CHAPTER 2

What Is User-Centered Design?

"Ease of use may be invisible, but its absence sure isn't."

—IBM

The most common and misguided presumption I find, especially within the developer community, is that the practice of usability is just subjective. These developers believe that usability decisions are arbitrary and can be decided by simply applying their own personal preference. Additionally, many of these decisions are made for reasons that have nothing to do with users. You better believe the CEO's current missive lands on the home page of the company portal. Who cares if it was written in lime green and has a dancing chili pepper on top? Therefore, if you're developing an application with a team or within an enterprise environment, you might be challenged when trying to implement user-centered design.

Perhaps you feel like you're the only member of your team who cares about the user's experience. Your colleagues or peers might roll their eyes when you talk about the importance of good layout and design. I realize this can be a long and lonesome journey, but it doesn't have to be. There are ways to spread sound, user-centered knowledge to disarm even your most vocal critics. One way to do this is by educating your team or organization about the value of user-centered design. To do that, we need to understand what user-centered design is; and most importantly, what it is not.

UCD Is Not Usability

I realize that my interchangeable use of user-centered design and usability might create confusion. Usability, also referred to as human factors, is the study of how humans relate to any product. Usability practices could be implemented in everything from a toaster to a doorknob, and even the packaging of both.

Human–computer interaction (HCI) is rooted in usability, but it focuses on how humans relate to computing products.

User-centered design (UCD) emerged from HCI and is a software design methodology for developers and designers. Essentially, it helps them make applications that meet the needs of their users.

Although this may be a bit of an over-simplification, Figure 2-1 is a diagram to help you understand the relationship between these methodologies.

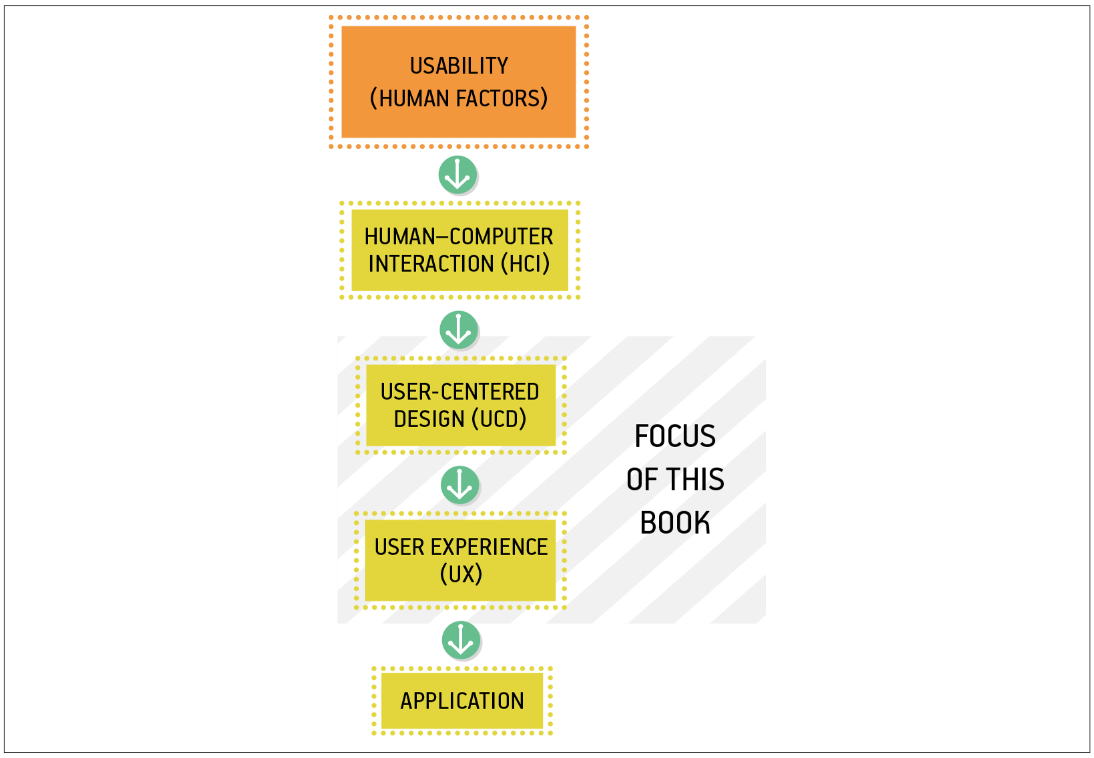
USABILITY (HUMAN FACTORS)
HUMAN–COMPUTER INTERACTION (HCI)
USER-CENTERED DESIGN (UCD)
USER EXPERIENCE (UX)
APPLICATION
FOCUS OF THIS BOOK

Figure 2-1. The relationship between usability, HCI, UCD, and UX

It's fair to say that practicing user-centered design will ensure that your application maintains good usability. That's the whole point! By placing users at the center of your development process, you remove ambiguity and get to the heart of what they need.

