Supercharged Human-centered Design in Open

Tony Santos
tony@notjustprettypictures.com
@tsmuse
Who is this guy?

* Who am I?

  * I'm Tony. A little about me, I've been designing and making things for the web since 1998, I have a BFA in Design from a school you've probably never heard of, the Columbus College of Art and Design, and an MS from one you probably have, the University of Washington. I've worked for a handful of large companies including Microsoft and T-Mobile, and a lot of small companies both as an employee and a consultant. Currently I'm a member of the Mozilla User experience team, working as an interaction and product designer on developer focused projects. But enough about me. What are we here to talk about today?
What is “Supercharged Human-centered Design?”

* How to make your already awesome open source idea/product/project reach its full potential awesomeness, cut down on building things that you end up throwing away or completely overhauling, how to save tons of money (but not on your car insurance), and how to engage your community to do all of these things better than non-opensource projects ever can.
In addition to the #OSCON2013 hashtag, if you're tweeting about this please use the hashtag #HCDRulez. I want to see if I can get it to trend on Twitter. Please tweet that :)
* Office Hours! 3:20p Expo Hall Table 1

Monday, July 29, 13

* Where I will be after for specific questions

  * It sounds like a lot, and it kinda is. I've tried to leave room for questions at the end, but I'll also be doing an Office Hour right after this talk at 3:20p at Expo Hall Table 1, if you have more questions or want to talk about more specific things than I can answer while standing on stage picturing all of you in your underwear.
What is Human-centered Design?

* What is Human Centered Design
  * HCD is, more or less, designing with a focus on the human beings who will be using your thing, and not on the people or materials or technologies you're using to make that thing. But that's still a little too vague for my taste. So let's break it down even more.
What is Design?

* What is Design? (in this context)
* It's really hip right now to say "Design is the details," thanks to Charles Eames, or "Design is how it works," thanks to Apple. The Oxford English dictionary says design is: A plan or scheme conceived in the mind and intended for subsequent execution. I like this one the best, because it's the most accurate. Design is planning. In graphic design it's planning how something will communicate visually, in UI design it's planning how a UI will present all the options to someone, react to their input and change as the user interacts with it, in industrial design it's planning how to use materials and industrial processes to bring a thing made of atoms into being on a mass scale. But it's all planning. Everything designers do and make supports the planning of a thing, sometimes those pieces get reused as part of that thing itself (this is fairly unique to digital design endeavors). And non-Designers, what one of my grad-school professors called "little d designers" (as opposed to "big D designers" who were those of us with formal training) do design too. Engineers are some of the most prolific designers there are!
What is Design?

* What is Design? (in this context)

    * It's really hip right now to say "Design is the details," thanks to Charles Eames, or "Design is how it works," thanks to Apple. The Oxford English dictionary says design is: A plan or scheme conceived in the mind and intended for subsequent execution. I like this one the best, because it's the most accurate. Design is planning. In graphic design it's planning how something will communicate visually, in UI design it's planning how a UI will present all the options to someone, react to their input and change as the user interacts with it, in industrial design it's planning how to use materials and industrial processes to bring a thing made of atoms into being on a mass scale. But it's all planning. Everything designers do and make supports the planning of a thing, sometimes those pieces get reused as part of that thing itself (this is fairly unique to digital design endeavors). And non-Designers, what one of my grad-school professors called "little d designers" (as opposed to "big D designers" who were those of us with formal training) do design too. Engineers are some of the most prolific designers there are!
* What is Design? (in this context)
  
  It's really hip right now to say "Design is the details," thanks to Charles Eames, or "Design is how it works," thanks to Apple. The Oxford English dictionary says design is: A plan or scheme conceived in the mind and intended for subsequent execution. I like this one the best, because it's the most accurate. Design is planning. In graphic design it's planning how something will communicate visually, in UI design it's planning how a UI will present all the options to someone, react to their input and change as the user interacts with it, in industrial design it's planning how to use materials and industrial processes to bring a thing made of atoms into being on a mass scale. But it's all planning. Everything designers do and make supports the planning of a thing, sometimes those pieces get reused as part of that thing itself (this is fairly unique to digital design endeavors). And non-Designers, what one of my grad-school professors called "little d designers" (as opposed to "big D designers" who were those of us with formal training) do design too. Engineers are some of the most prolific designers there are!
What is Design? (in this context)

