

Saving the Task from the Tool: Techniques for User Experience Requirements Analysis

by [Wayne Lee Jones](#)



Lead, Presentation Architecture Group
Advanced Technology Solutions Division
Oracle Consulting

As a consultant at Oracle focused on enterprise clients who hope to realize cost savings stemming from ease-of-use, I know that at some level I am working to render myself obsolete. Though I enjoy my job, the industry will be better off when I don't have to do it, when the skills of usability analysis and user experience management are part of the toolbox of every designer, rather than in the hands of a specialist. One day, the idea of user-centered design will catch on to such an extent that it will be taken for granted, executed perfectly, and integrated into every aspect of the technology that surrounds us.



The sobering realization that we are far from that time usually hits me when I sit down at my laptop in the morning to check my e-mail or begin a design. Despite the fact that user experience and its related buzzwords have become commonplace, there is little evidence that the actual production systems we use on a daily basis have been built with the user in mind. Too many things that we do with technology are too difficult, and the blame lies squarely with the population of architects, designers, and implementers; in other words, with us.

One of my responsibilities at Oracle and at client sites is to transfer knowledge about user-centered design. Often, when I critique existing products in front of their developers or project managers as part of this process, I am met with defensiveness. "Hey, don't blame me; I hired a usability consultant," the project manager will say. Or the programming lead will demur, saying, "I thought that system was a little complicated, but I'm just responsible for building it; the guys upstairs are the ones who couldn't figure out what they wanted." There are a host of other excuses, and the worst part is, they are all true. The fact is that, in most cases, everyone involved with the design of an application means well, and has

 [subscribe](#) [contact us](#) [submit an article](#) [rational.com](#) [issue contents](#) [archives](#) [mission statement](#) [editorial staff](#)

the intent to build the best and most usable system possible. Somehow, though, at the end of the project, tests invariably show users struggling as they interface with the finished product. It's too late to redesign the system, so the technical writers shoulder the burden of explaining the system to the helpless users, and the cycle begins again.

In this article, I will explore how well-meaning developers and project managers typically lose their way in the effort to design a usable system, and show how they can become familiar with the concepts of user experience and apply them to their own projects. The first part of the article will discuss what is meant by usability, its components, and how it is measured. The second part will explore where gaps typically appear between expectations and the finished product, and I provide techniques for contributors at all levels of a project to ensure that the completed application is friendly to its users. For designers eager to implement these techniques in their own projects, I will also present example workflows that designers can put in place during the Inception and Elaboration phases of the Rational Unified Process.®

Usability and Other Jargon

One of the problems of usability is that everyone has a different sense of what it means. Part of this problem is attributable to the usability experts themselves, who might use any of the following (or many other) terms for their discipline:

- User-centered design
- User experience
- Experience design
- Interaction design
- Human-computer interaction
- Ease-of-use
- User-friendliness
- Interface design
- User interface development
- Usability engineering

At the risk of raising the ire of my colleagues, some of whom have staked their reputations on the subtle differences of technique implied by the terms above, all of these phrases mean the same thing: designing a system that helps me accomplish my task, rather than being a hindrance to accomplishing that task. Where experts differ is in their evaluation of the relative importance of different components of the problem, and the precise way in which fixes can be applied. For a designer starting out on the journey to user-centered design, these distinctions are less important than an understanding of the general issues of usability presented in this article.

Put simply, most applications are developed to help users solve complex problems. The problem is complex enough; your application should not be a problem in itself.

So how do we define usability?

Usability is a metric for evaluating the extent to which an application helps, rather than hinders, the user. It is typically measured in three ways:

1. How often is the user able to accomplish the intended task?
2. How often does the user require help, need to ask for directions, or consult the manual?
3. How much time does the user take overall?

A Quick Usability Assessment

It is outside the scope of this article to discuss the many ways in which formal user testing is conducted, but I can offer a simple technique for a baseline assessment of a system's usability. First, sit your most competent user in front of the system and give him or her three to five tasks of increasing complexity. If you were testing Windows, for instance, the tasks might be:

1. Make a new file named "test.txt."
2. Make a new directory on the root drive called "My Stuff" and move test.txt to that directory.
3. Set the file sharing permissions on "My Stuff" so that another person on the LAN can read test.txt.

Then, sit back with a stopwatch and time the total process, noting the time for the completion of each intermediary task. This provides a methodologically questionable, but at least reasonable, baseline to assess the next set of users, who should be drawn from a range of different exposure levels. Ideally, one would get at least a casual user from management, a casual user from the rank and file, an administrative assistant or other non-specialist, and then a specialist whose primary responsibility is using the system.

