Agile Software Development

Refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. The term was coined in the year 2001 when the Agile Manifesto was formulated.

Agile methods generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals.

Conceptual foundations of this framework are found in modern approaches to operations management and analysis, such as lean manufacturing, soft systems methodology, speech act theory (network of conversations approach), and Six Sigma.

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

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 helps minimize overall risk, and lets the project adapt to changes quickly. Stakeholders produce documentation as required. An iteration may 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.[1] Multiple iterations may 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 Software Development

Agile methods emphasize face-to-face communication over written documents when the team is all in the same location. When a team works in different locations, they maintain daily contact through videoconferencing, voice, e-mail, etc.

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 help make team communication and team collaboration easier. Larger development efforts may be delivered by multiple teams working toward a common goal or different parts of an effort. This may also require a coordination of priorities across teams.

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 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 and ensuring alignment with customer needs and company goals.

Most agile implementations use 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 yesterday, what they intend to do today, and what their roadblocks are. This standing face-to-face communication prevents problems from being hidden.

Agile 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—though, in an agile project, documentation and other artifacts rank equally with a working product. The agile method encourages stakeholders to prioritize wants with other iteration outcomes based exclusively on the business value perceived at the beginning of the iteration.

Specific tools and techniques such as continuous integration, automated or xUnit tests, 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.

Agile Software Development

History

The modern definition of agile software development evolved in the mid-1990s as part of a reaction against "heavyweight" methods, perceived to be typified by a heavily regulated, regimented, micro-managed use of the waterfall model of development. The processes originating from this use of the waterfall model were seen as bureaucratic, slow, demeaning, and inconsistent with the ways that software developers actually perform effective work. A case can be made that agile and iterative development method mark a return to development practice from early in the history of software development.[2] Initially, agile methods were called "lightweight methods."

An adaptive software development process was introduced in a paper by Edmonds (1974).[3] Notable early Agile 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.

In 2001, 17 prominent figures[4] in the field of agile development (then called "lightweight methods") came together at the Snowbird ski resort in Utah to discuss ways of creating software in a lighter, faster, more people-centric way. They coined the terms "Agile Software Development" and "agile methods", and they created the Agile Manifesto, widely regarded as the canonical definition of agile development and accompanying agile principles. Later, some of these people formed The Agile Alliance, a non-profit organization that promotes agile development.

Principles

Agile methods are a family of development processes, not a single approach to software development. The Agile Manifesto states:

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.

Some of the principles behind the Agile Manifesto are:

• Customer satisfaction by rapid, continuous delivery of useful software

• Working software is delivered frequently (weeks rather than months)

• Working software is the principal measure of progress

• Even late changes in requirements are welcomed

• Close, daily cooperation 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

• Self-organizing teams

• Regular adaptation to changing circumstances

The manifesto spawned a movement in the software industry known as agile software development.

In 2005, Alistair Cockburn and Jim Highsmith gathered another group of people—management experts, this time—and wrote an addendum, known as the PM Declaration of Interdependence.

The functioning principles of Agile can be found in lean manufacturing and six sigma. These concepts include error-proofing, eliminating waste, creating flow, adding customer value, and empowering workers. The concepts were first formally espoused in the 14 principles of the Toyota Way, the two pillars of the Toyota Production System (Just-in-time and smart automation), the 5S methodology, and Deming's 14 points. These have been summarized in the seven points of lean software development.

Principles

Agile methods are a family of development processes, not a single approach to software development. The Agile Manifesto states:

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.

Some of the principles behind the Agile Manifesto are:

• Customer satisfaction by rapid, continuous delivery of useful software

• Working software is delivered frequently (weeks rather than months)

• Working software is the principal measure of progress

• Even late changes in requirements are welcomed

• Close, daily cooperation 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

• Self-organizing teams

• Regular adaptation to changing circumstances

The manifesto spawned a movement in the software industry known as agile software development.

In 2005, Alistair Cockburn and Jim Highsmith gathered another group of people—management experts, this time—and wrote an addendum, known as the PM Declaration of Interdependence.

The functioning principles of Agile can be found in lean manufacturing and six sigma. These concepts include error-proofing, eliminating waste, creating flow, adding customer value, and empowering workers. The concepts were first formally espoused in the 14 principles of the Toyota Way, the two pillars of the Toyota Production System (Just-in-time and smart automation), the 5S methodology, and Deming's 14 points. These have been summarized in the seven points of lean software development.

Comparison with other methods

Agile methods are sometimes characterized as being at the opposite end of the spectrum from "plan-driven" or "disciplined" methods. This distinction is misleading, as it implies that agile methods are "unplanned" or "undisciplined". A more accurate distinction is that methods exist on a continuum from "adaptive" to "predictive". 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 can report exactly what tasks are being done next week, but only which features are planned for next month. When asked about a release six months from now, an adaptive team may only be able to report 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 cause completed work to be thrown away and done over differently. Predictive teams will often institute a change control board to ensure that only the most valuable changes are considered.