It's really hip right now to say "Design is the details," thanks to Charles Eames, or "Design is how it works," thanks to Apple. The Oxford English dictionary says design is: A plan or scheme conceived in the mind and intended for subsequent execution. I like this one the best, because it's the most accurate. Design is planning. In graphic design it's planning how something will communicate visually, in UI design it's planning how a UI will present all the options to someone, react to their input and change as the user interacts with it, in industrial design it's planning how to use materials and industrial processes to bring a thing made of atoms into being on a mass scale. But it's all planning. Everything designers do and make supports the planning of a thing, sometimes those pieces get reused as part of that thing itself (this is fairly unique to digital design endeavors). And non-Designers, what one of my grad-school professors called "little d designers" (as opposed to "big D designers" who were those of us with formal training) do design too. Engineers are some of the most prolific designers there are!
* What is Design? (in this context)
* It's really hip right now to say "Design is the details," thanks to Charles Eames, or "Design is how it works," thanks to Apple. The Oxford English dictionary says design is: A plan or scheme conceived in the mind and intended for subsequent execution. I like this one the best, because it's the most accurate. Design is planning. In graphic design it's planning how something will communicate visually, in UI design it's planning how a UI will present all the options to someone, react to their input and change as the user interacts with it, in industrial design it's planning how to use materials and industrial processes to bring a thing made of atoms into being on a mass scale. But it's all planning. Everything designers do and make supports the planning of a thing, sometimes those pieces get reused as part of that thing itself (this is fairly unique to digital design endeavors). And non-Designers, what one of my grad-school professors called "little d designers" (as opposed to "big D designers" who were those of us with formal training) do design too. Engineers are some of the most prolific designers there are!
“Most people make the mistake of thinking design is what it looks like. People think it’s this veneer – that the designers are handed this box and told, ‘Make it look good!’ That’s not what we think design is. It’s not just what it looks like and feels like. Design is how it works.” – Steve Jobs

* What is Design? (in this context)
* It's really hip right now to say "Design is the details," thanks to Charles Eames, or "Design is how it works," thanks to Apple. The Oxford English dictionary says design is: A plan or scheme conceived in the mind and intended for subsequent execution. I like this one the best, because it's the most accurate. Design is planning. In graphic design it's planning how something will communicate visually, in UI design it's planning how a UI will present all the options to someone, react to their input and change as the user interacts with it, in industrial design it's planning how to use materials and industrial processes to bring a thing made of atoms into being on a mass scale. But it's all planning. Everything designers do and make supports the planning of a thing, sometimes those pieces get reused as part of that thing itself (this is fairly unique to digital design endeavors). And non-Designers, what one of my grad-school professors called "little d designers" (as opposed to "big D designers" who were those of us with formal training) do design too. Engineers are some of the most prolific designers there are!
“Design: A plan or scheme conceived in the mind and intended for subsequent execution.”
— Oxford English Dictionary

What is Design? (in this context)

It's really hip right now to say "Design is the details," thanks to Charles Eames, or "Design is how it works," thanks to Apple. The Oxford English dictionary says design is: A plan or scheme conceived in the mind and intended for subsequent execution. I like this one the best, because it's the most accurate. Design is planning. In graphic design it's planning how something will communicate visually, in UI design it's planning how a UI will present all the options to someone, react to their input and change as the user interacts with it, in industrial design it's planning how to use materials and industrial processes to bring a thing made of atoms into being on a mass scale. But it's all planning. Everything designers do and make supports the planning of a thing, sometimes those pieces get reused as part of that thing itself (this is fairly unique to digital design endeavors). And non-Designers, what one of my grad-school professors called "little d designers" (as opposed to "big D designers" who were those of us with formal training) do design too. Engineers are some of the most prolific designers there are!
“Design is a plan for arranging elements in such a way as best to accomplish a particular purpose.”
— Charles Eames

* What is Design? (in this context)
  * It's really hip right now to say "Design is the details," thanks to Charles Eames, or "Design is how it works," thanks to Apple. The Oxford English dictionary says design is: A plan or scheme conceived in the mind and intended for subsequent execution. I like this one the best, because it's the most accurate. Design is planning. In graphic design it's planning how something will communicate visually, in UI design it's planning how a UI will present all the options to someone, react to their input and change as the user interacts with it, in industrial design it's planning how to use materials and industrial processes to bring a thing made of atoms into being on a mass scale. But it's all planning. Everything designers do and make supports the planning of a thing, sometimes those pieces get reused as part of that thing itself (this is fairly unique to digital design endeavors). And non-Designers, what one of my grad-school professors called "little d designers" (as opposed to "big D designers" who were those of us with formal training) do design too. Engineers are some of the most prolific designers there are!
Design is planning.

