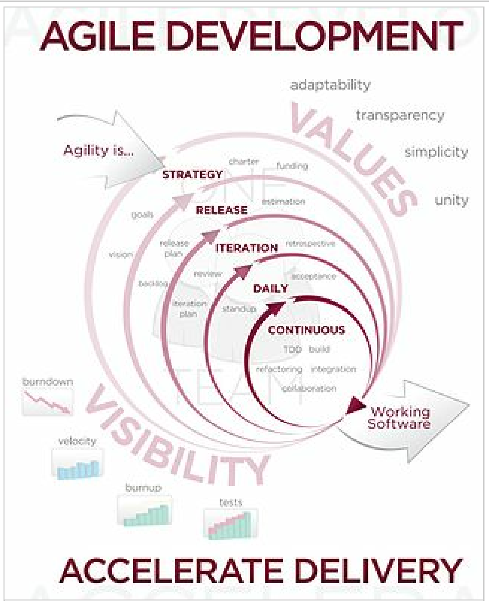
Agile software development

From Wikipedia, the free encyclopedia

Agile software development is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle. The *Agile Manifesto*[1] introduced the term in 2001.



Agile software development poster

Contents

History

Predecessors

Incremental software development methods have been traced back to 1957.[2] In 1974, a paper by E. A. Edmonds introduced an adaptive software development process.[3] Concurrently and independently the same methods were developed and deployed by the New York Telephone Company's Systems Development Center under the direction of Dan Gielan. In the early 1970s, Tom Gilb started publishing the concepts of Evolutionary Project Management (EVO), which has evolved into Competitive Engineering [4]. During the mid to late 1970s Gielan lectured extensively throughout the U.S. on this methodology, its practices, and its benefits.[*citation needed*]

Martin Fowler, widely recognized as one of the key founders of Agile methods

So-called *lightweight* software development methods evolved in the mid-1990s as a reaction against *heavyweight* methods, which were characterized by their critics as a heavily regulated, regimented, micromanaged, waterfall model of development. Proponents of lightweight methods (and now *agile* methods) contend that they are a return to development practices from early in the history of software development.[2]

Early implementations of lightweight methods include Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now typically referred to as agile methodologies, after the Agile Manifesto published in 2001.[5]

Agile Manifesto

In February 2001, 17 software developers[6] met at the Snowbird, Utah resort, to discuss lightweight development methods. They published the *Manifesto for Agile Software Development*[1] to define the approach now known as agile software development. Some of the manifesto's authors formed the Agile Alliance, a nonprofit organization that promotes software development according to the manifesto's principles.

The Agile Manifesto reads, in its entirety, as follows:[1]

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

The meanings of the manifesto items on the left within the agile software development context are described below:

- Individuals and Interactions – in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- Working software – working software will be more useful and welcome than just presenting documents to clients in meetings.
- Customer collaboration – requirements cannot be fully collected at the beginning of the software development cycle, therefore continuous customer or stakeholder involvement is very important.
- Responding to change – agile development is focused on quick responses to change and continuous development.[7]

Twelve principles underlie the Agile Manifesto, including:[8]

- Customer satisfaction by rapid delivery of useful software
- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Close, daily co-operation between business people and developers
- Face-to-face conversation is the best form of communication (co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity- The art of maximizing the amount of work not done - is essential
- Self-organizing teams
- Regular adaptation to changing circumstances

In 2005, a group headed by Alistair Cockburn and Jim Highsmith wrote an addendum of project management principles, the Declaration of Interdependence,[9] to guide software project management according to agile development methods.

Characteristics

There are many specific agile development methods. Most promote development, teamwork, collaboration, and process adaptability throughout the life-cycle of the project.

Pair programming, an agile development technique used by XP

Agile methods break tasks into small increments with minimal planning and do not directly involve long-term planning. Iterations are short time frames (timeboxes) that typically last from one to four weeks. Each iteration involves a team working through a full software development cycle, including planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders. This minimizes overall risk and allows the project to adapt to changes quickly. Stakeholders produce documentation as required. An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration.[10] Multiple iterations might be required to release a product or new features.

