DOI:10.1145/2133806.2133823

Even after almost a dozen years, they still deliver solid guidance for software development teams and their projects.

BY LAURIE WILLIAMS

What Agile Teams Think of Agile Principles

IN THE MID-1990s, the prescribed means of keeping software development projects out of trouble and on schedule was to follow a heavyweight software development methodology consisting of a complete requirements document, including architecture and design, followed by coding and testing based on a thorough test plan. The philosophy was often summarized as "Do it right the first time." Common belief among software engineers at the time was that projects run into trouble when they do not strictly adhere to a methodology, and, if only they did, all would be well. In reality, all was rarely well.

At the same time, a simmering undercurrent that had begun to undercut this doctrine was to follow an exceedingly iterative, lightweight software development methodology. Purportedly, a number of independent "rogue" consultants were rescuing projects in trouble through variations of these methodologies. The first to stand up and say, "Look at me," and attract wide attention, was Extreme Programming[1] in about 1999. The creators of other methodologies, including Adaptive Software Development, or ASD,[6] Crysta,[4] Dynamic Systems Development Method, or DSDM,[11] Feature Driven Development, or FDD,[8] and Scrum,[10] followed suit with "Hey, I'm doing something like that, too!"

Then, in February 2001, something remarkable happened: Rather than focus on their differences and the "competitive advantage" of their own methodologies, 17 creators and supporters[a] of the lightweight methodologies gathered in Snowbird, UT, to discuss their

[a] Software engineers in attendance in Snowbird included Kent Beck, Mike Beedle, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andy Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas, and Arie van Bennekum.

common interests and philosophies, coining the term "agile software development" to describe their methodologies. This unity rocked the software industry. In Snowbird, the Manifesto for Agile Software Development[b] and Principles Behind the Agile Manifesto[c] were born and endorsed by all 17 attendees, spelling out their values like this:

Manifesto for Agile Software Development

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; and*
- ***Responding to change*** *over following a plan.*

That is, while there is value in the items below (not bold), we value the items above (bold) more.

The Agile Manifesto and the agile principles thus began to serve as a rallying cry for some and the bull's-eye in the dartboard for others. "Religious" methodology wars ensued between the agilists and those supporting what came to be known as "plan driven,[2] methodologies, the term that came to be used for "not agile" methodologies.

These wars have since subsided. Observations at international agile conferences indicate that companies in all industrial domains have generally come to coexist peacefully with agile methodologies. Many have embraced them, while some use many agile practices and others just a few. Meanwhile, agile practices have evolved, with new ones emerging and others fading away.

So how well do the Agile Manifesto and its 12 principles still capture what is valued by practicing software engineers in industry and by teams that have adopted agile methodologies as their own practices have matured and evolved? How do agile teams regard the principles today? Here, "agile teams" refers to teams claiming to use an agile software development methodology.

Surveys

I conducted two surveys in 2010 at North Carolina State University to weigh the community's view of the principles and use of associated practices. I administered them through surveymonkey.com, advertising the first survey on a number of agile-related user groups (such as those on Yahoo! and LinkedIn). Additionally, I emailed approximately 100 personal contacts, inviting them to participate and forward the survey to their colleagues. Respondents from the first survey could optionally provide their email address if they wanted me to send aggregated results of the survey. When respondents received these results, I further invited them to participate in a follow-on survey.

The first survey focused on the original principles and commonly used software development practices, as of 2010, beginning the first set of questions with the following instruction, followed by a list of the principles in random order:

How important is this principle that comes from the original agile principles authored in 2001 for agile teams in 2010? (1=not very important; 5=essential, the team is not agile if it doesn't follow this principle)

I began the second set of questions with the following instruction, followed by a list of 45 software development practices typically associated with agile:

What practices are essential for a team to be considered agile? (1=not important; 5=essential, a team is not agile unless it does this practice)

With each set of questions, I offered respondents space to provide textual commentary to augment their quantitative responses.

The first survey was completed by 326 respondents with extensive experience in agile software development (see the figure here). Those indicating they had been using an agile methodology for 10 years or more were using what came to be called an "agile methodology" post-Manifesto. Respondents were primarily from North America (59%) and Europe (29%). Of the 326, 18 (55%) indicated they worked on teams with 30 or more members; 313 (96%) worked in a distributed fashion, with 110 (34%) having teams all in the same country, 42 (13%) all in the same continent, and 160 (49%) spread across different continents; and 52 (16%) indicated they worked on safety-critical projects.

