Agile software development is a conceptual framework for undertaking software engineering projects that embraces and promotes evolutionary change throughout the entire life-cycle of the project. There are a number of agile software development methods, such as those espoused by The Agile Alliance. Most agile methods attempt to minimize risk by developing software in short timeboxes, called iterations, which typically last one to four weeks. Each iteration is like a miniature software project of its own, and includes all of the tasks necessary to release.
the mini-increment of new functionality: planning, requirements analysis, design, coding, testing, and documentation.

While an iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software at the end of every iteration. In many cases, software is released at the end of each iteration. This is particularly true when the software is web-based and can be released easily. Regardless, at the end of each iteration, the team reevaluates project priorities.

Agile methods emphasize realtime communication, preferably face-to-
face, over written documents. Most agile teams are located in a bullpen and include all the people necessary to finish software. At a minimum, this includes programmers and their “customers” (customers are the people who define the product; they may be product managers, business analysts, or actual customers). The bullpen may also include testers, interaction designers, technical writers, and managers.

Agile methods also emphasize working software as the primary measure of progress. Combined with the preference for face-to-face communication, agile methods produce very little written
documentation relative to other methods. This has resulted in criticism of agile methods as being undisciplined.

History

The modern definition of agile software development evolved in the mid 1990s as part of a reaction against "heavyweight" methods, as 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 engineers actually perform effective work. A case can be made that agile and iterative development methods are a return to development practice seen early in the history of software development.\[1\] Initially, agile methods were called "lightweight methods." In 2001, prominent members of the community met at Snowbird, Utah, and adopted the name "agile methods." Later, some of these people formed The Agile Alliance\[1\], a nonprofit organization that promotes agile development.

While it may not have been the first agile method, Extreme Programming (usually abbreviated as "XP") established the popularity of agile methods. Extreme Programming was created by Kent Beck in 1996 as a way to rescue the struggling Chrysler Comprehensive Compensation (C3) project. While that project was eventually canceled, the methodology was
refined by Ron Jeffries' full-time XP coaching, public discussion on Ward Cunningham's Portland Pattern Repository wiki and further work by Beck, including a book in 1999.\[2\] Elements of Extreme Programming appear to be based on Scrum and Ward Cunningham's Episodes pattern language.

[edit] Principles behind agile methods

The Agile Manifesto

Agile methods are a family of development processes, not a single approach to software development. In 2001, 17
prominent figures in the field of agile development (then called "lightweight methodologies") came together at the Snowbird ski resort in Utah to discuss ways of creating software in a lighter, faster, more people-centric way. They created the Agile Manifesto, widely regarded as the canonical definition of agile development, and accompanying agile principles. 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. 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 publishing of 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.

Agile methods are sometimes
characterized as being at the opposite
end of the spectrum from "plan-
driven" or "disciplined" methodologies. This
distinction is misleading, as it implies
that agile methods are "unplanned" or "undisciplined". A more accurate
distinction is to say that methods exist
on a continuum from "adaptive" to "Predictive". Agile methods exist 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's as espoused by James Martin and others (see RAD).

Contrasted with iterative development, most agile methods share iterative development's emphasis on building releasable software in short time periods. Agile methods differ from iterative methods in that their time period is measured in weeks rather than months and work is performed in a highly collaborative manner. Most agile methods also...
Contrasted with the waterfall model, Agile development does not have much in common with the waterfall model. As of 2004, the waterfall model is still in common use.[6] The waterfall model is the most predictive of the methodologies, 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.
The main problem of the waterfall model is the inflexible nature of the division of a project into separate stages, so that commitments are made early on, and it is difficult to react to changes in requirements. 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 radically 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 or months. The emphasis is on obtaining the smallest workable piece of functionality to deliver.
business value early, and continually improving it/adding further functionality throughout the life of the project.