Observe and note down times as with the most competent user, but also record instances of hesitation, erroneous clicks, and other difficulties. Feel free to offer help if asked, but record the number of times help was given, with each additional clarification counting as another instance of help. Finally, compare the notes of each of the second set of users with the baseline time for the first user. There is no way to determine what the *ideal* numbers should be, but you might start with the rule of five: If the median user made more than five errors, asked for help more than five times, or took more than five times as long as the baseline to complete the tasks, there is likely a usability issue with the system. You may wish to be even stricter; usually the experience of testing users in this way, even

casually, is eye opening to the designers who built these systems in the first place.

Who Is That "User"?

In the section above there was a lot of talk about the *user*, and it is worth defining what is meant by this term. *A user is a person who might reasonably be expected to engage with a system in the process of accomplishing a task or reaching a goal.* Key to this definition is the concept of audience, from which the "reasonably expected" part derives. It is not appropriate to consider everybody in the whole world when designing a system, as not everybody in the whole world will likely use the system. For example, cars are not designed to be usable to five-year-old drivers, because five-year-olds do not drive. However, five-year-olds are in the audience of passengers, and therefore cars have features to accommodate the comfort and safety of small children traveling in the back seat. Sometimes with an application it can be complicated to determine who the audience actually is. It might be helpful to bear in mind the following three types of audiences for three different types of systems (though certainly there are more): your mom, your boss, and yourself.

For a public Web site or an application with a very broad audience, you can never go wrong designing for your mother. I say "your mother," not "my mother," because *my* mother was the first programmer in the family, teaching herself Pascal back in the days of the Apple II so that she could build custom multimedia applications for the education market. Now my father, on the other hand let's just say he's not a techie. To this day I'm convinced he thinks that Yahoo! and the Internet are one and the same, because Yahoo! is his home page. The point here is that you want to build for a user you find personally exasperating, but of whom you are fond, so that you will take the time to consider all the many ways the user can get lost in your designs.

For business applications or systems primarily used by specialists, it's helpful to think about your boss: reasonably intelligent, though certainly not as smart as you, and much more interested in apparent functionality than in the elegance of the back-end. This will help keep you focused on the user experience rather than the technical architecture, the natural place to which most developers gravitate.

If you are like most technical people, everything you use everyday is to some degree annoying, primarily because of usability concerns. So the final point in considering the user is to try not to ask the user to do anything that you yourself would find irritating and, conversely, to build in helpful shortcuts that you would enjoy having, even if they require more work. Some who have worked with usability before will note that my advice here is different from the advice typically given to "power users" (who are, after all, a theoretical category -- drunk on features and obsessed with customizability). Frankly, at an enterprise level, there are no power users. Not one person in your target audience wants to devote his or her whole life to maximizing the value of your product. They all just want to do their job without your product getting in their way.

The Extended Family

To conclude this discussion of usability, it is helpful to touch briefly on some related concepts that are, emphatically, *not the same* as usability. Nevertheless, a well-implemented usability process will address all of these concerns.

Accessibility. This is a metric of how available a system is to a user with some form of limitation or impairment. Such limitations include blindness or visual impairment, color blindness, motor difficulties, and so on. Many usability experts classify the problem of culture and language under accessibility. Though not speaking English is certainly not an *impairment*, a fully accessible system is able to provide information in all the languages used by its potential audience, and present information and symbols in a way that are intelligible across cultural boundaries. The United States government has recently put into force Section 508 of the United States Code, which governs the accessibility of software systems used by the government and military to prevent discrimination against users with various limitations. The Web site at www.section508.gov is a good place to start learning about compliance issues with this and other standards.

Information Architecture. The organization of content in a repository, on a Web site, or in any system with complex content, is known as information architecture. It is vitally important to the overall usability of a system that content be made available in a manner that is comprehensible and efficient to navigate. In most cases, the process of constructing an information architecture begins with the development of a taxonomy, a set of information classes or categories. This taxonomy is then subdivided into as many logical groupings as needed to make information easy to find and navigate. The two dangers of this kind of classification are fragmentation and ambiguity. Fragmentation occurs when there are two or more categories so narrowly defined that pieces of content can fit into more than one. Ambiguity, in contrast, occurs when categories are so broad that they combine unrelated information.

Information Design. The appearance and structure of data output, charts, maps, and other condensed forms of information are the subject of a discipline known as information design or information visualization. Again, this is a subject of great depth, but designers of systems can make great strides simply by looking to the standards of their industry before building a custom solution to the problem of data display. A helpful experiment is to limit oneself to using only black and white when building displays of complex information; this limitation forces designers to avoid layering data in a manner that obscures its meaning, the most common sin of corporate information designers.

