Distinguishing Between Application and Interface: An Alternative Approach to User Interface Development

by Wayne Lee Jones
Lead, Presentation Architecture Group
Advanced Technology Solutions Division
Oracle Consulting

One of the core values of the Rational Unified Process® is the separation of business requirements from technical implementation; during the Inception phase of a new software project, analysts strive to capture the objectives of the application, independent of the manner in which the processes that support those objectives will be realized. Usability experts share a similar value in their approach to the design of a user interface: The needs of the user must be identified and analyzed without consideration of the constraints of a particular device or method of implementation. In both cases, it is only after the requirements are determined that questions concerning how to prioritize and implement those requirements are addressed.

Odd, then, that user interface design is so often a complication in the larger process of developing applications using the Rational Unified Process. Even teams that are intimately familiar with iterative methodologies often treat the user interface as a monolithic entity, developed in isolation by "creatives" and dropped into a project from above sometime during Construction, an immutable object to which developers must attach their code and requirements. This article will explore the conditions and preconceptions that lead to this situation, and present an alternative approach by which it may be addressed. Along the way, the article will provide a map for how user interface development can be integrated with ongoing iterative design activities for other components.
The Peculiar Problem of User Interfaces

It cannot be denied that the user interface is distinct from other components of an application's architecture. Chiefly, this is because it is the only part of the application to which subjective measures of success apply. Granted, customers, users, and engineers all have opinions about the value of one technical architecture or another, but at the end of the day, the real success of a component such as a data model or persistent object will always be measured by whether it enables an application to achieve a degree of objective, externally verifiable performance. The user interface, on the other hand, is entirely subjective; even the problem of whether something is "usable" or not has its root in human habits and preferences. Usability can be a precise discipline to the extent that users have a lot in common regarding what they like and dislike, and can easily understand and learn. But by no means can a user interface be measured in the repeatable and reproducible manner that, say, persistence can. If an application puts something into memory and can get it back, it will be able to do so again and again, in the same way, without fail -- assuming no bugs exist. Yet even the most carefully designed user interface will perform differently for each user, not because the code that makes it available to the user varies, but because users themselves vary in their approaches to the same interface every time they use the application.

This subjective nature of user experience (UX) has caused it to become the province of those whose responsibility is the management of user perception: marketing, human resources, and functional management. In most cases, the requirements for the user interface are provided by those who have a stake in what is really one small piece of the interface: the look-and-feel. Requirements from functional professionals regarding the user interface typically fall into the following categories:

- Make it look like what we have; in other words, make it look and feel like existing applications, or mesh with existing branding.
- Make it work like the competition's UI does; in other words, match the features or functions of another product.
- Make it exciting/high-tech/retro/hot/cool/ or any other adjective or buzzword that is floating around the collective business consciousness.

Of course, none of these categories lends itself to positive, concrete requirements. To the contrary, all of them represent boundaries or constraints that, without other context, can severely distort the user interface development process.

Given this situation, software engineers tend to do one of two things: berate functional professionals for not understanding what "requirements" are, or give up and take whatever mock-up the functional team generates as the prototype and build around it. Both courses are unhelpful; the former generates ill will among team members, while the latter imposes a look and feel or set of metaphors that can be entirely inappropriate to the
application under development. The consequence is often disintegration of the product's feature set due to the imposition of an incompatible or inappropriate user interface.

**Two Distinct Products**

Of course, this wouldn't be much of an article if we ended on the dismal scene I just presented. Steps can be taken to avoid the insertion of an uniterated user interface prototype into the application design process. In fact, it is possible to integrate the iteration of the user interface with the iteration of other components of the application to produce a method that is more faithful to the principles behind the Rational Unified Process. You simply need to understand that the development of an application is actually the development of two completely independent products: an application, comprising business logic and its technical infrastructure; and an interface, a particular deployment of that application in the context of a device or presentation layer.

Consider the implications of two of the core principles of user-centered interface design:

1. Interfaces should make use of the facilities typically available to the user as part of the devices on which the interfaces are deployed. For example, a Web-based application deployed in a browser should accommodate use of that browser's back button. An application deployed as a native Windows application, meanwhile, should make use of the common APIs for open/save dialog boxes and maximize/minimize widgets.