* What is Design? (in this context)

* It's really hip right now to say "Design is the details," thanks to Charles Eames, or "Design is how it works," thanks to Apple. The Oxford English dictionary says design is: A plan or scheme conceived in the mind and intended for subsequent execution. I like this one the best, because it's the most accurate. Design is planning. In graphic design it's planning how something will communicate visually, in UI design it's planning how a UI will present all the options to someone, react to their input and change as the user interacts with it, in industrial design it's planning how to use materials and industrial processes to bring a thing made of atoms into being on a mass scale. But it's all planning. Everything designers do and make supports the planning of a thing, sometimes those pieces get reused as part of that thing itself (this is fairly unique to digital design endeavors). And non-Designers, what one of my grad-school professors called "little d designers" (as opposed to "big D designers" who were those of us with formal training) do design too. Engineers are some of the most prolific designers there are!
What do I mean when I say “Human-centered?”

* What do I mean when I say Human Centered?
  * Human centered, sometimes called User-centered but personally I think that's way too narrow a way to think about it, simply means putting the humans that will be involved with the final thing you're making at the center of your process. Every decision you make while you're planning revolves around the thing you know about these humans and how they work, think, act, etc etc. This is opposed to other design processes that revolve solely around what a designer thinks is best, or around processes that revolve around a specific technology or manufacturing method. In these other ways of designing the needs of the people who will have the thing you're designing in their lives is often secondary, or not considered at all. [show some examples of things that are clearly not HCD derived]
A few disclaimers:

* What do I mean when I say Human Centered?
  * Disclaimers: HCD is NOT a silver bullet. There are other design methodologies and they’re perfectly valid in the contexts they were intended to use. Just because something is not designed in a human-centered way doesn’t make it bad, often things that are not designed in a human-centered way are designed so because of some other constraint. There are constraints we can’t know about a finished design that we’re discussing unless we were involved with that design or we discussed them with the team doing the design work. HCD is a great way to approach most consumer and professional software, but it isn’t always possible and sometimes it doesn’t work inside the real world constraints you have. Adapt accordingly and ask yourself “what constraints may I be missing?” when judging someone else’s work. With that said, let’s look at some examples of non-HCD things.
What do I mean when I say Human Centered?
The classic example, straight from The Design of Everyday Things. Is this door a pull or a push? It's got pull bars on both sides.
* What do I mean when I say Human Centered?
  * Information overload. There are labeled colors on the left, buttons labeled S, P1, P2–P10, and unlabeled colors on the right...
* What do I mean when I say Human Centered?
* But we're here to talk about software. This is probably what you thought of when I said “let's look at some examples of non-human-centered design” right?
What do I mean when I say Human Centered?

* But what about this? This popup light-boxes out the content of this page, and all I did was click the link my girlfriend sent me. I love that there’s a “Login for Price” button behind the “give us your email address to be able to see this page” modal, btw. This is a pattern that has popped up a lot in the last couple years, Quora being maybe the biggest site doing this. There are certainly reasons to do this, but they are not human-centered. Remember just because it’s not human-centered doesn’t make it a bad design, it just makes it a not-human centered design.
What is Design? (in this context)

Last one. What about this? Anyone know what this regex string does? Super powerful, super not human-centered. In this instance, validating an email address format, it’s a lot less work in this instance to type out this arcane string. It’s a tool that fits a need and does well by not being very human-centered, because in this instance it’s more efficient in the long run for me to train to think more like a computer than it is to try and do the work to make a computer work more like the way I think. So what’s the point of these last two examples? Know who you’re making things for. The popup collecting my email address before I can view a page is clearly more concerned with fulfilling the companies need to collect my email address than it is allowing me to browse their catalog, and regex is an instance where being more human friendly actually increases the amount of work I need to do routinely. Know who you’re making things for. HCD doesn’t = “correct” it just = human-centered.
Why should we care about human-centeredness?

* Why should we care?

* The tl;dr answer to this question is simple: if you don't care about the humans involved in your thing, they won't care about your thing. We live in an age of unprecedented choice. There are literally millions of web apps, smart phone apps, web sites, web services, desktop apps, tools and just products in general for people to spend their limited money and most importantly time and attention on. If you don't care about them, someone will come along who does (if they aren't there already) and then what will you do? These are things you've probably heard a thousand times, but here may be one you haven't: Human Centered Design goes well beyond graphical user interfaces. If your back-end infrastructure or technology in general isn't set up in a way that supports users working in a way that makes sense to them, your project isn't human-centered, even if it has the most researched, tested and well designed UI ever. Human-centered design isn't just a front end thing, it's a project wide thing, it's a user experience thing and even though people like me (reluctantly) have the phrase "User Experience" in our titles, user experience is everyone's responsibility, no matter how low level your code may be.
So, who are your users?

