The whole reason we make software is so someone can use it. Yet just who these users are—their needs, skills, and goals—is so hard to determine that they are often simply ignored.

INTERACTION DESIGN *helps you stop treating your customers like* AN ELEPHANT IN THE ROOM.

By Jeff Patton

"I THINK WE NEED TO SPEND MORE TIME THAN WE ORIGINALLY planned on the configuration user interface for this new feature I'm coding. The configuration is more complex than the rest of the application's configuration, and it takes a long time for the user to complete all the settings," Sylaja tells me.

Sylaja's new to working with this product, and this will be one of the first new features she's designed and built herself. It's important to her that the quality of the features is high.

Info to Go

- Start by describing users and their goals.
- Determine whom you most need to satisfy.
- Make design decisions based on these insights.

YVETTA FEDOROVA

Having said that, I know the customer is paying a pretty steep hourly rate for her time, so the sky *isn't* the limit when it comes to spending that time.

"What's this feature for again?" I ask.

"It's an optional plug-in for a feature some of our customers have requested. Our biggest customer is helping with the development cost to add it."

"If it's the configuration you're concerned about, remind me who will configure this feature."

"The application's administrator."

"What do you know about the person in that role? How skilled do you think this person is?" I ask. I know who the guy is for our primary customer, and he's pretty sharp. But I'm curious to see if Sylaja knows.

Sylaja does know and announces, "Rob, at our customer's site, is doing that job now. He knows the domain better than anyone, and he's pretty good with a computer. He knows the application very well, almost better than I do."

So Sylaja knows who's doing this configuration. "How often do you think he'll go to the configuration screen you're designing?"

"Well, just once when the feature is initially set up. Maybe again in a few months if he wants to change it to work differently."

While we'd never intentionally forget the users working with our software, **we can get a little disconnected from them.**

OK … she knows the task's frequency, let's get some idea of risk. "What if Rob configures it wrong? What would happen?"

"Well, the application would generate incorrect information. Then he'd have to change the configuration. It wouldn't damage anything. He'd just have to go back to the configuration screen, reconfigure, and then regenerate the information. That might take him a few minutes."

"So, given that this feature is for the application's most skilled user, this configuration task is performed infrequently, and the risk for doing it wrong is low, how much extra time do you think you should spend making the configuration screen more friendly?"

"Hmm … not much."

Of course, we'd never intentionally forget the users working with our software, but sometimes we get a little disconnected from them. Sometimes we attribute skills to them that they don't have or forget those skills they do have. Sometimes we forget how frequently a task in our software is performed and the value or risks associated with that task. Keeping those people front and center during the design, development, and testing of software is the primary goal of a class of techniques referred to as *interaction design*. The dialogue above, which took place recently between a co-worker and me, is an example of how to practice interaction design in real time.

In *The Inmates Are Running the Asylum*, Alan Cooper defines interaction design: "Interaction design refers to the selection of [software] behavior, function and information and their presentation to users." So what exactly does that mean? We can see that applying some different questions to the problem in the dialogue above changed the design and the development time of the add-on. In

INTERACTION DESIGN CLASS

The term "interaction design" describes a much broader class of activities than can be described in one article. You'll find authors and practitioners who have been working on specific approaches and techniques for as long as people have been writing software.

Interaction design is often considered to be an activity practiced in private by an enlightened few. These right-brained, creative types seem to have little connection with the left-brained engineers, QA professionals, and business people who normally populate a software project. What's different about the approach described in this article is that interaction design is considered to be a collaborative activity. Not only are fundamental interaction design principles easily approachable by anyone, they're useful throughout the entire software design, development, and deployment lifecycle.

When you practice interactive design as described above, you're leveraging the good effects that working collaboratively can bring (see Ellen Gottesdiener's *Requirements by Collaboration*). You've also started to edge into the world of agile software development (see Alistair Cockburn's book of the same name).

However, if the exercise in the article sounds interesting, give it a try. If interaction design concepts in general sound interesting, start with the references from this article, and then hit the Web for more interaction design resources. It's valuable to have an interaction design professional on your software development team. It's even more valuable to inject interaction design concepts into other parts of software development, starting from requirements gathering, through development and testing, to the final evaluation by all the important people who actually get value from the software.

Keep your eye on the elephant in the room. Just as other goals for the software project should be kept visible throughout the software project, so should the people using your software and *their* goals. Use prominently posted reminders, such as the role model built in this exercise, to keep your users present throughout the development of the software.

fact, if we apply interaction design principles, it can change the way the software development process works from requirements through to test QA.

