# Agile software 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).[1] It advocates adaptive planning, evolutionary development, empirical knowledge, and continual improvement, and it encourages rapid and flexible response to change.[2]

The term *agile* (sometimes written *Agile*)[3] was popularized, in this context, by the *Manifesto for Agile Software Development*.[4] The values and principles espoused in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban.[5][6]

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.[7][8]

## Contents



## History

Iterative and incremental development methods can be traced back as early as 1957,[9] with evolutionary project management[10][11] and adaptive software development[12] emerging in the early 1970s.

During the 1990s, a number of *lightweight* software development methods evolved in reaction to the prevailing *heavyweight* methods that critics described as overly regulated, planned, and micro-managed. These included: rapid application development (RAD), from 1991;[13][14] the unified process (UP) and dynamic systems development method (DSDM), both from 1994; Scrum, from 1995; Crystal Clear and extreme programming (XP), both from 1996; and feature-driven development, from 1997. Although these all originated before the publication of the *Agile Manifesto*, they are now collectively referred to as agile software development methods.[6] At the same time, similar changes were underway in manufacturing[15] and aerospace.[16]

In 2001, seventeen software developers met at a resort in Snowbird, Utah to discuss these lightweight development methods, including among others Kent Beck, Ward Cunningham, Dave Thomas, Jeff Sutherland, Ken Schwaber, Jim Highsmith, Alistair Cockburn, and Bob Martin. Together they published the *Manifesto for Agile Software Development*.[4]

In 2005, a group headed by Cockburn and Highsmith wrote an addendum of project management principles, the PM Declaration of Interdependence,[17] to guide software project management according to agile software development methods.

In 2009, a group working with Martin wrote an extension of software development principles, the Software Craftsmanship Manifesto, to guide agile software development according to professional conduct and mastery.

In 2011, the Agile Alliance created the *Guide to Agile Practices* (renamed the *Agile Glossary* in 2016),[18] an evolving open-source compendium of the working definitions of agile practices, terms, and elements, along with interpretations and experience guidelines from the worldwide community of agile practitioners.

# 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:[4]

- ***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:[19]

- 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.

Some of the authors formed the Agile Alliance, a non-profit organization that promotes software development according to the manifesto's values and principles. Introducing the manifesto on behalf of the Agile Alliance, Jim Highsmith said,

> 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. Those who would brand proponents of XP or SCRUM or any of the other Agile Methodologies as "hackers" are ignorant of both the methodologies and the original definition of the term hacker.
>
> — Jim Highsmith, History: The Agile Manifesto[20]

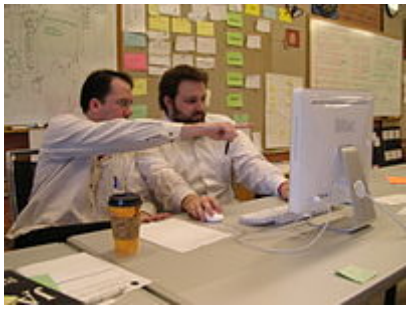
## Agile software development principles

The *Manifesto for Agile Software Development* is based on twelve principles:[21]

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

# Overview

## Iterative, incremental and evolutionary



Pair programming, an agile development technique used by XP. Note information radiators in the background.

Most agile development methods break product development work into small increments that minimize the amount of up-front planning and design. Iterations, or sprints, are short time frames (timeboxes) that typically last from one to four weeks. Each iteration involves a cross-functional team working in all functions: planning, analysis, design, coding, unit testing, and acceptance testing. At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the product to adapt to changes quickly.[22] 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.[23] Multiple iterations might be required to release a product or new features. Working software is the primary measure of progress.[21]

## Efficient and face-to-face communication

The principle of co-location is that co-workers on the same team should be situated together to better establish the identity as a team and to improve communication.[24] This enables face-to-face interaction, ideally in front of a whiteboard, that reduces the cycle time typically taken when questions and answers are mediated through phone, persistent chat, wiki, or email.[25]

No matter which development method is followed, every team should include a customer representative ("Product Owner" in Scrum). This person is agreed by stakeholders to act on their behalf and makes a personal commitment to being available for developers to answer questions throughout the iteration. 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.

In agile software development, an **information radiator** is a (normally large) physical display located prominently near the development team, where passers-by can see it. It presents an up-to-date summary of the product development status.[26][27] A build light indicator may also be used to inform a team about the current status of their product development.

## Very short feedback loop and adaptation cycle

A common characteristic in agile software development is the daily stand-up (also known as the *daily scrum)*. In a brief session, team members report to each other what they did the previous day toward their team's iteration goal, what they intend to do today toward the goal, and any roadblocks or impediments they can see to the goal.[28]

## Quality focus

Specific tools and techniques, such as continuous integration, automated unit testing, pair programming, test-driven development, design patterns, behavior-driven development, domain-driven design, code refactoring and other techniques are often used to improve quality and enhance product development agility.[29] This is predicated on designing and building quality in from the beginning and being able to demonstrate software for customers at any point, or at least at the end of every iteration.[30]

# Philosophy

Compared to traditional software engineering, agile software development mainly targets complex systems and product development with dynamic, non-deterministic and non-linear characteristics. Accurate estimates, stable plans, and predictions are often hard to get in early stages, and confidence in them is likely to be low. Agile practitioners will seek to reduce the *leap-of-faith* that is needed before any evidence of value can be obtained.[31] Requirements and design are held to be emergent. Big up-front specifications would probably cause a lot of waste in such cases, i.e., are not economically sound. These basic arguments and previous industry experiences, learned from years of successes and failures, have helped shape agile development's favor of adaptive, iterative and evolutionary development.[32]

## Adaptive vs. predictive

Development methods exist on a continuum from *adaptive* to *predictive*.[33] Agile software development methods lie on the *adaptive* side of this continuum. One key of adaptive development methods is a *rolling wave* approach to schedule planning, which identifies milestones but leaves flexibility in the path to reach them, and also allows for the milestones themselves to change.[34]

*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 has difficulty describing exactly what will happen in the future. The further away a date is, the more vague an adaptive method is 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 analysing and planning the future in detail and cater for known risks. In the extremes, a predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive methods rely on effective early phase analysis and if this goes very wrong, the project may have difficulty changing direction. Predictive teams often institute a change control board to ensure they consider only the most valuable changes.

Risk analysis can be used to choose between adaptive (*agile* or *value-driven*) and predictive (*plan-driven*) methods.[35] Barry Boehm and Richard Turner suggest that each side of the continuum has its own *home ground*, as follows:[36]

Home grounds of different development methods

| Value-driven methods | Plan-driven methods | 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 |

## Agile vs. waterfall

One of the differences between agile software development methods and waterfall is the approach to quality and testing. In the waterfall model, there is always a separate *testing phase* after a *build phase*; however, in agile software development testing is completed in the same iteration as programming.

Because testing is done in every iteration—which develops a small piece of the software—users can frequently use those new pieces of software and validate the value. After the users know the real value of the updated piece of software, they can make better decisions about the software's future. Having a value retrospective and software re-planning session in each iteration—Scrum

typically has iterations of just two weeks—helps the team continuously adapt its plans so as to maximize the value it delivers. This follows a pattern similar to the PDCA cycle, as the work is *planned*, *done*, *checked* (in the review and retrospective), and any changes agreed are *acted* upon.

