IEEE Software
www.computer.org/software

Understanding User Centricity

Jeff Patton

Vol. 24, No. 6
November/December 2007

Jeff Patton ■ ThoughtWorks ■ jpatton@acm.org

Understanding User Centricity

Jeff Patton

My goal for this column is to discuss user centricity as it applies to software design and development. While I'm happy to rant for 1,500 words from a soapbox every couple of months about how I think software development could and should be better, it might be more useful to illustrate what I'm talking about using real-world examples of common problems. In many cases, the example illustrates a common problem in software development. And rather than leave you with a problem, I'll describe a simple technique to help solve the problem. The emphasis is on simple. Applying the technique will be a first—but hopefully not last—step toward designing and building better software.

Developing software people can use

Compared to designing and constructing buildings, cars, or washing machines, software development is still an immature discipline. It's reasonably complex, and we're learning more every day. Consequently, how we design and develop software is constantly changing. Given the complexity and change, it's easy to get lost in building the software and forget that, like buildings, cars, and washing machines, our software is a tool that people will use to improve their lives.

We often focus on being productive designers and coders without realizing that the consumer of our code doesn't much care about productivity or elegance of design. What matters is how effectively the tool we build helps the consumer reach a goal or solve a problem.

Of course, this doesn't mean that design quality and effective coding aren't important—just that if you miss the "build something useful for people" target, nothing else matters. So to be successful, we must spend a fair bit of time trying to understand who uses what we build and how they use it. What do you do in your software development process to understand this? How do you apply that understanding to design, develop, and validate your software?

Redefining user emphasis

"We care a lot about users." This is a phrase I hear often, and I suspect you do too. And those who say it likely believe it.

When making decisions about what software should do and how it should do it, I commonly hear, "That will be hard to use" and "Users need to be able to do such and such"—and it's often from two teammates arguing with each other. When pressed further, I usually learn that the teammates don't actually know much about their users. They're imagining some fictitious user, usually somebody very much like themselves. The truth is, designers and developers—myself included—often think about users. However, in the absence of a strong mental model of specific users, we self-substitute. Self-substitution isn't user-centric—it's self-centric. Expressing concern for users without understanding them leads to self-centric evaluation.

Think back to those on your team engaged in arguments about what users would or wouldn't like or do or don't need. Is that argument based on an understanding of the users or on personal bias and assumption?

Redefining user involvement

I also often hear, "We involve our users continuously throughout the development process." But when I ask about the type of involvement, I usually hear stories about meetings in which designers and developers ask their users what software they'd like built. The users often respond by describing software functionality—after a fair bit of head scratching, of course. Developers then build software to meet those "requirements." After seeing the software, the users often say, "That's not quite it … change this to that." The developers then often respond: "You're changing your requirements. That's going to delay the project."

Although asking users what they want might seem like a good thing, in the end, we only understand their guess of the tool they need built. We still don't understand much about them or how they intend to use the tool to solve problems or achieve goals.

A friend of mine recently told me how his company's development process had changed over the years. One motivation behind the change was when their CIO said in a meeting one day, "We spend a huge amount of money writing software for the people in this company. Do we know if they're even using this stuff?" After watching a little head scratching and listening to some hemming and hawing, the organization appointed someone to head "user experience."

This head of user experience uses adoption rate as his primary metric for success. The development organization, which had always directly involved its users, found that users didn't actually like the applications they had described. The organization realized it had to change how it involved users—to treat them as experts in what they knew and the organization's software developers as experts in software design and development. This meant the developers had to watch their users work with the software and ask them about how they worked and what their goals were. The development organization could then prototype software that their users could test to see if it met their objectives. The result was software with higher adoption rates.

How much application specification do you delegate to users in the process of requirements gathering? Who's responsible when software doesn't meet user's objectives—the development organization or the users themselves?

Creating a user model

If you're in the process of building software, you're likely not completely in the dark about who your users are and why they use the software. So, try to distill a common understanding of your users into a user model.

Give this simple user-model creation technique a try: assemble a group of four to six people who know something about your software's target users, then grab a deck of index cards and several felt-tip markers. (I like to build models using index cards, because it makes it easy to rearrange or rebuild parts of the model.)

Brainstorm on user types

If we were building an internal accounting system, our user types might include an accounts-payable clerk, an accounts-receivable clerk, and a chief financial officer. If you're using a use-case driven approach, these might be the "actors" from your use cases. You might also think of users in terms of the role they play in the system such as "AP voucher enterer" or "payable approver."

Don't worry about which approach you use, just come up with a list. Write each user type on its own index card. A typical piece of software has at least a half-dozen user types—and usually lots more. If you're only coming up with two or three, think harder. This arrangement of cards is the beginning of your user model.