* Probably not you or your co-workers
  * There's a tendency to assume we are just like our users. Everyone does this from time to time, even me when I'm not thinking about it. "Of course this thing I want makes sense, everyone obviously wants this! Who couldn't think about it this way?" Usually the answer is "Almost nobody thinks about it like you do," for a variety of reasons. If you take one thing from this talk and only one thing, please let it be this new golden rule: YOU ARE NOT YOUR USERS. Pick up any "UX" book and this will be in there somewhere. It seems really simple, but it's so easy to forget. And I have an addendum: YOU ARE NOT YOUR USERS...and your coworkers probably aren't either although they'll do in a pinch. Think about it for a second. If you are making the thing you are immediately different from everyone who isn't. You have thought about it more, you've looked at it longer, you know things that no one who isn't making your thing can know. This is your superpower, and like all super powers it's also a burden. You have to work to be like your users because of it. But this is where open source works to your advantage, because open source communities almost always comprise more than just other people making the thing, and at the very least they usually consist of very different people all making the thing. The users of open source software are generally more involved and more enthusiastic about your software than they are about some software that's build in the shadows. Because they're a part of it, even if they aren't making it, and by adopting a human-centered approach you can identify your users AND create a whole new class of collaborator all at once. Cool, huh? So before we get into how, let's walk about user types. I feel obligated to mention the p-word here. Personas. Personas are awesome, they're also hard to do right because they either take a lot of time and effort up front or they take a few releases worth of research and testing to generate. If you can do them, do them. If you can't, profiles are better than nothing. Profiles are simply a rough character that you can refer to by name instead of talking about "our users." The worst thing you can do is pretend you know who your users are but not write it down. You will inevitably fall into the trap of the "elastic user." The elastic user is what happens when you don't have a hard definition, in a form everyone involved can reference, and your user conveniently stretches to fit whatever ideas you happen to have. Refer to your profiles by name! Bad: "Well, I don't think our users will actually want real-time atomic debugging, because while it might be a cool technology, they really haven't shown any interest in it and it's an expensive feature to build anyway." Good: "Well, I don't think Chad will actually like real-time atomic debugging, because he is running older hardware that can't support the feature." See the difference? It's easy to assign almost any value to the face-less entity that is "our users" because if you imaging that group large enough whatever your opinion is will certainly be represented by at least one of them. If you have a profile that you refer to like a real person, you can make decisions based on it. And if someone comes up with something that is their opinion but that isn't in line with the real user concerns, everyone can see it.
Golden Rule:
YOU ARE NOT YOUR USERS

So, Who are your users?
* Probably not you or your co-workers
  * There's a tendency to assume we are just like our users. Everyone does this from time to time, even me when I'm not thinking about it. "Of course this thing I want makes sense, everyone obviously wants this! Who couldn't think about it this way?"
Usually the answer is "Almost nobody thinks about it like you do," for a variety of reasons. If you take one thing from this talk and only one thing, please let it be this new golden rule: YOU ARE NOT YOUR USERS. Pick up any "UX" book and this will be in there somewhere. It seems really simple, but it's so easy to forget. And I have an addendum: YOU ARE NOT YOUR USERS...and your coworkers probably aren't either although they'll do in a pinch. Think about it for a second. If you are making the thing you are immediately different from everyone who isn't. You have thought about it more, you've looked at it longer, you know things that no one who isn't making your thing can know. This is your superpower, and like all super powers it's also a burden. You have to work to be like your users because of it. But this is where open source works to your advantage, because open source communities almost always comprise more than just other people making the thing, and at the very least they usually consist of very different people all making the thing. The users of open source software are generally more involved and more enthusiastic about your software than they are about some software that's build in the shadows. Because they're a part of it, even if they aren't making it, and by adopting a human-centered approach you can identify your users AND create a whole new class of collaborator all at once. Cool, huh? So before we get into how, let's walk about user types. I feel obligated to mention the p-word here. Personas. Personas are awesome, they're also hard to do right because they either take a lot of time and effort up front or they take a few releases worth of research and testing to generate. If you can do them, do them. If you can't, profiles are better than nothing. Profiles are simply a rough character that you can refer to by name instead of talking about "our users." The worst thing you can do is pretend you know who your users are but not write it down. You will inevitably fall into the trap of the "elastic user." The elastic user is what happens when you don't have a hard definition, in a form everyone involved can reference, and your user conveniently stretches to fit whatever ideas you happen to have. Refer to your profiles by name! Bad: "Well, I don't think our users will actually want real-time atomic debugging, because while it might be a cool technology, they really haven't shown any interest in it and it's an expensive feature to build anyway." Good: "Well, I don't think Chad will actually like real-time atomic debugging, because he is running older hardware that can't support the feature." See the difference? It's easy to assign almost any value to the face-less entity that is "our users" because if you imaging that group large enough whatever your opinion is will certainly be represented by at least one of them. If you have a profile that you refer to like a real person, you can make decisions based on it. And if someone comes up with something that is their opinion but that isn't in line with the real user concerns, everyone can see it.
Golden Rule:
YOU ARE NOT YOUR USERS... and your coworkers probably aren’t either, although they’ll do in a pinch.