Some agile teams use the waterfall model on a small scale, repeating the entire waterfall cycle in every iteration. [citation needed] Other teams, most notably Extreme Programming teams, work on activities simultaneously. Contrasted with "cowboy coding" Cowboy coding is the absence of a defined method: team members do whatever they feel is right. Agile development's
frequent reevaluation of plans, emphasis on face-to-face communication, and relatively sparse use of documents sometimes causes people to confuse it with cowboy coding. Agile teams, however, do follow defined (and often very disciplined and rigorous) processes. As with all methodologies, the skill and experience of the users define the degree of success and/or abuse of such activity. The more rigid controls systematically embedded within a process offer stronger levels of accountability of the users. The degradation of well-intended procedures can lead to activities often categorized as cowboy coding.
Suitability of agile methods

Although agile methods differ in their practices, they share a number of common characteristics, including iterative development, and a focus on interaction, communication, and the reduction of resource-intensive intermediate artifacts.

The suitability of agile methods in general can be examined from multiple perspectives. From a product perspective, agile methods are more suitable when requirements are emergent and rapidly changing; they are less suitable for systems that have high criticality, reliability and safety requirements, although there is
no complete consensus on this point. From an organizational perspective, the suitability can be assessed by examining three key dimensions of an organization: culture, people, and communication. In relation to these areas a number of key success factors have been identified (Cohen et al.,
The culture of the organization must be supportive of negotiation.

People must be trusted.

Fewer but more competent people.

Organizations must live with the decisions developers make.

Organizations need to have an environment that facilitates rapid communication between team members.

The most important factor is probably project size. As size grows, face-to-face communication becomes more difficult. Therefore, most agile methods are more suitable for projects with small...
teams, with fewer than 20 to 40 people. Large scale agile software development remains an active research area\[9\]\[10\] Another serious problem is that initial assumptions or overly rapid requirements gathering up-front may result in a large drift from an optimal solution, especially if the client defining the target product has poorly formed ideas of their needs. Similarly, given the nature of human behaviour, it's easy for a single "dominant" developer to influence or even pull the design of the target in a direction not necessarily appropriate for the project. Historically, the developers can, and often
do, impose solutions on a client then convince the client of the appropriateness of the solution, only to find at the end that the solution is actually unworkable. In theory, the rapidly iterative nature should limit this, but it assumes that there's a negative feedback, or even appropriate feedback. If not, the error could be magnified rapidly. This can be alleviated by separating the requirements gathering into a separate phase (a common element of Agile systems), thus insulating it from the developer's influence, or by keeping the client in the loop during development by
having them continuously trying each release. The problem there is that in the real world, most clients are unwilling to invest this much time. It also makes QAing a product difficult since there are no clear test goals that don't change from release to release.

In order to determine the suitability of agile methods individually, a more sophisticated analysis is required. The DSDM method, for example, provides a so-called ‘suitability-filter’ for this purpose. Also, the Crystal family of methods provides criteria on how to select the method for a given project. The selection is based on project size, criticality and priority.
However, other agile methods do not provide such explicit instruments to assess their suitability for a project. Some agile methods, like DSDM and Feature Driven Development (FDD), are claimed to be suitable for any agile software development project, regardless of situational characteristics (Abrahamson et al., 2003).[11]

A comparison of agile methods will reveal that they support different phases of a software development lifecycle to varying degrees. This individual characteristic of agile methods can be used as a selection criterion for selecting candidate agile methods.
Agile development has been widely documented (see Experience Reports, below, as well as Beck[2] pg. 157, and Boehm and Turner[12] pg. 55-57) as working well for small (<10 developers) co-located teams. Agile development is expected to be particularly suitable for teams facing unpredictable or rapidly changing requirements. Agile development's applicability to the following scenarios is open to question:

* Large scale development efforts (>20 developers), though scaling strategies have been described.[13]
Distributed development efforts (non-co-located teams). Strategies have been described in Bridging the Distance[14] and Using an Agile Software Process with Offshore Development[15].