Team composition in an agile project is usually cross-functional and self-organizing, without consideration for any existing corporate hierarchy or the corporate roles of team members. Team members normally take responsibility for tasks that deliver the functionality an iteration requires. They decide individually how to meet an iteration's requirements.

Agile methods emphasize face-to-face communication over written documents when the team is all in the same location. Most agile teams work in a single open office (called a bullpen), which facilitates such communication. Team size is typically small (5-9 people) to simplify team communication and team collaboration. Larger development efforts can be delivered by multiple teams working toward a common goal or on different parts of an effort. This might require a coordination of priorities across teams. When a team works in different locations, they maintain daily contact through videoconferencing, voice, e-mail, etc.

No matter what development disciplines are required, each agile team will contain a customer representative. This person is appointed by stakeholders to act on their behalf[11] and makes a personal commitment to being available for developers to answer mid-iteration problem-domain questions. At the end of each iteration, stakeholders and the customer representative review progress and re-evaluate priorities with a view to optimizing the return on investment (ROI) and ensuring alignment with customer needs and company goals.

Most agile implementations use a routine and formal daily face-to-face communication among team members. This specifically includes the customer representative and any interested stakeholders as observers. In a brief session, team members report to each other what they did the previous day, what they intend to do today, and what their roadblocks are. This face-to-face communication exposes problems as they arise.

Agile development emphasizes working software as the primary measure of progress. This, combined with the preference for face-to-face communication, produces less written documentation than other methods. The agile method encourages stakeholders to prioritize "wants" with other iteration outcomes, based exclusively on business value perceived at the beginning of the iteration (also known as *value-driven*).[12]

Specific tools and techniques, such as continuous integration, automated or xUnit test, pair programming, test-driven development, design patterns, domain-driven design, code refactoring and other techniques are often used to improve quality and enhance project agility.

Testing in agile and waterfall

One of the differences between agile and traditional testing methods, such as the waterfall model of software design, is that testing of the software is conducted at different points during the software development lifecycle. Traditional methods (waterfall, RUP) advocate testing after development, either after complete system development (waterfall) or after increments in a more iterative process (RUP). Agile XP advocates techniques such as Test Driven Development where the tests are written before the code or Behaviour Driven Development where the scenarios are written before the code and form part of an automated suite of tests that can verify whether the code meets the required functionality and/or quality standards.

Comparison with other methods

Agile methods are sometimes characterized as being at the opposite end of the spectrum from *plan-driven* or *disciplined* methods. Agile teams may, however, employ highly disciplined formal methods.[13] A more accurate distinction is that methods exist on a continuum from *adaptive* to *predictive*.[14] Agile methods lie on the *adaptive* side of this continuum. Adaptive methods focus on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well. An adaptive team will have difficulty describing exactly what will happen in the future. The further away a date is, the more vague an adaptive method will be about what will happen on that date. An adaptive team cannot report exactly what tasks they will do next week, but only which features they plan for next month. When asked about a release six months from now, an adaptive team might be able to report only the mission statement for the release, or a statement of expected value vs. cost.

Predictive methods, in contrast, focus on planning the future in detail. A predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive teams have difficulty changing direction. The plan is typically optimized for the original destination and changing direction can require completed work to be started over. Predictive teams will often institute a change control board to ensure that only the most valuable changes are considered.

Formal methods, in contrast to adaptive and predictive methods, focus on computer science theory with a wide array of types of provers. A formal method attempts to prove the absence of errors with some level of determinism. Some formal methods are based on model checking and provide counter examples for code that cannot be proven. Generally, mathematical models (often supported through special languages - see SPIN model checker) map to assertions about requirements. Formal methods are dependent on a tool-driven approach and can be combined with other development approaches. Some provers do not easily scale. Like agile methods, manifestos relevant to high-integrity software have been proposed in Crosstalk (http://elsmar.com/pdf_files/A%20Manifesto%20for%20High-Integrity%20Software.pdf) .

