Agile software development

Agile software development is an umbrella term for approaches to developing software that reflect the values and principles agreed upon by *The Agile Alliance*, a group of 17 software practitioners in 2001.[1] As documented in their *Manifesto for Agile Software Development* the practitioners value:[2]

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

The practitioners cite inspiration from new practices at the time including extreme programming, scrum, dynamic systems development method, adaptive software development and being sympathetic to the need for an alternative to documentation driven, heavyweight software development processes. [3]

Many software development practices emerged from the agile mindset. These agile-based practices, sometimes called *Agile* (with a capital A)[4] include requirements, discovery and solutions improvement through the collaborative effort of self-organizing and cross-functional teams with their customer(s)/end user(s).[5][6]

While there is much anecdotal evidence that the agile mindset and agile-based practices improve the software development process, the empirical evidence is limited and less than conclusive.[7][8][9]

History

Iterative and incremental software development methods can be traced back as early as 1957,[10] with evolutionary project management[11][12] and adaptive software development[13] emerging in the early 1970s.[14]

During the 1990s, a number of *lightweight* software development methods evolved in reaction to the prevailing *heavyweight* methods (often referred to collectively as *waterfall*) that critics described as overly regulated, planned, and micromanaged.[15] These lightweight methods included: rapid application development (RAD), from 1991;[16][17] 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 (FDD), from 1997. Although these all originated before the publication of the *Agile Manifesto*, they are now collectively referred to as agile software development methods.[3]

Already since 1991 similar changes had been underway in manufacturing[18][19] and management thinking[20] derived from Lean management.

In 2001, seventeen software developers met at a resort in Snowbird, Utah to discuss lightweight development methods. They were: Kent Beck (Extreme Programming), Ward Cunningham (Extreme Programming), Dave Thomas (Pragmatic Programming, Ruby), Jeff Sutherland (Scrum), Ken Schwaber

(Scrum), Jim Highsmith (Adaptive Software Development), Alistair Cockburn (Crystal), Robert C. Martin (SOLID), Mike Beedle (Scrum), Arie van Bennekum, Martin Fowler (OOAD and UML), James Grenning, Andrew Hunt (Pragmatic Programming, Ruby), Ron Jeffries (Extreme Programming), Jon Kern, Brian Marick (Ruby, Test-driven development), and Steve Mellor (OOA). The group, The Agile Alliance, published the *Manifesto for Agile Software Development*.[2]

In 2005, a group headed by Cockburn and Highsmith wrote an addendum of project management principles, the PM Declaration of Interdependence,[21] 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),[22] 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.

Values and Principles

Values

The agile manifesto reads:[2]

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions*** *over processes and tools*
- ***Working software*** *over comprehensive documentation*
- ***Customer collaboration*** *over contract negotiation*
- ***Responding to change*** *over following a plan*

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

Scott Ambler explained:[23]

- 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 not a 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.

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[24]

Principles

The values are based on these principles:[25]

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

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)[26] that typically last from one to four weeks.[27]:20 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.[28][29] 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.[30] Through incremental development, products have room to "fail often and early" throughout each iterative phase instead of drastically on a final release date.[31] Multiple iterations might be required to release a product or new features. Working software is the primary measure of progress.[25]

A key advantage of agile approaches is speed to market and risk mitigation. Smaller increments are typically released to market, reducing the time and cost risks of engineering a product that doesn't meet user requirements.

Efficient and face-to-face communication

The 6th principle of the agile manifesto for software development states "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation". The manifesto, written in 2001 when video conferencing was not widely used, states this in relation to the communication of information, not necessarily that a team should be co-located.

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.[32] 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.[33] With the widespread adoption of remote working during the COVID-19 pandemic and changes to tooling, more studies have been conducted[34] around co-location and distributed working which show that co-location is increasingly less relevant.

