

## **Form Feeds Function**

### **The role of storyboards in requirements elicitation**

from The Rational Edge: Among iterative development practitioners, a sharp debate centers on the merits of use cases versus storyboards as techniques for eliciting user requirements. This article explores how user interaction with the details of a storyboard can lead to helpful discovery regarding functional requirements.

*Functionality is an important -- some would say the most important -- success factor for an IT system. Users expect IT systems to help them perform their jobs easier, faster, and more reliably. Users interact with a system via a human-computer interface (HCI), and, with the emergence of the graphical user interface (GUI) in the early 1990s, many users have become more articulate about their needs and how they want to interact with the system. Focusing on HCI in software engineering raises two related issues that need to be resolved: a) users are commonly not usability experts and b) software engineers need to identify and build the functionality behind the HCI and are typically less concerned about the usability of the HCI itself. This article will introduce a technique that helps identify the functionality for a system while still allowing users to visualize and think in HCI.*

*Note: For many readers, the difference between a GUI and the more general HCI concept may be a bit fuzzy. But for a variety of reasons (e.g., performance, size), not all IT systems provide a graphical user interface: for example, systems that offer command or voice-based interfaces. The technique presented in this article happens to use a GUI-based example, but a GUI is only one of many possible ways that humans and computers might interact. (See, for instance, this issue's cover story on building accessible applications for users with physical disabilities.) For that reason, I decided to distinguish between these two concepts, HCI and GUI. Bottom line: There are many forms of HCI, a GUI being one example.*

### **Functionality and human nature**

In a GUI, a logical set of functionality is grouped and made visible in a logical visual flow so that users can do their job without being overloaded with unnecessary, unwanted information. When a user starts an application, all they see is a window, command line, Webpage, or some other form of HCI that allows them to interact with a system. That might suggest to a user that the HCI *is* the system, because the HCI covers and hides all technical details behind a façade. For example, a user of a system clicks on a button and expects that the request associated with this button will be executed without requiring his or her understanding the system internals.

The difference between users' expectations and the design concerns of a software engineer became clear to me some years ago when, together with users, I developed a visual prototype for a new system. I did my homework, and a few days later I came back with a prototype in which I proposed the business flow mapped to a GUI.

## Form Feeds Function

My users were so excited about the prototype that they asked me, "When can we have the system?" You can imagine how disappointed everyone was after I told them that the screens were filled with static data hard-wired into the screens, without any functionality programmed or databases linked to them. It became clear to me that my users anticipated the HCI as a working system. But the actual business logic, typically represented in an object model, had not yet been developed and hooked to the GUI, and it was difficult to explain that only the "invisible" object model would bring the HCI to life.

In another situation, the users told me how they wanted to interact with a system and drew actual windows. The windows ended up overloaded with information and simple entry fields typically seen on mainframes.

But even a fabulous HCI does not entertain users for a long time. Soon questions like, "How can I generate a receipt for this order?" or "Where does the manager approve orders greater than US\$5,000?" arise. It is very challenging for end-users to picture a process-flow without HCI suggestions. That might be one reason why they brainstorm the functionality after a system is developed. At that point, the original requirements on paper have been transformed into software, and users at last experience the software as a real, rather than theoretical, system and express their real impressions of the system experience.

This particular risk -- functionality brainstorms occurring late in the development cycle -- can of course be reduced through iterative development practice.

Another reason, particularly for business IT systems, is that a software application is seen as a tool to make business workflows more efficient. Curious users turn into a jury to assess the benefits and impact very carefully. For example, why do users still have calculators next to their computers on their desk? One reason might be that the IT system in front of them does not provide quick and easy calculations. In scenarios like that, the functionality and the HCI did not convince users to give up their original previous workflow facilitator, in this case, the calculator.

