

# Making "Useful" Use Cases

- Chris Gieger: Director, UI Design Practice – Computer Horizons Corp

**Summary:** This article discusses the problems with use cases and how they could be made more useful.

## The Problem with Use Cases

Use cases have become a staple in many software development methodologies, yet many development teams do not write use cases that provide enough value to their intended readers. Software engineering expert Ivar Jacobson originally documented use cases in his book "Object-Oriented Software Engineering: A Use Case Driven Approach" as "a sequence of transactions in a system whose task is to yield a measurable value to an individual actor of the system." The problem is that over the years too many different use case styles and formats have emerged, which has led to a general lack of consistency and purpose for their original intent. Use case writers are now mixing "concrete use cases" with "essential use cases" with "narrative use cases", etc. In addition, many use cases are also often written with too many words and not enough meaning for their most important readers – the client.

I have been involved with numerous projects where the Business Analysts and Development teams spent several weeks (even months) writing hundreds of use cases only to find that the client couldn't understand them. In fact, often times the uses cases (usually kept in huge 3-ring binders) were never used – even by the developers. Rarely do clients need to (or want to) fully understand each and every system response documented in use cases. What they usually want is to "see" the interface that will provide them or their users with the functionality that they want to have built. The bottom line is, if the client is footing the bill for the project then they will most likely want to approve each of the deliverables. If the client cannot understand the deliverables, chances are they won't approve them.

**Tip:** Show your client some annotated screen mockups or wireframes and you'll see a client that can quickly and easily understand its intended functionality.

### Real World Experience:

I once had a client tell me that he was able to define more requirements from a 2-hour presentation of preliminary UI mockups than he was from 4 days of Joint Application Development (JAD) sessions. The reason he stated this was obvious – he could visualize the elements of the interface and therefore define how they should function. He was also able to then realize what was missing, what he hadn't thought of during the JAD sessions, and prioritize what functionality should be built for the initial release.

## Use Cases Should Never Define the User Interface

Use cases are intentionally focused on the system needs, not the user's needs. Yet, use cases often blur that line and include UI requirements without any input from the end-users. This seems to be an area where many software development projects begin heading the down the path to an end-product that may function properly but is not usable, sellable or marketable to its end-users. Use cases are inherently too system-oriented to determine UI requirements, thus use cases should never define or be used as UI requirements. Ideally, an Information Architect and/or UI designer capture User Interface requirements with input from end-users at the very beginning of the development process.

Often times use cases will be written to include things that are not appropriate for their purpose, such as:

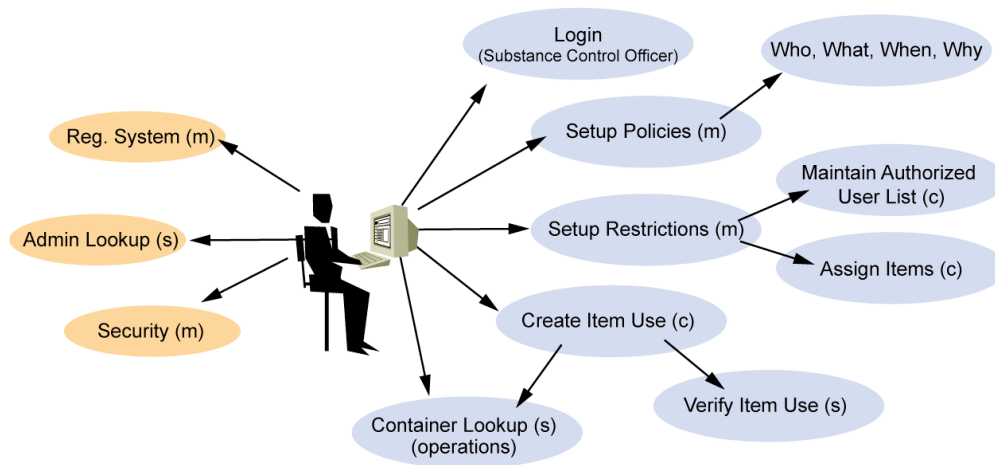
- Use cases often define user task flows based on system or functionality needs instead of involving end-users to help determine what flow would be best for them.
- Label names are often stated in use cases that end up becoming concrete without testing with users first. For example, a use case should never define a label name such as: "user selects 'Submit'". Instead, the use case

should simply state the system need, such as: “system stores data to database”. The user should help determine the best label name for the UI with the help of an IA or UI designer.