Agile methods have much in common with the *Rapid Application Development* techniques from the 1980/90s as espoused by James Martin and others.[*citation needed*] In addition to technology-focused methods, customer- and design-centered methods, such as Visualization-Driven Rapid Prototyping developed by Brian Willison, work to engage customers and end users to facilitate agile software development.[*citation needed*]

In 2008 the Software Engineering Institute (SEI) published the technical report "CMMI or Agile: Why Not Embrace Both" [15] to make clear that CMMI and agile can co-exist.

Agile methods

Well-known agile software development methods include:

- Agile Modeling
- Agile Unified Process (AUP)
- Crystal Clear
- Dynamic Systems Development Method (DSDM)
- Essential Unified Process (EssUP)
- Exia Process (ExP)
- Extreme Programming (XP)
- Feature Driven Development (FDD)
- GSD
- Kanban (development)
- Open Unified Process (OpenUP)
- Scrum
- Velocity tracking

Method tailoring

In the literature, different terms refer to the notion of method adaptation, including ‘method tailoring’, ‘method fragment adaptation’ and ‘situational method engineering’. Method tailoring is defined as:

> A process or capability in which human agents through responsive changes in, and dynamic interplays between contexts, intentions, and method fragments determine a system development approach for a specific project situation.[16]

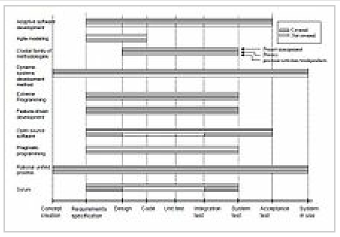
Potentially, almost all agile methods are suitable for method tailoring. Even the DSDM method is being used for this purpose and has been successfully tailored in a CMM context.[17] Situation-appropriateness can be considered as a distinguishing characteristic between agile methods and traditional software development methods, with the latter being relatively much more rigid and prescriptive. The practical implication is that agile methods allow project teams to adapt working practices according to the needs of individual projects. Practices are concrete activities and products that are part of a method framework. At a more extreme level, the philosophy behind the method, consisting of a number of principles, could be adapted (Aydin, 2004).[16]

Extreme Programming (XP) makes the need for method adaptation explicit. One of the fundamental ideas of XP is that no one process fits every project, but rather that practices should be tailored to the needs of individual projects. Partial adoption of XP practices, as suggested by Beck, has been reported on several occasions.[18] Mehdi Mirakhorli (http://portal.acm.org/citation.cfm?id=1370143.1370149&coll=ACM&dl=ACM&CFID=69442744&CFTOKEN=96226775,) proposes a tailoring practice that he feels provides a sufficient roadmap and guidelines for adapting all the practices. RDP Practice is designed for customizing XP. This practice, first proposed as a long research paper in the APSO workshop at the ICSE 2008 conference, is currently the only proposed and applicable method for customizing XP. Although it is specifically a solution for XP, this practice has the capability of extending to other methodologies. At first glance, this practice seems to be in the category of static method adaptation but experiences with RDP Practice says that it can be treated like dynamic method adaptation. The distinction between static method adaptation and dynamic method adaptation is subtle.[19] The key assumption behind static method adaptation is that the project context is given at the start of a project and remains fixed during project execution. The result is a static definition of the project context. Given such a definition, route maps can be used to determine which structured method fragments should be used for that particular project, based on predefined sets of criteria. Dynamic method adaptation, in contrast, assumes that projects are situated in an emergent context. An emergent context implies that a project has to deal with emergent factors that affect relevant conditions but are not predictable. This also means that a project context is not fixed, but changing during project execution. In such a case prescriptive route maps are not appropriate. The practical implication of dynamic method adaptation is that project managers often have to modify structured fragments or even innovate new fragments, during the execution of a project (Aydin et al., 2005).[19]