I based the follow-on survey on the optional textual commentary provided by respondents of the first survey (see Table 1), distilling the most common comments from the first survey in a revised agile principle and asking their opinion of the revised principle. The motivation behind creating the suggested revision was to highlight emerging industry trends and possible missing subtleties and/or evolution of the original principles; for example, I changed the original principle "Working software is the primary measure of progress" to "Valuable, high-quality software is the primary measure of progress at the end of each short, timeboxed iteration." The follow-on survey sought feedback on the revised principle, though my intent was not to replace the original principle. Respondents in the follow-on survey reflected roughly the same characteristics as participants in the first survey in terms of professional experience and geographic location, with 93 of the original 326 respondents providing feedback on each of the revised principles.

The second survey also asked how valuable respondents considered the principles, as well as "Why are the agile principles valuable?," letting respondents pick as many responses as they felt were applicable, along with the opportunity to provide additional comments.

The personal comments in the survey's "Other" category are best represented by this one: "The purpose of any principle is to provide a simple, clear source of guidance and inspiration. The agile principles are important because they distill the values of 'agile' into as little text as possible. By review-

[b] http://agilemanifesto.org/

[c] http://agilemanifesto.org/principles.html

» key insights

- The 12 original agile principles created by 17 software engineers in 2001 defined the agile trend that continues to transform the entire software industry.
- Rather than view one another solely as competition, these same engineers also wrote the Agile Manifesto, cooperatively focusing on their common interest in agile development and greatly magnifying any of their potential individual contributions.
- Supported by this foundation, agile practices used by software development teams today continue to evolve to address ever-changing user expectations and development team challenges.

ing them as we consider implementation specifics, we can make sure our day-to-day processes are serving the purposes that presumably we decided we wanted to meet.”

Survey respondents’ experience with agile software development.

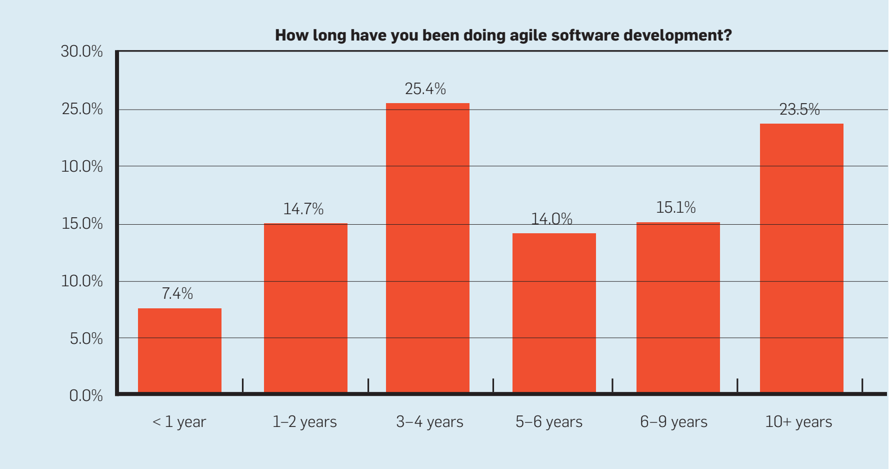

Table 1. Value of agile principles.

Why are the agile principles valuable?	Response Percent
They aren't really; the important thing is to use agile practices.	9.8%
They aren't really; the important thing is to look at the Agile Manifesto.	0%
Because they guide teams new to agile.	48.8%
Because all agile teams choose among software development practices, but, if they want to be agile, they should choose practices that are in line with the principles.	64.6%
They aren't really; no one looks at them anyway.	0%
Other [please comment]	20.7%

Original Principles

My discussion here highlights the most noteworthy of the 93 textual comments from both surveys and the supporting data analysis rather than looking to explain each principle. Note that 11 of the 12 had a mean score of 4.1 out of 5 or higher, indicating a high level of support for principles that had been spelled out 10 years earlier.

Tier One (mean 4.6)

Principle 1 (standard deviation 0.8). Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Principle 3 (standard deviation 0.7). Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.

Respondents’ commentary emphasized delivery of a solution with “high business value” to a customer early and often, along with willingness to respond to feedback. One respondent suggested that principles 1 and 3 were probably redundant, a view supported by statistics based on overall survey responses. The Pearson’s r values for these two principles was 0.31, among the highest correlations between any two principles.