Look and Feel. Perhaps the area most often confused with genuine usability, the look and feel of an application are related to usability only to the extent that they impair it. In other words, the structure of the interface and the flow between pages or processes should be independent of, but supported by, details such as color palette, font selection, and graphics. For many stakeholders, however -- as will be discussed -- look and feel are a major obstacle to understanding usability; care must be taken not to compromise usability in pursuit of aesthetic satisfaction.

Closing the Gaps

Once you understand the nature of usability, your next step is to capture requirements in a manner that supports user-centered design. Although a recently available Rational "User Experience Plug-In¹ has addressed some of the gaps that existed in previous version of the Rational Unified Process, the development community as a whole tends to regard user experience as a task relegated to the late stages of design -- in other words, a mold into which one fits an existing product or process. The techniques described below can assist developers with introducing the dialogue about user experience to their stakeholders at the very beginning of the design process. This guarantees that usability requirements will be captured in a manner appropriate to the problem of usability itself, rather than made to conform to processes more appropriate to technical development.

RUP Artifacts

One of the most significant problems in ensuring usability in the final product is that the key drivers of usability requirements are often far removed from the implementers. Although management or business experts may have had a strong vision of the way in which users would interact with the software, their vision rarely survives in the development process because such "soft" information is inadequately captured in the artifacts that form the basis of development. Three types of documents used in the current Rational Unified Process are discussed below in terms of how they can be adapted to preserve information critical for the conduct of proper user-centered design.²

Stakeholder Requests. The Stakeholder Requests interview template is an extremely valuable device for gathering and maintaining requirements that will yield a usable product. The template has a lot going for it, particularly the way it helps stakeholders break down the system audience. Of course, this document is not typically used as a repository for implementation requirements, and a supplementary specification might seem a more appropriate container for the information discussed below. However, there are several reasons why you should consider using the Stakeholder Requests template. First, it is one of the few places where the business drivers of a project can communicate their conceptual understanding directly to developers, and it represents a rare opportunity to elevate user experience issues to a project-wide priority. Second, a complex development project often has many supplementary specifications or other documents that are, in practice, largely ignored by high-level stakeholders. Third, highly usable interfaces result when user experience is treated not merely as one of several categories of requirements, but rather as a conceptual focus of development. Hence, elevating user experience issues early on, as part of the basic stakeholder requirements, can change the priorities of business drivers, designers, and implementers to the benefit of end users.

One important caveat is that non-technical stakeholders (and, for that matter, technical stakeholders) rarely differentiate between their requirements for interface usability and their requirements for look and

feel. Making interface usability a discussion point during the stakeholder interview will make stakeholders more comfortable that their concerns are being heard, and also educate them about the difference between the concepts. "Make it look like Amazon" is not a usability requirement, but "Provide a dynamic, personalized environment from which the user has one-click access to his or her shopping cart, wish list, and previously browsed titles" is an excellent one. Stakeholders should be encouraged to communicate their requirements from the perspective of the user, so that the resulting language can be more easily verified and addressed by developers later in the process.

Stakeholders should also take time to capture look-and-feel requirements, as there may be marketing, branding, and competitive concerns that seem trivial to developers but are very important to business stakeholders. Ideally, during the stakeholder interview, agreement should be reached on a preliminary look and feel, probably derived from previous development projects or from rapid prototyping on the part of a third-party or internal graphic designer. This first prototype, which should not be more than one or two example screens, is primarily intended to concentrate all parties in the direction of a particular color scheme and general arrangement of elements. Although this look-and-feel prototype is not critical to system usability, getting it out of the way early will prevent a lot of heartache later in the process, when business stakeholders might not see the value of the application because they are blinded by colors they find atrocious.

Software Architecture Document. On the development side, the initial Software Architecture Document should note the way in which usability requirements will influence the architecture of the final product. For example, advanced user interactions requested by stakeholders might mandate the use of JavaScript. Moreover, performance can be a component of usability; a Web page that takes a long time to refresh can confuse users and generate error if those users invoke the "Back" function on their browsers. The selected software architecture must not only address technical performance requirements, but must also support the fluid user interaction that stakeholders typically expect.

Storyboards Based on Use Cases. In the current incarnation of the Rational Unified Process, storyboards are not presented in great detail and are not assigned the same level of importance as requirements. But for capturing and properly delivering on usability requirements, storyboards are indispensable; they must be promoted from supplemental material to full requirements, with the status of use cases. In fact, it is possible to incorporate storyboard-related information into the use case itself, though designers working with usability for the first time will find it easier to keep it separate.

