Interaction Design

Kristen Vaccaro
September 2021

Introduction

Interaction Design is the study of the design, evaluation, and implementation of computing systems for human use.

People first! The people using a computer system should come first. Their needs, capabilities and preferences for their tasks should direct how the system is built. People should not have to change the way that they use a system in order to fit in with it.

Bad experiences with tools are almost never the user's fault. Bad design is to blame.

What makes a good interaction? (with some examples of bad interactions)

1 *Easy to learn:* how easy is it to accomplish basic tasks the first time people use the system?

> Bad Interaction: Snapchat. The New York Times published an article in 2018 titled "A Guide to Snapchat for People Who Don't Get Snapchat," whose primary goal was to explain the interface. Snapchat itself has parts of its website dedicated to explaining the meanings of various shapes and colors of its icons (Figure 1).

2 *Efficient:* how quickly can someone perform tasks, once they're used to the system?

> Bad Interaction: any tool with long menus (without shortcuts).

3 *Memorable:* how quickly can someone remember how to use the system, especially if they've been away for some time?

> Bad interaction: using the command line to untar a file (Figure 1). Similarly, system designs that personalize or update over time can make it more difficult for people to remember what they need to do.

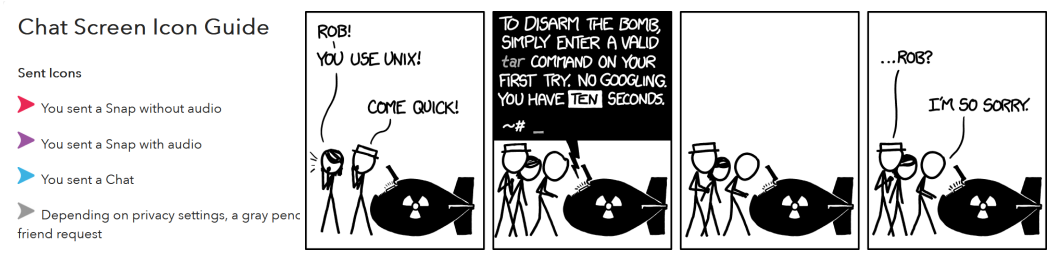

Figure 1: A system that is not easy to learn (left) and one that is not memorable (right)

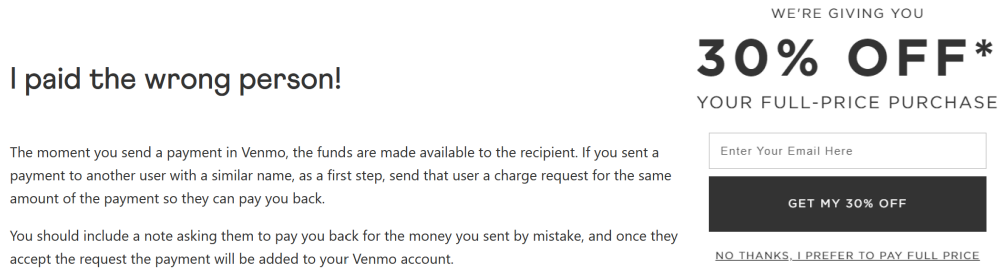

Figure 2: The Venmo app makes serious errors commonplace (left), while negative opt outs (right) are simply unpleasant.

4 *Few errors:* how many errors does someone make when using the system? How severe are those errors? How easy or difficult is it for them to recover from that error?

> Bad Interaction: When you send money to the wrong person in Venmo, there's no way to undo or cancel the transaction. This is a serious error, that is easy to make, and difficult to recover from.

5 *Subjectively satisfying:* how much do they like using it?

> Bad Interaction: Some companies try to convince people not to opt out of an offer, subscription, etc. (Figure 2). This design, which is called a negative opt out, manipulink or confirmshaming is not appreciated by users.

How should I design things?

Start by identifying a problem

It is very common for developers to start with a solution and look for a problem.

For example, you might be very familiar with topic modeling. You think it would be cool to apply topic modeling to fashion. So you build a tool that uses topic modeling in fashion, and then try to figure out what it could be used for. This is a bad idea.

Instead, start by identifying a problem. This could be a problem you have already experienced in your life or you could learn from others. Learning from others is a process called *needfinding*. Needfinding can either use observation (i.e., viewing users and their behavior in context) or interviews (i.e., directly engaging people with questions). Think about who you're engaging with for the needfinding – who are the experts? who are the extreme users in that context?

Start rough, refine over time

It is very common for developers to start programming as soon as they've decided on their idea for a tool. This is a bad idea.

Your first idea is rarely the best solution. Instead, *iterate* on your ideas.

A better approach involves generating a handful of rough sketches early in the decision making process. Get feedback on these ideas. Decide which elements, concepts, ideas from the first round

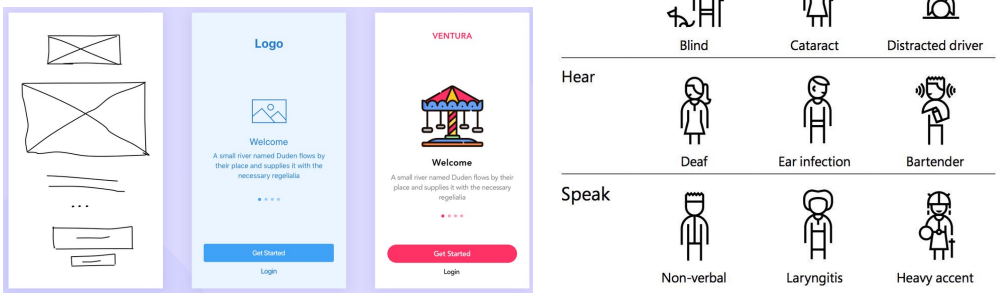

Figure 3: Examples of low, medium, and high fidelity prototypes of the same design.

to keep and which to discard. Create a second round of sketches – perhaps slightly more polished. Get more feedback on these sketches. Do a third round, becoming progressively "higher fidelity," that is, closer to the final product you plan to actually build (Figure 3).
You also waste less time on bad ideas when you follow this approach. It also helps people give you better feedback; they're more willing to be honest (particularly if it involves rethinking major aspects of your prototype), if they see an early draft.

Get feedback from others

It is very common for developers to build systems that they like. Often they have no idea whether other people would like (or even be able) to use that system. This is a bad idea.

Unless you're 100% sure that everyone who will want or need to use your tool is very similar to you, there are probably some assumptions, habits, preferences, etc. that you have and others don't.

You can catch some issues in your early draft designs using checklists (e.g., Nielsen's heuristics), but your most powerful approach will be having others try out your systems. You can do these tests even with early, low fidelity prototypes. Describe the sketch, ask what they would do, and then describe what would happen next, or show them the next sketch. The process can reveal issues very early in the design process, when you have plenty of time to fix them.

Other tips

Pay attention to the designs around you. What do you like? What works well? And when are you annoyed? What is a pain? There are nearly infinite examples to learn from...

Being explicit about the values in your design (e.g., community, security, trustworthiness) can also help you think through decisions you need to make.

Solving a problem for one group allows many more to participate (Figure 3). And most people encounter at least temporary or situational challenges to their use of systems. These are principles underlying universal and inclusive design.