Additionally, there is the subject of user experience (UX). UX is a term often used to summarize the entire experience of a software product. It not only encompasses functionality, but also how engaging and delightful an application is to use. An application's UX is greater than the sum of its parts.

User-centered design can be implemented to ensure that your application maintains a great user experience.

UCD Is Not Subjective

The entire discipline of usability, and all of its underlying methodologies, is a conglomeration of many scientific disciplines. Through the implementation of ergonomics, psychology, anthropology, and many other fields, usability is rooted in scientific knowledge. It's far from subjective thinking or conjecture.

The user-centered design process works *against* subjective assumptions about user behavior. It requires proof that your design decisions are effective. If user-centered design is done correctly, your application becomes an outcome of actively engaging users. Therefore, any design decisions that were made by observing and listening to them will not be based on whims or personal preferences.

As the saying goes, "numbers don't lie." The user-centered design practice relies on data to support your design decisions. One way to do this is by completing usability studies (see Chapter 9). By observing users directly, we remove assumptions and statistically prove what is actually happening. This gives us a more stable foundation for the direction of our development.

Effectively, the data collected throughout the user-centered design process should make it difficult to argue against the changes your application needs.

UCD Is Not Just Design

This is probably the most common misunderstanding about user-centered design. Some people (and I find this mostly amongst our developer friends) believe that user-centered design practitioners are only focused on aesthetics or making things look pretty. While an application's aesthetic can be important, it's not the whole picture.

Being user-focused is more than just deliberating on how things look or creating flashy animations and slick transitions. User-centered design ensures that we examine how effective an application is in achieving its designed purpose. It's possible, as Figure 2-2 shows, to have a stunningly beautiful application that's a usability nightmare.

Of course, the reverse could be true. A usability study can identify flaws in your application's user interface (UI) that make it difficult to complete tasks. In this case, your application's UI plays a huge role in achieving success; however, it would be a mistake to make it our only focus.