Software development life cycle

See also: Software development life cycle

The agile methods are focused on different aspects of the software development life cycle. Some focus on the practices (extreme programming, pragmatic programming, agile modeling), while others focus on managing the software projects (the scrum approach). Yet, there are approaches providing full coverage over the development life cycle (dynamic systems development method, or DSDM, and the IBM Rational Unified Process, or RUP), while most of them are suitable from the requirements specification phase on (feature-driven development, or FDD, for example). Thus, there is a clear difference between the various agile software development methods in this regard. Whereas DSDM and RUP do not need complementing approaches to support software development, the others do to a varying degree. DSDM can be used by anyone (although only DSDM members can offer DSDM products or services). RUP, then, is a commercially sold development environment (Abrahamsson, Salo, Rankainen, & Warsta, 2002).[18]



Software development life-cycle support[18]

Measuring agility

While agility can be seen as a means to an end, a number of approaches have been proposed to quantify agility. *Agility Index Measurements* (AIM)[20] score projects against a number of agility factors to achieve a total. The similarly named *Agility Measurement Index*,[21] scores developments against five dimensions of a software project (duration, risk, novelty, effort, and interaction). Other techniques are based on measurable goals.[22] Another study using fuzzy mathematics[23] has suggested that project velocity can be used as a metric of agility. There are agile self-assessments to determine whether a team is using agile practices (Nokia test,[24] Karlskrona test,[25] 42 points test[26]).

While such approaches have been proposed to measure agility, the practical application of such metrics has yet to be seen.

Historically, there is a lack of data on agile projects that failed to produce good results. Studies can be found that report poor projects due to a deficient implementation of an agile method, or methods, but none where it was felt that they were executed properly and failed to deliver on its promise. "This may be a result of a reluctance to publish papers on unsuccessful projects, or it may in fact be an indication that, when implemented correctly, Agile Methods work." [27] However, there is agile software development ROI data available from the DACS ROI Dashboard.[28]

Experience and reception

One of the early studies reporting gains in quality, productivity, and business satisfaction by using Agile methods was a survey conducted by Shine Technologies from November 2002 to January 2003.[29] A similar survey conducted in 2006 by Scott Ambler, the Practice Leader for Agile Development with IBM Rational's Methods Group reported similar benefits.[30] In a survey conducted by VersionOne (a provider of software for planning and tracking agile software development projects) in 2008, 55% of respondents answered that agile methods had been successful in 90-100% of cases.[31] Others claim that agile development methods are still too young to require extensive academic proof of their success.[32]

Suitability

Large-scale agile software development remains an active research area.[33][34]

Agile development has been widely seen as being more suitable for certain types of environment, including small teams of experts.[35][36]:157

Positive reception towards Agile methods has been observed in Embedded domain across Europe in recent years.[37]

Some things that may negatively impact the success of an agile project are:

- Large-scale development efforts (>20 developers), though scaling strategies[34] and evidence of some large projects[38] have been described.
- Distributed development efforts (non-colocated teams). Strategies have been described in *Bridging the Distance*[39] and *Using an Agile Software Process*

with Offshore Development[40]

- Forcing an agile process on a development team[41]
- Mission-critical systems where failure is not an option at any cost (e.g. software for surgical procedures).

The early successes, challenges and limitations encountered in the adoption of agile methods in a large organization have been documented.[42]

In terms of outsourcing agile development, Michael Hackett, Sr. Vice President of LogiGear Corporation has stated that "the offshore team ... should have expertise, experience, good communication skills, inter-cultural understanding, trust and understanding between members and groups and with each other."[43]

Risk analysis can also be used to choose between adaptive (*agile* or *value-driven*) and predictive (*plan-driven*) methods.[12] Barry Boehm and Richard Turner suggest that each side of the continuum has its own *home ground*, as follows:[35]

