Agile sw development

Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s). It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change.
The term *agile* (sometimes written *Agile*) was popularized, in this context, by the *Manifesto for Agile Software Development*. The values and principles espoused in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban.

There is significant anecdotal evidence that adopting agile practices and values improves the agility of software professionals, teams and organizations; however, some empirical studies have found no scientific evidence.

The Manifesto for Agile Software Development

Agile software development values

Based on their combined experience of developing software and helping others do that, the seventeen signatories to the manifesto proclaimed that they 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 to say, the items on the left are valued more than the items on the right.

As Scott Ambler elucidated:

- Tools and processes are important, but it is more important to have competent people working together effectively.
- Good documentation is useful in helping people to understand how the software is built and how to use it, but the main point of development is to create software, not documentation.
- A contract is important but is no substitute for working closely with customers to discover what they need.
- A project plan is important, but it must not be too rigid to accommodate changes in technology or the environment, stakeholders' priorities, and people's understanding of the problem and its solution.

The Agile movement is not anti-methodology, in fact many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment.

XP

Extreme programming (**XP**) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.
Other elements of extreme programming include: programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management

structure, code simplicity and clarity, expecting changes in the customer's requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers. The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels. As an example, code reviews are considered a beneficial practice; taken to the extreme, code can be reviewed *continuously*, i.e. the practice of pair programming.

History

Extreme programming was created by Kent Beck during his work on the Chrysler Comprehensive Compensation System (C3) payroll project. Beck became the C3 project leader in March 1996 and began to refine the development methodology used in the project and wrote a book on the methodology (in October 1999, *Extreme Programming Explained* was published). Chrysler cancelled the C3 project in February 2000, after seven years, when the company was acquired by Daimler-Benz.

Many extreme programming practices have been around for some time; the methodology takes "best practices" to extreme levels. For example, the "practice of test-first development, planning and writing tests before each micro-increment" was used as early as NASA's Project Mercury, in the early 1960s. To shorten the total development time, some formal test documents (such as for acceptance testing) have been developed in parallel (or shortly before) the software is ready for testing. A NASA independent test group can write the test procedures, based on formal requirements and logical limits, before the software has been written and integrated with the hardware. In XP, this concept is taken to the extreme level by writing automated tests (perhaps inside of software modules) which validate the operation of even small sections of software coding, rather than only testing the larger features.

Origins

Software development in the 1990s was shaped by two major influences: internally, object-oriented programming replaced procedural programming as the programming paradigm favored by some in the industry; externally, the rise of the Internet and the dot-com boom emphasized speed-to-market and company growth as competitive business factors. Rapidly changing requirements demanded shorter product life-cycles, and were often incompatible with traditional methods of software development.

The Chrysler Comprehensive Compensation System (C3) was started in order to determine the best way to use object technologies, using the payroll systems at Chrysler as the object of research, with Smalltalk as the language and GemStone as the data access layer. They brought in Kent Beck, a prominent Smalltalk practitioner, to do performance tuning on the system, but his role expanded as he noted several problems they were having with their development process. He took this opportunity to propose and implement some changes in their practices based on his work with his frequent collaborator, Ward Cunningham. Beck describes the early conception of the methods:

The first time I was asked to lead a team, I asked them to do a little bit of the things I thought were sensible, like testing and reviews. The second time there was a lot more on the line. I thought, "Damn the torpedoes, at least this will make a good article," [and] asked the team to crank up all the knobs to 10 on the things I thought were essential and leave out everything else.

Beck invited Ron Jeffries to the project to help develop and refine these methods. Jeffries thereafter acted as a coach to instill the practices as habits in the C3 team.

Information about the principles and practices behind XP was disseminated to the wider world through discussions on the original wiki, Cunningham's WikiWikiWeb. Various contributors discussed and expanded upon the ideas, and some spin-off methodologies resulted (see agile software development). Also, XP concepts have been explained, for several years, using a hypertext system map on the XP website at http://www.extremeprogramming.org circa 1999.

Beck edited a series of books on XP, beginning with his own *Extreme Programming Explained* (1999, ISBN 0-201-61641-6), spreading his ideas to a much larger audience. Authors in the series went through various aspects attending XP and its practices. The series included a book that was critical of the practices.

Current state

XP generated significant interest among software communities in the late 1990s and early 2000s, seeing adoption in a number of environments radically different from its origins.

The high discipline required by the original practices often went by the wayside, causing some of these practices, such as those thought too rigid, to be deprecated or reduced, or even left unfinished, on individual sites. For example, the practice of end-of-day integration tests for a particular project could be changed to an end-of-week schedule, or simply reduced to mutually agreed dates. Such a more relaxed schedule could avoid people feeling rushed to generate artificial stubs just to pass the end-of-day testing. A less-rigid schedule allows, instead, for some complex features to be more fully developed over a several-day period.

Meanwhile, other agile development practices have not stood still, and XP is still evolving, assimilating more lessons from experiences in the field, to use other practices. In the second edition of *Extreme Programming Explained* (November 2004), five years after the first edition, Beck added more values and practices and differentiated between primary and corollary practices.

The Theory of Sustainable Software Development explains why extreme programming teams can thrive in spite of team disruptions.