Agile Development

by Brian Marick

In February, I participated in a convocation of people who've created or worked on "Agile Methods." You may not be familiar with that relatively new umbrella term, but individual Agile Methods have been gaining attention lately—the most well-known of these being eXtreme Programming (XP). Other examples include Adaptive Software Development, Crystal, Dynamic Systems Development Method, Feature-Driven Development, Scrum, and XBreed. Our group's goal was to find what these different methods had in common. The resulting *Manifesto for Agile Software Development* (posted at **www.AgileAlliance.org**) offers the statement of values shown in Figure 1.

For me, two key imperatives underlie these values.

Working Code

In his fine book *Hackers*, author Steven Levy describes the Hands-On Imperative. It says to get your hands on the computer and make it *do* something. Programmers are often in the grip of the Hands-On Imperative; that's why they don't like to write documentation. Agile Methods accept the Hands-On Imperative, but extend it to the users.

Why? Author Reinhard Keil-Slawik puts it this way: "Thinking does not take place inside our heads...Most of our mental activities need external resources." Different resources lead to different ways of thinking. Someone who adds with an abacus thinks differently about addition than does someone who uses the Indian decimal numbering system and a pencil and paper. And a user asked to evaluate an abstraction like a requirements document or a design model thinks profoundly differently than one presented with working software.

As German researchers Reinhard Budde and Heinz Züllighoven have written, "When we formalize, we explain how an object functions; when we work purposively with a thing, we understand what it means." So when we sit a user down in front of the screen for the first time, her reaction—"Wow, this is not what I expected"—doesn't represent any sort of failure. It's not that we didn't do a good job "capturing" requirements, or that the reviews weren't planned carefully enough, or that the user wasn't properly trained in our modeling notation, or anything like that. It's just that we humans simply cannot imagine the real experience by studying an abstraction.

Because Agile projects understand that, they deliver working software (or perhaps executable prototypes) as quickly as possible and as frequently as practical. Development is organized into a rapid series of functionally complete releases, each one made available for the user to try. Since each such release is really the first chance the user has had to think about the new features, rework is just part of the job—not a crisis.

Conversing People

With less documentation, how do Agile projects keep everyone in synch? With an imperative toward human contact: face-to-face conversation and collaboration. XP has people program in pairs and tries hard to have a customer representative working every day in the same bullpen as the developers. Another Agile method, Scrum, relies on carefully crafted daily standup meetings that create and preserve group understanding. Crystal, perhaps the least dogmatic process conceivable, nevertheless insists on frequent retrospectives. These techniques foster the communication that documents cannot replace.

All Agile methods want customers to be part of the team. With a suitable customer representative at hand, you don't need a detailed requirements document. When you have a question, you turn around and ask. Worried that you won't know to ask the right question? Implement something and show it to the customer for a quick reaction—you'll quickly learn if you're going off track.

In an Agile project, conversation is carefully designed to avoid a lot of idle chatter. Talk will be pragmatic, concentrating on some object of work. But there'll be a background of tacit understanding of the fundamentals, so

We are uncovering better ways of developing software by doing it and helping others do it. We value:

Individuals and interactions	over	processes and tools
Working software	over	comprehensive documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan

That is, while there is value in the items on the right, we value the items on the left more.

FIGURE 1 Manifesto for Agile Software Development

that different people will reflexively make consistent choices when confronted with similar problems.

Agile Testing

These same imperatives can underlie a practice of Agile *Testing* (**www.testing.com/agile**). Agile Testing would most obviously apply to Agile Development projects, but it should work—perhaps less well—on conventional projects too.

The first step is to abandon the notion that others communicate "at" us with requirements and design documents, and that we communicate back at them with test plans and bug reports. We've always realized that the documents we base our tests on are flawed—incomplete, incorrect, and ambiguous—but our reaction has been to insist, in our usually powerless way, that the document producers should do better. But now we can see that "better" will never be good enough. Documents can't be an adequate representation of working code. So we can let go of the illusion that documents will save us. We can view them as they are: interesting texts, partly fictional, often useful.

Rather than communicating *at* people, we need to join and encourage the ongoing project conversation. Testers and developers should sit in the same bullpen, share offices, or occupy alternate cubicles. Except for some types of testing in which it doesn't make sense (such as configuration testing and some types of stress testing), testers should be assigned to help particular developers, rather than to test pieces of the product. The test plan should evolve through a series of what testing consultant James Bach calls "drop-in meetings": short, low-preparation, informal discussions of particular topics. These will result in what Microsoft test manager James Tierney calls "test doclets": short memos addressing a specific issue. Test status should be reported via big, public, simple-to-read charts that answer specific development questions like "what parts of the product can we stop worrying about?"

Conversation with the customer is as important as with the developers. Remember: Customers are trying to figure out what they need, want, and are getting—in large part by trying out the working code. Testers should sit down with them as they do that. Creating some tests together is an excellent way for both of you to learn what matters—and also to describe it to the developers in a clear and concrete way.

That's an instance of the Hands-On Imperative. Exploit that with developers as well. The normally strained relationship with them will be less stressed if they see you wanting to get started testing, even on something unfinished, especially if your expressed goal is to help them improve and complete it. They'll value tests they can run as they continue development.

Agile Testing is not the answer for all projects. Neither is any of the Agile Methods named at the beginning of this article. No single approach *can* be. But the Agile approach is a conscious and well-crafted departure from conventional software development styles—one that deserves your careful attention. STQE

Brian Marick (marick@testing.com, www.testing.com) *is an* STQE *technical editor. [Author's note: Quotes are from* Software Development and Reality Construction.*]*

STQE **magazine is produced by STQE Publishing, a division of Software Quality Engineering.**