Suitability of different development methods

Agile home ground	Plan-driven home ground	Formal methods
Low criticality	High criticality	Extreme criticality
Senior developers	Junior developers	Senior developers
Requirements change often	Requirements do not change often	Limited requirements, limited features see Wirth's law
Small number of developers	Large number of developers	Requirements that can be modeled
Culture that responds to change	Culture that demands order	Extreme quality

Criticism

One common criticism of agile software development methods is that it is developer-centric rather than user-centric. Agile software development focuses on processes for getting requirements and developing code and does not focus on product design. Mike Gualtieri, principal analyst of agile software development at Forrester Research, published a widely read criticism stating that software developers are not coders, but experience creators.[44]

Agile methodologies can also be inefficient in large organizations and certain types of projects. Agile methods seem best for developmental and non-sequential projects. Many organizations believe that agile methodologies are too extreme, and adopt a hybrid approach that mixes elements of agile and plan-driven approaches.[45]

See also

- Agile testing
- Collaborative software development model
- Lean Startup
- List of software development philosophies
- Perpetual beta
- Small-scale Project Management
- Software Craftsmanship
- Agile Methodology (http://hyperjots.com/readJots.aspx?ID=12)

References

1. ^ *a b c* Beck, Kent; et al. (2001). "Manifesto for Agile Software Development" (http://agilemanifesto.org/) . Agile Alliance. http://agilemanifesto.org/. Retrieved 14 June 2010.
2. ^ *a b* Gerald M. Weinberg, as quoted in Larman, Craig; Basili, Victor R. (June 2003). "Iterative and Incremental Development: A Brief History". *Computer* **36** (6): 47–56. doi:10.1109/MC.2003.1204375 (http://dx.doi.org/10.1109%2FMC.2003.1204375) . ISSN 0018-9162 (//www.worldcat.org/issn/0018-9162) . "We were doing incremental development as early as 1957, in Los Angeles, under the direction of Bernie Dimsdale [at IBM's ServiceBureau Corporation]. He was a colleague of John von Neumann, so perhaps he learned it there, or assumed it as totally natural. I do remember Herb Jacobs (primarily, though we all participated) developing a large simulation for Motorola, where the technique used was, as far as I can tell .... All of us, as far as I can remember, thought waterfalling of a huge project was rather stupid, or at least ignorant of the realities. I think what the waterfall description did for us was make us realize that we were doing something else, something unnamed except for 'software development.'"
3. ^ Edmonds, E. A. (1974). "A Process for the Development of Software for Nontechnical Users as an Adaptive System". *General Systems* **19**: 215–18.
4. ^ Evolutionary Project Management (EVO) (http://www.gilb.com/Project-Management)
5. ^ Larman, Craig (2004). *Agile and Iterative Development: A Manager's Guide.* Addison-Wesley. p. 27. ISBN 978-0-13-111155-4
6. ^ 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, Stephen J. Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas
7. ^ Ambler, S.W. "Examining the Agile Manifesto" (http://www.ambysoft.com/essays/agileManifesto.html) . Retrieved 6 April 2011.
8. ^ Beck, Kent; et al. (2001). "Principles behind the Agile Manifesto" (http://www.agilemanifesto.org/principles.html) . Agile Alliance. Archived (http://web.archive.org/web/20100614043008/http://www.agilemanifesto.org/principles.html) from the original on 14 June 2010. http://www.agilemanifesto.org/principles.html. Retrieved 6 June 2010.
9. ^ Anderson, David (2005). "Declaration of Interdependence" (http://pmdoi.org) . http://pmdoi.org.
10. ^ Beck, Kent (1999). "Embracing Change with Extreme Programming". *Computer* **32** (10): 70–77. doi:10.1109/2.796139 (http://dx.doi.org/10.1109%2F2.796139) .
11. ^ Gauthier, Alexandre (17 August 2011). "What is scrum" (http://www.planbox.com/resources/agile-artifacts#roles) . Planbox. http://www.planbox.com/resources/agile-artifacts#roles.

