software development The lifecycle starts here.

The Agile Manifesto

August 2001

Facilitating change is more effective than attempting to prevent it. Learn to trust in your ability to respond to unpredictable events; it's more important than trusting in your ability to plan for disaster.

by Martin Fowler and Jim Highsmith

In the past 12–18 months, a wide range of publications—*Software Development, IEEE Software, Cutter IT Journal, Software Testing and Quality Engineering,* and even *The Economist*—has published articles on what Martin Fowler calls the New Methodology (see www.martinfowler.com/articles/newMethodology.html), reflecting a growing interest in these new approaches to software development (Extreme Programming, Crystal Methodologies, SCRUM, Adaptive Software Development, Feature-Driven Development and Dynamic Systems Development Methodology among them). In addition to these "named" methodologies, scores of organizations have developed their own "lighter" approach to building software.

Formation of the Agile Alliance

On February 11–13, 2001, at The Lodge at Snowbird ski resort in the Wasatch mountains of Utah, 17 people met to talk, ski, relax and try to find common ground. What emerged was the Agile Software Development Alliance.

A bigger gathering of organizational anarchists would be hard to find, so what emerged from this meeting was symbolic—a *Manifesto for Agile Software Development*—signed by all participants. Although the Manifesto provides some specifics, a deeper theme drives many Alliance members. At the close of the two-day meeting, Extreme Programming mentor Bob Martin joked that he was about to make a "mushy" statement. Though tinged with humor, Bob's sentiments were shared by the group—we all enjoyed working with people who shared compatible goals and values based on mutual trust and respect, promoting collaborative, people-focused organizational models, and building the types of professional communities in which we would want to work.

The agile methodology movement is not anti-methodology; in fact, many of us want to restore credibility to the word. We also want to restore a balance: We embrace modeling, but not merely to file some diagram in a dusty corporate repository. We embrace documentation, but not to waste reams of paper in never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment. Those who brand proponents of XP, SCRUM or any of the other agile methodologies as "hackers" are ignorant of both the methodologies and the original definition of the term (a "hacker" was first defined as a programmer who enjoys solving complex programming problems, rather than someone who practices ad hoc development or destruction).

Early on, Alistair Cockburn identified the general disgruntlement with the word *light:* "I don't mind the methodology being called light in weight, but I'm not sure I want to be referred to as a 'lightweight' attending a 'lightweight methodologists' meeting. It sounds like a bunch of skinny, feebleminded people trying to remember what day it is." So our first task was to come up with a new adjective that we could live with. Now our processes are "agile," even if some of us are a bit creaky.

The result of this meeting (and the ensuing frenzied online interaction) was the Agile Manifesto (see sidebar). While the purpose and principles of the Manifesto were developed by the entire group, we (Jim and Martin, both authors of the Manifesto) have added, for this article, our own interpretations and explanations.

The Agile Manifesto: Purpose

"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."

This statement has a number of fascinating aspects, not the least of which was getting 17 people to agree to it. First, the word *uncovering*. While this was a group of experienced and recognized software development "gurus," the word *uncovering* was selected to assure (or frighten) the audience that the Alliance members don't have all the answers and don't subscribe to the silver-bullet theory.

Second, the phrase *by doing it* indicates that the members actually practice these methods in their own work. Ken Schwaber (a proponent of SCRUM) told of his days of selling tools to automate comprehensive, "heavy" methodologies. Impressed by the responsiveness of Ken's company, Jeff Sutherland (SCRUM) asked him which of these heavy methodologies he used internally for development. "I still remember the look on Jeff's face," Ken remarked, "when I told him, 'None—if we used any of them, we'd be out of business!'"

Third, this group is about helping, not telling. The Alliance members want to help others with agile methods, and to further our own knowledge by learning from those we try to help.

The value statements have a form: In each bullet point, the first segment indicates a preference, while the latter segment describes an item that, though important, is of lesser priority. This distinction lies at the heart of agility, but simply asking people to list what's valuable doesn't flesh out essential differences. Roy Singham, Martin's boss at ThoughtWorks, put it well when he said that it's the edge cases, the hard choices, that interest him. "Yes, we value planning, comprehensive documentation, processes and tools. That's easy to say. The hard thing is to ask 'what do you value *more*?'"

The Alliance recognizes the importance of process and tools, with the additional recognition that the interaction of skilled individuals is of even greater importance. Similarly, comprehensive documentation is not necessarily bad, but the primary focus must remain on the final product—delivering working software. Therefore, every project team needs to determine for itself what documentation is absolutely essential.

Contract negotiation, whether through an internal project charter or external legal contract, isn't a bad practice, just an insufficient one. Contracts and project charters may provide some boundary conditions within which the parties can work, but only through ongoing collaboration can a development team hope to understand and deliver what the client wants.