This iterative approach supports a *product* rather than a *project* mindset. This provides greater flexibility throughout the development process; whereas on projects the requirements are defined and locked down from the very beginning, making it difficult to change them later. Iterative product development allows the software to evolve in response to changes in business environment or market requirements.[37]

Because of the short iteration style of agile software development, it also has strong connections with the lean startup concept.

### Code vs. documentation

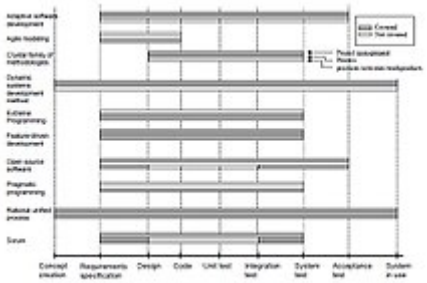
In a letter to *IEEE Computer*, Steven Rakitin expressed cynicism about agile software development, calling it "yet another attempt to undermine the discipline of software engineering" and translating "working software over comprehensive documentation" as "we want to spend all our time coding. Remember, real programmers don't write documentation."[38]

This is disputed by proponents of agile software development, who state that developers should write documentation if that is the best way to achieve the relevant goals, but that there are often better ways to achieve those goals than writing static documentation.[39] Scott Ambler states that documentation should be "just barely good enough" (JBGE),[40] that too much or comprehensive documentation would usually cause waste, and developers rarely trust detailed documentation because it's usually out of sync with code,[39] while too little documentation may also cause problems for maintenance, communication, learning and knowledge sharing. Alistair Cockburn wrote of the *Crystal Clear* method:

> Crystal considers development a series of co-operative games, and intends that the documentation is enough to help the next win at the next game. The work products for Crystal include use cases, risk list, iteration plan, core domain models, and design notes to inform on choices...however there are no templates for these documents and descriptions are necessarily vague, but the objective is clear, **just enough documentation** for the next game. I always tend to characterize this to my team as: what would you want to know if you joined the team tomorrow.
>
> — Alistair Cockburn.[41]

# Agile software development methods

Agile software development methods support a broad range of the software development life cycle.[42] Some focus on the practices (e.g., XP, pragmatic programming, agile modeling), while some focus on managing the flow of work (e.g., Scrum, Kanban). Some support activities for requirements specification and development (e.g., FDD), while some seek to cover the full development life cycle (e.g., DSDM, RUP).



Software development life-cycle support[42]

Popular agile software development frameworks include (but are not limited to):

- Adaptive software development (ASD)
- Agile modeling
- Agile unified process (AUP)
- Disciplined agile delivery
- Dynamic systems development method (DSDM)
- Extreme programming (XP)
- Feature-driven development (FDD)
- Lean software development
- Kanban
- Rapid application development (RAD)
- Scrum
- Scrumban

## Agile software development practices

Agile software development is supported by a number of concrete practices, covering areas like requirements, design, modeling, coding, testing, planning, risk management, process, quality, etc. Some notable agile software development practices include:[43]

- Acceptance test-driven development (ATDD)
- Agile modeling
- Agile testing
- Backlogs (Product and Sprint)
- Behavior-driven development (BDD)
- Business analyst designer method (BADM)[44]
- Continuous integration (CI)
- Cross-functional team
- Domain-driven design (DDD)
- Information radiators (scrum board, task board, visual management board, burndown chart)
- Iterative and incremental development (IID)
- Low-code development platforms
- Pair programming
- Planning poker
- Refactoring
- Retrospective
- Scrum events (sprint planning, daily scrum, sprint review and retrospective)
- Story-driven modeling
- Test-driven development (TDD)
- Timeboxing
- User story
- User story mapping
- 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 determine a system development approach for a specific project situation through responsive changes in, and dynamic interplays between contexts, intentions, and method fragments.
>
> — Mehmet Nafiz Aydin et al., An Agile Information Systems Development Method in use[45]

Situation-appropriateness should be considered as a distinguishing characteristic between agile methods and more plan-driven software development methods, with agile methods allowing product development teams to adapt working practices according to the needs of individual products.[46][45] Potentially, most agile methods could be suitable for method tailoring,[42] such as DSDM tailored in a CMM context.[47] and XP tailored with the *Rule Description Practices* (RDP) technique.[48][49] Not all agile proponents agree, however, with Schwaber noting "that is how we got into trouble in the first place, thinking that the problem was not having a perfect methodology. Efforts [should] center on the changes [needed] in the enterprise".[50] Bas Vodde reinforced this viewpoint, suggesting that unlike traditional, large methodologies that require you to pick and choose elements, Scrum provides the basics on top of which you add additional elements to localise and contextualise its use.[51] Practitioners seldom use system development methods, or agile methods specifically, by the book, often choosing to omit or tailor some of the practices of a method in order to create an in-house method.[52]

In practice, methods can be tailored using various tools. Generic process modeling languages such as Unified Modeling Language can be used to tailor software development methods. However, dedicated tools for method engineering such as the Essence Theory of Software Engineering of SEMAT also exist.[53]

## Large-scale, offshore and distributed

Agile software development has been widely seen as highly suited to certain types of environments, including small teams of experts working on greenfield projects,[36][54]:157 and the challenges and limitations encountered in the adoption of agile software development methods in a large organization with legacy infrastructure are well-documented and understood.[55]

In response, a range of strategies and patterns has evolved for overcoming challenges with large-scale development efforts (>20 developers)[56][57] or distributed (non-colocated) development teams,[58][59] amongst other challenges; and there are now several recognised frameworks that seek to mitigate or avoid these challenges.

- Scaled agile framework (SAFe),[60] Dean Leffingwell *inter alia*
- Disciplined agile delivery (DAD), Scott Ambler *inter alia*
- Large-scale scrum (LeSS), Craig Larman and Bas Vodde
- Nexus (scaled professional Scrum),[61] Ken Schwaber
- Scrum at Scale,[62] Jeff Sutherland, Alex Brown
- Enterprise Scrum,[63] Mike Beedle
- Setchu (Scrum-based lightweight framework),[64] Michael Ebbage
- Xscale[65]
- Agile path[66]
- Holistic Software Development [67]

There are many conflicting viewpoints on whether all of these are effective or indeed fit the definition of agile development, and this remains an active and ongoing area of research.[56][68]

When agile software development is applied in a distributed setting (with teams dispersed across multiple business locations), it is commonly referred to as distributed agile development. The goal is to leverage the unique benefits offered by each approach. Distributed development allow organizations to build software by strategically setting up teams in different parts of the globe, virtually building software round-the-clock (more commonly referred to as follow-the-sun model). On the other hand, agile development provides increased transparency, continuous feedback and more flexibility when responding to changes.

