Dr. Dobb's Portal
The World of Software Development

The Fragile Manifesto

August 2001 proved to be a watershed issue for Software Development magazine, with the publication of "The Agile Manifesto" by Martin Fowler and Jim Highsmith. Since then, we've published many articles about agile software development and have even repositioned this column to focus on this emerging discipline. In doing so, we may have become biased and ignored existing and proven development approaches in the process. It's time to rectify this situation by presenting an alternative to the Agile Manifesto.

By Scott Ambler, Software Development
Aug 01, 2002
URL:http://www.ddj.com/architect/184414888

August 2001 proved to be a watershed issue for *Software Development* magazine, with the publication of "The Agile Manifesto" by Martin Fowler and Jim Highsmith. Since then, we've published many articles about agile software development and have even repositioned this column to focus on this emerging discipline. In doing so, we may have become biased and ignored existing and proven development approaches in the process. It's time to rectify this situation by presenting an alternative to the Agile Manifesto.

To counteract the Agile Alliance's (www.agilealliance.org) subversive rhetoric, I've set down a collection of values more suitable for traditional software development environments, particularly those found in large corporations or government agencies. These values define preferences—while the concluding phrases describe important items, traditional developers should value the initial items more. These four values should be used as a philosophical foundation for your software development activities.

Processes and Tools Over Individuals and Interactions
Software development is an engineering discipline, and as such, should depend upon a highly rigorous collection of procedures. Some would argue that this approach didn't work in the past because it resulted in shelves of binders that developers ignored. Luckily, this problem has been solved—you can now write your development procedures as HTML pages, providing developers with easy access to the software processes that they so desperately crave.

You can also reduce your payroll with a well-defined process, because you'll no longer need to hire highly skilled developers. Instead, you'll only need people who can follow instructions—a clear cost savings for your organization. Just think of the quality of the work that will be produced with everyone following common procedures.

Unfortunately, a well-defined process isn't sufficient: You'll also need good tools that support your process. Developers will require a robust set of tools to help them complete the status reports, time sheets, traceability matrices, meeting agendas and minutes that are critical to the success of any software project. You should consider a workflow system to integrate and manage these tools, configured to reflect the complex reporting structures among the individual specialists on your team.

Critical to your success is the definition of a standard development workstation configuration, going beyond hardware issues to include software tools such as a code editor, compiler and testing environment. When developers are given the freedom to choose their own tools, they'll naïvely select products specific to the task at hand instead of your organization's standard toolset. Worse yet, they may even download open-source software tools—yet another reason to restrict developers access to the Net—and clearly, anything that's free mustn't be any good. Developers don't realize that tools from a single vendor will reduce the license-negotiating burden on your purchasing department. A standardized development workstation also eases the configuration management burden of your internal support group, and when you do it right, you can deploy the same configuration for several years without having to revisit your decisions. You clearly don't need to be "agile" to have a streamlined process.

Comprehensive Documentation Over Working Software

Customers want lots of documentation—that's why they're paying you to develop software. Senior management wants assurance that you understand what should be built; hence the need to create a comprehensive requirements specification. You'll also want to invest significant time in up-front modeling and documenting the architecture. Architects are your smartest people, so you know they'll be able to get it right the first time, particularly if you leave them alone. They shouldn't dirty their hands by writing code; instead, they can base their models on the documentation provided by vendors and reports by industry experts, because, as we all know, technology always works exactly as claimed. Fortunately, most architects wrote Cobol code years ago, so they can apply that experience to modern J2EE and .NET systems. This approach works because developers have great respect for architects and will always follow their documentation to the letter.

It's important to have comprehensive documentation, so you don't lose critical knowledge when someone leaves the team. For many organizations, this is a critical issue because they just don't seem to be able to retain their good people (yet another reason to focus on process), and once the economy picks up, many organizations can expect to lose good people in droves. Developers are really flaky that way, so you'd better get them writing documentation while they're still working for you.

Comprehensive documentation is critical for future maintenance efforts, as well. Maintenance professionals want as much system documentation as they can get, the more minutely detailed the better, because they can trust such documentation to be accurate. With comprehensive documentation, maintainers won't need to talk to the original developers (who probably aren't around anyway), and better yet, they won't even have to learn the source code at least, this is what a lot of my management friends tell me.

With comprehensive documentation, you can show that you actually got something done when your project fails or funding gets cut. This strategy seems to work for many big consulting firms: They've had projects that go on for years that produce a hefty "telephone book" instead of a working product, only to nab a similar contract from senior management for another "system." If it works for them, why not you?

Contract Negotiation Over Customer Collaboration

A critical goal for any software project is to write a comprehensive requirements document, review it, negotiate it down to a reasonable list of demands, and get customers to sign off on it before you start coding. This process effectively creates a contract with your customers that defines exactly what you're to build. Without such a contract in place, you're at serious risk if your customers change their minds—and they always do—or audaciously claim that what you delivered isn't what they asked for. You can't trust your customers—therefore, you need to protect yourself with a firmly negotiated contract.

Customer collaboration is overrated: More often than not, customers just get in the way, and, if you're smart, you can minimize their participation. At the beginning of a project, you need customers to provide initial requirements. They should also attend a series of reviews of your comprehensive documents to ensure that they're getting what they're asking for. After sign-off, they should just leave you alone to build the system. Then, months or years later, you might need your customers to be involved with user acceptance testing, although you're likely to cut that effort short because the project will probably be late and over budget by that point. The good news is that this approach reflects your customers' expectations. For some reason, they feel that you have little chance of success—apparently, they've forgotten all about that system you delivered five years ago—so they should be glad that you're doing the best you can to reduce their participation in the first place.

Following a Plan Over Responding to Change

A professional project manager, particularly one who is adept at creating and maintaining Gantt charts using Microsoft Project or another similar tool, is the most valuable person on any project. Even if she has to spend several days a week keeping the schedule up to date, it's a worthy investment because your developers desperately need the schedule's guiding wisdom to determine what to do next. Senior management and your customers want to see that you know what you're doing, and nothing says that better than a detailed schedule with hundreds or even thousands of minute, interconnected tasks. And when you print it out (something you should do at least weekly), you can use it to cover up wall space where someone might have put up a messy whiteboard.

Experience demonstrates that change is bad: It's much easier to plan and execute a project if you don't allow change to creep into it. This is another reason why you should freeze requirements early in the project, because your customers know they shouldn't introduce changes without arduous renegotiation of the project cost and schedule. Your customers need to learn to live with exactly what you decide to give them, should sacrifice changes, however "necessary," and embrace the plan instead.

And now for the punch line: As you may have gathered, this has all been an April Fools' joke. Since this is the April issue *[Editor's note: Because Scott took a fragile approach to developing this column, he turned it in four months late.]*, I thought it'd be fair to poke fun at some of the folks who doubt the veracity of agile software development. Warning: If you find that you're currently following some or all of this column's "advice," you might want to rethink your approach.