SOFTWARE DESIGN SERIES

USABLE SOFTWARE DESIGN

by Alexandru Bolboacă

2016

Contents

Chapter 1

2 Minute Introduction

One thing is missing when comparing software design with other design disciplines: who is its user? The common assumption is that the user is the end user of the product, but that is incorrect. The end user doesn't care how the product is built, only about what it can do.

So, who is the user of software design? It took me a while to figure out the answer to this simple question. But, once found, it opened up a world of possiblities. The answer, is also obvious once we state it: the user of software design is *the developer*.

The next question is: can we use some of the techniques from user-centric design and apply them to software design? Could we benefit from the experience of UX and Usability practitioners?

Since I spent some time writing this book, it's obvious that I believe the answer is yes. Usability is a very succesful idea in design, because it's a win-win situation. Users get to solve their problems and enjoy the process, while businesses see higher conversion and therefore more money. Usability is where user happiness meets business growth.

I hope I made you curios. In the next chapters, we will explore the characteristics of usability, how we can apply them to software design and, why we should look into it.

Chapter 2

The Key Questions

Why Usable Software Design?

I discuss the topic of software development with many developers and managers. Some of the common complaints of developers are:

- The code I work with is very complicated
- I don't understand my colleague's code
- The others don't get how to write better code

Some of the common complaints of managers are:

- It takes months for a new developer to become productive in our team
- We have low productivity
- Customers complain about bugs

What if I told you that there's a way of thinking that has the potential to solve all these problems?

Ergonomics, User-Centric Design, Usability

In the early 1900s, industry was mostly dependent on human power. Attempts to increase productivity, like Taylor's "Scientific Management", led to the development of ergonomics. Ergonomics is the study of people's efficiency in their working environment. Adapting tools to the job became something very important; Taylor managed to increase the production and wages at Bethlehem Steel by using different types of shovels for different types of materials.

While Taylorism is far from being popular nowadays, ergonomics persisted as a domain of study and experimentation. It especially developped during World War II. It also expanded past efficiency into areas such as work safety. This is why we still have things like "ergonomic chairs" for programming.

Fast forward to 1988, when Dr. Donald Norman wrote the seminal book for user centered design: “The Design of Everyday Things”. The book details examples of good and bad design and documents issues with them. For example, the Three Mile Island nuclear accident was partially due to confusion over the status of a valve. The typical reaction to such an error is that the workers made a mistake. Instead, Donald Norman argues that it’s a design problem and that the design of machines should change to prevent such mistakes. The book expands over the design practices and mindset that lead to user centered design.

Since Donald Norman’s book, user centered design has permeated most design disciplines, from IKEA’s furniture, to electronic devices such as those created by Apple, Samsung or HP and even kitchenware and coffee machines. We should thank him for the frustrations eliminated by designers who followed his principles.

The last step in this revolution has been usability. Usability is mostly associated with computer programs, how easy it is for end users to learn and to use them. Any developer doing web applications has certainly heard of this concept, since it is so ingrained into the way we build applications nowadays.

Given the complaints from developers and managers from over the world, it looks like the software industry has a similar problem. Because software designs are not usable, developers have a hard time learning them, using them and avoiding mistakes. What makes the problem even more complex is the fundamental *flexibility of software design*. There are tens, hundreds, thousands of potential solutions for each software problem, and almost every developer has a different view on what makes a good solution. It is also easier to change the design of software than to change the design of a chair, of a car or of a plane. These two things combined create many of the issues we face when designing software and using code written by other people.

The software industry has reached the point where usability has become increasingly important. The solution to the usability problem in software design is the same as for teh design for end-users: a shift in mindset. If we want more productive developers we should create software designs that help developers be more productive and make less mistakes. Based on the experience we had with usability in other areas, it’s very likely that developers will also become happier in the process. Everybody wins.

What Is Usable Software Design?

First of all, Usable Software Design is a change in mindset. We no longer accept the idea that “the code is what it is and a developer should be able to deal with it”. We actively pursue the change in design that allows developers to be more productive and make less mistakes. We do this incrementally and throughout the lifetime of a product.

Second of all, Usable Software Design means structuring the code such that the software design exhibits qualities similar with usability. We need therefore to start from the design qualities that define usability in other domains and explore what they mean for software design. Here are the five design qualities that define usability for products:

1. **Learnability**: How easy it is to accomplish basic tasks the first time you’re using the design?

2. **Efficiency**: How quickly can you perform a task once you've learned the design?
3. **Memorability**: When you go back to the design after a period of not using it, how easy is to become efficient again?
4. **Errors**: How many errors do you make, how severe are they and how easily can you recover from them?
5. **Satisfaction**: How pleasant it is to use the design?

As you can see, these characteristics link with the economical benefits:

- **Learnability**: how quickly can we **integrate a new developer** into the team
- **Efficiency**: how fast can a **developer do common (not simple) tasks**
- **Errors**: **how many bugs do developers introduce** and how fast can they fix them

and with management challenges:

- **Satisfaction**: directly influences the **motivation** of the team

Memorability can be approximated with efficiency and learnability, so we will only discuss it in passing.

When is Usable Software Design appropriate?

It should be obvious at this point that any manager or business owner would like to obtain the benefits of usable software design. The next question is: can we apply usable software design in any context?

The simple answer: Yes! A simple analogy can help you see that this is so. Web usability is used for a wide range of applications, from social networks to accounting applications. This is possible because web usability is a set of overarching principles that have a very wide applicability. The specific techniques and design decisions are contextual dependent, but the web usability principles apply across all web applications.

The same is true for Usable Software Design. Usable Software Design is not a set of design principles that you should blindly follow. Instead, it's a target formed by the 5 qualities of usability: Learnability, Efficiency, Memorability, Errors, Satisfaction.

This also means that Usable Software Design, like user-centered design, is a *shift in mindset*. Instead of letting developers live with problems like over-complicated code or error-prone designs, we put the developers in the center of the system and challenge the software design with the question: how should the design change so that we don't have < problem X > again?

Achieving usable software design will require a number of specific techniques that are applicable in concrete contexts. Some of these techniques can be borrowed from usability (Personas, user flows, usability testing, continuous user feedback etc) while others are unique to the practice of Usable Software Design. In this book we will look at both kinds of techniques.