Placing users and goals at the epicenter of our design and development decisions can impact the design of the resulting product as well as our day-to-day decisions throughout development and testing. Once we understand that, users and their goals are hard to ignore. The expression "ignoring the elephant in the living room" comes to mind. The elephant in the living room is visible to all, but we collectively choose to ignore it because we don't know what to do about it. The people who use our software loom equally large in our development process. Yet many steps are taken to push them pretty far into the background, to make them more abstract, to make them, somehow, less important.

Assuming you're somehow involved in software development, as you work with a feature in your software ask yourself what you know about the people who will be using it. Who are they? How much do they know about their domain? … your company's software? … software in general? Where are they? What kind of working environment do you picture them in? What is their goal at the time they're using the software? Is the software, as it is, helping them meet that goal? If you start to answer those questions, you'll find the heart of interaction design—you'll acknowledge the elephant.

Build a User Role Model

While practitioners of interaction design may use different specific approaches, they will agree that the design of the software starts with understanding the people who intend to use and benefit from it. Start with identifying who cares about this software. There are lots of possible ways to do that.

One approach that I've found works well is identifying user roles.

Let's say you need to write a new piece of software or add features or test revisions to an existing piece. In each of the cases, you have some idea of the software features that will be, or already have been, built. Let's build a user role model to put some human context around those features.

You'll use low-tech tools like 3×5 cards, sticky poster paper, markers, tape, Post-It notes, and individually wrapped pieces of candy. (I'm partial to bite-sized Tootsie Rolls, myself.) These easy-to-use, high-touch tools get both the left and right brains engaged in the process. You should try to involve the people with goals for this software—the actual stakeholders. Involve users who will end up using this software. Involve designers, developers, and QA people. Sometimes it's hard to get each important group represented, but keep the number at around eight. Involve as many people in your immediate team as possible. Be sure to do this as a collaborative activity. It's the discussion among participants that really helps important details emerge. If you're just experimenting with this technique, you don't need to gather all these people right now, but do involve a couple of people who know something about those using the software.

STEP 1

Determine who these people are and what they do

Start by brainstorming user roles onto 3×5 cards. "Role" is often used to refer to a specific responsibility taken on by a user. For example, I may be the Assistant Night Manager at my local fast food establishment, but when the store shuts down, I take on the role of Janitor in order to get the place cleaned up for tomorrow. Larry Constantine and Lucy Lockwood's *Usage-Centered Design* puts particular emphasis on choosing roles that describe the goals of the user. Using U-CD, a more appropriate role name than Janitor might be Restaurant Tidier. Even better might be Hurried Restaurant Tidier, because that name indicates not only what my goal is but my frame of mind while doing it.

When I'm identifying user roles, I prefer the form "thing-doer" (like Restaurant Tidier, just mentioned). The adjective "hurried" adds a bit of context and color to our Restaurant Tidier. With a little discussion and imagination, we can draw a mental picture of this Hurried-Restaurant-Tidier guy. He looks just like my brother-in-law.

Now that we understand the "role," let's start. Constantine and Lockwood refer to this type of brainstorming as "cardstorming." I choose one person to be the recorder. Start the brainstorming by calling out user roles. With everyone calling them out, the recorder should be writing as fast as he can. Remember the primary rule of brainstorming: do not judge any of the suggestions made, just record them, and shut up. I find it helpful to shout out obviously silly responses; "Lion Tamer" is a favorite. In addition to lightening things up, it often loosens something in participants' heads and good ideas follow. Stop brainstorming when 2–3 seconds pass between new suggestions popping up—the same way you know when microwave popcorn is done.

After brainstorming, you'll boil down your role candidates. Sift through the roles and remove duplicates and those silly suggestions you made while brainstorming. For each remaining candidate role, look at the role name. If it's not already in the "thing-doer" form, can it be converted to it? For extra credit, can you advance it to the "adjective-thing-doer" form?

STEP 2

Decide what these people want from you

Now that you've got the properly named roles, determine the goals for each. The name of the role may get you most of the way there, but make sure it specifically states what this user's goals are.

Be careful—the goals may not be what you expect. Management may believe that a call center employee's goals are to quickly and effectively answer as many calls as possible. The employee's own goals, however, may be to get through the workday without incident, to look good to their boss, and to not have the software they use make them feel stupid.

Write the goal or goals for each role in short statements on the 3×5 cards.