12. ^ *a b* Sliger, Michele; Broderick, Stacia (2008). *The Software Project Manager's Bridge to Agility*. Addison-Wesley. p. 46. ISBN 0-321-50275-2.
13. ^ Black, S. E.; Boca., P. P.; Bowen, J. P.; Gorman, J.; Hinchey, M. G. (September 2009). "Formal versus agile: Survival of the fittest". *IEEE Computer* **49** (9): 39–45.
14. ^ Boehm, B.; R. Turner (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley. ISBN 0-321-18612-5. Appendix A, pages 165-194
15. ^ TECHNICAL NOTE CMU/SEI-2008-TN-003 CMMI or Agile: Why Not Embrace Both (http://www.sei.cmu.edu/library/abstracts/reports/08tn003.cfm)
16. ^ *a b* Aydin, M.N., Harmsen, F., Slooten, K. v., & Stagwee, R. A. (2004). An Agile Information Systems Development Method in use. *Turk J Elec Engin, 12(2),* 127-138
17. ^ Abrahamsson, P., Warsta, J., Siponen, M.T., & Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. *Proceedings of ICSE'03*, 244-254
18. ^ *a b c* Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. *VTT Publications 478*
19. ^ *a b* Aydin, M.N., Harmsen, F., Slooten van K., & Stegwee, R.A. (2005). On the Adaptation of An Agile Information(Suren) Systems Development Method. *Journal of Database Management Special issue on Agile Analysis, Design, and Implementation, 16(4),* 20-24
20. ^ "David Bock's Weblog : Weblog" (http://jroller.com/page/bokmann?entry=improving_your_processes_aim_high) . Jroller.com. http://jroller.com/page/bokmann?entry=improving_your_processes_aim_high. Retrieved 2 April 2010.
21. ^ "Agility measurement index" (http://doi.acm.org/10.1145/1185448.1185509) . Doi.acm.org. http://doi.acm.org/10.1145/1185448.1185509. Retrieved 2 April 2010.
22. ^ Peter Lappo; Henry C.T. Andrew. "Assessing Agility" (http://www.smr.co.uk/presentations/measure.pdf) . http://www.smr.co.uk/presentations/measure.pdf. Retrieved 6 June 2010.
23. ^ Kurian, Tisni (2006). Agility Metrics: A Quantitative Fuzzy Based Approach for Measuring Agility of a Software Process, *ISAM-Proceedings of International Conference on Agile Manufacturing'06(ICAM-2006)*, Norfolk, U.S.
24. ^ Joe Little (2 December 2007). "Nokia test, A scrum-specific test" (http://agileconsortium.blogspot.com/2007/12/nokia-test.html) . Agileconsortium.blogspot.com. http://agileconsortium.blogspot.com/2007/12/nokia-test.html. Retrieved 6 June 2010.
25. ^ Mark Seuffert, Piratson Technologies, Sweden. "Karlskrona test, A generic agile adoption test" (http://www.piratson.se/archive/Agile_Karlskrona_Test.html) . Piratson.se. http://www.piratson.se/archive/Agile_Karlskrona_Test.html. Retrieved 6 June 2010.
26. ^ "How agile are you, a scrum-specific test" (http://www.agile-software-development.com/2008/01/how-agile-are-you-take-this-42-point.html) . Agile-software-development.com. http://www.agile-software-development.com/2008/01/how-agile-are-you-take-this-42-point.html. Retrieved 6 June 2010.
27. ^ David Cohen, Mikael Lindvall, Patricia Costa "Agile Software Development" (http://www.thedacs.com/techs/abstract/345473) , *Data & Analysis Center for Software*, January 2003
28. ^ DACS ROI Dashboard (http://www.thedacs.com/databases/roi/) Retrieved 11 November 2011.
29. ^ "Agile Methodologies Survey Results" (http://www.shinetech.com/attachments/104_ShineTechAgileSurvey2003-01-17.pdf) (PDF). Shine Technologies (http://www.shinetech.com) . January 2003. http://www.shinetech.com/attachments/104_ShineTechAgileSurvey2003-01-17.pdf. Retrieved 3 June 2010. "95% [stated] that there was either no effect or a cost reduction . . . 93% stated that productivity was better or significantly better . . . 88% stated that quality was better or significantly better . . . 83% stated that business satisfaction was better or significantly better"
