April 2008 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

GET TO YES
How to win management buy-in

REV UP REUSE
Popular source code search engines

Incremental and Iterative Development

HOW they are different and WHY you should be doing both

Learning the Hardware Lessons

by Michael Bolton

ISTOCKPHOTO

The other night, I went to a local hardware store to replace a broken piece of a shelving unit in my daughter's closet. The store was closed, so I went to the competitor across the street—a chain store that I usually avoid because of patterns of familiar problems. I found something that looked like the part that I needed and took it to the cashier. I spent the usual several minutes in line, watching the cashiers address one problem after another related to the point-of-sale system that the store uses. Some items were missing the barcode stickers that allow items to be scanned and recognized by the software; the prices of some sale items were inconsistent with the prices displayed on the shelves; and some items apparently weren't in the system at all, even though they had seemingly legitimate stickers. Each customer transaction took several unnecessary minutes to resolve.

The fellow ahead of me was sipping a cup of coffee. I should have gotten one for myself, I thought—we're likely to be here for a while. As usual. The coffee drinker looked around. "Bloody computers," he said to me. "They're always broken at this store."

"That's my experience, too," I replied. "Although, technically, the *computers* are probably working just fine; it's the *programs* that are broken." We had time for a chat, and eventually he learned that I was a software tester.

"Ha! I guess the program for *this* system wasn't tested very well," my new friend said.

"Well, without specific information about the project, I can't be sure of that," I replied. "When we testers find a problem, it's up to programmers and managers to decide what to do about it. They might decide to fix it, or they might decide that the problem isn't serious enough to bother with. They often decide not to fix problems because they perceive that it might be expensive or risky, which can be a reasonable decision in a lot of cases. On the other hand, you only really get to find out about the quality of your risk assessment by paying attention to what happens in the field."

He finished his cup of coffee. "It doesn't look like they're paying much attention here," he said.

"It doesn't, does it? It's too bad, because confusion, delays, or annoyances—for the sales clerks or the customers—are real problems. They affect employee morale, the length of time that it takes to pump a sale through the system, and things that the customers value—like their time. Ultimately, *that* affects the bottom line. The store needs more cashiers to handle the same number of customers, or employees get frustrated and quit."

"Or customers head for the competition."

"Right. Managers here don't seem to observe the problems that the cashiers are having, and they don't seem to take notice of the amount of time that customers spend in line. I've been avoiding this chain for years because I can depend on having to wait more time than I think is reasonable. I only come here when I'm desperate."

"Me too," he said. "Don't managers realize how much that costs them?"

"Well, people say that missed opportunities are hard to measure. Plus, it's hard to evaluate things when you don't see how they interact with the rest of the system. Most testers sit in front of computer screens, testing the software but not the process that it's designed to support. If they tested the whole system, good testers would discover more important weaknesses, and they'd be able to tell better stories about how the problems threaten value. Smart developers and smart managers would notice possibilities for increased value if the system worked better. I can't say much about the testing, but I can pretty much guarantee that the system hasn't been developed, or managed, very well."

It was finally my new friend's turn to go through the checkout. He did so without incident. He looked over his shoulder and grinned as he left, "Good luck."

My turn. I realized to my horror that my shelf bracket didn't have a sticker on it. "It costs a dollar ninety-seven," I said weakly. The cashier asked me if I could wait for someone to do a price check. I had noticed that staff members were constantly being paged for price checks, and that people were waiting a long time for the information to come back. I decided to go and get the information myself.

There was a label on the shelf with a description, a bar code, and two numbers: 71924-20 and A434-300. There was no picture to match the product with the code, but most of the brackets in the same bin looked the same as the one I had. The description of the product was easy to remember. I fumbled for my notebook, but I had left it in my other jacket. I had a receipt in my pocket, but no pen and no staff around to lend me one. I memorized the numbers using some mnemonic tricks (7pm is 1900 hours, of which there are 24 in a day, 20 is easy. A's the first letter of the alphabet, 434's a palindrome, and 300 is a lousy movie.) I took the long walk back to the cashier.

"71924-20," I said. She typed it in.

"Wait. That's too many numbers," she said, clearly frustrated.

I said, "The only things that work in this place are the people. I really do appreciate that you're struggling here." She smiled. "What about A434-300?"

She tried it. "Nope. Six numbers is right, but it can't start with a letter." I looked all over the item. There was a number on it—but only four digits.

"What about dropping the A?" I suggested. "434-300?"

She tried backspacing, but that displayed only equal signs in the text field. Shift-backspace allowed her to backspace over the whole field, and she typed in the number. "Ah, that works. Shelf bracket. A dollar ninety-seven."

That was a lot of work for a two-dollar part, and the system didn't do very much to help. Why are there two numbers on the bin label? Why is neither number identified? Why does the software accept neither and reject both? Why isn't the acceptable number printed on the label? When I mistype something in the Google search window that doesn't match an entry in its database, Google offers a plausible suggestion—a close-but-not-exact match. Couldn't the store's system have tried a database search based on the numeric portion of the number? Couldn't it have cross-referenced one catalog with another to offer at least a choice or a guess as to what the product might be? Meanwhile, the clerk had an idea about the correct number format, but not how to deal with an incorrect number. Had management realized this as a training issue?

I'll guess that when the developers for this product designed, built, and tested it, they thought in terms of confirming that it worked. I'll guess that when they tested it, they used all kinds of boundary testing and field validation checks to make sure that the software accepted properly formatted numbers and rejected improperly formatted numbers. Those things are important. But I'll also guess that they tested the software in isolation from the environment in which it was intended to operate, and that they looked for functional correctness without testing the product in the field and asking, "Is there a problem here?" And finally, I'll guess that, to this day, they don't know about the frustrations that customers and cashiers alike are having with the system.

Systems and software aren't just about correctness; they're also about solving problems for people. One principle of the context-driven software testing movement says that if the problem isn't solved, the product doesn't work. Testers: Could we find more bugs—and *more important* bugs—in our systems by observing something other than the software itself? And managers: What problems could we prevent by letting our testers see more than some screens and some specifications? **{end}**

How would you answer the questions above? What additional questions should we be asking?

▼

Follow the link on the **StickyMinds.com** homepage to join the conversation.

26TH ANNUAL
PACIFIC NW SOFTWARE QUALITY CONFERENCE
October 13 -15, 2008
Portland, OR

World-class quality does not happen in a vacuum. Agile-inspired collaboration spans levels, disciplines, and industries.

2008 keynote speakers Ron Jeffries and Chet Hendrickson will address *Quality Dynamics in Agile Software Development.*

www.pnsqc.org