Storyboards are narratives that follow the flow of use cases and provide additional information pertinent to the designer of the user interface. There should be one storyboard for every use case or group of use cases that has a user interface component. Specifically, storyboards provide detail for each facility of the user interface about:

- Guidance the user interface can provide to make things easier for

users.

- The average volume of the data object to which the facility corresponds.
- The frequency with which users typically interact with the facility.

Remember that what constitutes a facility of the user interface can vary from system to system. On a Web site, this might be a form field or a button. On a voice-driven telephony application, this might be an input opportunity.

The first part of the storyboard's discussion of a user interface facility -- what the Rational Unified Process calls **desired guidance** -- deals with ways the interface can be made less ambiguous. For example, if the user is asked by the interface, "How long should this process run?" the form field in which to supply the answer should be labeled "hours," "days," or whatever is appropriate to communicate to the user that the request is for that particular unit of time.

The second part of the storyboard explains just how big the set of choices might be from which the user chooses or the amount of input the user will submit. For example, a Web site interface for choosing one's favorite color from the basic Crayola set of eight could easily take the form of a pull-down menu. Choosing an employee name from the global database at my company (Oracle), however, would require some kind of search and filtering facility. Otherwise I'd be there scrolling a long, long time. The goal of this section of the storyboard is to help interface designers select appropriate facilities with which users can perform their tasks.

The final section of the storyboard helps interface designers set priorities for the appearance of user interface facilities. The most important and frequently accessed information should be presented most prominently. For example, most bank and credit card call center applications place the option to hear one's balance first, because it is the most common selection. Putting it first reduces costs and load on the system measurably, because callers don't have to spend time reading through prior options. Conversely, if the designer expects a facility to be rarely used or supplemental, then its position in the interface can be minimized.

An example storyboard for the part of an ATM sequence wherein the user inputs a monetary amount is shown in the sidebar.

User Interface Standards

User interface standards are unique for every project, even if the look and feel they represent is kept the same across a company's product line. User interface standards change because each project deals with its own unique set of data objects and user interactions. In addition to maintaining consistency for some aspects of look and feel, it is also

Portion of a Storyboard for an ATM Sequence

Storyboards record insights into usability from business drivers in a manner that is accessible to implementers. The following brief storyboard shows some insights that might be useful in developing the portion of an ATM's user interface that requests an amount of money to be withdrawn.

important to maintain consistency of implementation across user interface facilities. That means adopting user interface standards as requirements against which to verify compliance. If users choose their state of residence via a pull-down menu in one part of an application, then they should not be required to input a two-letter state abbreviation in another part. The same facility should appear every time the user interacts with a particular data object. To make this possible, user interface standards should address the following topics, at a minimum:

- How will the user navigate between one part of the system and another?
- How will the user know which part of the system he or she is using, and how that part relates to other parts of the system?
- What features will be available to the user at all times (e.g., help, logout)?
- What are the most common multi-step processes in the system, and how will their flow be implemented?
- What colors will be used in the interface, and what will these colors communicate?
- What typographic conventions will be employed to communicate various types of information to the user?
- How will quantitative and/or tabular data be presented to the user, and what options will the user have for filtering and sorting this information?
- What standard terms will comprise the lexicon of this application? (E.g., the word "Search" will be used in lieu of "Find.")
- How will the user receive feedback that an error was made or input was not understood?
- How will look and feel, branding, and other requirements be implemented?
- What are the most common user interface objects, and what are their peculiarities? (E.g., pull-down menus are used when the volume of the data set being invoked is greater than three but less than fifty objects.)

Input of Amount of Withdrawal

The user will be provided with a facility to input an amount of money to be withdrawn from the system.

Desired Guidance

The user will be provided with a numeric keypad consisting of the numbers 0 through 9 with which to enter the amount. The amount entered by the user will be prefaced by a dollar sign (\$) to indicate that the amount entered is in United States dollars. A decimal point followed by two zeroes will follow user input to show that withdrawals can be made only in whole dollar amounts. The user will be provided with a facility to cancel input and re-enter it. The word "Withdrawal" will appear prominently to remind the user that the amount entered will be deducted from his or her account.

Average Volume of Object

90 percent of users enter two or three digits when making withdrawals, for a total of between \$10 and \$990 per withdrawal. The maximum amount that may be withdrawn is five digits, or \$99,999.

Average Usage of Facility

All users making withdrawals interact with the system to input an amount to be withdrawn.