### Regulated domains

Agile software development methods were initially seen as best suitable for non-critical product developments, thereby excluded from use in regulated domains such as medical devices, pharmaceutical, financial, nuclear systems, automotive, and avionics sectors, etc. However, in the last several years, there have been several initiatives for the adaptation of agile methods for these domains.[69][70][71][72][73]

There are numerous standards that may apply in regulated domains, including ISO 26262, ISO 9000, ISO 9001, and ISO/IEC 15504. A number of key concerns are of particular importance in regulated domains:[74]

- Quality assurance (QA): Systematic and inherent quality management underpinning a controlled professional process and reliability and correctness of product.
- Safety and security: Formal planning and risk management to mitigate safety risks for users and securely protecting users from unintentional and malicious misuse.
- Traceability: Documentation providing auditable evidence of regulatory compliance and facilitating traceability and investigation of problems.
- Verification and Validation (V&V): Embedded throughout the software development process (e.g. user requirements specification, functional specification, design specification, code review, unit tests, integration tests, system tests).

# Experience and adoption

Although agile software development methods can be used with any programming paradigm or language in practice, they were originally closely associated with object-oriented environments such as Smalltalk and Lisp and later Java. The initial adopters of agile methods were usually small to medium-sized teams working on unprecedented systems with requirements that were difficult to finalize and likely to change as the system was being developed. This section describes common problems that organizations encounter when they try to adopt agile software development methods as well as various techniques to measure the quality and performance of agile teams.[75]

### Measuring agility

The best agile practitioners have always emphasized thorough engineering principles. As a result, there are a number of best practices and tools for measuring the performance of agile software development and teams.

#### Internal assessments

The *Agility measurement index*, amongst others, rates developments against five dimensions of product development (duration, risk, novelty, effort, and interaction).[76][77] Other techniques are based on measurable goals[78] and one study suggests that velocity can be used as a metric of agility.[79] There are also agile self-assessments to determine whether a team is using agile software development practices (Nokia test,[80] Karlskrona test,[81] 42 points test).[82]

#### Public surveys

One of the early studies reporting gains in quality, productivity, and business satisfaction by using agile software developments methods was a survey conducted by Shine Technologies from November 2002 to January 2003.[83]

A similar survey, the *State of Agile*, is conducted every year starting in 2006 with thousands of participants from around the software development community. This tracks trends on the benefits of agility, lessons learned, and good practices. Each survey has reported increasing numbers saying that agile software development helps them deliver software faster; improves their ability to manage changing customer priorities; and increases their productivity.[84] Surveys have also consistently shown better results with agile product development methods compared to classical project management.[85][86] In balance, there are reports that some feel that agile development methods are still too young to enable extensive academic research of their success.[87]

## Common agile software development pitfalls

Organizations and teams implementing agile software development often face difficulties transitioning from more traditional methods such as waterfall development, such as teams having an agile process forced on them.[88] These are often termed *agile anti-patterns* or more commonly *agile smells*. Below are some common examples:

#### Lack of overall product design

A goal of agile software development is to focus more on producing working software and less on documentation. This is in contrast to waterfall models where the process is often highly controlled and minor changes to the system require significant revision of supporting documentation. However, this does not justify completely doing without any analysis or design at all. Failure to pay attention to design can cause a team to proceed rapidly at first but then to have significant rework required as they attempt to scale up the system. One of the key features of agile software development is that it is iterative. When done correctly design emerges as the system is developed and commonalities and opportunities for re-use are discovered.[89]

#### Adding stories to an iteration in progress

In agile software development, *stories* (similar to use case descriptions) are typically used to define requirements and an *iteration* is a short period of time during which the team commits to specific goals.[90] Adding stories to an iteration in progress is detrimental to a good flow of work. These should be added to the product backlog and prioritized for a subsequent iteration or in rare cases the iteration could be cancelled.[91]

This does not mean that a story cannot expand. Teams must deal with new information, which may produce additional tasks for a story. If the new information prevents the story from being completed during the iteration, then it should be carried over to a subsequent iteration. However, it should be prioritized against all remaining stories, as the new information may have changed the story's original priority.

#### Lack of sponsor support

Agile software development is often implemented as a grassroots effort in organizations by software development teams trying to optimize their development processes and ensure consistency in the software development life cycle. By not having sponsor support, teams may face difficulties and resistance from business partners, other development teams and management. Additionally, they may suffer without appropriate funding and resources.[92] This increases the likelihood of failure.[93]

#### Insufficient training

A survey performed by VersionOne found respondents cited insufficient training as the most significant cause for failed agile implementations[94] Teams have fallen into the trap of assuming the reduced processes of agile software development compared to other methodologies such as waterfall means that there are no actual rules for agile software development.

#### Product owner role is not properly filled

The product owner is responsible for representing the business in the development activity and is often the most demanding role.[95]

A common mistake is to have the product owner role filled by someone from the development team. This requires the team to make its own decisions on prioritization without real feedback from the business. They try to solve business issues internally or delay work as they reach outside the team for direction. This often leads to distraction and a breakdown in collaboration.[96]

#### Teams are not focused

Agile software development requires teams to meet product commitments, which means they should focus only on work for that product. However, team members who appear to have spare capacity are often expected to take on other work, which makes it difficult for them to help complete the work to which their team had committed.[97]

#### Excessive preparation/planning

Teams may fall into the trap of spending too much time preparing or planning. This is a common trap for teams less familiar with agile software development where the teams feel obliged to have a complete understanding and specification of all stories. Teams should be prepared to move forward only with those stories in which they have confidence, then during the iteration continue to discover and prepare work for subsequent iterations (often referred to as backlog refinement or grooming).

#### Problem-solving in the daily standup

A daily standup should be a focused, timely meeting where all team members disseminate information. If problem-solving occurs, it often can only involve certain team members and potentially is not the best use of the entire team's time. If during the daily standup the team starts diving into problem-solving, it should be set aside until a sub-team can discuss, usually immediately after the standup completes. [98]

#### Assigning tasks

One of the intended benefits of agile software development is to empower the team to make choices, as they are closest to the problem. Additionally, they should make choices as close to implementation as possible, to use more timely information in the decision. If team members are assigned tasks by others or too early in the process, the benefits of localized and timely decision making can be lost.[99]

Being assigned work also constrains team members into certain roles (for example, team member A must always do the database work), which limits opportunities for cross-training.[99] Team members themselves can choose to take on tasks that stretch their abilities and provide cross-training opportunities.

#### Scrum master as a contributor

Another common pitfall is for a scrum master to act as a contributor. While not prohibited by the Scrum methodology, the scrum master needs to ensure they have the capacity to act in the role of scrum master first and not working on development tasks. A scrum master's role is to facilitate the process rather than create the product.[100]

Having the scrum master also multitasking may result in too many context switches to be productive. Additionally, as a scrum master is responsible for ensuring roadblocks are removed so that the team can make forward progress, the benefit gained by individual tasks moving forward may not outweigh roadblocks that are deferred due to lack of capacity.[100]

#### Lack of test automation

Due to the iterative nature of agile development, multiple rounds of testing are often needed. Automated testing helps reduce the impact of repeated unit, integration, and regression tests and frees developers and testers to focus on higher value work.[101]

Test automation also supports continued refactoring required by iterative software development. Allowing a developer to quickly run tests to confirm refactoring has not modified the functionality of the application may reduce the workload and increase confidence that cleanup efforts have not introduced new defects.