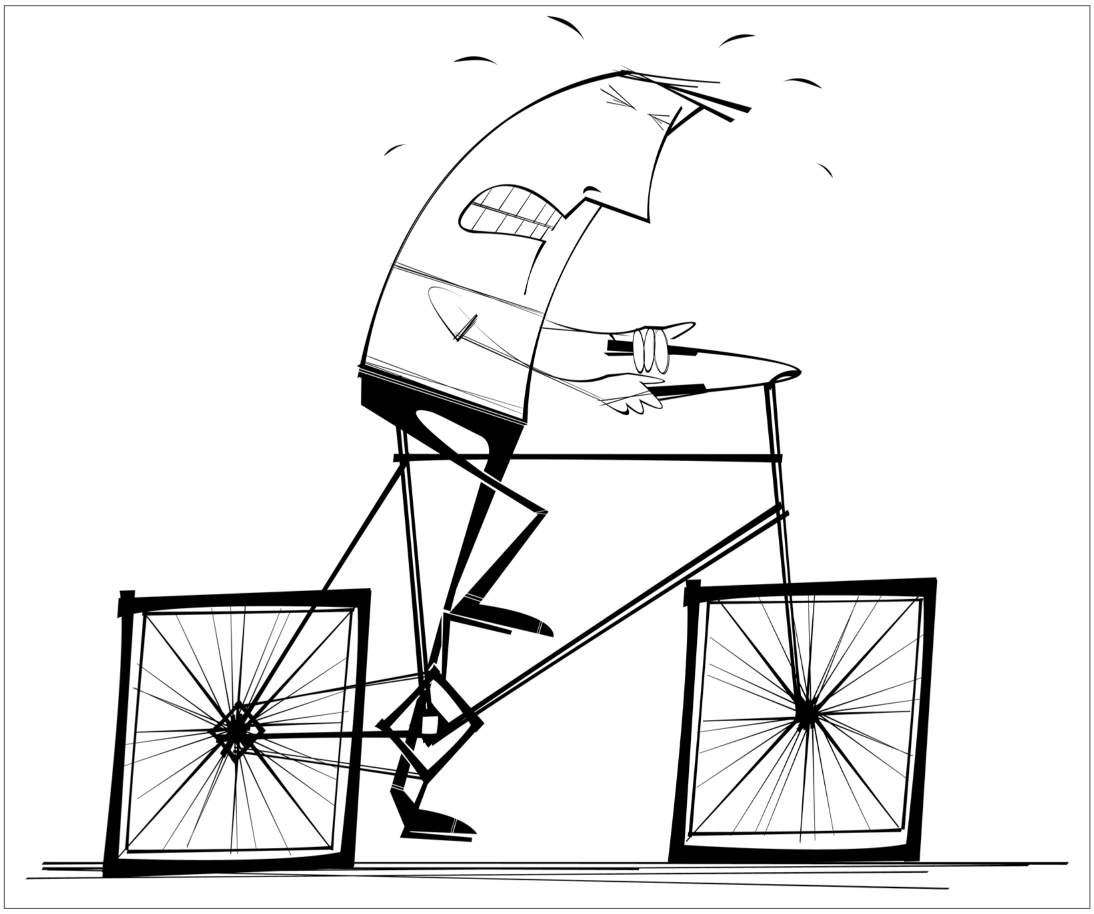

Figure 2-2. The designers of this bicycle should have conducted a usability study!

UCD Is Not a Waste of Time or Money

Dedicating time to proper user-centered design practices can be a difficult thing to do. The very nature of UCD requires reflection and observation. Let's face it, if you're spending your development time reflecting on design choices, it can feel like you're not moving forward. Also, if your usability research reveals design problems, you may end up having to remove previous efforts. That can feel like you're moving backwards!

User-centered design requires that we ask users what they don't like about our applications. Sometimes we don't want to hear their criticisms, or we assume we know what they're going to say. Opening up to feedback means opening up to complaints, and no one wants to hear how terrible of a job he's doing. Sure, Sally was just being "constructive" when she said your application was "worthless."

To avoid these criticisms, we ignore our users and shut them out. We focus on finishing our code, hoping all the other things will just sort themselves out.

Listen, I get it. As developers, the toolset of knowledge we must become familiar with is always expanding. New technologies emerge, form factors change, and new coding frameworks spring up daily. One minute, you've got a complex set of APIs all figured out, and the next minute you're reading a blog article about how it's dead, with a brand new API taking its place!

Getting into the business of being a programmer is to forever agree that you're constantly learning; and as technologies become more complicated, it requires more of our time and investment. With all of these challenges, the temptation is to dive deeper into our code and shut out any "distractions" like usability testing.

Billy Hollis, a developer-evangelist who promotes the value of usability practices, says that this resistance is a big challenge for our industry. He suggests the developer community loses valuable leaders in the usability space because they can't balance learning new coding techniques and spending time with users. They end up having to choose one over the other. Therefore, the community is full of code-only developers willing to spend their entire focus on learning the next API:

> I think that's one of the reasons we see such tremendous resistance [from developers] to design. The very people who've survived in the developer ecosystem are the ones who love code so much that they shut everything else out.

The point of user-centered design is that it doesn't have to be an either/or. Involving users' feedback in your development process can be a powerful both/and. Also, you don't need to be a user experience expert to implement good usability principles.

Hollis likens the process to learning how to ski:

> When you set out to ski, you're not trying to learn to become an Olympic skier. You're trying to learn to get to the bottom of the hill without falling down.

We have to break through the mindset that if we're not writing code, then our application isn't progressing. We need to accept that time spent with our users is a necessary part of the development process. It's just as necessary as learning and writing code. It seems that some developers spend more time deciding what framework they're going to use than how they plan to provide value for their users. It's as if these developers assume their application idea is inherently valuable.

If implemented correctly, user-centered design can actually *save you time*. By making sure you understand users' needs, you eliminate misunderstandings and costly mistakes. Remember, rebuilding your application because you didn't meet your users' expectations is a waste of time, too!

I would argue that problem solving outside of code would expose you to new ways of critical thinking. Taking a break from your code and conducting a survey or usability study can allow you to look at the problem from a different angle. You may find that when you return to your code, you're more focused and directed about what needs to be improved.

You may have the desire to conduct user research, but it doesn't seem financially viable to do so. After all, time is money; therefore, the perception is that time spent on anything other than writing code is costly.

Producing evidence of the return on investment (ROI) of user-centered design is outside the scope of our discussion; however, I would encourage you to think of usability as a way to avoid *losing* money. Fixing production bugs and supporting users through confusing and broken workflows requires a significant financial commitment as well.

In an article published in *interactions* magazine, Arnold Lund makes a similar case:

> An alternative approach is to view usability testing as part of a software quality management program and to justify it through the reduction in costs that would otherwise be incurred if usability bugs were not removed early in development. These costs include support costs for deployed software and the costs of fixing software once it is deployed.

If you're having trouble convincing management (or yourself) on the financial benefits of embracing user-centered design methodologies, consider making a financial case from another angle like cost avoidance.

UCD Is Not a Bug Report

You might believe that you're already performing user-centered design by simply giving users the ability to submit a bug or issue within your application.