So, Who are your users?
* Probably not you or your co-workers
  * There's a tendency to assume we are just like our users. Everyone does this from time to time, even me when I'm not thinking about it. "Of course this thing I want makes sense, everyone obviously wants this! Who couldn't think about it this way?"
Usually the answer is "Almost nobody thinks about it like you do," for a variety of reasons. If you take one thing from this talk and only one thing, please let it be this new golden rule: YOU ARE NOT YOUR USERS. Pick up any "UX" book and this will be in there somewhere. It seems really simple, but it's so easy to forget. And I have an addendum: YOU ARE NOT YOUR USERS... and your coworkers probably aren't either although they'll do in a pinch. Think about it for a second. If you are making the thing you are immediately different from everyone who isn't. You have thought about it more, you've looked at it longer, you know things that no one who isn't making your thing can know. This is your superpower, and like all super powers it's also a burden. You have to work to be like your users because of it. Even in open source where a large number of users are also owners of this superpower, this is still an issue because probably not all of them are, and maybe they're not really who you're making this for. But this is where open source works to your advantage, because open source communities almost always comprise more than just other people making the thing, and at the very least they usually consist of very different people all making the thing. The users of open source software are generally more involved and more enthusiastic about your software than they are about some software that they just bought. Because they're a part of it, even if they aren't making it, and by adopting a human-centered approach you can identify your users AND create a whole new class of collaborator all at once. Cool, huh? So before we get into how, let's walk about user types. I feel obligated to mention the p-word here. Personas. Personas are awesome, they're also hard to do right because they either take a lot of time and effort up front or they take a few releases worth of research and testing to generate. If you can do them, do them. If you can't, profiles are better than nothing. Profiles are simply a rough character that you can refer to by name instead of talking about "our users." The worst thing you can do is pretend you know who your users are but not write it down. You will inevitably fall into the trap of the "elastic user." The elastic user is what happens when you don't have a hard definition, in a form everyone involved can reference, and your user conveniently stretches to fit whatever ideas you happen to have. Refer to your profiles by name! Bad: "Well, I don't think our users will actually want real-time atomic debugging, because while it might be a cool technology, they really haven't shown any interest in it and it's an expensive feature to build anyway." Good: "Well, I don't think Chad will actually like real-time atomic debugging, because he is running older hardware that can't support the feature, and he isn't concerned with the speed of non-atomic debugging anyway, what he really wants is intelligent-code complete 5000." See the difference? It's easy to assign almost any value to the face-less entity that is "our users" because if you imaging that group large enough whatever your opinion is will certainly be represented by at least one of them. If you have a profile that you refer to like a real person, you can make decisions based on it. And if someone comes up with something that is their opinion but that isn't in line with the real user concerns, everyone can see it.
So, Who are your users?

* Defining your primary users

  * Your primary user type is the one you care the most about. It's really that simple. Everything you do is for the primary users, because they're the most important to you. Why are they the most important? That depends. Maybe they're the ones who spend the most money. Maybe they're the ones who tweet the most about your project. Maybe they're the ones you just like the best and have decided "This thing, it's for them."
So, Who are your users?
  * Secondary users are nice too.
  * Often, but not always, we also define secondary users. These are one or more user types who we mostly satisfy, but if our project deviates from their needs a little bit we're ok with it. That's why they're secondary.
So, Who are your users?
  * Who are we ignoring?

  * Not often enough, IMHO, do we define who we’re not making stuff for. These are called, Negative users (or Negative personas), and they’re a really handy thing to have, a lot of the times these get made while you’re working on defining your primary user, because it’s pretty easy to define that user by first defining who they are not. Take the time to define these, you’ll be glad you did later.
So, Who are your users?

