Kicking Off the Slow Software Movement

by Jeff Patton

Over breakfast, an old friend who happens to run a software development company was complaining that too many of his people focus on software development—engineering, requirements, and project management stuff. When confronted with a problem, they jump in and start gathering requirements, putting together project plans, and developing.

What is now a problem used to be cause for celebration.

He went on to explain that his staff members don't take enough time to understand and appreciate the problems they're trying to solve. They are quick to launch into a project, but the result—no matter how quickly or effectively it was built—just isn't right.

My friend wants people to better understand what success means before starting to build. He said, only half joking, "I want to start the Slow Software movement!" He was alluding to the Slow Food movement—a non-profit international group trying to bring diversity and quality back into the food we eat (see the StickyNotes for a link). His comments made sense to me and set me to wondering why we're in such a hurry.

Years ago it was incredible that we could build software at all. We were constrained by the tools we used—both the development languages and the computer systems. Clever people worked within these constraints to create software applications that allowed users to perform tasks that were nearly impossible before.

But constraints are changing. Computers are now commonplace and every place. Our mobile phones have more power than computers that once filled entire rooms. There is a variety of programming languages that leverage libraries of code to solve problems so we don't have to solve them any more.

As technical constraints lift, computers and software have become ubiquitous. Computers—once only operated by scientists—are now in my daughter's kindergarten classroom. With the proliferation of software-based tools has come the expectation from users that these tools will be as good, as useful, and as valuable as other tools, like kitchen appliances or tools in their garages.

In his book *The Inmates Are Running the Asylum*, Alan Cooper refers to most software as "dancing bear-ware." It's not that the bear is a good dancer; we're just amazed that the bear dances at all. Things have changed since 1999 when Cooper wrote that. These days not only are we surrounded by dancing bears, but some of them are getting pretty good. We no longer consider acceptable the lumbering shuffle achieved by some software—we expect a tango.

In response, I've observed over the past dozen or so years a drift in the way we think about, design, and build software. While in the past there was more emphasis on engineering discipline and process, now they're not sufficient. When we sit down to use software today, we're not impressed by the engineering discipline that went into it, we don't ponder the process used to build it, and we don't wonder whether the product delivered on time with its intended scope. We're no longer amazed that we have software to use. We now expect software to work well. We expect that the value we get from using it is worth the price we pay for it.

Our development approaches have changed, too. Popular processes now place emphasis on delivering business value, not lines of code. The quality of software is increasingly judged not by its lack of bugs but by its usability. Writing high-quality, bug-free code no longer seems to be a measure of success. Delivering on time now seems less important than delivering value.

If value realized and quality of use really are our new measures of success, maybe it is time to slow down.

Before we plan and build, we should take time to understand what value is. Make sure the first thing we gather is how the people paying for the software will get value from it. That's likely not a list of features but rather a list of goals or a description of a world that's a little better because of this software. Then we will decide what to build based on what best meets those goals.

We also might want to understand better the people using our software. They've got goals, too, which likely are met using other software or manual processes today. What we decide to build should outperform what users already have. The quality of what we build will be judged alongside the other tools on which users currently rely.

Gathering business goals, talking to and observing people, and validating software with usability testing likely will slow us down. But maybe it wasn't really "fast" that we wanted in the first place. **{end}**

Jeff Patton leads teams of agile developers to build the best software possible. He proudly works at ThoughtWorks. Jeff's series of columns on software design and pre-design tips appears regularly on StickyMinds.com.

Sticky Notes

For more on the following topic go to www.StickyMinds.com/bettersoftware.

- Slow Food movement