The Secrets of User Interaction for Programmers

The science of interaction design.

Sheep Dalton

The Secrets of User Interaction for Programmers

The science of interaction design.

Sheep Dalton

This book is for sale at http://leanpub.com/userinteractionforprogrammers

This version was published on 2015-04-30

Contents

Chapter One Why Usability for programmers?

Why this book ?

The design/build divide

The motivation for this book came from the simple observation. If you go into any shop that sells computer books. Most books on how to program user interaction site on one shelf and the books on how to design use interaction normally sits on a completely different shelf. Most books on how to build a graphic user interface assume that you magically know how to design a application. They give you the details on how to build a particular interaction using the very large libraries most systems give you. For many programmers this can be the starting point for their programming careers or for their next hot app. The problem is that knowing how to program a graphic user interface is just a very small part of how to create an effective application. Equally, if you read a book on designing human computer interaction they typically fail to mention anything about how to build a program. Sometimes as Einstein said the secret is in the details.

I believe this separation is the principal cause for a lot of badly design software. The idea behind this book is to show you not only how to program a graphic user interface but also how to evaluate it.

Things how-to books don't tell you

This separation of how and why creates a number of new problems. Most books on how to program graphic user interface is tend to stick with the details of the particular API you are dealing with. They tend to assume that you know how to build a graphic user interface or rather there is a huge amount they don't tell you. Most how-to books don't mention things like how-to implement undo. This is because it's not difficult or complicated or part of the API and you're normally assumed to be grown up enough to build this yourself. Generally you are either learn how to do this by trial and error or finding an ephemeral article somewhere on the Internet. This is what in design circles people call a pattern. The problem is that most books don't include general design patterns for user interactions (UI). So the objective of this book is to not only tell you how to do something also why you must do something. For example Alan Kay (more on him later) thought that undo was one of the most important elements of any user interface.

Interaction design and user interface is changing rapidly

We are currently standing at the beginning of a significant shift in user interaction design. The first of these is the rise of *remote usability testing*. Doing a usability experiment right can be a long time consuming process. So much so that quite often you as a programmer want to off load it onto some specialist. If you have the money it's well worth doing. The problem is that we are now living in the long tail of application development. In the days when people were developing Excel and Word it was possible to have teams of 100 people developing it and so having dedicated user interaction specialists to test the software was a sensible division of labour. These days development teams are a lot smaller, the typical app developer has a team size of two. At this point it becomes very difficult to have a specialist. However remote user testing changes the whole equation. We will go into the details of remote user testing later, but for the moment it reduces the overhead of getting users to test your software down to a size where it is now possible to integrate user testing in your nightly build cycle.

At the same time it is possible to use analytics or remote analytics (for apps) to get a great deal of information about how your users are using your application. This can add a huge potential to understand your audience but it is not something you can let an external party do. In code analytics requires you to be part of the user experience.

Currently user interaction books find it hard to keep up-to-date with the rapid development in the world of software. To be frank there's a good reason for this, while software and code has changed radically over the last 30 years people haven't. We still have the same cognitive abilities we had 100 years ago. Most development processes still make the same basic mistake. They are too busy to talk to the users. So the central message for most interaction books is the same (and this book is no different) 'it's the user is stupid'.

Then there was lean

Another major development that has swept silicon valley is the rise of the lean philosophy. If you have ever set up a start-up company pre-lean your first response from reading the lean startup book is if I had read this book back then I'd be rich by now. My surprise was how much lean had inadvertently stolen from user interaction. For example in the Lean Startup[1] there is the 'concierge service' method where you just pretend to be a computer, from the 80s onwards people have taught the 'Wizard of Oz' method which is precisely the same thing. Both user interaction action design and the lean Startup method both seek to find ways to avoid writing code by finding clever ways to talk to users. While there are overlaps between market research, and true interaction design the only real difference is that interaction design never asks about the price. Importantly there are many techniques in the interaction design arsenal which lean has yet to discover. We are in quite a golden period for the lean Startup, eventually there is likely to be some huge malfunction. Lean processes

[1]amzn.to/1EFnrno

get quite close to fraud (taking money for products that don't exists) but have not so far crossed the line. It's only a matter of time before lean development becomes more regulated. The longer we can develop products in an ethical way the longer that golden age before regulation will last.