No matter which development method is followed, every team should include a customer representative (known as *product owner* in Scrum). This representative 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, the project stakeholders together with 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. The importance of stakeholder satisfaction, detailed by frequent interaction and review at the end of each phase, is why the approach is often denoted as a customer-centered methodology.[35]

Information radiator

In agile software development, an **information radiator** is a (normally large) physical display, board with sticky notes or similar, located prominently near the development team, where passers-by can see it.[36] It presents an up-to-date summary of the product development status.[37][38] 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 (known as *daily scrum* in the Scrum framework). In a brief session (e.g., 15 minutes), team members review collectively how they are progressing toward their goal and agree whether they need to adapt their approach. To keep to the agreed time limit, teams often use simple coded questions (such as what they completed the previous day, what they aim to complete that day, and whether there are any impediments or risks to progress), and delay detailed discussions and problem resolution until after the stand-up.[39]

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.[40] 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.[41]

Pair programming, an agile development technique used in XP

Philosophy

Compared to traditional software engineering, agile software development mainly targets complex systems and product development with dynamic, indeterministic and non-linear properties. 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 use their free will to reduce the "leap of faith" that is needed before any evidence of value can be obtained.[42] 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.[43]

Adaptive vs. predictive

Development methods exist on a continuum from *adaptive* to *predictive*.[44] 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.[45]

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 analyzing 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.[46] Barry Boehm and Richard Turner suggest that each side of the continuum has its own *home ground*, as follows:[47]

Home grounds of different development methods

Value-driven methods (agile)	Plan-driven methods (waterfall)	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, work moves through software development life cycle (SDLC) phases—with one phase being completed before another can start—hence the ***testing phase*** is separate and follows a ***build phase***. In agile software development, however, 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 plan-do-check-act (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.

Code vs. documentation

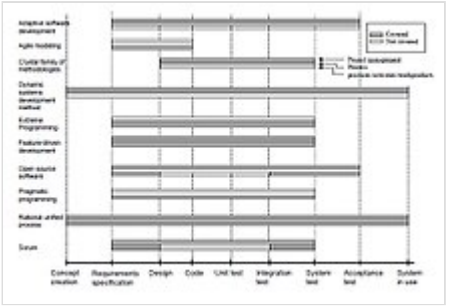
In a letter to *IEEE Computer*, Steven Rakitin expressed cynicism about agile software development, calling it "*yet another attempt to undermine the discipline of software engineering*" and translating "*working software over comprehensive documentation*" as "*we want to spend all our time coding. Remember, real programmers don't write documentation*."[48]

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.[49] Scott Ambler states that documentation should be "just barely good enough" (JBGE),[50] 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,[49] 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[51]

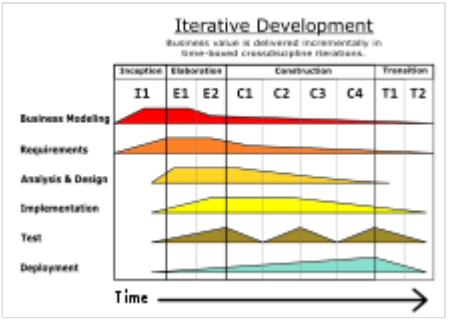
Methods

Agile software development methods support a broad range of the software development life cycle.[52] Some methods 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[52]

Notable agile software development frameworks include:



Agile unified process (AUP) is based on unified process (an iterative and incremental software development process framework).

Framework	Main contributor(s)
Adaptive software development (ASD)	Jim Highsmith, Sam Bayer
Agile modeling	Scott Ambler, Robert Cecil Martin
Agile unified process (AUP)	Scott Ambler
Disciplined agile delivery	Scott Ambler
Dynamic systems development method (DSDM)	Jennifer Stapleton
Extreme programming (XP)	Kent Beck, Robert Cecil Martin
Feature-driven development (FDD)	Jeff De Luca
Lean software development	Mary Poppendieck, Tom Poppendieck
Lean startup	Eric Ries
Kanban	Taiichi Ohno
Rapid application development (RAD)	James Martin
Scrum	Ken Schwaber, Jeff Sutherland
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:[53]

Practice	Main contributor(s)
Acceptance test-driven development (ATDD)	Ken Pugh
Agile modeling	Scott Ambler
Agile testing	Lisa Crispin, Janet Gregory
Backlogs (Product and Sprint)	Ken Schwaber, Jeff Sutherland
Behavior-driven development (BDD)	Dan North, Liz Keogh
Continuous integration (CI)	Grady Booch
Cross-functional team	
Daily stand-up / Daily Scrum	James O Coplien
Domain-driven design (DDD)	Eric Evans
Iterative and incremental development (IID)	
Pair programming	Kent Beck
Planning poker	James Grenning, Mike Cohn
Refactoring	Martin Fowler
Retrospective	Esther Derby, Diana Larsen, Ben Linders, Luis Gonçalves
Scrum events (sprint planning, sprint review and retrospective)	Ken Schwaber, Jeff Sutherland
Specification by example	
Story-driven modeling	Albert Zündorf
Test-driven development (TDD)	Kent Beck
Timeboxing	
User story	Alistair Cockburn
Velocity tracking	