STEP 3

Figure out what these people have to do with each other

Now we'll start building a model that helps show how user roles relate to each other. Grab a sheet of poster paper and lay it on a large flat surface. Take up the role cards, and start placing them on the poster paper in an arrangement that feels right. Don't think too hard about it, but try to cluster the cards a bit. Do this by placing similar role cards closer together or overlapping, and placing dissimilar role cards far apart. The group should discuss the arrangement until they decide it feels correct.

Using clear tape, fix the cards to the sheet of poster paper. This is the start of a good role model.

STEP 4

It's time for the candy

Each participant should grab a couple of pieces of candy. Look at the arrangement of roles and the goals of each. Of these roles, which is most important to satisfy for this software to be successful? Alternatively, ask which of these roles will we be in big trouble with if we don't satisfy them. Each participant should vote for the role that answers those questions by placing a piece of candy on that role.

As you place your candy votes, you'll find that not everyone agrees. The reasons for those disagreements are important to understand. Remove the candy, prominently writing a letter "F" for each piece of candy you remove on the bottom left corner of the card. Some cards may have two, three, or more Fs. These are our *focal* roles—the ones we need to focus on. The more Fs, the higher the priority. Now eat the candy.

STEP 5

Discover hidden meanings

Now look more closely at the clusters. Why did they cluster? What makes them similar? Circle the cluster and label it with what makes these roles similar. While doing this we often end up with labels like "back-office accounting," "customers," or "process oversight people." Is there a common goal that all these roles share? Label them with whatever makes sense to the team.

Now look at all the clusters. Do they relate to each other? Wherever possible, draw a line from one cluster to another and label that line. We often end up with line labels like "sends work orders to," "supervises," or "tries to hide from."

Look at where the role cards are placed on the poster paper. Some cards are higher than other cards and some cards lower; some cards are to the left and some to the right. It's common to find those who participate early in a process located on the top left and those with later participation on the bottom right. Senior roles often end up at the top of the model. The imaginary x-axis seems to approximate time, while the virtual y-axis indicates seniority. Look for evidence that your roles are arranged on axis lines, decide what they are, draw some lines on the model, and label them.

STEP 6

Take note of interesting details

Now it's time to add whatever other markings or notations to the model you want. Are there other bits of information about these users that are important? Think of their work environments—are they hectic or peaceful? Think of their skill levels—are they highly skilled, or just trying to get by? Think of their computer experience—are they real geeks, or do mice frighten them? Make notes on the model about these things and any other interesting details you can think of.

Lastly, give this model a title that indicates which software the role model pertains to. Something such as "Widget Performance Analysis Role Model."

This should now look like a kindergarten art project

Congratulate yourselves. You may even want to grab a blanket and a graham cracker and take a short nap. (I'm only partially joking. When everyone is fully engaged in the process, a huge amount of information is being exchanged over the creation of this model. It can be pretty exhausting.)

What you've built is a wildly creative variation of a user role model from Constantine and Lockwood's *Usage-Centered Design*. This form of role model has a few strong advantages and a couple of disadvantages.

You'll find that when you look at the role model it will remind you of the conversations you had while creating it. It will be easier to recall specific users and the reasons behind the goals you captured. The notations you invented will remind you of the relationships the roles have with each other. Knowing which are the focal roles will help you make critical decisions about the software, and how to design and test it.

On the other hand, while there's a lot of information packed in this role model, it's very specific to those present during its creation. When you need to describe the people using your software, you can't simply direct others to the role model. You'll have to stop and explain your kindergarten artwork. After some initial shock, they'll catch on.

Also, you can't easily email this model to someone. You can't give it to top-level executives without some orientation. In the same way, the model can't be used to permanently document details about the project, since much of its value is in the minds of the people who created it.

Post your elephant in a prominent place

In his book *Agile Software Development,* Alistair Cockburn describes the concept of an information radiator. When posted in a prominent place where your team can see the model, you'll find it radiates information, like heat, into the room. As the development project progresses, you'll find the model becomes the elephant in the room—only now you'll know why it's there, what it means, and have some ideas on how to use it when specific decision points arrive during your project. As other people pass through your area, notice their reactions. Do they pretend the elephant's not there, or do they stop and ask questions? **{end}**

Jeff Patton works with Tomax Corporation, designing and developing software for chain retailers. He's active in agile software development, object-oriented design, and interaction design.

Sticky Notes

For more on the following topics go to www.bettersoftware.com

- Pictures of Interaction Design in progress
- Further reading

page 23
full-page ad
Serena