While this is admirable and certainly something you should continue doing, do not substitute bug reports for comprehensive user research.

By continually looking at user feedback as a task list of items that need to be fixed, you never get to the root of what your users need.

Suppose a user submits a bug report for a feature that is not working correctly. You may be tempted to immediately drill into your code, find the source of the issue, and fix it as soon as possible.

If you're not taking the time to question what the user was trying to achieve when they encountered the issue, you gain no meaningful insight into how you could improve the application as a whole.

By exploring and asking questions that are unrelated to specific bugs, you might discover that users are trying to use your application in a way that you didn't realize. You might consider rewriting features to make those workflows more clear, or, better yet, the discussion could generate new ideas on how your application could provide more value.

Once I received a support ticket because a user was encountering an error every time she tried to submit a record in my application. The ticket provided all the technical details, including the entire error message. I thought I was so clever to include automated bug reporting within my application. Every time a user encountered an error, a ticket was automatically generated and sent to me. No more wasted time having to talk with users! Big mistake.

For hours I dug into the code and tried to discover a syntax or programming logic error. I looked through all my database connections and even reviewed the databases themselves. I reviewed the error message in the automated ticket several times.

Finally, feeling like I was getting nowhere, I decided to call the user. I asked her what she was trying to do. It turned out that the cause of the error was because she was including some invalid characters in a comments field on one of the application's forms.

Admittedly, it should've occurred to me that a user might input these special characters, and I should've conditioned my code to allow it. However, the larger issue was that she was trying to use a general comments field to document important medical information.

After spending some more time with the user, I realized that I needed to create more fields on the form to capture the information she was trying to document. My automated bug report was not the complete picture of the issue. If I had missed the opportunity to speak with the user by just fixing the problem within the code and moving on, I would have been unaware of her need for additional documentation.

Thus, users would've continued to use a general comments field to document vital medical information.

This is why the totality of your user feedback should not be just a list of errors within your application. Consider using bug reports only as a way to augment your overall user-design strategy.

UCD Is Not a Distraction

Have you ever been in a meeting listening to your users' requests, and your mind drifts to the dreamy land of the solution?

- Should I use a web service, or should I connect directly to the data?
- I wonder if we could build this on our company portal.
- What programming language should I use?
- I bet I could build this as a module on top of our main product.
- I'd love to use this as an opportunity to finally build a mobile application.

All of this technical thinking is fine. However, it has no bearing on what clients need because we haven't collected their user requirements yet! User-centered design helps us remain focused on the user's core needs. It ensures that we get solid information first and prevents us from trying to make the problem fit the technology.

Granted, there are real technological constraints that we have to deal with, but developers often make the mistake of addressing those issues first. User-centered design helps us move properly from our users' requirements to our technological solution. It's a purposeful approach that makes sure that we complete tasks in the proper order. We'll talk more about this process in Chapter 4.

For now, understand that usability is not a distraction. In fact, it actually works against distractions by helping you focus on the right things. It puts you in the correct mindset so that you can ask the right questions and challenge any preconceived notions.

Rather than contemplating technology, here are some questions we should be asking first:

- What is the source of the user's request? Can a technical solution solve his problem? Perhaps his problem is procedural or even political. Maybe it's a process, workflow, or education issue.
- Why is the user confused about this message? How does he interpret its meaning? Should I explain it a different way?
- Why does the user get lost between these two screens?
- Why did he miss this alert? Is he just ignoring it? If so, why?
- Why is the user completing tasks in the wrong order? Is there a better way to organize the layout to ensure he does it the right way?

A common theme throughout the study of usability is asking *why*. User-centered design helps us to become hyperfocused on understanding user behavior. It's a framework to help us discover the most effective response to their needs.

By combining usability, user-centered design, and user experience, you're ensuring a more complete approach to your application's development. It requires focus, determination, and even a little sacrifice. However, ignoring these aspects of your application, especially in today's ever-competitive market, is doing yourself and your users a disservice.

The Short Version

- The world of usability is broad and focuses on the study of humans interacting with *any* product.
- Human–computer interaction (HCI) is a subset of usability that focuses specifically on humans interacting with *computing* products.
- User-centered design (UCD) is a methodology used by developers and designers to ensure they're creating products that meet users' needs.
- User experience (UX) is one of the many focuses of UCD. It includes the user's entire experience with the product, including physical and emotional reactions.
- UCD is not subjective and often relies on data to support design decisions.
- UCD involves much more than making applications aesthetically pleasing. Design plays an important role; however, it's not the only focus.
- UCD can actually save time by helping you avoid costly mistakes.
- UCD doesn't distract us from getting work done. It ensures that we focus on the right things: meeting users' needs with the proper technological solution.