Open source-ify your personas/profiles

* This is an opensource conference, do what you do best. Share them with your project community. This will do two things: 1) it will allow you to get feedback from people and build better ones and 2) it will put an end to the "I think we should do this, and you're stupid if you disagree" style arguments that we all have in bug-trackers and projects forums ALL THE TIME, because you'll have a source of truth to point the community back to. If you can get community members talking about making things for the profiles by name, you have won the HCD in open source game.
How to figure out how your users think about your problem a.k.a. Empathy

The one real goal for any user research or user testing is to figure out who your users really are, what they like and dislike, how they think, what their problems are, and what they think the solutions are. If you know all of this, you can mash it all up with what you know about design and engineering and technology and make something that they will love, and you can do it cheaper, faster and better than your competition. This is the idea of "designing with" instead of "designing for." Simply put, "designing for" is a one way act, it's not human centered. "Designing with" is a two-way act, it requires us to admit that we don't know everything, actively try to empathize with the people we're trying to serve, and let them inform our process and final solution. Designing with is what Human Centered Design is all about. Open source is already doing a lot of the work of "designing with" by allowing the community around a project to help shape the project, but here are a few ways to super-charge it:
Building Empathy: Ask

* How to figure out how your users think about your problem a.k.a. Empathy
  * Ask them
    * This one is pretty simple. Talk with them, this is where interviews and (and sometimes surveys, if they have long form answer type questions) come in. Find out what people think, what they know, what they say they do (this is an important distinction we'll get to in a minute), what they say their problems are, what they say they like, what they dislike, and most importantly, how they think about the way a product like yours works, or how they think about how it fits into their world. Don't assume that if they dislike something they will file a bug about it, submit a patch or even bring it up in a forum. Sometimes people don't really know what they want, (and that's why you don't ask them that question) but what they do know if how they think about stuff (mostly) and how they live their lives and how they do their jobs. Your project probably fits into one of those categories. Create a new contributor type and official recognize them as user advocates or user representatives or something like that. Engage them, keep them talking, show them things early and often.

  * Pros
    * It's easy. People love to talk about themselves, and most of them are very happy to know that their opinion is important. This is especially true with open source projects, and usually drives participation WAY up. It gives non-technical people (or people who are not technically inclined in the right way) a way to contribute to your project. People like to feel heard, and when you take an open source community (contributors and users) and empower them like this, it creates an awesome feedback loop of excitement and promotion and participation that leads to everyone winning.

  * Cons
    * It turns out people don't always do what they say they do. It's important to understand what people think they actually do, but we have to keep in mind that there is often times a disconnect, especially if it's something they do often enough that they can do it on auto pilot.
The Henry Ford Problem

* How to figure out how your users think about your problem a.k.a. Empathy
  * The Henry Ford problem

* Any conversation about "asking people what they want" can't happen without mentioning Henry Ford. I wish this wasn't the case, but c'est la vie. When I first started working as an interaction designer, I had a manager throw this in my face when I tried to do research. You've probably seen this quote: "If I had asked people what they wanted, they'd have said a faster horse." This is an argument used in product design and development against doing research. It assumes people don't know what they want and are just waiting for some genius to figure it out and tell them. To that I say, "If you're asking people point-blank what they want, you're not a very good researcher." The job of research, especially user or consumer research, is to learn about the people who will use your thing. If you try to get them to do your job for you, figuring out the problem and solving it, then you will never succeed. Ford obviously saw a problem of people needing faster, reliable transportation to deal with the country that had grown considerably. If he had asked them "what is it about travel that you dislike the most" people may have overwhelmingly told him "it's too slow. It takes so long to get anywhere. I can't go more than a few hours without resting the horses." and from there he could have envisioned the Model–T as a solution to the problem he had clearly identified.
Building Empathy: Observe