Tier Two (mean 4.5)

Principle 5 (standard deviation 0.9). Build projects around motivated individuals. Give them an environment and support they need, and trust them to get the job done.

Principle 7 (standard deviation 0.8). Working software is the primary measure of progress.

Principle 12 (standard deviation 0.8). The team regularly reflects on how to be more effective, tuning and adjusting its behavior accordingly.

Respondents’ comments concerning principle 5 emphasized the need to empower and respect motivated individuals while making them “feel they can make a difference and [are] part of building something out of the ordinary.” Some respondents said providing the “support they need” included removing obstacles so the team could operate efficiently.

Principle 7 attracted the most comments, with respondents saying that it, in particular, along with the full set of principles, in general, did not adequately emphasize the need to produce high-quality software and test and elicit nonfunctional requirements. The short-term, functional focus of iterations can lead to trouble. “Flaccid Scrum”[d] is the term coined by Martin Fowler, a noted author and speaker on software development, to refer to teams using only Scrum’s project-management practices without also following sound engineering practices. Progress eventually slows for all Flaccid Scrum teams, according to Fowler, because they have not paid enough attention to the quality of the code. In some cases, only the easiest scenario of a feature (often called the “happy path”) is demonstrated at the end of an iteration. The feature can then be considered “done,” with project focus then turning to implementing a new set of features. Teams today more often define and adhere to sound “done criteria,” stipulating the quality and testing steps a team must take before a feature is considered done (see Table 2).

Reacting to principle 12, respondents showed strong enthusiasm for holding retrospectives at least every iteration if not more often “for feedback and creating a culture of continuous improvement and building respect.”

Tier Three (mean 4.4)

Principle 9 (standard deviation 0.8). Continuous attention to technical excellence and good design enhances agility.

Respondents gave strong support for this principle but provided no further commentary or clarification of their views.

Tier Four (mean 4.3)

Principle 2 (standard deviation 0.8). Welcome changing requirements even late in development; agile processes harness change for the customer’s competitive advantage.

Principle 10 (standard deviation 1.0).

[d] http://www.martinfowler.com/bliki/Flaccid-Scrum.html

Simplicity, the art of maximizing the amount of work not done, is essential.

Some commenters on principle 2 suggested a project's requirements should change only at the beginning of each iteration. Agile methodologies aim to reduce waste associated with "thrashing," or progress implementing a feature, stopping and starting and maybe never being complete due to constantly changing priorities. This wasted effort can be reduced through a rule stating that once a feature is started, it must be completed.

One commenter described principle 10 as "Build great software…that addresses users' needs without unnecessary features."

Table 2. Agile principles.

	Mean	Standard Deviation
Continuous integration	4.5	0.8
Short iterations (30 days or less)	4.5	0.8
"Done" criteria	4.5	0.8
Automated tests run with each build	4.4	0.9
Automated unit testing	4.4	0.9
Iteration reviews/demos	4.3	0.8
"Potentially shippable" features at the end of each iteration	4.3	0.9
"Whole" multidisciplinary team with one goal	4.3	0.8
Synchronous communication	4.4	0.9
Embracing changing requirements	4.3	0.8
Features in iteration are customer-visible/customer-valued	4.3	0.8
Prioritized product backlog	4.4	0.9
Retrospective	4.2	1.0
Collective ownership of code	4.2	0.9
Sustainable pace	4.2	0.8
Refactoring	4.2	1.0
"Complete" feature testing done during iteration	4.1	0.9
Negotiated scope	4.1	0.9
Stand up/Scrum meeting	4.1	1.1
Timeboxing	4.1	1.1
Test-driven development unit testing	4.0	1.0
Just-in-time requirements elaboration	4.0	1.0
Small teams (12 people or less)	4.0	1.1
Emergent design	4.0	1.0
Configuration management	4.0	1.2
Daily customer/product manager involvement	3.9	1.0
Release planning	3.9	1.1
Test-driven development acceptance testing	3.8	1.0
Team documentation focuses on decisions rather than planning	3.8	1.2
Informal design; no big design up front	3.7	1.0
Co-located team	3.6	1.1
Team velocity	3.6	1.1
Requirements written as informal stories	3.6	1.1
10-minute build	3.6	1.3
Task planning	3.5	1.2
Coding standard	3.5	1.2
Kanban	3.4	1.6
Acceptance tests written by product manager	3.4	1.2
Pair programming	3.3	1.2
Burndown charts	3.3	1.3
Code inspections	3.2	1.3
Design inspections	3.3	1.3
Planning Poker	3.1	1.4
Stabilization iterations	3.0	1.5

Tier Five (mean 4.1)

Principle 4 (standard deviation 1.0). Businesspeople and developers must work together daily throughout the project.