No one can argue that following a plan is a good idea—right? Well, yes and no. In the turbulent world of business and technology, scrupulously following a plan can have dire consequences, even if it's executed faithfully. However carefully a plan is crafted, it becomes dangerous if it blinds you to change. We've examined plenty of successful projects and few, if any, delivered what was planned in the beginning, yet they succeeded because the development team was agile enough to respond again and again to external changes.

The Agile Manifesto: Principles

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. In a recent workshop, a software development manager questioned the feature or story approach to iterative cycle planning. "But aren't requirements specifications and architecture documents important?" he asked. "Yes," Jim replied, "They *are* important, but we need to understand that customers don't care about documents, UML diagrams or legacy integration. Customers care about whether or not you're delivering working software to them every development cycle—some piece of business functionality that proves to them that the evolving software application serves their business needs."

Implementing a "customer value" principle is one of those "easier said than done" activities. Traditional project management practices assume that achieving a plan equals project success equals demonstrated customer value. The volatility associated with today's projects demands that customer value be reevaluated frequently, and meeting original project plans may not have much bearing on a project's ultimate success.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. The growing unpredictability of the future is one of the most challenging aspects of the new economy. Turbulence—in both business and technology—causes change, which can be viewed either as a threat to be guarded against or as an opportunity to be embraced.

Rather than resist change, the agile approach strives to accommodate it as easily and efficiently as possible, while maintaining an awareness of its consequences. Although most people agree that feedback is important, they often ignore the fact that the result of accepted feedback is change. Agile methodologies harness this result, because their proponents understand that facilitating change is more effective than attempting to prevent it.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale. For many years, process gurus have been telling everyone to use an incremental or iterative style of software development, with multiple deliveries of ever-growing functionality. While the practice has grown in use, it's still not predominant; however, it's essential for agile projects. Furthermore, we push hard to reduce delivery cycle time.

However, remember that *deliver* is not the same as *release*. The business people may have valid reasons for not putting code into production every couple of weeks. We've seen projects that haven't achieved releasable functionality for a year or more. But that doesn't exempt them from the rapid cycle of internal deliveries that allows everyone to evaluate and learn from the growing product.

Business people and developers work together daily throughout the project. Many folks want to buy software the way they buy a car. They have a list of features in mind, they negotiate a price, and they pay for what they asked for. This simple buying model is appealing, but for most software projects, it doesn't work. So agile developers respond with a radical change in our concept of the requirements process.

For a start, we don't expect a detailed set of requirements to be signed off at the beginning of the project; rather, we see a high-level view of requirements that is subject to frequent change. Clearly, this is not enough to design and code, so the gap is closed with frequent interaction between the business people and the developers. The frequency of this contact often surprises people. We put "daily" in the principle to emphasize the software customer's continuing commitment to actively take part in, and indeed take joint responsibility for, the software project.

Build projects around motivated individuals, give them the environment and support they need and

trust them to get the job done. Deploy all the tools, technologies and processes you like, even our agile processes, but in the end, it's people who make the difference between success and failure. We realize that however hard we work in coming up with process ideas, the best we can hope for is a second-order effect on a project. So it's important to maximize that first-order people factor.

For many people, trust is the hardest thing to give. Decisions must be made by the people who know the most about the situation. This means that managers must trust their staff to make the decisions about the things they're paid to know about.

The most efficient and effective method of conveying information with and within a development team is face-to-face conversation. Inevitably, when discussing agile methodologies, the topic of documentation arises. Our opponents appear apoplectic at times, deriding our "lack" of documentation. It's enough to make us scream, "the issue is *not* documentation—the issue is *understanding*!" Yes, physical documentation has heft and substance, but the real measure of success is abstract: Will the people involved gain the understanding they need? Many of us are writers, but despite our awards and book sales, we know that writing is a difficult and inefficient communication medium. We use it because we have to, but most project teams can and should use more direct communication techniques.

"Tacit knowledge cannot be transferred by getting it out of people's heads and onto paper," writes Nancy Dixon in *Common Knowledge* (Harvard Business School Press, 2000). "Tacit knowledge can be transferred by moving the people who have the knowledge around. The reason is that tacit knowledge is not only the facts but the relationships among the facts—that is, how people might combine certain facts to deal with a specific situation." So the distinction between agile and document-centric methodologies is not one of extensive documentation versus no documentation; rather a differing concept of the blend of documentation and conversation required to elicit understanding.

Working software is the primary measure of progress. Too often, we've seen project teams who don't realize they're in trouble until a short time before delivery. They did the requirements on time, the design on time, maybe even the code on time, but testing and integration took much longer than they thought. We favor iterative development primarily because it provides milestones that can't be fudged, which imparts an accurate measure of the progress and a deeper understanding of the risks involved in any given project. As Chet Hendrickson, coauthor of *Extreme Programming Installed* (Addison-Wesley, 2000), remarks, "If a project is going to fail, I'd rather know that after one month than after 15."