* How to figure out how your users think about your problem a.k.a. Empathy
  * Observe them
    * Watch people doing whatever it is your thing does. Email client? Watch people read their email, see how the manage it, see how they organize it, do they delete it? Do they tag it? Is it all in their inbox even after it's read? You can do this in person, although it's time consuming and expensive. You can also use online services like 2Loop? or usability.com. On my team at Mozilla, we use skype screen sharing to watch how people interact with our stuff and other company's things to get a better understanding of how they do whatever it is we're interested in (e.g. Find help to a problem that comes up when they're writing code).
  * Why this is best
    * Remember when I said people don't always tell you the truth when they tell you how they do something? It's not intentional, they usually really believe that's how they act, but our brains dump any information we don't really need when we're remembering things. This is why if you ask someone to detail how they do something they do all the time, most people have a really hard time breaking down the steps. They don't have to think about it anymore and so they don't remember doing it. People also tend to believe they will act in ways that are more virtuous than they are. WHen I say virtuous what I really mean is "more idea". When asked if seeing the calorie totals on fast food menus would make them order more healthy options, and overwhelming majority of people said "Oh yes, if I only knew there was 8000 calories in that 72oz frozen custard, fudge, peanut butter, snickers, gummy bear, toffee milk shake, I would never order it!" When they were then observed in restaurants with these new menus, they continued to order the same unhealthy things. They weren't lying, they really did believe they would act this way, because we are all for the most part positive creatures and we want to believe we can be better. (citation?) Observing people allows you to see how people really act, and spot the discrepancies in their descriptions about their behavior. Its important to not only know how people do something, but when they think they actually do something else entirely.
  * Pros
    * gives you the richest data. All that stuff I just said about people doing things differently than they report them when asked.
  * Cons
    * It can be expensive and time consuming. Doing it remotely causes you to miss out on things they're doing not on the computer.
Building Empathy: Participatory Design

* How to figure out how your users think about your problem a.k.a. Empathy
  * Participatory Design -- The best method for open source projects, IMO
  * When you head the term participatory design, what do you think of? Crowdsourcing? Design by committee?
  Participatory design isn't really even a design technique, it's still a research technique. Neat huh?

* What isn't it?
  * Having your users design it for you. The "designs" they come up with are simply an artifact to help you understand their thought processes.
  * Why is it particularly good for open source projects?
  * Participatory design was practically made for open source projects, because they don't have to worry about IP issues. Larger, closed-source companies are often not willing to take the risk of having users come in and design something, for fear that they may get sued later for "stealing someone's idea." This isn't a problem for open source, and participatory design is, in my experience with it, one of the fastest ways to understand a user's way of thinking about a problem I'm working on.
* How to figure out how your users think about your problem a.k.a. Empathy
  * How it works: Both asking and observing
    * In a participatory design session, you sit down with one of your users (or coworkers in a pinch) and you explain to them the problem you've identified and are trying to solve. You then let them correct your explanation, because it will always be a little different with each person. Then you invite them to envision a solution to that problem, with no constraints. This can be done through drawing, story-telling, or collage (yes collage, more on this in a second.) The point is you get them to explain to you the solution as they see it, which will show you how they're thinking about the problem. You do not, or at least I've yet to see it be a good idea, expect to take the designs from your sessions and implement them. You have designers for that, or that's your job. In fact, it's usually a good idea to tell them that these aren't going to be implemented as is, but are going to greatly inform the final solution. This 1) takes the pressure off them to make something that will ship and 2) removes any expectation they might have to seeing their design in the final release. You do a handful of these sessions with different people, and then you go back and look at how they solved the problem. How they're thinking about the problem, and you use that point of view, combined with your knowledge of the technology and the constraints of the project, and you design a final solution. If you're design problem is a UI, have them design a new UI. If your problem is trying to build you APIs, have them tell you a story or draw you a diagram of how they think the information your API will be moving around works. Again, this is not just a UI thing. It's especially important in data modeling, because you can better anticipate the patterns that users will be calling for data out of a system, instead of optimizing on something else and paying for it later when they use your system in some unexpected way. I've worked on multi-million dollar pieces of software that suffer from this specific problem.
Tell me a story...

It's not just drawing -- Story telling is a great way to do participatory design with participants who are uncomfortable drawing or doing anything too "artsy." Try to get them to write it down, that way you have it in their words, but if the only way you can get it is to have them tell you a story about it, record them speaking so you get it in their words. Throughout, keep asking them questions, get them to enrich the story, give it more detail, examine their solution and observe the way they're thinking about the problem. The final story is good, but understanding where they left holes in their story, where they decided they had to go back and make changes, and why they felt they had to are almost more important for when you're figuring out your solutions later. Writing is also a great solution for remote sessions, where collaboratively drawing or collaging are a little more difficult to do. You could technically do this unmoderated, but then it really just turns into a survey and you're less likely to get good results, it's not very "participatory" if you aren't there participating too.
I can't draw -- Sometimes you need a solution to problem that is visual, usually UI issues, but maybe even data modeling issues, and you really need to see how they would build an interface to see how they understand the problem. But your participants are terrified of drawing, they tell you they can't draw, and they're embarrassed. No problem, it just means it's collage time. Set up a space (digitally or on paper) for them to organize pre-built components into a UI to solve the problem you've presented. They can be super high fidelity (like the tehan and lax iOS components Photoshop templates) or they can be super lo-fidelity, simple boxes with labels on them, or they can be something in between. Choose based on the problem you're trying to solve and how much value you think having the higher fidelity is going to add. When you do this, it's important to have A LOT of components to choose from. Way more than you need. You don't want to have too few and have them feel like they need to arrange all of them into the correct solution. Also make sure you have a lot of each kind of component, they shouldn't be constrained by how many graphics you've printed out. Just like with the story telling, it's important to ask them questions about what they've put together every time they think they're done, get them to tell you the story behind their design, and why they made the choices they did. This will often cause them to want to make changes as they've reconsidered something while answering your question. This is a good thing! Just make sure you take a picture of each finished solution before they make changes, so you can see the evolution of the thinking later.
Let’s all draw!