Principle 6 (standard deviation 1.0). The most effective method of conveying information to and within a development team is face-to-face conversation.

Principle 8 (standard deviation 0.9). Agile processes promote sustainable development; sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Concerning principle 4 several respondents said that developers (often seen as those writing the code) should not be the only ones to work with businesspeople (a.k.a product owners). Rather, the whole team, including user-interface analysts, testers, project managers, developers, and businesspeople should collaborate. Others commenters said, "Every day often isn't realistic, nor is it necessarily needed."

Principle 6 was generally supported, though some commenters said the "requirement for face-to-face conversation is a severely limiting factor for distributed teams, and it seems to be a generational issue as well." In today's connected world, synchronous communication through instant messaging, Voice over IP, and WebEx may effectively stand in for face-to-face communication.

Several representative comments on principle 8 indicating the commenters' negative experience with relatively intense iterations ad infinitum:

"Agile does not promote sustainable development but increases the kind of focus that leads to burnout";

"Sustainable pace is extremely important, but we also sometimes have to slow down and think about things a little";

"Emphasize scheduled downtime as part of sustainable pace"; and

"The team should have dedicated exploratory study time that contributes to its ability to produce innovation."

Tier Six (mean 3.8)

Principle 11 (standard deviation 1.0). The best architectures, requirements, and designs emerge from self-organizing teams.

Concerning principle 11, several commenters suggested the need for a release vision and that teams should understand how the product "contribute[s] to the larger goals of the [user] organization." The principles do not explicitly state that a release plan must be developed, with agile teams often beginning their iterations without such a vision communicated to the whole team. However, agile methodologies have always advocated producing a feasible, prioritized release backlog that also serves as the release vision. One commenter said, "You really need to do some systems engineering when building large systems," while some agilists may consider such systems engineering the equivalent of "big design up front."

Respondents also commented that the lean software development[9] concept of minimizing work-in-process was missing from the principles but still important for agile teams. An emerging lean trend in agile software development that does not appear in the original principles is the use of kanban, or signboard or billboard in Japanese, as a possible replacement for iterations and, in general, a focus on limiting work-in-process. However, this trend is not inconsistent with the original principles. Finally, commenters also said the principles did not cover planning, learning, and collaboration, and communication was not emphasized enough.

Overall, the results of both surveys suggested overwhelming support for the original principles, even after more than 10 years of use. However, survey commenters also said three concepts were missing from the principles but had still been part of the agile software development methodologies from the beginning. First, two principles included the term "developer" in places where the intended connotation was more likely the "whole team." Second, the principles did not explicitly say that a release vision would be created prior to starting incremental development. Finally, the principles did not insist that the working software produced should be valuable and of high quality, though this notion has been part of agile since it was first laid out.

Revised Principles

I assimilated the most common comments from the first survey to revise the original principles through the follow-on survey seeking feedback on the revisions, including:

Principle 1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. [no change]

Principle 2. Welcome changing requirements at the start of each iteration, even late in development; agile processes harness change for the customer's competitive advantage.

Principle 3. [delete; redundant with Principle 1]

Principle 4. The whole team, from businesspeople through testers, must communicate and collaboratively work together throughout the project.

Principle 5. Build projects around empowered, motivated individuals with a shared vision of success; give them the environment and support they need, clear their external obstacles, and trust them to get the job done.

Principle 6. The most efficient, effective method for conveying information to and within a development team is through synchronous communication; important decisions are documented so are not forgotten.

Principle 7. Valuable, high-quality software is the primary measure of progress at the end of each short time-boxed iteration.

Principle 8. Agile processes promote sustainable development. The whole team should be able to maintain a reasonable work pace that includes dedicated time for exploration, visioning, refactoring, and obtaining and responding to feedback.

Principle 9. Continuous attention to technical excellence and good design enhances agility. [no change]

Principle 10. Simplicity—the art of maximizing the amount of work not done—is essential. [no change]

Principle 11. The best architectures, requirements, and designs emerge from self-organizing teams guided by a vision for product release.

Principle 12. With each iteration, the team candidly reflects on the success of the project, feedback, and how to be more effective, then tunes and adjusts its plans and behavior accordingly.