#### Allowing technical debt to build up

Focusing on delivering new functionality may result in increased technical debt. The team must allow themselves time for defect remediation and refactoring. Technical debt hinders planning abilities by increasing the amount of unscheduled work as production defects distract the team from further progress.[102]

As the system evolves it is important to refactor as entropy of the system naturally increases.[103] Over time the lack of constant maintenance causes increasing defects and development costs.[102]

#### Attempting to take on too much in an iteration

A common misconception is that agile software development allows continuous change, however an iteration backlog is an agreement of what work can be completed during an iteration.[104] Having too much work-in-progress (WIP) results in inefficiencies such as context-switching and queueing.[105] The team must avoid feeling pressured into taking on additional work.[106]

#### Fixed time, resources, scope, and quality

Agile software development fixes time (iteration duration), quality, and ideally resources in advance (though maintaining fixed resources may be difficult if developers are often pulled away from tasks to handle production incidents), while the scope remains variable. The customer or product owner often push for a fixed scope for an iteration. However, teams should be reluctant to commit to the locked time, resources and scope (commonly known as the project management triangle). Efforts to add scope to the fixed time and resources of agile software development may result in decreased quality.[107]

#### Developer burnout

Due to the focused pace and continuous nature of agile practices, there is a heightened risk of burnout among members of the delivery team.[108]

# Agile management

The term *agile management* is applied to an iterative, incremental method of managing the design and build activities of engineering, information technology and other business areas that aim to provide new product or service development in a highly flexible and interactive manner, based on the principles expressed in the *Manifesto for Agile Software Development*.[109]

Agile X techniques may also be called extreme project management. It is a variant of iterative life cycle[110] where deliverables are submitted in stages. The main difference between agile and iterative development is that agile methods complete small portions of the deliverables in each delivery cycle (iteration),[111] while iterative methods evolve the entire set of deliverables over time, completing them near the end of the project. Both iterative and agile methods were developed as a reaction to various obstacles that developed in more sequential forms of project organization. For example, as technology projects grow in complexity, end users tend to have difficulty defining the long-term requirements without being able to view progressive prototypes. Projects that develop in iterations can constantly gather feedback to help refine those requirements.

Agile management also offers a simple framework promoting communication and reflection on past work amongst team members.[112] Teams who were using traditional waterfall planning and adopted the agile way of development typically go through a transformation phase and often take help from agile coaches who help guide the teams through a smooth transformation. There are typically two styles of agile coaching: push-based and pull-based agile coaching. Agile management approaches have also been

employed and adapted to the business and government sectors. For example, within the federal government of the United States, the United States Agency for International Development (USAID) is employing a collaborative project management approach that focuses on incorporating collaborating, learning and adapting (CLA) strategies to iterate and adapt programming.[113]

Agile methods are mentioned in the *Guide to the Project Management Body of Knowledge* (*PMBOK Guide*) under the Project Lifecycle definition:

> **Adaptive project life cycle**, a project life cycle, also known as change-driven or agile methods, that is intended to facilitate change and require a high degree of ongoing stakeholder involvement. Adaptive life cycles are also iterative and incremental, but differ in that iterations are very rapid (usually 2-4 weeks in length) and are fixed in time and resources.[114]

### Applications outside software development

Agile Brazil 2014 conference

According to Jean-Loup Richet (Research Fellow at ESSEC Institute for Strategic Innovation & Services) "this approach can be leveraged effectively for non-software products and for project management in general, especially in areas of innovation and uncertainty." The end result is a product or project that best meets current customer needs and is delivered with minimal costs, waste, and time, enabling companies to achieve bottom line gains earlier than via traditional approaches.[115]

Agile software development methods have been extensively used for development of software products and some of them use certain characteristics of software, such as object technologies.[116] However, these techniques can be applied to the development of non-software products, such as computers, motor vehicles,[117] medical devices, food, clothing, and music.[118] Agile software development methods have been used in non-development IT infrastructure deployments and migrations. Some of the wider principles of agile software development have also found application in general management[119] (e.g., strategy, governance, risk, finance) under the terms business agility or agile business management.

Under an agile business management model, agile software development techniques, practices, principles and values are expressed across five domains.[120]

1. Integrated customer engagement: to embed customers within any delivery process to share accountability for product/service delivery.
2. Facilitation-based management: adopting agile management models, like the role of Scrum Master, to facilitate the day-to-day operation of teams.
3. Agile work practices: adopting specific iterative and incremental work practices such as Scrum, Kanban, test-driven development or feature-driven development across all business functions (from sales, human resources, finance[121] and marketing).
4. An enabling organisational structure: with a focus on staff engagement, personal autonomy and outcomes based governance.
5. Applications of agile process (along with DevOps and lean manufacturing), to data analytics, business intelligence, big data, and data science is called DataOps

Agile software development paradigms can be used in other areas of life such as raising children. Its success in child development might be founded on some basic management principles; communication, adaptation, and awareness. In a TED Talk, Bruce Feiler shared how he applied basic agile paradigms to household management and raising children.[122]

# Criticism

Agile practices can be inefficient in large organizations and certain types of developments.[123] Many organizations believe that agile software development methodologies are too extreme and adopt a Hybrid approach [124] that mixes elements of agile software development and plan-driven approaches.[125] Some methods, such as dynamic systems development method (DSDM) attempt this in a disciplined way, without sacrificing fundamental principles.

The increasing adoption of agile practices has also been criticized as being a management fad that simply describes existing good practices under new jargon, promotes a *one size fits all* mindset towards development strategies, and wrongly emphasizes method over results.[126]

Alistair Cockburn organized a celebration of the 10th anniversary of the *Manifesto for Agile Software Development* in Snowbird, Utah on 12 February 2011, gathering some 30+ people who had been involved at the original meeting and since. A list of about 20 elephants in the room ('undiscussable' agile topics/issues) were collected, including aspects: the alliances, failures and limitations of agile software development practices and context (possible causes: commercial interests, decontextualization, no obvious way to make progress based on failure, limited objective evidence, cognitive biases and reasoning fallacies), politics and culture.[127] As Philippe Kruchten wrote:

> The agile movement is in some ways a bit like a teenager: very self-conscious, checking constantly its appearance in a mirror, accepting few criticisms, only interested in being with its peers, rejecting en bloc all wisdom from the past, just because it is from the past, adopting fads and new jargon, at times cocky and arrogant. But I have no doubts that it will mature further, become more open to the outside world, more reflective, and therefore, more effective.
>
> — Philippe Kruchten[127]

