An Ivar Jacobson Consulting White Paper

IVAR JACOBSON CONSULTING

www.ivarjacobson.com

Integrating Use Cases, Storyboarding and Prototyping

By Kurt Bittner, CTO Americas

January 2008

Table of Contents

Introduction

I am often asked whether teams should use use cases, storyboarding or prototyping to capture requirements, and my usual answer is "Yes!" These techniques each have something unique to offer, and when used together appropriately they are highly complementary. The key here is the word *appropriately* - without some guidance the techniques can get in each other's way, and at the very least can be confusing. In this article I will share the combined approach to using these techniques together that I have found most successful in practice. But first a little background.

Strengths and weaknesses of the three approaches

To start the discussion, it's important to remember the goal of any requirements approach: to drive effective discussions about what the system should do, with the result that everyone - the team, management, and stakeholders - come to a shared understanding of what is going to get built, *and* to continue these discussions throughout the project so that the right trade-offs can be made when decisions must be made. A technique is good if it encourages open and frank discussions, and it is bad if everyone is simply going through the motions. There is no "magic" in any approach - they are just different representation techniques - so if you are not having the right kind of discussions with the right people, your results will fall short of expectations. Garbage in, garbage out.

Strengths and weaknesses of Use Cases

So what are use cases good for? My opinion is that they are very good for capturing flows of behavior - i.e. first one thing happens, then another, then another, and so on. A *flow* is inherently sequential, things happen in a specific order. There can be branches in the flow, but there is still a flow. The structure of the use case description helps to organize the branches and alternatives, and when done well comprehensibility is improved.

So what are use cases not particularly good for, or when is a different approach more appropriate? I usually think of two things when this question arises: state-driven behavior, and CRUD. In a system whose behavior is state-driven, often real-time systems, events can occur in any order. The system's response to these events depends on the events that have already occurred, which have resulted in the system being in a particular *state* or condition. While it is not impossible to represent state-driven behavior in a use case, why bother? There is already a well-established technique for dealing with these sort of problems called a *state-machine diagram*. Because a state machine has no real "flow", it is difficult to represent the behavior as a linear sequence of steps, and the strengths of the use case approach begin to break down and I've found it more expedient to use other techniques.

The exception to this rule of thumb is the case where the behavior is state-driven, but where the responses to the event is really a flow unto itself, where there are a series of steps that need to be undertaken. Then you can use a state machine diagram to organize the use case into flows that occur when the event is detected when the system is in a particular state.

The second thing I don't find use cases very useful for is what I call CRUD, which is short for Create, Read, Update, Delete behavior. As I mentioned before, I think the use case technique is useful for describing things that have a flow, but when the behavior is simply entering data, validating data, storing data and simple retrieval of data (usually for editing), the resulting use cases are not very interesting and are probably not the best way of describing the desired behavior. A logical data model that describes the data and associated validation rules, coupled with prototyping tools that can let you explore the look and feel of the editing experience tend to be far more useful for achieving good results.

Strengths and weaknesses of Storyboarding and Prototyping

Storyboarding and prototyping can be thought of as more or less the same kind of thing - they focus on creating a visualization of the user interface that implements some set of behaviors. A storyboard tends to be more conceptual and still retains most of the flow information, but unlike a use case a storyboard is usually focused on just one path through the behavior of interest, while a use case captures all possible paths. Storyboards are often used as early, informal, prototypes that evolve into richer executable prototypes over time.

With a prototype, it's harder to see the flow information. Some prototyping tools capture some of the flow information for simulation purposes, but this information is usually not a complete substitute for having a use-case description. Most often I find that the use case is useful as a kind of "script" for walking through a prototype.

Prototypes are especially useful for capturing requirements related to the look and feel of the solution - details that will clog a use-case description with unnecessary information, rendering it unreadable.

So each of these different forms - use cases, storyboards, and prototypes - are useful for certain kinds of information, and are not as useful for others. Taken together they are useful techniques to have in your skill set as you work on the requirements of a system.

An analogy from the film industry

Taking a cue from Hollywood (or Bollywood, if that suits your tastes better) we can see how different techniques similar to the ones described above can be used together. The analogue of the use case model on a movie set is the script - it describes all the scenes (scenarios) of the movie, and the dialogue, and some of the set directions, but it does not discuss "implementation" details such as lighting, camera angles, and other things left up to the film crew.

As detailed as the script is, it is usually not sufficient to work out details such as the look and feel of the movie, as well as various aspects of set layout, camera angle, and other technical aspects. For this they use a storyboard, rough sketches of how the scenes will look when filmed. Storyboards are also useful for communicating among all members of the cast and crew certain critical ideas about how the movie will work. The movie storyboard has a similar roles to the use-case storyboard in software development - the use-case storyboard is a way to explore various aspects of look and feel without spending a lot of time and money on developing something. The advent of easy to use prototyping tools may seem to remove some of the need for use-case storyboarding, but my perspective is that the prototyping tools simply make it easier to create storyboards quickly.