* Let's all draw -- Sometimes you have participants who are comfortable drawing. If that's the case let them start drawing and then talk about each drawing. Like the other two methods, it's important to use the design they're making to tell you a story about how they are thinking about the problem and how they came to the solution they did. Have them start a new drawing each time they decide they want to change things, having paper that's easy to trace through and using a market instead of a pencil makes this a whole lot easier to enforce.
How many do we do?

* How many of these do we do?
  * This is qualitative research, so you don't need a gigantic sample for this to have value. You're going to start seeing patterns after 1 or 2 participants and those patterns in thinking are what you're looking for. I like to do 6–8 participants, but if you have time and want to you could do as many as 15 or 20, but you'd probably start seeing the same kinds of solutions way before you finished that many.
I have all these designs...now what?

* We have all these designs, now what?
* Use this data to construct your profiles, or answer specific questions about your users. If you're directly involved in designing or building, you're very likely to start having ideas while you're watching or running the sessions. Write them down, and save them for later. Note all the patterns you find in the solutions, what is it that everyone is doing or saying similarly? What are they all doing differently? How is what they came up with different than your initial ideas for solving this problem? How is it the same? Because this is open source, it's important to share your information with your project community so they can benefit from it too. Remember, above all else, these results are here to guide you, and to help you think like the people using your software, not to give you a pool of answers to choose from.
One more thing...
Usability Testing

* Usability Testing -- A quick pitch for something everyone should be doing in every software project ever ever ever
* Like QA before you've even built anything
  * Usability tests are at their best when you run them before you write a single line of code. Test a prototype, test a sketch, a system diagram (if it's for a technical audience). Test something that's cheap and fast to make that represents your final product. That way you don't have to throw away stuff people spent a lot of time building.
* or even if you've built something
  * In reality, it rarely happens that usability tests happen before any thing is built, or at least in my experience. You're either testing the current version of something to figure out what you need to work on for the next version, or you're running usability tests at the last second before it's too late to make any more changes. That last scenario is a particularly bad idea. If you're already hours away from QA, usability testing isn't going to save you anything, except it might give you a head start on problems you're going to uncover when you release.
* You too can plan and run a usability test
  * Tasks -- it's what people are already doing in your software. Come up with a few tasks that you want to test. Usually they're the main functions of your project, but whatever they are, make sure they're defined enough to have a specific end point. "Login to your account and check your balance" is an example of a usability task. It's testing both the login part of my software, as well as whatever I did to give people their balance information, and it's well defined enough that the participant will know when they're finished with the task.
* How to use the results

* Usability results are really like bug reports for designs. "4 out of 6 participants couldn't complete task 1 because of they couldn't find the right screen." They expose problems and often times give you additional insight into the way people are actually thinking about your software and what it enables them to do. You triage the problems from a usability study the same way you would bugs, and then start fixing them.
1. Design is planning
1. Design is planning
2. Human-centered == human needs first
1. Design is planning
2. Human-centered == human needs first
3. YOU ARE NOT YOUR USER
1. Design is planning
2. Human-centered == human needs first
3. YOU ARE NOT YOUR USER
4. Define your users and write it down
1. Design is planning
2. Human-centered == human needs first
3. YOU ARE NOT YOUR USER
4. Define your users and write it down
5. Empathy is key
1. Design is planning
2. Human-centered == human needs first
3. YOU ARE NOT YOUR USER
4. Define your users and write it down
5. Empathy is key
6. Participatory Design == best of asking and observing
1. Design is planning
2. Human-centered == human needs first
3. YOU ARE NOT YOUR USER
4. Define your users and write it down
5. Empathy is key
6. Participatory Design == best of asking and observing
7. Usability testing == QA for design
8. I’ll be at Expo Hall Table 1 from 3:20-4:20 for office hours
#HCDRulez

Questions?