# See also

- Workers' self-management

# References


1. Collier, Ken W. (2011). *Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing*. Pearson Education. pp. 121 ff. ISBN 9780321669544. "What is a self-organizing team?"
2. "What is Agile Software Development?" (http://www.agilealliance.org/the-alliance/what-is-agile/). Agile Alliance. 8 June 2013. Retrieved 4 April 2015.
3. Rally (2010). "Agile With a Capital "A" Vs. agile With a Lowercase "a"" (https://web.archive.org/web/20160105105258/https://www.rallydev.com/blog/engineering/agile-capital-vs-agile-lowercase). Archived from the original on 5 January 2016. Retrieved 9 September 2015.
4. Kent Beck; James Grenning; Robert C. Martin; Mike Beedle; Jim Highsmith; Steve Mellor; Arie van Bennekum; Andrew Hunt; Ken Schwaber; Alistair Cockburn; Ron Jeffries; Jeff Sutherland; Ward Cunningham; Jon Kern; Dave Thomas; Martin Fowler; Brian Marick (2001). "Manifesto for Agile Software Development" (http://agilemanifesto.org/). Agile Alliance. Retrieved 14 June 2010.
5. *Which is better – Kanban or Scrum?* (http://www.cleverpm.com/2016/03/04/which-is-better-kanban-or-scrum/)
6. Larman, Craig (2004). *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley. p. 27. ISBN 978-0-13-111155-4.
7. Dybå, Tore; Dingsøyr, Torgeir (1 August 2008). "Empirical studies of agile software development: A systematic review" (https://www.sciencedirect.com/science/article/pii/S0950584908000256). *Information and Software Technology*. **50** (9–10): 833–859. doi:10.1016/j.infsof.2008.01.006 (https://doi.org/10.1016%2Fj.infsof.2008.01.006). ISSN 0950-5849 (https://www.worldcat.org/issn/0950-5849).
8. Lee, Gwanhoo; Xia, Weidong (2010). "Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility". *MIS Quarterly*. **34** (1): 87–114. doi:10.2307/20721416 (https://doi.org/10.2307%2F20721416). JSTOR 20721416 (https://www.jstor.org/stable/20721416).
9. Gerald M. Weinberg, as quoted in Larman & Basili 2003, pp. 47–56 "We were doing incremental development as early as 1957, in Los Angeles, under the direction of Bernie Dimsdale at IBM's Service Bureau 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.'"
10. "Evolutionary Project Management (Original page, external archive)" (https://web.archive.org/web/20160327214807/http://www.gilb.com/Project-Management). Gilb. Archived from the original (https://www.gilb.com/Project-Management) on 27 March 2016. Retrieved 2017-04-30.
11. "Evolutionary Project Management (New page)" (http://concepts.gilb.com/tiki-page.php?pageName=Evolutionary-Project-Management). Gilb. Retrieved 2017-04-30.
12. Edmonds, E. A. (1974). "A Process for the Development of Software for Nontechnical Users as an Adaptive System". *General Systems*. **19**: 215–18.
13. Martin, James (1991). *Rapid Application Development*. Macmillan. ISBN 978-0-02-376775-3.

14. Kerr, James M.; Hunter, Richard (1993). *Inside RAD: How to Build a Fully Functional System in 90 Days or Less*. McGraw-Hill. p. 3. ISBN 978-0-07-034223-1.
15. Iacocca Institute (1991). "21st Century Manufacturing Enterprise Strategy: An Industry Led View". Iacocca Institute, Lehigh University, Bethlehem, PA.
16. Presley, A., J. Mills and D. Liles (1995). "Agile Aerospace Manufacturing". Nepcon East 1995, Boston.
17. Anderson, David (2005). "Declaration of Interdependence" (https://web.archive.org/web/20180127094805/http://www.pmdoi.org/). Archived from the original (http://pmdoi.org) on 27 January 2018. Retrieved 4 October 2018.
18. McDonald, Kent (1 November 2016). "How You Can Help Agile Alliance Help You" (https://www.agilealliance.org/how-you-can-help-the-agile-alliance-help-you/). *Agile Alliance Blog*. Retrieved 4 July 2017.
19. "Examining the Agile Manifesto" (http://www.ambysoft.com/essays/agileManifesto.html). Ambysoft Inc. Retrieved 6 April 2011.
20. Jim Highsmith (2001). "History: The Agile Manifesto" (http://agilemanifesto.org/history.html). agilemanifesto.org.
21. Kent Beck; James Grenning; Robert C. Martin; Mike Beedle; Jim Highsmith; Steve Mellor; Arie van Bennekum; Andrew Hunt; Ken Schwaber; Alistair Cockburn; Ron Jeffries; Jeff Sutherland; Ward Cunningham; Jon Kern; Dave Thomas; Martin Fowler; Brian Marick (2001). "Principles behind the Agile Manifesto" (http://www.agilemanifesto.org/principles.html). Agile Alliance. Archived (https://web.archive.org/web/20100614043008/http://www.agilemanifesto.org/principles.html) from the original on 14 June 2010. Retrieved 6 June 2010.
22. Moran, A. (2014). *Agile Risk Management*. Springer Verlag. ISBN 978-3319050072.
23. Beck, Kent (1999). "Embracing Change with Extreme Programming". *Computer*. **32** (10): 70–77. doi:10.1109/2.796139 (https://doi.org/10.1109%2F2.796139).
24. Preuss, Deborah Hartmann (13 October 2006). "Study: Co-Located Teams vs. the Cubicle Farm" (https://www.infoq.com/news/collaborative-team-space-study). *InfoQ*. Retrieved 2018-10-23.
25. Cockburn, Alistair (2007). "Agile Software Development: The Cooperative Game" (https://www.pearson.com/us/higher-education/program/Cockburn-Agile-Software-Development-The-Cooperative-Game-2nd-Edition/PGM38838.html). *www.pearson.com* (2nd ed.). Addison-Wesley Professional. Retrieved 2018-10-23.
26. Cockburn, Alistair (19 June 2008). "Information radiator" (http://alistair.cockburn.us/Information+radiator).
27. Ambler, Scott (12 April 2002). *Agile Modeling: Effective Practices for EXtreme Programming and the Unified Process*. John Wiley & Sons. pp. 12, 164, 363. ISBN 978-0-471-20282-0.
28. Vasiliauskas, Vidas (2014). "Developing agile project task and team management practices" (http://www.eylean.com/Publications/DownloadPublication/3443705e-1697-4557-8327-ff8644fab40b?name=Whitepaper---Developing-agile-project-task-and-team-management-practices). Eylean.
29. Jeffries, Ron; Anderson, Ann; Hendrickson, Chet (2001). *Extreme Programming installed*. Addison-Weslsy. pp. 72–147. ISBN 978-0201-70842-4.
30. Lisa Crispin; Janet Gregory (2009). *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley.
31. Mitchell, Ian (2016). *Agile Development in Practice*. Tamare House. p. 11. ISBN 978-1-908552-49-5.
32. Larman, Craig (2004). *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley. p. 27. ISBN 978-0-13-111155-4.
33. Boehm, B.; R. Turner (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley. ISBN 978-0-321-18612-6. Appendix A, pages 165–194
34. Larman, Craig (2004). "Chapter 11: Practice Tips". *Agile and Iterative Development: A Manager's Guide* (https://books.google.com/?id=76rnV5Exs50C&pg=PA253&dq=adaptive%20predictive%20%22rolling%20wave%22#v=onepage&q=adaptive%20predictive%20%22rolling%20wave%22). p. 253. ISBN 9780131111554. Retrieved 14 October 2013.
35. Sliger, Michele; Broderick, Stacia (2008). *The Software Project Manager's Bridge to Agility*. Addison-Wesley. p. 46. ISBN 978-0-321-50275-9.
36. Boehm, B.; R. Turner (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley. pp. 55–57. ISBN 978-0-321-18612-6.
37. "At the Kickoff: Project Development vs Product Development" (https://www.altexsoft.com/blog/business/at-the-kickoff-project-development-vs-product-development/). *AltexSoft Inc*. 12 February 2016. Retrieved 31 May 2016.
38. Rakitin, Steven R. (2001). "Manifesto Elicits Cynicism: Reader's letter to the editor by Steven R. Rakitin". *IEEE Computer*. **34**: 4. "The article titled 'Agile Software Development: The Business of Innovation' ... is yet another attempt to undermine the discipline of software engineering ... We want to spend all our time coding. Remember, real programmers don't write documentation."
39. Scott Ambler. "Agile/Lean Documentation: Strategies for Agile Software Development" (http://www.agilemodeling.com/essays/agileDocumentation.htm).
40. Scott Ambler. "Just Barely Good Enough Models and Documents: An Agile Best Practice" (http://www.agilemodeling.com/essays/barelyGoodEnough.html).