It depends on who you talk to, but in my experience, end-users usually do not think that building IT systems is exciting. They know they need a new IT system and play along with the software engineers as it's being developed, but they are less enthused about the prospect of conducting requirements workshops and design review sessions. They are rarely specialists in requirements gathering and are most likely not interested in learning the IT-lingo. And why should they? It is the responsibility of business analysts and software engineers to manage the set of user needs and speak the user's language.

## Use cases and storyboards

Tools are available to the software engineer to help manage these user needs. One good example is the Unified Modeling Language (UML), which is commonly used to depict all sorts of artifacts in software engineering. Of course it would be nice for software engineers to receive documents written in *their* language, but this is unrealistic. Along with the UML, another successful approach in bridging the language barrier has been iterative, incremental development, where users have a chance to direct the efforts and provide feedback on functionality developed in small intervals.

Furthermore, use cases and storyboards, combined with iterative-incremental development, can help jump-start software projects, bringing missing requirements to light early on. The most important thing before starting to write use cases is getting clear about the goals and the audience of these documents.

Let's explore use cases and storyboards and consider their distinct roles in requirements analysis.

## Use cases

Among software engineers, use cases have become a very popular approach to capturing functional requirements. The scenario-driven approach supported by use cases allows engineers to focus on nicely scoped portions of requirements. The same scenarios can then be used by members of the quality assurance team to test the implemented stories. As a positive side effect, even project managers and end-users can monitor the progress of a project and provide essential feedback to the project team. This form of use case is meant to capture the *essence* of a particular business scenario. Instead of getting distracted by technical details and discussions, the use-case format focuses on capturing the interaction between one or more actors and the system. From looking at a use case, we learn what the actor wants to accomplish and what the steps are between start and achievement. In a simplified picture of a layered architecture, as shown in Figure 1, the use cases usually collect the requirements for the middle layer providing the functionality of a system (class and object model).

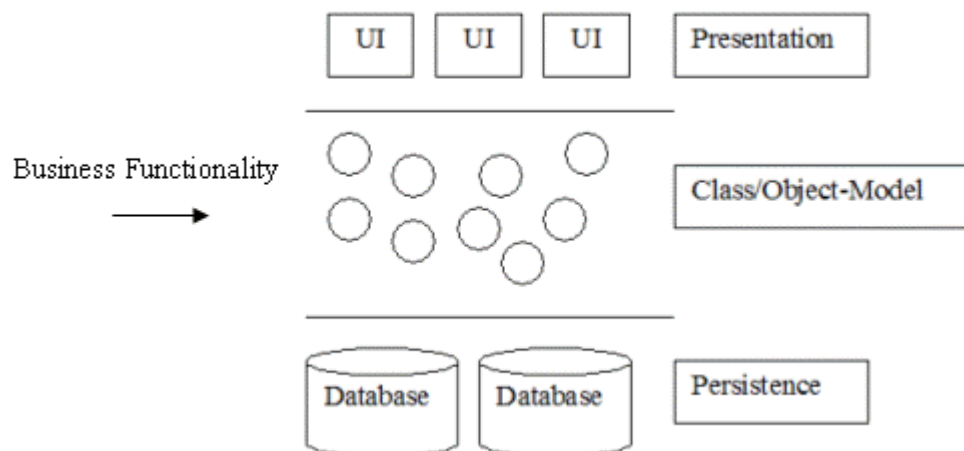


Figure 1: Target layer for use cases in a layered architecture

Because the use case describes the scenario without technical (hardware or software) commitment, the engineers building the middle layer where the business logic resides can focus on the functionality described in the flow of events. Another advantage of use-case style is that the scenario is written in natural language and users can review and discuss the implications. Using the same vocabulary as the users, the engineer can now perform linguistic analyses and convert the requirements, piece by piece, into software. Once the project members agree to an essential use case, it can serve as a contract between business and IT.

A disadvantage, though, is that essential use cases can be written on different abstraction levels; the user who verifies the scenario needs to map the scenario to his world. That puts some distance between the user and the engineer. Another downside of this approach is that nonfunctional requirements might be ignored.