- Development teams will often feel that the use cases tell them enough information to build the UI and so they proceed without the help of UI designer until the system is built and tested—at which point they will look to a design team to “dress up” the interface. I sometimes referred to this as “putting lipstick on a pig” or making a bad UI simply look aesthetically better.

## A Picture Tells a Thousand Words

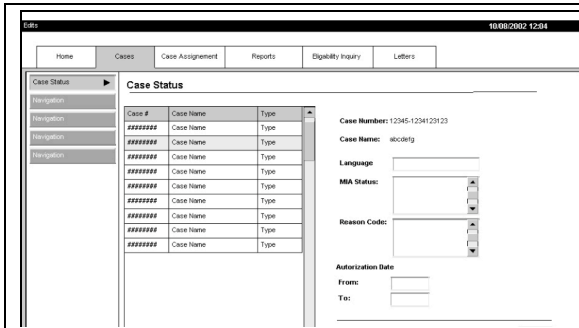
The best possible use case is one that helps all its intended readers understand what it is trying to communicate so that all project team members and the client are on the “same page”. In the high-level use case diagram below, all the major pieces of system functionality are captured.



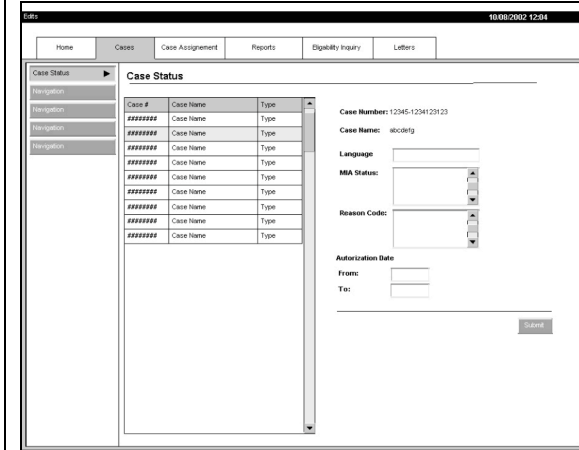
In the use case scenario template example provided below, the following key elements are used to help the reader understand the use case:

- A Use Case Title that clearly labels the use case.
- Short and clearly segmented topics and content on the right so the reader can quickly scan the use case and understand its intent. Note: the usual term “Actor” is purposely replaced by “User”.
- Visual wireframe layouts on the left to help the reader visualize and follow the use case scenario and understand how the words apply to an actual interface – even if it won’t be the final UI.

<Use Case Title>	
	<b>Summary:</b> <A short blurb of the scenario>
	<b>User(s):</b> <Identify each potential user type>
	<b>Preconditions:</b> <what did the user or system do before this scenario>
	<b>Basic Sequence:</b> 1. <action taken by the user(s)> 2. <action taken by the user(s)>



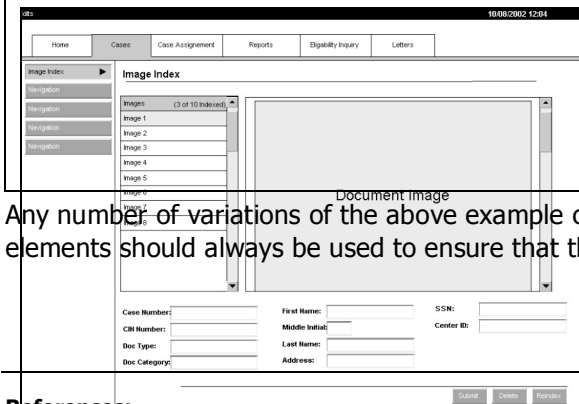
**Summary:**  
 <A short blurb of the scenario>



**Exceptions:**  
 <steps that don't follow the flow of intended scenario but can still occur>

**Post Conditions:**  
 <what will the user do or be presented with once they have completed this scenario>

(wireframe 1a)



Any number of variations of the above example can be applied to the unique needs of the project but the key elements should always be used to ensure that the use case is easily understood.

**References:**

- Ivar Jacobson - "Object-Oriented Software Engineering: A Use Case Driven Approach"
- Constantine & Lockwood, Ltd - "Structure and Style in Use Cases"
- Simon Ståhl, Matti Kauppinen and Samma Rekola - "Bridging the Gap Between User Needs and User Requirements"