Acceptance test-driven development

Acceptance test–driven development (ATDD) is a development methodology based on communication between the business customers, the developers, and the testers.[54] ATDD encompasses many of the same practices as specification by example (SBE),[55][56] behavior-driven development (BDD),[57] example-driven development (EDD),[58] and support-driven development also called story test–driven development (SDD).[59] All these processes aid developers and testers in understanding the customer's needs prior to implementation and allow customers to be able to converse in their own domain language.

Agile modeling

Agile modeling (AM) is a methodology for modeling and documenting software systems based on best practices. It is a collection of values and principles that can be applied on an (agile) software development project. This methodology is more flexible than traditional modeling methods, making it a better fit in a fast-changing environment.[60] It is part of the agile software development tool kit.

Agile testing

Agile testing is a software testing practice that follows the principles of agile software development. Agile testing involves all members of a cross-functional agile team, with special expertise contributed by

testers, to ensure delivering the business value desired by the customer at frequent intervals, working at a sustainable pace. Specification by example is used to capture examples of desired and undesired behavior and guide coding.

Backlogs

Within agile project management, product backlog refers to a prioritized list of functionality which a product should contain. It is sometimes referred to as a to-do list,[61] and is considered an 'artifact' (a form of documentation) within the scrum software development framework.[62] The product backlog is referred to with different names in different project management frameworks, such as *product backlog* in scrum,[62][63] work item list in disciplined agile,[63][64] and option pool in lean.[63] In the scrum framework, creation and continuous maintenance of the product backlog is part of the responsibility of the product owner.[65]

Behavior-driven development

Behavior-driven development (BDD) involves naming software tests using domain language to describe the behavior of the code.

Continuous integration

Continuous integration (CI) is the practice of integrating source code changes frequently and ensuring that the integrated codebase is in a workable state.

Cross-functional team

A cross-functional team (XFN), also known as a multidisciplinary team or interdisciplinary team,[66][67][68] is a group of people with different functional expertise working toward a common goal.[69] It may include people from finance, marketing, operations, and human resources departments. Typically, it includes employees from all levels of an organization. Members may also come from outside an organization (in particular, from suppliers, key customers, or consultants).

Daily stand-up

A stand-up meeting (stum) is a meeting in which attendees typically participate while standing. The discomfort of standing for long periods is intended to keep the meetings short.

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[70]

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.[71][70] Potentially, most agile methods could be suitable for method tailoring,[52] such as DSDM tailored in a CMM context.[72]

and XP tailored with the *Rule Description Practices* (RDP) technique.[73] 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".[74] 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 localize and contextualize its use.[75] 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.[76]

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.[77]

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,[47][78] 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.[79]

In response, a range of strategies and patterns has evolved for overcoming challenges with large-scale development efforts (>20 developers)[80][81] or distributed (non-colocated) development teams,[82][83] amongst other challenges; and there are now several recognized frameworks that seek to mitigate or avoid these challenges.

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.[80][84]

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 software development. The goal is to leverage the unique benefits offered by each approach. Distributed development allows 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.[85][86][87][88][89]

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:[90]

- 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, Lisp and later Java, C#. 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.[91]

Measuring agility

Internal assessments

The *Agility measurement index*, amongst others, rates developments against five dimensions of product development (duration, risk, novelty, effort, and interaction).[92] Other techniques are based on measurable goals[93] and one study suggests that velocity can be used as a metric of agility. There are also agile self-assessments to determine whether a team is using agile software development practices (Nokia test,[94] Karlskrona test,[95] 42 points test).[96]

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.[97]

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 perceived 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.[98] Surveys have also consistently shown better results with agile product development methods compared to classical project management.[99][100] In balance, there are reports that some feel that agile development methods are still too young to enable extensive academic research of their success.[101]

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.[102] 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 require significant rework 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, agile software development allows the design to emerge as the system is developed and helps the team discover commonalities and opportunities for re-use.[103]

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.[104] 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.[105]

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.[106] This increases the likelihood of failure.[107]

Insufficient training

A survey performed by VersionOne found respondents cited insufficient training as the most significant cause for failed agile implementations[108] Teams have fallen into the trap of assuming the reduced processes of agile software development compared to other approaches 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.[109]

A common mistake is to fill the product owner role with 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.[110]

Teams are not focused

Agile software development requires teams to meet product commitments, which means they should focus on work for only 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.[111]

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 with only 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 involve only 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.[112]

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.[113]

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.[113] Team members themselves can choose to take on tasks that stretch their abilities and provide cross-training opportunities.