## Storyboards

Use cases are like the general plot of a film, whereas storyboards are the script. In contrast to use cases, these storyboards now make use of very real and concrete concepts. That includes examples of user interfaces and verbiage about using hardware. This approach can go even further by personalizing stories (giving actor names) or adding cartoon-like elements to use cases. The creativity of the author is the only limit for the content, because no standard for storyboards exists. Storyboards can be put together by one person in electronic format, or they can be crafted by a team on whiteboards using markers, sticky notes, or other media. The workshop outcomes can be scanned or photographed to preserve and share the results.

Because storyboards exist independently of the software system they describe, they have many advantages over regular prototypes. They cannot crash, are very easy to share with large groups, and do not give the false impression that the system is already developed. Additionally, feedback is easier to accommodate.

However, one of the biggest problems with storyboards is that they can become outdated very quickly. User interfaces originally defined often change over time, and that creates a maintenance burden.

## Does form only follow function?

The architect Louis Henry Sullivan (1856-1924) coined the phrase "form follows function," a concept developed by the Bauhaus movement in the 1920s. The idea was to focus entirely on the functionality in a given design and derive the appearance and form from the functional concern. Adolf Loos (1870-1933) extended the concept with his more judgmental notion of "ornament and crime." These design theories were not limited to buildings only, but were also applied to designs for furniture and other pop-culture items. For example, the form of a jet airliner is very much derived from its functionality, both inside and out.

In the software development industry, the form-follows-function design concept is comparable to the concept of use cases. The "plot" of the user interaction with a system is less likely to change than the technology that supports it. We can build the entire business layer from essential use cases and eventually derive user interfaces from it, which drive the configuration and aesthetics of the HCI. So the question is: "Are storyboards necessary?"

Well, it depends on the audience and goals. There is little question that use cases have a bigger impact on software engineering than storyboards. Use cases are the origin and basis for a well-defined object model that not only manages the functionality and business rules of a system, but also serves more than one HCI (for example, checking cellphone minutes over a a Webpage or the cellphone itself).

The importance of a solid business layer is therefore crucial and of highest importance in software engineering, which reflects the use case. The HCI, on the other hand, needs to provide and show the necessary information from the business layer and contains clues about data and functionality, which reflects the storyboard. It is neither the purpose of a storyboard to help development teams "sign-off" a proposed GUI with the client nor to assure GUI compliance concerns. For example, various books about visual standards in HCIs exist. One sample user interface and a book reference can easily direct a conversation about look-and-feel discussions. It is not always beneficial to depict each individual user interface if nothing new is learned from it.

But storyboards *can* serve a niche in software engineering that I believe is often overlooked. Within this niche, they can be extremely powerful when applied in the right context. Storyboards, like HCIs and GUIs, communicate design notions more clearly to users than use cases alone can. They give users real content that is easy to digest. For some industries, for example, the electronic gaming industry, where HCIs are crucial for the success of the system, a storyboard gets even more attention.

In a sense, the primary beneficiary of the storyboard document is not the users, but we the software engineers. Imagine that, somewhere in a use-case description, it says: "The accountant enters the billing number." In reviewing this use case, our end-user accountant might simply nod in agreement, recognizing that the use case captures an essential element in the billing process. But in the more detailed storyboard, we might add a simplified dialog box to the statement, as depicted in Figure 2.

## Form Feeds Function

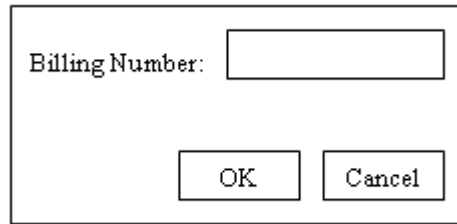


Figure 2: Sample user interface

This sample GUI makes the billing number more obvious for the accountant and directs the attention to this aspect of the story. Now, imagine that in reviewing this visual storyboard, our end-user accountant says, "Well, actually, billing numbers are divided into two parts, the year and a unique number. This drawing shows only one field for the account number." Then he adds: "And by the way, I don't want to enter the year all the time, so please initialize this value with the current year, which I can overwrite if necessary."