Mission- and life-critical efforts
Command-and-control company cultures

It is worth noting that several large scale project successes have been documented by organisations such as BT which have had several hundred developers situated in the UK, Ireland and India, working collaboratively on projects and using Agile methodologies. While questions undoubtedly still arise about the suitability of some Agile methods to certain project...
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. The authors suggest that each side of the continuum has its own home ground:

**Agile home ground:**
- Low criticality
- Senior developers
- Requirements change very often
- Small number of developers
- Culture that thrives on chaos
Plan-driven home ground:
* High criticality
* Junior developers
* Low requirements change
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.[11]

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 which 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]
In the case of XP the need for method adaptation is made explicit. One of the fundamental ideas of XP is that there is no process that fits every project as such, but rather practices should be tailored to the needs of individual projects. There are also no experience reports in which all the XP practices have been adopted. Instead, a partial adoption of XP practices, as suggested by Beck, has been reported on several occasions.[17]
A distinction can be made between static method adaptation and dynamic method adaptation. 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.
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).

[edit] 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. PRINCE2 has been suggested as a suitable, complementary project management system.

Some of the well-known agile software development methods:
- Extreme Programming (XP)
- Scrum
Agile Modeling

Adaptive Software Development (ASD)

Crystal Clear and Other Crystal Methodologies

Dynamic Systems Development Method (DSDM)

Feature Driven Development (FDD)

Lean software development

Agile Unified Process (AUP)

Other approaches:

Agile Documentation

ICONIX Process

Microsoft Solutions Framework (MSF)

Agile Data Method

Database refactoring
Examples of the use of agile principles in non-software activities:

* Lean manufacturing
While agility is seen by many as a means to an end, a number of approaches have been proposed to quantify agility. Agility Index Measurements (AIM)\[2\] score projects against a number of agility factors to achieve a total. The similarly-named Agility Measurement Index \[3\], scores developments against five dimensions of a software project (duration, risk, novelty, effort, and interaction). Other techniques are based on measurable goals \[4\]. Another study using fuzzy mathematics\[20\] has suggested that project velocity can be used as a metric of agility.
While such approaches have been proposed to measure agility, the practical application of such metrics has yet to be seen.

[Criticism]

Agile development is sometimes criticized as cowboy coding. Extreme Programming's initial buzz and controversial tenets, such as pair programming and continuous design, have attracted particular criticism, such as McBreen[21] and Boehm and Turner.[12] However, much of the criticism was believed by Agile practitioners as misunderstandings about agile development.[22]
In particular, Extreme Programming is reviewed and critiqued by Matt Stephens's and Doug Rosenberg's *Extreme Programming Refactored.*[23]

Criticisms include:

- lack of structure and necessary documentation
- only works with senior-level developers
- incorporates insufficient software design
- requires too much cultural change to adopt
- can lead to more difficult contractual negotiations
The criticisms regarding insufficient software design and lack of documentation are addressed by the Agile Modeling method which can easily be tailored into agile processes such as XP.

[edit] Post-Agilism

In software engineering, post-Agilism is an informal movement of former “Agilistas” (Agile Software Development evangelists) who have chosen to draw from a much wider range of methods and schools of thought on software development, preferring to avoid being constrained by what they consider to be “Agile Dogma”. The term Fragilism is
sometimes used to mean the same thing.

Much of the debate around Post-Agilism centres around the meaning of the word Agile - with a capital 'a' - vs. "agile" (the dictionary definition of the word). In late 2005, Jason Gorman argued that the meaning of Agile was ambiguous and was being inappropriately applied to a very wide range of approaches like Six Sigma and CMMi. He also argued that "Agile", "evolutionary", and "lean" (as in Lean software development or Lean Manufacturing) did not mean the same thing in practice, even though they are all lumped under the banner of "Agile" -
possibly for marketing purposes. Gorman argued that process-oriented methods, especially methods that incrementally reduce waste and process variation like Six Sigma, have a tendency to limit an organisation's adaptive capacity (their "slack"), making them less able to respond to discontinuous change - i.e., less agile. He also argues in later posts that "agile", "lean" and "evolutionary" are strategies that need to be properly understood and appropriately applied to any specific context. That is, there is a time to be "agile", a time to be "lean" and a time to be "evolutionary".
The debate continued on various discussion groups, and transferred into the blogosphere in December 2005. In June 2006 the debate widened and the term Post-Agilism was coined by Jonathan Kohl to describe the growing—but still very loose—association of people extolling "post-Agile" sentiments in their work. Much of the post-Agile thinking centres around Nonlinear Management, a superset of management techniques that include many Agile practices.