41. Geoffrey Wiseman (July 18, 2007). "Do Agile Methods Require Documentation?" (http://www.infoq.com/news/2007/07/agile-methods-documentation). InfoQ. quoting Cooper, Ian (6 July 2007). "Staccato Signals:Agile and Documentation" (https://ianhammondcooper.wordpress.com/2007/07/06/agile-and-documentation/). *WordPress.com*.
42. Abrahamson P, Salo O, Ronkainen J, Warsta J (2002). *Agile software development methods: Review and analysis* (http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf) (PDF) (Technical report). VTT. 478.
43. "Guide to Agile Practices" (https://web.archive.org/web/20140209152034/http://guide.agilealliance.org/). the Agile Alliance. Archived from the original (http://guide.agilealliance.org/) on 9 February 2014.
44. Heap, Tony. "Business Analyst Designer Method" (http://www.its-all-design.com/business-analyst-designer-method/). *www.its-all-design.com*. Retrieved 2015-08-08.
45. Aydin, M.N.; Harmsen, F.; Slooten; Stagwee, R. A. (2004). "An Agile Information Systems Development Method in use". *Turk J Elec Engin*. **12** (2): 127–138.
46. Morris, David (2015). *The Paradox of Agile Transformation: Why trying too hard to be Agile stops organisations from becoming truly agile*. NZ: University of Auckland. doi:10.13140/RG.2.2.32698.08640 (https://doi.org/10.13140%2FRG.2.2.32698.08640).
47. Abrahamsson, P., Warsta, J., Siponen, M.T., & Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. *Proceedings of ICSE'03*, 244-254
48. Mirakhorli, M.; Rad, A.K.; Shams, F.; Pazoki, M.; Mirakhorli, A. (2008). "RDP technique: a practice to customize xp". *Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral (APOS '08)* (http://dl.acm.org/citation.cfm?id=1370143.1370149). ACM. pp. 23–32. doi:10.1145/1370143.1370149 (https://doi.org/10.1145%2F1370143.1370149). ISBN 978-1-60558-021-0.
49. Aydin, M.N.; Harmsen, F.; van Slooten, 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.
50. Schwaber, K (2006) Scrum is hard and disruptive.
51. Vodde, B (2016) The Story of LeSS. Closing Keynote. Scrum Australia, Melbourne. April, 2016.
52. Lagstedt, A., and Dahlberg, T. (2018). Understanding the Rarity of ISD Method Selection – Bounded Rationality and Functional Stupidity. PACIS 2018 Proceedings. 154. https://aisel.aisnet.org/pacis2018/154.
53. Park, J. S., McMahon, P. E., and Myburgh, B. (2016). Scrum Powered by Essence. ACM SIGSOFT Software Engineering Notes, 41(1), pp. 1-8.
54. Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Boston, MA: Addison-Wesley. ISBN 978-0-321-27865-4.
55. Evans, Ian. "Agile Delivery at British Telecom" (http://www.methodsandtools.com/archive/archive.php?id=43). Retrieved 21 February 2011.
56. W. Scott Ambler (2006) Supersize Me (http://www.drdobbs.com/184415491) in Dr. Dobb's Journal, 15 February 2006.
57. Schaaf, R.J. (2007). Agility XL Systems and Software Technology Conference 2007 (http://www.sstc-online.org/Proceedings/2007/pdfs/RJS1722.pdf) Archived (https://web.archive.org/web/20160313105019/http://sstc-online.org/proceedings/2007/pdfs/rjs1722.pdf) 13 March 2016 at the Wayback Machine, Tampa, FL
58. "Bridging the Distance" (http://www.drdobbs.com/architecture-and-design/184414899). Sdmagazine.com. Retrieved 1 February 2011.
59. Fowler, Martin. "Using an Agile Software Process with Offshore Development" (http://www.martinfowler.com/articles/agileOffshore.html). Martinfowler.com. Retrieved 6 June 2010.
60. Leffingwell, Dean. "Scaled Agile Framework" (http://www.scaledagileframework.com/). *Scaled Agile Framework*.
61. Schwaber, Ken. "Nexus Guide: The Definitive Guide to Nexus: The exoskeleton of scaled Scrum development" (https://www.scrum.org/Portals/0/NexusGuide%20v1.1.pdf) (PDF). *scrum.org*. Retrieved 14 September 2015.
62. Sutherland, Jeff; Brown, Alex. "Scrum At Scale: Part 1" (http://www.scruminc.com/scrum-scale-part-1/). Retrieved 14 September 2015.
63. Beedle, Mike. "Enterprise Scrum" (http://www.enterprisescrum.com/). Retrieved 25 September 2015.
64. Ebbage, Michael. "Setchu – Agile at Scale" (http://agile-setchu.org/). Retrieved 30 September 2015.
65. http://agiletng.org/2014/04/21/xscale/
66. http://www.agile-path.com/
67. http://www.holistic-software.com
68. Agile Processes Workshop II Managing Multiple Concurrent Agile Projects. Washington: OOPSLA 2002
69. Cawley, Oisín; Wang, Xiaofeng; Richardson, Ita (2010). Abrahamsson, Pekka; Oza, Nilay, eds. "Lean/Agile Software Development Methodologies in Regulated Environments – State of the Art" (https://link.springer.com/chapter/10.1007/978-3-642-16416-3_4). *Lean Enterprise Software and Systems*. Lecture Notes in Business Information Processing. **65**: 31–36. doi:10.1007/978-3-642-16416-3_4 (https://doi.org/10.1007%2F978-3-642-16416-3_4). hdl:10344/683 (https://hdl.handle.net/10344%2F683). ISBN 978-3-642-16415-6.