30. ^ Ambler, Scott (3 August 2006). "Survey Says: Agile Works in Practice" (http://www.drdobbs.com/architecture-and-design/191800169;jsessionid=2QJ23QRYM3H4PQE1GHPCKH4ATMY32JVN?queryText=agile+survey) . *Dr. Dobb's*. http://www.drdobbs.com/architecture-and-design/191800169;jsessionid=2QJ23QRYM3H4PQE1GHPCKH4ATMY32JVN?queryText=agile+survey. Retrieved 3 June 2010. "Only 6 percent indicated that their productivity was lowered . . . No change in productivity was reported by 34 percent of respondents and 60 percent reported increased productivity. . . . 66 percent [responded] that the quality is higher. . . . 58 percent of organizations report improved satisfaction, whereas only 3 percent report reduced satisfaction."
31. ^ "The State of Agile Development" (http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf) (PDF). VersionOne, Inc.. 2008. http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf. Retrieved 3 July 2010. "Agile delivers"
32. ^ "Answering the "Where is the Proof That Agile Methods Work" Question" (http://www.agilemodeling.com/essays/proof.htm) . Agilemodeling.com. 19 January 2007. http://www.agilemodeling.com/essays/proof.htm. Retrieved 2 April 2010.
33. ^ Agile Processes Workshop II Managing Multiple Concurrent Agile Projects. Washington: OOPSLA 2002
34. ^ *a b* W. Scott Ambler (2006) Supersize Me (http://www.drdobbs.com/184415491) in Dr. Dobb's Journal, 15 February 2006.
35. ^ *a b* Boehm, B.; R. Turner (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley. pp. 55–57. ISBN 0-321-18612-5.
36. ^ Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Boston, MA: Addison-Wesley. ISBN 0-321-27865-8.
37. ^ K Petersen's doctoral research in Sweden Implementing Lean and Agile Software Development in Industry (http://www.bth.se/tek/aps/kps.nsf/pages/phd-studies)
38. ^ Schaaf, R.J. (2007). Agility XL Systems and Software Technology Conference 2007 (http://www.sstc-online.org/Proceedings/2007/pdfs/RJS1722.pdf) , Tampa, FL
39. ^ "Bridging the Distance" (http://www.drdobbs.com/architecture-and-design/184414899) . Sdmagazine.com. http://www.drdobbs.com/architecture-and-design/184414899. Retrieved 1 February 2011.
40. ^ Martin Fowler. "Using an Agile Software Process with Offshore Development" (http://www.martinfowler.com/articles/agileOffshore.html) . Martinfowler.com. http://www.martinfowler.com/articles/agileOffshore.html. Retrieved 6 June 2010.
41. ^ [The Art of Agile Development James Shore & Shane Warden pg 47]
42. ^ Evans, Ian. "Agile Delivery at British Telecom" (http://www.methodsandtools.com/archive/archive.php?id=43) . http://www.methodsandtools.com/archive/archive.php?id=43. Retrieved 21 February 2011.
43. ^ [1] (http://www.logigear.com/in-the-news/973-agile.html) LogiGear, PC World Viet Nam, Jan 2011
44. ^ Gualtieri, Mike (2011 [last update]). "Agile Software Is A Cop-Out; Here's What's Next | Forrester Blogs" (http://blogs.forrester.com/mike_gualtieri/11-10-12-agile_software_is_a_cop_out_heres_whats_next) . *blogs.forrester.com*. http://blogs.forrester.com/mike_gualtieri/11-10-12-agile_software_is_a_cop_out_heres_whats_next. Retrieved 28 November 2011.
45. ^ Barlow, Jordan B.; Justin Scott Giboney, Mark Jeffery Keith, David W. Wilson, Ryan M. Schuetzler, Paul Benjamin Lowry, Anthony Vance (2011). "Overview and Guidance on Agile Development in Large Organizations" (http://aisel.aisnet.org/cais/vol29/iss1/2/) . *Communications of the Association for Information Systems* **29** (1): 25–44. http://aisel.aisnet.org/cais/vol29/iss1/2/.