User interface standards should be owned jointly by the business stakeholders and product development leads, and they should be reviewed periodically to ensure that all parties are still aligned. A good practice once user interface standards have been established is to create a library of user interface examples. When a designer has created a particularly usable or high-quality facility, an example of that facility should be stored in a repository for all implementers to reference. Ultimately, all major facilities should be in the repository, so that when developers need to implement a phone number entry field, for example, they can simply choose from the repository rather than breaking consistency with other parts of the application by building it from scratch.

Prototyping: A Blessing and a Curse

A final technique for ensuring usability is prototyping. If you're not careful, prototyping can set false expectations for stakeholders -- especially early in the project -- that lead to conflicts later, when the final output looks nothing like the prototypes. Nevertheless, prototypes are quite necessary, since they offer the development team the only opportunity to demonstrate flows and functionality to stakeholders prior to construction. Further, prototypes can be tested with users to ensure usability prior to construction, reducing some of the risk that usability concerns will require extensive reworking of the application. To avoid the most common pitfalls of prototyping, bear in mind some of the following practices:

1. If you are going to prototype, go the whole way. Build a complete, seemingly functional demo that looks and feels like the end product. Obviously, the demo can only show predictable paths through the application, but it should show at least a few alternate paths and one error subroutine. I'm sure many of the developers reading this are groaning right now, because this process can be long and painful, and it really needs to be completed and signed off on by all stakeholders long before construction begins. You cannot make the assumption that stakeholders can "visualize" what is meant by a more skeletal prototype, or that you can "fill in the blanks" on your own. In fact, a lot of unpleasant experiences have convinced me that these things are *never* possible. No one can see what is in your head except you, and even if everyone around the table nods and says they "get" it, project managers and development leads should assume that people actually *don't* get it.
2. The full prototype discussed above should be owned exclusively by development resources -- those individuals with a stake in the technical architecture of the product. Too often, business stakeholders lead the demo process, and hire outside graphic designers to build fancy demos in Flash. This is possibly the most horrible mistake that can be made with prototyping. First, graphic designers are generally not programmers; and second, if they do not have a stake in the technical architecture, then they will almost certainly be led, although they mean well, to depict user interface facilities or functionality that cannot be realized within the bounds

of the project's technical architecture or performance constraints. Third, there is also a very large risk that stakeholder expectations will be raised to the "anything is possible" level, which is always dangerous. Finally, development resources should build the prototype because they will ultimately build the product. If the resources are not available for building an attractive and usable prototype, they will not be available to build an attractive and usable product. Instead of viewing the prototype as a waste of time to placate management (a common assumption among developers), development leads should use the opportunity to test the feasibility of the technical architecture and the interface; in doing so, they should use as much of the intended technology as possible in the time frame and begin to build the library of user interface facilities discussed above.

Putting It All Together

User-centered design takes work, particularly in the Inception and Elaboration phases of a project. When stakeholders and developers are brought together to consider the needs and goals of the user, the resulting system invariably benefits. By elevating storyboards and user interface standards to the status of requirements, and completing an extensive and carefully scrutinized prototype, a project can have both satisfied stakeholders and satisfied users. In a subsequent article, I will explore some of these concepts in detail, providing examples of how usability requirements are analyzed and recorded on a variety of software projects.

Related Reading

Apple Computer, Inc. *Macintosh Human Interface Guidelines*, 1996.
Download at
<http://developer.apple.com/techpubs/mac/HIGuidelines/HIGuidelines-2.html>

Jakob Nielsen, *Designing Web Usability*. New Riders Publishing, 1999.

Donald Norman, *The Design of Everyday Things*. Currency/Doubleday, 1990.

Edward Tufte, *The Visual Display of Quantitative Information*. Graphics Press, 1992.

Acknowledgments

The author wishes to thank Timothy Dwyer and Oracle Consulting for their support of the mission of user-centered design in application development, as well as Mike Perrow, John Ringoen, and Jeff Campbell at Rational for their assistance in the editing and publishing of this article.

Notes

¹ For members of the Rational Developer Network, a new version of the "User Experience Plug-In" is available, which provides valuable tools for enhancing the capture and communication of user experience requirements. Here is the description of the plug-in, as it appears on the Rational Developer Network: "User-Experience Modeling Plug-In Version 1.5 -- This Rational Unified Process Plug-In provides a set of best practices for the development of a user experience model. It shows how the development team can define a simple model of the proposed user experience and use that model to prime analysis and design activities, and test models." The Rational Developer Network can be found at www.rational.net.

² The focus of this article is on applying traditional artifacts in a creative manner to allow the mindset of user experience to inform the overall process of software development. You may find that the new plug-in available from www.rational.net will facilitate the needs described in the three areas below.



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