Agile methods have much in common with the "Rapid Application Development" techniques from the 1980/90s as espoused by James Martin and others.

Contrasted with other iterative development methods

Most agile methods share other iterative and incremental development methods' emphasis on building releasable software in short time periods. Agile development differs from other development models: in this model time periods are measured in weeks rather than months and work is performed in a highly collaborative manner. Most agile methods also differ by treating their time period as a timebox.

Contrasted with the waterfall model

Agile development has little in common with the waterfall model. As of 2009, the waterfall model is still in common use. The waterfall model is the most structured of the methods, stepping through requirements-capture, analysis, design, coding, and testing in a strict, pre-planned sequence. Progress is generally measured in terms of deliverable artifacts: requirement specifications, design documents, test plans, code reviews and the like.

A common criticism of the waterfall model is its inflexible division of a project into separate stages, where commitments are made early on, making it difficult to react to changes in requirements as the project executes. Iterations are expensive. This means that the waterfall model is likely to be unsuitable if requirements are not well understood or are likely to change in the course of the project.

Agile methods, in contrast, produce completely developed and tested features (but a very small subset of the whole) every few weeks. The emphasis is on obtaining the smallest workable piece of functionality to deliver business value early, and continually improving it, and adding further functionality throughout the life of the project. If a project being delivered under the waterfall method is cancelled at any point up to the end, there is nothing to show for it beyond a huge resources bill. With the agile method, being cancelled at any point will still leave the customer with some worthwhile code that has likely already been put into live operation.

Adaptations of Scrum show how agile methods are augmented to produce and continuously improve a strategic plan.

Some agile teams use the waterfall model on a small scale, repeating the entire waterfall cycle in every iteration. Other teams, most notably Extreme Programming teams, work on activities simultaneously.

Contrasted with "cowboy coding"

Cowboy coding is the absence of a defined method; i.e., team members do whatever they feel is right. The Agile approach is often confused with cowboy coding, due to Agile's frequent re-evaluation of plans, emphasis on face-to-face communication, and relatively sparse use of documentation. However, Agile teams follow defined—and often very disciplined and rigorous—processes.

As with all development methods, the skill and experience of users determine the degree of success and/or abuse of such activity. Rigid controls, which are systematically embedded within an Agile process, offer stronger levels of accountability. The degradation of well-intended procedures can lead to activities that are often categorized as cowboy coding.

Suitability of agile methods

There is little if any consensus on what types of software projects are best suited for the agile approach. Many large organizations have difficulty bridging the gap between the traditional waterfall method and an agile one.

Large scale agile software development remains an active research area.

Agile development has been widely documented as working well for small (<10 developers) co-located teams.

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

• Large scale development efforts (>20 developers), though scaling strategies and evidence to the contrary[17] have been described.

• Distributed development efforts (non-co-located teams). Strategies have been described in Bridging the Distance and Using an Agile Software Process with Offshore Development[19]

• Command-and-control company cultures

• Forcing an agile process on a development team

• Mission critical systems where failure is not an option at any cost (Software for surgical procedures).

Several successful large scale agile projects have been documented. BT has had several hundred developers situated in the UK, Ireland and India working collaboratively on projects and using Agile methods. While questions undoubtedly still arise about the suitability of some Agile methods to certain project types, it would appear that scale or geography, by themselves, are not necessarily barriers to success.

Barry Boehm and Richard Turner suggest that risk analysis be used to choose between adaptive ("agile") and predictive ("plan-driven") methods.[15] The authors suggest that each side of the continuum has its own home ground as follows:

Agile home ground:

• Low criticality

• Senior developers

• Requirements change often

• Small number of developers

• Culture that thrives on chaos

Plan-driven home ground:[15]

• High criticality

• Junior developers

• Requirements do not change often

• Large number of developers

• Culture that demands order

Agile methods and 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.

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. 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).

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.[22] A tailoring practice is proposed by Mehdi Mirakhorli which provides a sufficient roadmap and guidelines for adapting all the practices. RDP Practice is designed for customizing XP. This practice was first time proposed as a long research paper in the APSO workshop at ICSE 2008 conference and yet it is 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 say that it can be treated like dynamic method adaptation. The distinction between static method adaptation and dynamic method adaptation is subtle.[23] 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 in order 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 changes 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).

Agile Methods and project management

Agile methods differ to a large degree in the way they cover project management. Some methods are supplemented with guidelines on project management, but there is generally no comprehensive support.

Both PMP and PRINCE2 have been suggested as suitable, complementary project management systems.