Scrum master as a contributor

In the Scrum framework, which claims to be consistent with agile values and principles, the *scrum master* role is accountable for ensuring the scrum process is followed and for coaching the scrum team through that process. A common pitfall is for a scrum master to act as a contributor. While not prohibited by the Scrum framework, the scrum master needs to ensure they have the capacity to act in the role of scrum master first and not work on development tasks. A scrum master's role is to facilitate the process rather than create the product.[114]

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.[115]

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.[116]

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.[117]

As the system evolves it is important to refactor.[118] Over time the lack of constant maintenance causes increasing defects and development costs.[117]

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.[119] Having too much work-in-progress (WIP) results in inefficiencies such as context-switching and queueing.[120] The team must avoid feeling pressured into taking on additional work.[121]

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 pushes 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.[122]

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.[123]

Agile management

Agile project management is an iterative development process, where feedback is continuously gathered from users and stakeholders to create the right user experience. Different methods can be used to perform an agile process, these include scrum, extreme programming, lean and kanban.[124] 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*.[125] Agile project management metrics help reduce confusion, identify weak points, and measure team's performance throughout the development cycle. Supply chain agility is the ability of a supply chain to cope with uncertainty and variability on offer and demand. An agile supply chain can increase and reduce its capacity rapidly, so it can adapt to a fast-changing customer demand. Finally, strategic agility is the ability of an organisation to change its course of action as its environment is evolving. The key for strategic agility is to recognize external changes early enough and to allocate resources to adapt to these changing environments.[124]

Agile X techniques may also be called extreme project management. It is a variant of iterative life cycle[126] 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),[127] 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.[128] 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 smoother transformation. There are typically two styles of agile coaching: push-based and pull-based agile coaching. Here a "push-system" can refer to an upfront estimation of what tasks can be fitted into a sprint (pushing work) e.g. typical with scrum; whereas a "pull system" can refer to an environment where tasks are only performed when capacity is available.[129] 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.[130]

Agile methods are mentioned in the *Guide to the Project Management Body of Knowledge* (*PMBOK Guide 6th Edition*) under the **Product Development Lifecycle** definition:

> Within a project life cycle, there are generally one or more phases that are associated with the development of the product, service, or result. These are called a development life cycle (...) Adaptive life cycles are agile, iterative, or incremental. The detailed scope is defined and approved before the start of an iteration. Adaptive life cycles are also referred to as agile or change-driven life cycles.[131]

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 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.[132]

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.[133] However, these techniques can be applied to the development of non-software products, such as computers, medical devices, food, clothing, and music.[134] 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[135] (e.g., strategy, governance, risk, finance) under the terms business agility or agile business management. Agile software methodologies have also been adopted for use with the learning engineering process, an iterative data-informed process that applies the learning sciences, human-centered design, and data informed decision-making to support learners and their development.[136]

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.[137]

Criticism

Agile practices have been cited as potentially inefficient in large organizations and certain types of development.[138] Many organizations believe that agile software development methodologies are too extreme and adopt a hybrid approach[139] that mixes elements of agile software development and plan-driven approaches.[140] 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.[141]

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.[142] 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[142]

The "Manifesto" may have had a negative impact on higher education management and leadership, where it suggested to administrators that slower traditional and deliberative processes should be replaced with more "nimble" ones. The concept rarely found acceptance among university faculty.[143]

Another criticism is that in many ways, agile management and traditional management practices end up being in opposition to one another. A common criticism of this practice is that the time spent attempting to learn and implement the practice is too costly, despite potential benefits. A transition from traditional management to agile management requires total submission to agile and a firm commitment from all members of the organization to seeing the process through. Issues like unequal results across the organization, too much change for employees' ability to handle, or a lack of guarantees at the end of the transformation are just a few examples.[144]

See also

- Agile Automation
- Cross-functional team
- Scrum (software development)
- Fail fast (business), a related subject in business management
- Kanban
- Agile leadership
- Agile contracts
- Rational unified process

References

1. "What is Agile?" (https://www.agilealliance.org/agile101/). *Agile Alliance*. Retrieved 16 July 2024.