Two things have happened in the scenario just described. Not only did we receive feedback about the HCI, but we also learned about the business logic and functionality: i.e., in this business, accountants archive bills by year. With this little mock-up of a screen, we actually gained new functional requirements. Looking back at our layered architecture example in Figure 1, we now see how storyboards not only serve to create the presentation layer, but also the business layer. True, form follows function; but as we've seen here, form *feeds* function, too. Storyboards provide the means for validating the HCI and can serve as a valuable requirements elicitation technique. Therefore, storyboards can target two distinct layers in a software architecture, the presentation layer and a middle layer that represents the object model, as shown in Figure 3.

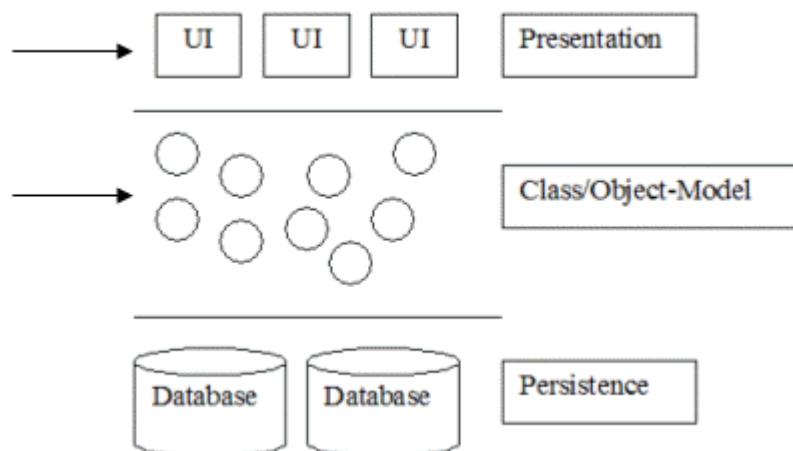


Figure 3: Target layers for storyboards in a layered architecture

"Form feeds function" is especially true for software engineering where iterative-incremental development is applicable.

To lessen the maintenance burden of keeping storyboards up-to-date (one of the drawbacks I noted earlier), many of them can be destroyed after the results have been recorded via the appropriate use case(s).

## Conclusion

Storyboards are fun to create and well-liked by the most important participant in the software engineering process, the customer. Using storyboards as a requirements elicitation technique is often underestimated. They provide a vital opportunity to discover potential new functional requirements much earlier. Storyboards can also be used selectively to explore one area of a future system or to validate existing functionality. That benefits scope, schedule, and costs in the earlier phases of a project. During my career, I have been party to many heated discussions about whether or not use cases should contain screenshots. I hope that this article clarified the role of storyboards in software engineering and presented new perspectives that might help the next time you need to identify requirements.

## Additional reading

Alistair Cockburn, *Writing Effective Use Cases*. Addison-Wesley 2000.

Suzanne Robertson and James Robertson, *Mastering the Requirements Process*. Addison-Wesley 2000.

Donald C. Gause and Gerald M. Weinberg, *Exploring Requirements: Quality Before Design*. Dorset Publishing House 1989.

Geri Schneider and Jason P. Winters, *Applying Use Cases*. Addison-Wesley 2001.

Hugh Beyer, *Contextual Design*. Elsevier 1997.

## About the author



Jochen (Joe) Krebs (<http://www.jochenkrebs.com>) is a senior IT specialist for the Rational software brand within IBM Software Group. He is responsible for successful enablement of Rational products and services for clients in the financial sector. Prior to joining IBM Rational, he worked as an instructor and senior consultant with a focus on project management, requirements management, software engineering processes, and object-oriented technologies using Smalltalk and Java. He holds his MSc in computing for commerce and industry from the Open University.