On a film, crews often shoot some early footage to prove out the ideas in the storyboards. Sometimes they find that what they thought would look right does not actually work in reality. Prototypes on software development projects play a similar role, and they are a natural outgrowth of the storyboarding work. But just as the early footage is not a substitute for a script (it's been tried, usually with dismal results), prototypes are usually not a substitute for having use-case descriptions if the details of the *flow* are significant or complex in their structure.

How to use Use Cases, Storyboards and Prototyping together

On a typical project, the pattern of work goes something like this:

The project kicks off with a brainstorming session to establish goals and desired outcomes for the project. This often takes a few discussions, and also can often benefit from having a skilled facilitator leading the discussions to keep the extended team focused.

Once desired outcomes are known, it's useful to have a use-case workshop to brainstorm what the system is going to have to do do deliver the desired outcomes. The result of this is a use-case model, with brief descriptions of the use cases that articulate the desired outcomes the use cases will produce. It is also a natural result of this workshop to start early drafts of outlines of flows of events of the use cases, as well as informal sketches of user interfaces and a start on the use-case storyboards. If this seems like a lot of work, keep in mind that everything is really just an early working version at this point, just enough to drive ongoing discussion.

From here the work continues to evolve in parallel. Use cases that are mostly CRUD are prototyped, reviewed with stakeholders and further evolved, and little or no work occurs on the actual use-case description itself. Use cases with significant *flow* behavior are storyboarded and discussed with stakeholders, all the while keeping the use-case description up to date. At some point the storyboards will evolve into working prototypes, but the use-case description should not be ignored as it will serve as useful input into test-case creation as well as providing useful user documentation.

At some point the prototyping effort will evolve into writing real code, at which point the use cases will serve mostly as reference on how things were supposed to work. Throughout this process there should be more or less continuous informal reviews and feedback sessions, but don't worry about sign-off - do this at iteration and phase boundaries. Remember - the goal is to get a working system, not to get "requirements signed-off".

Working Iteratively

The most effective way to work after the initial use-case workshop early in the project is to break the work into a series of iterations, with each iteration tackling a set of scenarios. A scenario is a subset of a use case comprised of the *basic flow* plus zero or more *alternative flows*. How to choose scenarios for iterations is a subject covered in some depth in "Managing Iterative Software Development Projects" (see notes below) so I won't cover it here, but the general principle is to choose the more technically challenging scenarios early so as to force the rapid evolution of a stable architecture.

Within an iteration, you will take the rough outline of the scenario (if one was created as an outcome of the use-case workshop) and work out the details of the flow of events, usually working side-by-side with a subject matter expert from the line of business. As you work out these details, it is often useful to create sketches of the user interface in the form of a storyboard, and walk through the scenario using these storyboards. The interaction of walking through the scenario using the storyboards will help to hone the flow of events description as you are forced to think about the overall user experience. As the description evolves, the storyboards will evolve into working prototypes, and these prototypes will evolve into code. The result of the iteration will be, hopefully, a number of completed and tested scenarios.

The general way of working should be clear: you do not write complete use cases early in the project, then storyboard, then prototype, then code. There is a natural and reinforcing parallelism between working on the use-case descriptions, storyboarding, and prototyping, and these activities flow together seamlessly during an iteration.

Conclusion

Use cases, storyboarding and prototyping are all useful techniques for eliciting and documenting requirements. Separately, each has its own weaknesses, but together they provide a powerful combination of techniques for working with requirements. Knowing how to do this is the trick, but working scenario-by-scenario, iteration by iteration, the application of the techniques becomes quite easy and natural, and your end results will be much improved.

About the Author

Kurt Bittner is CTO-Americas for Ivar Jacobson Consulting. He is the author of numerous articles on software engineering, and is the co-author with Ian Spence of "Use Case Modeling", published by Addison-Wesley in 2002, and "Managing Iterative Software Development Projects", published by Addison-Wesley in 2006. His industry experience spans more than twenty-four years of successfully applied iterative approaches to delivering software solutions in a number of industries and problem domains, and he is a past contributor to the Rational Unified Process.

About Ivar Jacobson Consulting

Regional Office	**Email**	**Phone**
Americas	us-enquiry@ivarjacobson.com	978-649-2856
Australia	info@ivarjacobson.com.au	+ 61 2 9994 8993
China	info@ivarjacobson.com	+86-10-62091361
Scandinavia	info@ivarjacobson.com	+46 (0)8 501 64 170
Singapore	info@ivarjacobson.com	+65 9772 3538
United Kingdom	info@ivarjacobson.com	+44 (0)20 7025 8070