Further reading

- *Annual State of Agile Development Survey: 2011 trends* (http://www.versionone.com/state_of_agile_development_survey/11/)
- *The Future of Agile Software Development* (http://www.targetprocess.com/rightthing.html)
- *SCRUM Roles and Responsibilities* (http://www.agiledevelopernotes.com/2011/07/scrum-roles-and-responsibilities.html)
- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile Software Development Methods: Review and Analysis. (http://agile.vtt.fi/publications.html) *VTT Publications 478*.
- Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. In *Advances in Computers* (pp. 1–66). New York: Elsevier Science.
- Dingsøyr, Torgeir, Dybå, Tore and Moe, Nils Brede (ed.): *Agile Software Development: Current Research and Future Directions* (http://www.amazon.co.uk/Agile-Software-Development-Research-Directions/dp/3642125743) , Springer, Berlin Heidelberg, 2010.

- Fowler, Martin. *Is Design Dead?* (http://www.martinfowler.com/articles/designDead.html) . Appeared in *Extreme Programming Explained*, G. Succi and M. Marchesi, ed., Addison-Wesley, Boston. 2001.
- Larman, Craig and Basili, Victor R. *Iterative and Incremental Development: A Brief History* IEEE Computer, June 2003 (http://www.highproductivity.org/r6047.pdf)
- Riehle, Dirk. *A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming: How Methodologies May Learn From Each Other* (http://www.riehle.org/computer-science/research/2000/xp-2000.html) . Appeared in *Extreme Programming Explained*, G. Succi and M. Marchesi, ed., Addison-Wesley, Boston. 2001.
- Rother, Mike (2009). (http://books.google.com/?id=_1lhPgAACAAJ&dq=toyota+kata) Toyota Kata. McGraw-Hill. ISBN 0-07-163523-8. http://books.google.com/?id=_1lhPgAACAAJ&dq=toyota+kata
- M. Stephens, D. Rosenberg. *Extreme Programming Refactored: The Case Against XP*. Apress L.P., Berkeley, California. 2003. (ISBN 1-59059-096-1)
- Shore, J., & Warden S. (2008). The Art of Agile Development. O'Reilly Media, Inc.
- Wik,Philip *Effective Top-Down SOA Management In An Efficient Bottom-Up Agile World* (http://soamag.com/I38/0410-1.php) Service Technology Magazine, April, 2010.
- Willison, Brian (2008). Iterative Milestone Engineering Model. New York, NY.
- Willison, Brian (2008). Visualization Driven Rapid Prototyping. Parsons Institute for Information Mapping.

External links

- Agile (http://www.dmoz.org/Computers/Programming/Methodologies/Agile/) at the Open Directory Project

- Article Two Ways to Build a Pyramid by John Mayo-Smith (http://www.informationweek.com/news/software/development/showArticle.jhtml?articleID=6507351)
- The New Methodology (http://martinfowler.com/articles/newMethodology.html) Martin Fowler's description of the background to agile methods
- Ten Authors of The Agile Manifesto Celebrate its Tenth Anniversary (http://www.pragprog.com/magazines/2011-02/agile--)
- A look into the PMI-ACP (Agile Certified Practitioner) (http://www.pmi.org/Certification/New-PMI-Agile-Certification.aspx)

Retrieved from "http://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=492081371"
Categories: Software project management | Software development philosophies | Agile software development