Discuss relevant user-type characteristics

Starting with the user type you believe is most critical to your system's success, discuss with your group various user characteristics:

- *Number of users*. How large is the user group (five, 10, …, 100,000)?
- *Typical activities*. What does a typical user type do with the software?
- *Computer skills*. How much does this group's computer and software expertise vary?
- *Domain expertise*. How much do they know about the subject of the software you're writing? (For example, a bank president knows more about banking than the typical ATM customer.)
- *Goals*. What are their goals (outside the software)? For example, when using an email application, a simple objective is to "send messages," but the real goal might be keep in touch with friends.
- *Pains*. What problems will using this software solve?
- *Technology ecosystem*. What other software or technology does this user group likely use? Knowing this gives us clues to what this user may be accustomed to using and a valuable source of usage idioms to borrow. For instance, I can easily get away with spreadsheet-usage idioms for a typical banker.
- *Usage frequency*. How often is this user type likely to use the software? Hourly? Daily? Weekly? Monthly?

Start with these general categories of characteristics. For each one, discuss and record what you know about the user

Although asking users what they want might seem like a good thing, we only understand their guess of the tool they need built.

types and look for "differences that make a difference." In other words, look for differences that would compel you to design the software differently for one user constituency than another.

Discuss and record design implications

Of course, none of this matters if it doesn't in some way affect your subsequent design decisions. For each user type, discuss and record on cards the opportunities for functionality that would be particularly helpful to this user and the characteristics that the software should have to be most valuable to this user.

For example, I recently worked with a group to help design a piece of software for research chemists. The target users were very smart people with generally strong computer skills, but they were going to use the software no more than once a month. The low usage frequency implied that the software had to be easy to learn. Despite the users' high domain knowledge and computer skills, we had to make it easy to remember how to use the software. Furthermore, the users had to process several batches of information at a time. Prior versions of the software made users wait for each batch to finish before creating the next, but the chemists needed the results for later analysis. So the new version of the software let them start several batches and emailed them the results when they became available.

Building useful software relies on understanding users, their goals, and their problems, and on determining the software that will help them reach those goals and solve those problems. User centricity isn't just caring about users or asking them what they want. It's understanding them and collaborating effectively with them to help make informed choices about what software to build.

Jeff Patton teaches, consults, and coaches independently and as a member of ThoughtWorks. He is the founder and list moderator of the agile-usability Yahoo discussion group, a columnist for StickyMinds.com, and the winner of the Agile Alliance's 2007 Gordon Pask Award for contributions to agile development. Contact him at jpatton@acm.org; www.agileproductdesign.com.

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Rose-Hulman Institute of Technology
Tenure-Track Faculty/Director of Software Engineering

The Department of Computer Science and Software Engineering invites applications for a tenure-track faculty position in software engineering beginning August 2008. Applicants may also be considered for the position of Director of Software Engineering. Rank and salary will be commensurate with experience. Applicants with significant software engineering experience are particularly encouraged to apply. We are especially interested in candidates who can contribute to our software engineering expertise in one or more of the following areas: software project management, software testing and quality assurance, software maintenance and evolution and software construction.

The Director of Software Engineering oversees all aspects of the software engineering program within the department and works in close conjunction with the department head. The department recently began offering a B.S. degree in software engineering to complement its B.S. degree in computer science. Both programs are ABET accredited.

Rose-Hulman is a highly selective, coeducational, primarily undergraduate college of engineering and science, with a vision to be the best institution of its kind in the world (www.rose-hulman.edu). Faculty members are expected to be outstanding teachers. Ongoing professional development is expected of all faculty so that their teaching continues to be outstanding. A Ph.D. in computer science, software engineering, or a closely related field is required.

Detailed information is available from: J.P. Mellor (CM 120), Associate Professor and Interim Head, Computer Science and Software Engineering, Rose-Hulman Institute of Technology, 5500 Wabash Ave., Terre Haute, IN 47803. Phone:(812)877-8085 Web: www.cs.rose-hulman.edu/faq.phtml. Applicants should submit a cover letter, a resume, a statement on their teaching philosophy, a statement on their professional development goals, copies of graduate school transcripts, and three letters of recommendation to https://jobs.rose-hulman.edu. EEO/AA

IEEE Software **How to Reach Us**

Writers
For detailed information on submitting articles, write for our Editorial Guidelines (software@computer.org) or access www.computer.org/software/author.htm.

Letters to the Editor
Send letters to

Editor, *IEEE Software*
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
software@computer.org

Please provide an email address or daytime phone number with your letter.

On the Web
Access www.computer.org/software for information about *IEEE Software*.

Subscribe
Visit www.computer.org/subscribe.

Subscription Change of Address
Send change-of-address requests for magazinesubscriptions to address.change@ieee.org. Be sure to specify *IEEE Software*.

Membership Change of Address
Send change-of-address requests for IEEE and Computer Society membership to member.services@ieee.org.

Missing or Damaged Copies
If you are missing an issue or you received a damaged copy, contact help@computer.org.

Reprints of Articles
For price information or to order reprints, send email to software@computer.org or fax +1 714 821 4010.

Reprint Permission
To obtain permission to reprint an article, contact the Intellectual Property Rights Office at copyrights@ieee.org.