Design, Usability and their Impact on Complex Systems

Kevin H Richardson, PhD

Senior User Experience Architect
D³ Infragistics Services
2 Commerce Drive
Cranbury, NJ 08512
krichardson@infragistics.com
Overview

While this is an article that is ultimately about user experience, it is necessary to understand the relationship between “design process” and usability. Historically, usability refers to post-design measurement and testing. Following existing software design methods, it is entirely possible to create a system that is neither useful nor intuitive but meets usability guidelines. This becomes increasingly problematic when the end result is the creation of complex business applications (e.g., internal business services software, control systems and enterprise software, to name a few). The solution to this problem, as implemented by Infragistics Services, is to follow a design process. Starting from a set of requirements rooted in user needs (rather than wish lists), usability analysts and design professionals collaborate to create quick, inexpensive models of the final system. Since this process of rapid “paper” prototyping does not require code, it is as easy to modify as erasing a line. It also provides the opportunity to put the prototype in front of users and test both the conceptual ideas generated during requirements gathering as well as screen-level usability, ensuring that the final, implemented product will be both usable and useful.

Cars, Houses and Software

Cars have been around since the late 1800s. They began as mechanized carriages, powered by primitive internal combustion engines. By all accounts, early cars were uncomfortable, unreliable and dangerous. But they did what the mature modes of transportation at the time could not. They could go farther, faster. And while the owner/operator also needed to act as their own mechanic and parts fabricator, it was enough.

Houses have been around quite a bit longer than cars. Early on, providing shelter from the elements was the deciding factor in choosing a worthy home. If it kept the rain off and the wind out, it was enough.

Software, particularly enterprise software, is the most recent of the three. Software is interesting in that it is not a physical object like a car or a house and therefore not tied to a particular form or size. However, like cars and houses, primitive software (and I would argue that almost all software today that requires human interaction is primitive software) is considered acceptable if it contains an impressive list of features and functions. Regardless of whether anyone can use it, if the software does more than the previous version (or its competitors’ software), it is enough.

![Figure 1. Don Norman’s Technology Lifecycle](image-url)
What becomes clear as you trace the evolution of these three is that, initially, complex systems begin their existence as engineering events. It’s enough that they work. As time goes by, complex systems evolve from the engineering to the personal. This is certainly true for cars and houses. Software, however, isn’t quite there. As shown in Figure 1, Don Norman referred to this idea in his 1999 book *The Invisible Computer*, though he was speaking about hardware rather than software (see, Norman, D., 1999: The Invisible Computer).

**Software, Usability and Design**

Software is a young product, from a product evolution perspective. While that may be sufficient to explain much of why it remains so difficult to use, it is not a reason to accept it. Every piece of software is the result of countless, well-meaning decisions. It was designed to be exactly what it is. That’s the problem. Not every product is designed well.

**Isn’t Usability Enough?**

Companies attempt to address the issue of poor design by handing the problematic screens to usability professionals. Through rigorous testing and adherence to time-honored guidelines, overall usability can certainly be improved. However, when dealing with complex systems in particular, improving usability after the system has been designed is akin to treating the symptoms rather than the disease. With that in mind, here is the answer to the question that begins this section:

**Usability is not enough.**

There are a number of ways to approach the design of a new system. In the first, a list of new features, functional updates and component wish lists are collected from key stakeholders and representative users. This list is then handed to a talented development team who do their absolute best to coordinate the requirements and create a unified system that will make sense to users. This is how terrible software is created. Why? Because developers are not usability analysts or designers and they are being asked to do the jobs of both.

In the second, a usability analyst works with key stakeholders and representative users to determine the needs and requirements of both groups. This results in a more appropriate set of requirements. Usability professionals are trained to uncover not only what users say they want but also to infer, based on observation and questioning, what they need. This is incredibly powerful. Neither users nor stakeholders are typically able to see beyond their immediate wish list and the difference between incremental improvement (giving users what they want) and innovation (giving users what they need) lies in skilled requirements elicitation. While this new system will be based on an excellent set of requirements, issues such as screen layout, interaction models and data visualization (i.e., design issues) are still being determined by roles whose core competencies are not design.

In the third, usability analysts and design professionals collaborate, gathering requirements and creating innovative ways of supporting users as they complete their work in ways they never dreamed. Sound a bit too
utopic? It’s true and when you can be part of such a “design process“ as either a stakeholder, user or team member, it’s a beautiful thing.

**Tales from the Trading Floor**

While visiting a financial client, the manager in charge of the project gave me a tour of his facility. As we walked through the trading floor, he pointed out his best trader and we paused for a moment to say hello and shake hands. The trader’s workstation looked similar to every other trader’s workstation I had ever seen: four monitors mounted two over two, on a single pedestal, angled such that they basically wrapped around the trader’s head. On the monitors was a riot of spreadsheets, line graphs, and scrolling, blinking and colored data points. It was so bad that I assumed my client was simply streaming in data from other sources. I asked him, “Who controls the way data is displayed on the screen?” His reply was surprising: “We do. We have complete control of the data and how it’s displayed.”

I then turned to the trader and asked him, “What would you want to make your job easier?” I was expecting something like “bubble up the important stuff” or even “stop these things from blinking all day”. Here is what he said: “A faster processor.” That’s right. He wanted a faster processor. Reeling, I asked why. He replied, “Because the one we have can only run 4 monitors. With a faster processor I could have six.” I could have stopped there, told the project manager to order everyone on the trading floor new hardware and called it a day. Would that have improved their performance? For the more experienced traders, it probably would have resulted in some degree of incremental improvement. But it’s not what they needed.