70. McHugh, Martin; McCaffery, Fergal; Coady, Garret (2014-11-04). Mitasiunas, Antanas; Rout, Terry; O'Connor, Rory V.; et al., eds. "An Agile Implementation within a Medical Device Software Organisation" (https://link.springer.com/chapter/10.1007/978-3-319-13036-1_17). *Software Process Improvement and Capability Determination*. Communications in Computer and Information Science. **477**: 190–201. doi:10.1007/978-3-319-13036-1_17 (https://doi.org/10.1007%2F978-3-319-13036-1_17). ISBN 978-3-319-13035-4.
71. Wang, Yang; Ramadani, Jasmin; Wagner, Stefan (2017-11-29). "An Exploratory Study on Applying a Scrum Development Process for Safety-Critical Systems" (https://link.springer.com/chapter/10.1007/978-3-319-69926-4_23). *Product-Focused Software Process Improvement*. Lecture Notes in Computer Science. **10611**: 324–340. arXiv:1703.05375 (https://arxiv.org/abs/1703.05375). doi:10.1007/978-3-319-69926-4_23 (https://doi.org/10.1007%2F978-3-319-69926-4_23). ISBN 9783319699257.
72. http://www.sintef.no/safescrum
73. Thor Myklebust, Tor Stålhane, Geir Kjetil Hanssen, Tormod Wien and Børge Haugset: Scrum, documentation and the IEC 61508-3:2010 software standard, http://www.sintef.no/globalassets/ec-61508-documentation-and-safescrum-psam12.pdf
74. Fitzgerald, B.; Stol, K.-J.; O'Sullivan, R.; O'Brien, D. (May 2013). "Scaling agile methods to regulated environments: An industry case study" (http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6606635). *2013 35th International Conference on Software Engineering (ICSE)*: 863–872. doi:10.1109/ICSE.2013.6606635 (https://doi.org/10.1109%2FICSE.2013.6606635). hdl:10344/3055 (https://hdl.handle.net/10344%2F3055). ISBN 978-1-4673-3076-3.
75. Beck, Kent (2000). *Extreme Programming Explained*. Addison-Wesley. pp. 1–24. ISBN 978-0201616415.
76. Datta, Subhajit (2006). "Agility measurement index: a metric for the crossroads of software development methodologies". *ACM-SE 44 Proceedings of the 44th annual Southeast regional conference*. p. 271. doi:10.1145/1185448.1185509 (https://doi.org/10.1145%2F1185448.1185509). ISBN 1595933158.
77. "David Bock's Weblog : Weblog" (https://web.archive.org/web/20060111041831/http://jroller.com/page/bokmann?entry=improving_your_processes_aim_high). Jroller.com. Archived from the original (http://jroller.com/page/bokmann?entry=improving_your_processes_aim_high) on 11 January 2006. Retrieved 2 April 2010.
78. Peter Lappo; Henry C.T. Andrew. "Assessing Agility" (http://www.smr.co.uk/presentations/measure.pdf) (PDF). Retrieved 6 June 2010.
79. 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.
80. Joe Little (2 December 2007). "Nokia test, A scrum-specific test" (http://agileconsortium.blogspot.com/2007/12/nokia-test.html). Agileconsortium.blogspot.com. Retrieved 6 June 2010.
81. Mark Seuffert; Mayberg, Sweden. "Karlskrona test, A generic agile adoption test" (http://mayberg.se/learning/karlskrona-test). Mayberg.se. Retrieved 5 Apr 2014.
82. "How Agile Are You? (Take This 42 Point Test)" (http://www.allaboutagile.com/how-agile-are-you-take-this-42-point-test/). allaboutagile.com/. Retrieved 3 April 2014.
83. "Agile Methodologies Survey Results" (https://web.archive.org/web/20100821225423/http://www.shinetech.com/attachments/104_ShineTechAgileSurvey2003-01-17.pdf) (PDF). Shine Technologies. January 2003. Archived from the original (http://www.shinetech.com/attachments/104_ShineTechAgileSurvey2003-01-17.pdf) (PDF) on 21 August 2010. 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"
84. "2013 State of Agile report: Why Agile?" (https://web.archive.org/web/20140828012224/http://stateofagile.versionone.com/why-agile/). stateofagile.com. 27 January 2014. Archived from the original (http://stateofagile.versionone.com/why-agile/) on 28 August 2014. Retrieved 13 August 2014.
85. Status Quo Agile (http://www.status-quo-agile.net), Second study on success and forms of usage of agile methods. Retrieved 1 July 2015
86. 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*. Retrieved 3 June 2010. "Only 6% indicated that their productivity was lowered ... No change in productivity was reported by 34% of respondents and 60% reported increased productivity ... 66% [responded] that the quality is higher ... 58% of organizations report improved satisfaction, whereas only 3% report reduced satisfaction."
87. "Answering the "Where is the Proof That Agile Methods Work" Question" (http://www.agilemodeling.com/essays/proof.htm). Agilemodeling.com. 19 January 2007. Retrieved 2 April 2010.
88. Shore & Warden 2008, p. 47
89. Beck, Kent (2000). *Extreme Programming Explained*. Addison-Wesley. pp. 48–49. ISBN 978-0201616415.
90. Rouse, Margaret. "Sprint (software development) definition" (http://searchsoftwarequality.techtarget.com/definition/Scrum-sprint). *searchsoftwarequality.techtarget.com*. Retrieved 2 October 2015.
91. Goldstein, Ilan. "Sprint issues – when sprints turn into crawls" (http://www.axisagile.com.au/blog/planning-and-metrics/sprint-issues-when-sprints-turn-into-crawls/). *www.axisagile.com.au*. Retrieved 2014-06-08.