The 93 respondents to the second survey also provided 164 textual comments on the principles. About 11% of the comments (18 of 164) indicated the respondents preferred fewer words, as in "keep it simple." The revised principles are often longer than the original.

Principle 6 (standard deviation 1.0). **The most effective method of conveying information to and within a development team is face-to-face conversation.**

However, the follow-on survey indicated general agreement with the revisions, with two exceptions:

First, the most prominent reaction among survey commenters was when the word "iterations" was added to a principle (such as in revised principles 2, 7, and 12). The recent introduction of the lean software development kanban[7] practice removed the notion of iterations for many teams. With kanban, a feature can begin at any time if the "pull system" indicates the team has the capacity to start new work. As a result, kanban teams often lack defined iterations.

Second, many commenters also reacted negatively to the switch from "face-to-face communication" to "synchronous communication." Despite the fact (discussed earlier) that 96% of survey respondents worked on distributed teams and the assertion by one commenter that the change was "a nice update for the digital world," survey respondents generally emphasized that nothing beats face-to-face for verbal and non-verbal communication alike, and wanted principle 6 to represent the ideal practice.

Agile Practices

Table 2 lists the results of the first survey, which asked whether agile practices are essential for a team to be considered agile; each agile practice is followed by the mean response and the standard deviation of the responses, with 1 indicating the practice is not very important and 5 that the practice is essential for agile teams. Note that the practices at the top of the list generally have a lower standard deviation (connoting greater consistency among survey respondents) than those at the bottom of the list.

Many original agile practices (such as continuous integration and short iterations) are often at the top of such a list, while the more recent, emergent practices (such as Planning Poker, kanban, and stabilization iterations) are at the bottom. Planning Poker[5] is a Wideband Delphi[3] practice for estimating team-based effort. Stabilization iterations (generally two weeks) can occur at the end of all feature-producing iterations or periodically throughout a longer release cycle. During stabilization iterations, testers can perform additional integration, regression, and performance testing; the defect backlog can be reduced; the product backlog can be more intensively groomed; and some preliminary architecture, design, and dependency analysis for the next group of iterations can take place. Some teams find that stabilization iterations mid-release reduce burnout and provide time for exploration and learning.

An agile practice that survey respondents said was left off the list was the "spike"; that is, teams do spikes when they do not know enough about a feature to effectively estimate the resources needed for its implementation. A spike is a timeboxed experiment that allows developers to learn just enough about something unknown about a feature implementation (such as a new technology) to be able to estimate the effort required to deliver the feature.

Conclusion

The authors of the Agile Manifesto and the original 12 principles spelled out the essence of the agile trend that has transformed the software industry over more than a dozen years. That is, they nailed it.

Acknowledgements

Funding for my two-part study was provided by the Scrum Alliance (http://www.scrumalliance.org/). Many thanks to the North Carolina State University Realsearch group (www.realsearchgroup.org) and to the survey respondents. C

References

1. Beck, K. *Extreme Programming Explained: Embrace Change, Second Edition.* Addison-Wesley, Reading, MA, 2005.
2. Boehm, B. and Turner, R. Using risk to balance agile and plan-driven methods. *IEEE Computer 36*, 6 (June 2003), 57–66.
3. Boehm, B.W. *Software Engineering Economics.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.
4. Cockburn, A. *Agile Software Development.* Addison Wesley Longman, 2001.
5. Grenning, J. *Planning Poker or How to Avoid Analysis Paralysis while Release Planning,* 2002; http://renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf
6. Highsmith, J. *Adaptive Software Development.* Dorset House, New York, 1999.
7. Kniberg, H. and Skarin, M. *Kanban and Scrum: Making the Most of Both.* C4Media, Lexington, KY, 2010.
8. Palmer, S.R. and Felsing, J.M. *A Practical Guide to Feature-Driven Development.* Prentice Hall PTR, Upper Saddle River, NJ, 2002.
9. Poppendieck, M. and Poppendieck, T. *Lean Software Development.* Addison Wesley, Boston, 2003.
10. Schwaber, K. and Beedle, M. *Agile Software Development with SCRUM.* Prentice-Hall, Upper Saddle River, NJ, 2002.
11. Stapleton, J. *DSDM: The Method in Practice, Second Edition.* Addison Wesley Longman, 2003.

Laurie Williams (williams@csc.ncsu.edu) is a professor of computer science in the Department of Computer Science at North Carolina State University, Raleigh, NC, and an agile trainer and coach.