve as software requirements. If they don't make sense to the person building the software, they are not effective. There are several sources on the web for writing effective use cases including the book by Alistair Cockburn.

**Tip 12. What's the difference between a User Story and a Use Case?**

With so many engineering teams making the paradigm shift from waterfall to Agile, people often get caught up in having a pure Agile process which would include the use of User Stories. So what's all of the hoopla with User Stories? What are they, how are they different from use cases, do I need them, and where do they fit in the process?

- **What is a User Story?** Simply put, written from the context of the user as a simple statement about their feature need. They should generally have this format. "As a -role-, I want -goal/desire- so that -benefit-"

- **How is a User Story different than a Use Case?** While a use case is highly structured and tells a story, the User Story sets the stage by stating the need. A User Story is the prelude to the use case by stating the need before the use case tells the story.

- **How does the User Story fit into the process?** User Stories are great as an activity in collecting and prioritizing the high level features. Getting this initial feedback from the customer is a simple way of trying to get all of their needs identified and prioritized. The User Stories will then morph themselves into the business requirements and use cases.