"Working software is the measure of progress because there's no other way of capturing the subtleties of the requirements: Documents and diagrams are too abstract to let the user 'kick the tires,'" says Dave Thomas, coauthor of *The Pragmatic Programmer* (Addison-Wesley, 1999).

Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely. Our industry is characterized by long nights and weekends, during which people try to undo the errors of unresponsive planning. Ironically, these long hours don't actually lead to greater productivity. Martin and Kent Beck have often recalled working at companies where they spent all day removing errors made late the previous night.

Agility relies upon people who are alert and creative, and can maintain that alertness and creativity for the full length of a software development project. Sustainable development means finding a working pace (40 or so hours a week) that the team can sustain over time and remain healthy.

Continuous attention to technical excellence and good design enhances agility. When many people look at agile development, they see reminders of the "quick and dirty" RAD (Rapid Application Development) efforts of the last decade. But, while agile development is similar to RAD in terms of speed and flexibility, there's a big difference when it comes to technical cleanliness. Agile approaches emphasize quality of design, because design quality is essential to maintaining agility.

One of the tricky aspects, however, is the fact that agile processes assume and encourage the alteration of requirements while the code is being written. As such, design cannot be a purely up-front activity to be completed before construction. Instead, design is a continuous activity that's performed throughout the project. Each and every iteration will have design work.

The different agile processes emphasize different design styles. FDD has an explicit step at the beginning of each iteration in which design is executed, usually graphically with the UML. XP places great emphasis on refactoring to allow the design to evolve as development proceeds. But all of these processes borrow from each other: FDD uses refactoring as developers revisit earlier design decisions, and XP encourages short design sessions before coding tasks. In all cases, the project's design is enhanced continually throughout the project.

Simplicity—the art of maximizing the amount of work not done—is essential. Any software development task can be approached with a host of methods. In an agile project, it's particularly important to use simple approaches, because they're easier to change. It's easier to add something to a process that's too simple than it is to take something away from a process that's too complicated. Hence, there's a strong taste of minimalism in all the agile methods. Include only what everybody needs rather than what anybody needs, to make it easier for teams to add something that addresses their own particular needs.

"Simple, clear purpose and principles give rise to complex, intelligent behavior," says Dee Hock, former CEO of Visa International. "Complex rules and regulations give rise to simple, stupid behavior." No methodology can ever address all the complexity of a modern software project. Giving people a simple set of rules and encouraging their creativity will produce far better outcomes than imposing complex, rigid regulations.

The best architectures, requirements and designs emerge from self-organizing teams. Contrary to what you've heard, form doesn't follow function: Form follows failure. "The form of made things is always subject to change in response to their real or perceived shortcomings, their failures to function properly," writes Henry Petroski, civil engineering professor and author of *The Evolution of Useful Things* (Vintage Books, 1994). Stuart Brand writes that the "form follows function" idea has misled architects into believing that they could predict how buildings would actually be used.

Petroski's views are similar to one of the two key points of this principle—that the best designs (architectures, requirements) emerge from iterative development and use rather than from early plans. The second point of the principle is that emergent properties (*emergence,* a key property of complex systems, roughly translates to innovation and creativity in human organizations) are best generated from self-organizing teams in which the interactions are high and the process rules are few.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. Agile methods are not something you pick and follow slavishly. You may start with one of these processes, but we all recognize that we can't come up with the right process for every situation. So any agile team must refine and reflect as it goes along, constantly improving its practices in its local circumstances.

Jim has been working with a consulting company to develop an Adaptive Software Development–Extreme Programming combination methodology. The first team to use it modified it immediately. Martin has worked with a number of teams at ThoughtWorks to tailor Extreme Programming practices to various project situations. Trust in people, believing that individual capability and group interaction are key to success extends to trusting teams to monitor and improve their own development processes.

Toward an Agile Future

Early response to the Agile Manifesto has been gratifying. Several e-mails expressed sentiments such as, "My product manager has already posted the Manifesto on her wall." Many of Martin's colleagues at ThoughtWorks have popped in to say how much they shared the values.

One question that arose immediately was whether or not the Alliance was a precursor to what one conference attendee tagged a Unified Lightweight Methodology. Absolutely not! While the group believes that a set of common purposes and principles will benefit the users of agile methodologies, we are equally adamant that variety and diversity of practices are necessary. When it comes to methodologies, each project is different and each project team is different—there's no one-size-fits-all solution.

What of the future? We can confidently say that we don't know. Agility is all about trusting in one's ability to respond to unpredictable events more than trusting in one's ability to plan ahead for them. We also know that the personal relationships formed by our collaboration matter far more than the document that we've produced. One thing is clear: we've only just started.

[return to article]

The Manifesto for Agile Software Development
Seventeen anarchists agree:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to 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 we value the items on the right, we value the items on the left more.

We follow the following principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

—Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas
www.agileAlliance.org