**What Users Need**

What a user needs becomes clear as you observe them work. It becomes clear as you ask them to explain why they took a certain action or why actions were taken in a certain order. Innovation becomes possible once you understand what the user is trying to accomplish. During one task, the trader was trying to decide whether to buy or sell a certain financial product. To do this, he was interested in knowing how certain currencies were trading relative to one another, whether the rates of change between the currencies were trending in the desired direction over time and how this compared to historic rates of change between the currencies. Rate of change, the rate of change of the rate of change, historical comparisons of rate of change. There are many ways to represent this information. One way includes 6 monitors worth of spreadsheets and line charts. Or a designer could create 3 novel ways of visualizing rate of change data at a glance (because it’s what they’re extraordinarily good at). Users cannot think in these terms. Frankly, neither can most usability analysts. They have neither the proclivity nor the training. Designers have both. That’s what happens when usability and design collaborate. You get **User Experience** - accurate requirements presented in an innovative, beautiful and useful form.

**Software Development is Risky Business**

I have made the case that the creation of more evolved software is possible through the implementation of a design process that relies on collaboration between usability and design professionals. What if it could also decrease the risk associated with software development? And since risk typically incurs cost, what if user
experience and design could reduce the cost of software development? Before we look at the two most prominent models of software development, it would be useful to consider another anecdote.

**A Tale of Two Harrys**

My wife and I have 2 boys, now grown and in college. When they were younger, they read all the Harry Potter books. By the time the first of the movies was released, they had already read the first three books and were anxious to see the film. But an interesting thing happened after waiting in line, paying $10 per ticket, buying soda, candy and popcorn and watching 90 minutes of Harry Potter’s adventures. They didn’t like it.

When I asked why they didn’t like the move, they said it wasn’t the same as the book. Apparently, neither the person who wrote the screenplay, the director nor the actors could match the “movie” the boys had constructed in their heads. Therefore, the film was a disappointment.

This is exactly the problem with traditional software development methods. Coding begins when everyone involved in the project agrees that the prose-based requirements documents, containing the sum total of everyone's wants, needs and desires, correctly captures said wants, needs and desires. This is the first of the three ways in which software can be created (described earlier). Not only are the stakeholders (and users) dependent on the developers to create a usable system, but they are depending on the developers to create a system that matches the system they’ve already created in their minds! Inevitably, many months later, when the curtain is pulled back and the alpha release is revealed, the team struggles to find the system they envisioned. This is followed by variants of “That’s not what I was expecting”. The only real difference between my movie example and this software development example is that I only spent $40. The software project spent more than that on coffee.

**Software Design vs Design**

The two main software design processes today are Waterfall and Agile. Each model begins with prose-based descriptions of the requirements that describe what the software should do. Each begins with a “Risk Potential” (my term) of zero. By this I mean that at the beginning of the development phase, everyone is confident and happy. Nothing bad has happened and everyone believes they share a common vision for the end-product. Then development begins and risk begins to increase. Why? Because decisions are being made that cause the final form of the product to veer away from the vision held by each member the team. Is it the fault of the development team? Absolutely not. They’ve been put in the uncomfortable position of needing to make design decisions as they try and do what they love to do – code. And so, as time passes, risk goes up and up.

As demonstrated in Figures 2 and 3, this happens whether the project is following a Waterfall or Agile methodology. Waterfall is the more problematic of the two because the end result isn’t available until the build is complete. Agile attempts to address this issue by chunking the build into short sprints. However, this only allows views into isolated pieces of functionality, requiring the team to imagine how they will work within the larger whole.
Neither process allows the team to view what the final system will look like, how it will behave or how it will interact with other systems and other pieces of functionality until the development process is complete. In
both, the initial development cost is high. Changing design decisions that have already been implemented in code is even more expensive. The likelihood of users accepting such a system is small.

The design process, as shown in Figure 4, coordinates the skills of usability analysts and design professionals and addresses this problem.

**Software Development Process Design**

![Diagram](image)

*Figure 4. A Design process minimizes development risk*

In addition to creating a better set of requirements (which, admittedly, could be the input to either Waterfall or Agile methods), the usability and design professionals also create quick, inexpensive models of the final system. Through a rapid series of sketches, wireframes and screen mock-ups, the team is able to provide a view of the complete system that enables everyone to gather around, question and come to agreement. Since this process of rapid prototyping does not require code, it is as easy to modify as erasing a line. It also provides the opportunity to put the prototype in front of users and test both the conceptual ideas generated during requirements gathering as well as screen-level usability. Once the bugs have been worked out and everyone shares a common understanding of the final product, the development team can begin working. And in addition to the requirements, they also have designs that concretely demonstrate the final visual form of those requirements. Development risk is therefore minimized before coding begins and held relatively constant throughout the development process.
Well Designed is Well...Designed

Good design is a research-based, iterative process that discovers and defines requirements. It provides an understanding of what users need to accomplish, in addition to what they want. It enables early consensus, modifications and user testing. It is fast. It is inexpensive. It is usable and intuitive, beautiful and inspiring. It reduces risk. It provides for innovation rather than incremental improvement. It fits within existing software development methods. It is scalable and supportable.

Software is an infinitely malleable product. It can be molded into absolutely anything. No need to worry about the availability of raw materials, transportation logistics or factory support. It will be what it was designed to be. All it requires is the desire, and a process, to make it great.
References