So Why get user interaction right?

Let's start with some simple bold facts. 90% of software doesn't make it. If you are working within a business then the likelihood is that your project will be canned. If you are building an app then you need to remember that only 5% of the apps on the app store and make money. The rest sit there unused and unloved. Some people can go through their entire careers and never get software that gets used by anyone. There are many reasons why this software doesn't get used, and why software projects fail. Most software projects failed because no one actually wants the program that was built. If you want to see more then follow this link to this academic work[2]

Here are 10 signs of IS project failure:

From T. Field, "When BAD Things Happen to GOOD Projects," CIO, 15 Oct. 1997, pp. 55-62.[3]

1. Project managers don't understand users' needs.
2. The project's scope is ill-defined.
3. Project changes are managed poorly.
4. The chosen technology changes.
5. Business needs change.
6. Deadlines are unrealistic.
7. Users are resistant.
8. Sponsorship is lost.
9. The project lacks people with appropriate skills.
10. Managers ignore best practices and lessons learned.

As you can see reasons 1,2,5,6, and 7 come down to basically understanding user interaction. As we will see later on user interaction failures tend not to be seen as such. Partly its because no one likes to admit they can't use a particular program. Partly most users don't have a away of articulating interaction problems (more of that later).

[2]http://www2.engr.arizona.edu/~ece473/readings/8-Critical%20Success%20Factors%20in%20Software.pdf
[3]http://www2.engr.arizona.edu/~ece473/readings/8-Critical%20Success%20Factors%20in%20Software.pdf

In house & contract programming

If you are working for an organisation as an in-house programmer, the description of the program comes from the ‘product manager’. Typically in-house software begins as a specification from someone in management. Even agile development has a ‘product owner’ who is responsible for giving each function point the go ahead. The problem is it’s not people in management who have to use it. If the people on the production floor cannot use the software this creates problems. At worst productivity just drops for no reason, at best the people on the shop floor pushback creating what is euphemistically called ‘Poor communication among customers, developers, and users’.

Start up development

Even if you are developing an app for yourself have you just switched one bad manager for less well-trained one? Most apps on the App Store only have a very tiny number of downloads. Most websites failed to gain traction. Remember when you are developing an app for yourself you must ask yourself how truly representative of the problem you actually are? (in this video by Tom Sharon a Google UI lead)[https://www.youtube.com/watch?v=v1KKsLukIBE] He interviewed 150 startup founders and VCs (venture capitalists) found that 86% of the people he interviewed said that the start-ups idea was baed on the founder’s personal pain. This is not necessarily a bad thing, but if the founder of your start-up is not representative you’re going to have problems.

While getting the user interaction right cannot guarantee that you get a successful product it will certainly up the probability of getting it right. Some programers I have met have been philosophically happy to work on one failure after another. After all they are being paid. Alternatively having people actually use your software is possibly one of the greatest experiences you can have. I have met people at parties who begin by describing this fantastic piece of software they have, their babbling about how brilliant it is, and eventually you realise that you wrote it. It’s a sweet moment and one that makes all the effort involved worthwhile.

Sometimes lives depend upon it.

There are many other excellent reasons to get the software right. Digital technology is becoming part of our everyday lives. As such getting it right helps everybody get on with living. If you have ever been played by bad software in an organisation then you have to ask yourself if it is time to break the circle. As software emerges in more critical areas it also becomes more important to get the interaction right. Many lives have already been lost in aircraft where pilots have inadvertently miss-read or miss-programmed the avionics. Clearly in the situations this kind of development you will be keenly aware of the dangers of failure.

However the greatest danger comes when you think there is no danger. You might think that the development of an in car entertainment system isn’t a safety critical operation. Yet distracting people for even half a second longer will likely lead to thousands of deaths. Harold Thimbleby[4] pointed out that people have been using a poorly designed phoned for LED font for displays in medical emergency equipment for years, showing that even the most minor of choices can have huge consequences.

[4]http://cs.swan.ac.uk/~csharold/cv/files/digitsCHI.pdf