92. "Project Roles and Responsibility Distribution" (http://agile-only.com/master-thesis/project-mgmt/pr-and-rd). *agile-only.com*. Retrieved 2014-06-15.
93. Bourne, Lynda. "What Does a Project Sponsor Really Do?" (http://blogs.pmi.org/blog/voices_on_project_management/2012/04/what-does-a-project-sponsor-re.html). *blogs.pmi.org*. Retrieved 2014-06-08.
94. "9th State of Agile Report" (http://www.versionone.com/state_of_agile_development_survey/09/page5.asp). *Stage of Agile Survey*. VersionOne. Retrieved 2014-06-08.
95. Sims, Chris; Johnson, Hillary Louise (2011-02-15). *The Elements of Scrum* (Kindle ed.). Dymaxicon. p. 73.
96. Rothman, Johanna Rothman. "When You Have No Product Owner At All" (http://www.jrothman.com/blog/mpd/2011/08/when-you-have-no-product-owner-at-all.html). *www.jrothman.com*. Retrieved 2014-06-08.
97. Fox, Alyssa. "Working on Multiple Agile Teams" (http://techwhirl.com/working-multiple-agile-teams/). *techwhirl.com/*. Retrieved 2014-06-14.
98. "Daily Scrum Meeting" (http://www.mountaingoatsoftware.com/agile/scrum/daily-scrum). *www.mountaingoatsoftware.com*. Retrieved 2014-06-14.
99. May, Robert. "Effective Sprint Planning" (https://web.archive.org/web/20140628102810/http://www.agileexecutives.org/Blogs/tabid/66/EntryId/18/Effective-Sprint-Planning.aspx). *www.agileexecutives.org*. Archived from the original (http://www.agileexecutives.org/Blogs/tabid/66/EntryId/18/Effective-Sprint-Planning.aspx) on 28 June 2014. Retrieved 2014-06-14.
100. Berczuk, Steve. "Mission Possible: ScrumMaster and Technical Contributor" (http://www.agileconnection.com/article/mission-possible-scrummaster-and-technical-contributor?page=0%2C1). *www.agileconnection.com*. Retrieved 2014-06-14.
101. Namta, Rajneesh. "Thoughts on Test Automation in Agile" (http://www.infoq.com/articles/thoughts-on-test-automation-in-agile). *www.infoq.com*. Retrieved 2014-06-14.
102. Band, Zvi. "Technical Debt + Red October" (http://zviband.com/posts/technical-debt-red-october/). Retrieved 8 June 2014.
103. Shore, James. "The Art of Agile Development: Refactoring" (http://www.jamesshore.com/Agile-Book/refactoring.html). *www.jamesshore.com*. Retrieved 2014-06-14.
104. "Step 4: Sprint Planning (Tasks)" (http://www.allaboutagile.com/how-to-implement-scrum-in-10-easy-steps-step-4-sprint-planning-tasks/). *www.allaboutagile.com*. Retrieved 2014-06-14.
105. George, Claire. "Why Limiting Your Work-in-Progress Matters" (http://leankit.com/blog/2014/03/limiting-work-in-progress/). *leankit.com*. Retrieved 2014-06-14.
106. "Sprint Planning Meeting" (http://www.mountaingoatsoftware.com/agile/scrum/sprint-planning-meeting/). *www.mountaingoatsoftware.com*. Retrieved 2014-06-14.
107. McMillan, Keith. "Time, Resources, Scope... and Quality" (http://www.adeptechllc.com/2010/05/13/time-resources-scope-and-quality/). *www.adeptechllc.com*. Retrieved 2014-06-15.
108. "Current study on limitations of Agile" (https://ac.els-cdn.com/S1877050916000582/1-s2.0-S1877050916000582-main.pdf?_tid=c457f048-fb52-11e7-898a-00000aacb35e&acdnat=1516172036_b8b055d4f50de7743058b83879f35f74//) (PDF).
109. Moran, Alan (2015). *Managing Agile: Strategy, Implementation, Organisation and People*. Springer. ISBN 978-3-319-16262-1.
110. ExecutiveBrief, Which Life Cycle Is Best For Your Project? (http://www.pmhut.com/which-life-cycle-is-best-for-your-project), PM Hut. Accessed 23 October 2009.
111. "Agile Project Management" (http://www.versionone.com/agile-project-management/). VersionOne. Retrieved 1 June 2015.
112. "What is Agile Management?" (http://www.project-laneways.com.au/certification-courses/agilepm/what-is-agile-management). Project Laneways. Retrieved 1 June 2015.
113. USAID. "ADS Chapter 201 Program Cycle Operational Policy" (https://www.usaid.gov/sites/default/files/documents/1870/201.pdf). Retrieved 19 April 2017
114. Project Management Institute, *A Guide to the Project Management Body of Knowledge* (PMBOK Guide), Fifth Edition
115. Richet, Jean-Loup (2013). *Agile Innovation*. Cases and Applied Research, n°31. ESSEC-ISIS. ISBN 978-2-36456-091-8
116. Smith, Preston G (2007). *Flexible Product Development*. Jossey-Bass. p. 25. ISBN 978-0-7879-9584-3.
117. "WIKISPEED – Applying Agile software principles and practices for fast automotive development" (http://agilebusinessmanagement.org/content/wikispeed-%E2%80%93-applying-agile-software-principles-and-practices-fast-automotive-development). Agile Business Management Consortium. 2013-12-03. Retrieved 2015-09-11.
118. Newton Lee (2014). *"Getting on the Billboard Charts: Music Production as Agile Software Development,"* Digital Da Vinci: Computers in Music. Springer Science+Business Media. ISBN 978-1-4939-0535-5.
119. Moran, Alan (2015). *Managing Agile: Strategy, Implementation, Organisation and People*. Springer Verlag. ISBN 978-3-319-16262-1.
120. Leybourn, Evan (2013). *Directing the Agile Organisation: A Lean Approach to Business Management*. IT Governance Publishing. ISBN 978-1-849-28491-2.
121. "Pair Trading: Collaboration in Finance" (http://theagiledirector.com/article/2015/03/11/pair-trading-collaboration-in-finance/). The Agile Director. 2015-03-11. Retrieved 2015-09-11.

122. "Agile programming – for your family" (http://www.ted.com/talks/bruce_feiler_agile_programming_for_your_family.html).
123. Larman, Craig; Bas Vodde (2009-08-13). "Top Ten Organizational Impediments to Large-Scale Agile Adoption" (http://www.informit.com/articles/article.aspx?p=1380615). InformIT.
124. "Introduction to Hybrid project management" (https://www.binfire.com/blog/2016/07/hybrid-project-management-methodology//).
125. 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.
126. Kupersmith, Kupe. "Agile is a Fad" (http://www.batimes.com/kupe-kupersmith/agile-is-a-fad.html).
127. Kruchten, Philippe (2011-06-20). "Agile's Teenage Crisis?" (http://www.infoq.com/articles/agile-teenage-crisis). InfoQ.

# Further reading

- 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" (https://books.google.com/books?id=N-06uoJ9iSsC&pg=PA1). In Zelkowitz, Marvin. *Advances in Software Engineering*. Advances in Computers. **62**. Academic Press. pp. 1–66. ISBN 978-0-08-047190-7.
- Dingsøyr, Torgeir; Dybå, Tore; Moe, Nils Brede (2010). *Agile Software Development: Current Research and Future Directions* (https://books.google.com/books?id=JRhGAAAAQBAJ). Springer. ISBN 978-3-642-12575-1.
- Fowler, Martin (2001). "Is Design Dead?" (http://www.martinfowler.com/articles/designDead.html). In Succi, Giancarlo; Marchesi, Michele. *Extreme Programming Examined*. Addison-Wesley. pp. 3–18. ISBN 978-0-201-71040-3.
- Larman, Craig; Basili, Victor R. (June 2003). "Iterative and Incremental Development: A Brief History" (https://pdfs.semanticscholar.org/f9b3/ca89c69bacfade039c8be40762c6857bda11.pdf) (PDF). *IEEE Computer*. **36** (3): 47–56. doi:10.1109/MC.2003.1204375 (https://doi.org/10.1109%2FMC.2003.1204375).
- "Handbook for Implementing Agile in Department of Defense Information Technology Acquisition" (https://www.mitre.org/publications/technical-papers/handbook-for-implementing-agile-in-department-of-defense-information-technology-acquisition). MITRE.
- Moran, Alan (2015). *Managing Agile: Strategy, Implementation, Organisation and People* (https://books.google.com/books?id=I6l_BwAAQBAJ). Springer. ISBN 978-3-319-16262-1.
- 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). In Succi & Marchesi 2001
- Shore, James; Warden, Shane (2008). *The Art of Agile Development* (https://books.google.com/books?id=2q6bAgAAQBAJ). O'Reilly Media. ISBN 978-0-596-52767-9.
- Stephens, M.; Rosenberg, D. (2003). *Extreme Programming Refactored: The Case Against XP*. Apress. ISBN 978-1-59059-096-6.
- Willison, Brian (2008). Iterative Milestone Engineering Model. New York, NY.
- Willison, Brian (2008). Visualization Driven Rapid Prototyping. Parsons Institute for Information Mapping.
- Sondra Ashmore Ph.D; Kristin Runyan (2015). *Introduction to Agile Methods*. Addison-Wesley.

# External links

- Agile Manifesto (http://agilemanifesto.org/)
- Agile Glossary (https://www.agilealliance.org/agile101/agile-glossary/)
- 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--)
- AgilePatterns.org (http://agilepatterns.org/)

---