2. Interfaces should be suited to the constraints of the systems on which they are deployed. Anyone who has used a wireless device to access Web pages that lack wireless-appropriate style sheets can appreciate the pain of users compelled to scroll horizontally to read content designed for a system with six times the resolution of the device in hand.

Taken together, these principles imply that each deployment of an application's functionality on a particular device will require the development of a unique interface, with distinct functional and technical requirements. Even in the case where an application is made available only through one device, the user interface for that device will have requirements and a technical architecture independent of those that support the application's business logic. Of course, because an interface is, in essence, the provision of facilities to a user for interaction with an application's business logic, the design and development of the interface depend in large part on knowledge of the functions performed by the application. The high-level design of an application, the concepts developed during its inception, are a dependency of similar work on the user interface. Likewise, the functions of the application must be elaborated before the means by which those functions will be made available to the user can be elaborated.

Interface development need not insert itself into application development,
disrupting the iterative process. Rather, interface development follows its own iterative course in parallel with application development, but one phase off; decisions made in the development of the application feed into the iteration of the interface as requirements. The final section of this article illustrates how such a process can take place.

An Example

Consider the example of a company developing a knowledge management system for internal use. The first round of activity should focus on the inception of the application itself; the team should consider questions such as "What do we mean by knowledge management?" and "What do we expect users to do with the system?" The outcome of the Inception phase provides the input that enables the elaboration of the functionality of the system: Given that the knowledge management system will serve as a repository for documents, the system must be able to provide a set of features to support the storage, retrieval, location, and versioning of documents. Yet the inception of the application also serves as an input to the inception of a parallel product: the user interface for the knowledge management system. Analysts must begin to consider high-level requirements for the user interface -- the devices it will support, the connections it must make with other systems. For the sake of our example, let's imagine that the knowledge management system will be deployed through a Web interface and a wireless interface.

While the high-level boundaries of the user interface are being defined, analysts working on the knowledge management system's functionality will be in the process of elaborating the system's features. This will provide a set of detailed requirements that feed into the elaboration of the user interface. For example, if the application requires a data model that supports seven user-provided types of metadata, the elaboration of the user interface must address the manner in which users will actually provide the necessary metadata through the Web interface and the wireless interface. Further, to ensure the consistency and usability of the interface and minimize user effort, it is a best practice to elaborate the user interface for the entire feature set of the application at once. This requires that the elaboration of the application's features must be complete before the elaboration of the user interface can begin.

The user interface iteration process continues to trail the application's iteration by one phase through Construction and Transition. The infrastructure that enables construction of the user interface must all exist prior to the development of anything but the most rudimentary user interface templates. It is very difficult, for example, to develop a JSP to support the Web interface of a knowledge management system unless that JSP has a persistence layer and a set of business logic validations to invoke. Because the user interface is a window into an application's functionality, that functionality must exist prior to the interface's construction. The Transition phase of the user interface, meanwhile, should focus on issues such as usability testing; the testing and transition of the application's infrastructure should already be largely complete to ensure that bugs discovered are actually part of the user interface implementation rather than aberrant behavior caused by issues with the application's infrastructure.
Figure 1 illustrates how the two parallel iterations proceed.

Figure 1: Parallel Iterations for Application and Interface Development

Benefits of Staggered RUP Phases for UI Development

The iteration of the user interface as an independent product prevents the dilution of application functionality due to overzealous prototyping. Indeed, by intentionally and appropriately placing user interface development on an iteration path one phase behind that of the application itself, the typical, problematic model is turned on its head. Moreover, this model enables the simultaneous development of multiple user interfaces customized for various devices and channels. Whether user interfaces are distinguished for business or technical reasons, the various prototypes deployed will always be based on a known set of application features and functions. This allows for maximum differentiation of user interface form for different audiences and output formats, while minimizing differentiation in function among these interfaces -- because all of the interfaces share a common set of application features as a point of reference.

Acknowledgments
The author wishes to thank Marc Horowitz, Hamidou Dia, and Oracle Consulting for supporting the mission of user-centered design in application development.

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