NoSQL

NoSQL (originally referring to "non-SQL" or "non-relational")[1] is an approach to database design that focuses on providing a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Instead of the typical tabular structure of a relational database, NoSQL databases house data within one data structure. Since this non-relational database design does not require a schema, it offers rapid scalability to manage large and typically unstructured data sets.[2] NoSQL systems are also sometimes called *"Not only SQL"* to emphasize that they may support SQL-like query languages or sit alongside SQL databases in polyglot-persistent architectures.[3][4]

Non-relational databases have existed since the late 1960s, but the name "NoSQL" was only coined in the early 2000s,[5] triggered by the needs of Web 2.0 companies.[6][7] NoSQL databases are increasingly used in big data and real-time web applications.[8]

Motivations for this approach include simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases),[5] finer control over availability, and limiting the object-relational impedance mismatch.[9] The data structures used by NoSQL databases (e.g. key–value pair, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.[10]

Many NoSQL stores compromise consistency (in the sense of the CAP theorem) in favor of availability, partition tolerance, and speed. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages (instead of SQL, for instance), lack of ability to perform ad hoc joins across tables, lack of standardized interfaces, and huge previous investments in existing relational databases.[11] Most NoSQL stores lack true ACID transactions, although a few databases have made them central to their designs.

Instead, most NoSQL databases offer a concept of "eventual consistency", in which database changes are propagated to all nodes "eventually" (typically within milliseconds), so queries for data might not return updated data immediately or might result in reading data that is not accurate, a problem known as stale read.[12] Additionally, some NoSQL systems may exhibit lost writes and other forms of data loss.[13] Some NoSQL systems provide concepts such as write-ahead logging to avoid data loss.[14] For distributed transaction processing across multiple databases, data consistency is an even bigger challenge that is difficult for both NoSQL and relational databases. Relational databases "do not allow referential integrity constraints to span databases".[15] Few systems maintain both ACID transactions and X/Open XA standards for distributed transaction processing.[16] Interactive relational databases share

conformational relay analysis techniques as a common feature.[17] Limitations within the interface environment are overcome using semantic virtualization protocols, such that NoSQL services are accessible to most operating systems.[18]

History

The term *NoSQL* was used by Carlo Strozzi in 1998 to name his lightweight Strozzi NoSQL open-source relational database that did not expose the standard Structured Query Language (SQL) interface, but was still relational.[19] His NoSQL RDBMS is distinct from the around-2009 general concept of NoSQL databases. Strozzi suggests that, because the current NoSQL movement "departs from the relational model altogether, it should therefore have been called more appropriately 'NoREL'",[20] referring to "not relational".

Johan Oskarsson, then a developer at Last.fm, reintroduced the term *NoSQL* in early 2009 when he organized an event to discuss "open-source distributed, non-relational databases".[21] The name attempted to label the emergence of an increasing number of non-relational, distributed data stores, including open source clones of Google's Bigtable/MapReduce and Amazon's DynamoDB.

Types and examples

There are various ways to classify NoSQL databases, with different categories and subcategories, some of which overlap. What follows is a non-exhaustive classification by data model, with examples:[22]

Type	Notable examples of this type
Key–value cache	Apache Ignite, Couchbase, Coherence, eXtreme Scale, Hazelcast, Infinispan, Memcached, Redis, Velocity
Key–value store	Azure Cosmos DB, ArangoDB, Amazon DynamoDB, Aerospike, Couchbase, ScyllaDB
Key–value store (eventually consistent)	Azure Cosmos DB, Oracle NoSQL Database, Riak, Voldemort
Key–value store (ordered)	FoundationDB, InfinityDB, LMDB, MemcacheDB
Tuple store	Apache River, GigaSpaces, Tarantool, TIBCO ActiveSpaces, OpenLink Virtuoso
Triplestore	AllegroGraph, MarkLogic, Ontotext-OWLIM, Oracle NoSQL database, Profium Sense, Virtuoso Universal Server
Object database	Objectivity/DB, Perst, ZODB, db4o, GemStone/S, InterSystems Caché, JADE, ObjectDatabase++, ObjectDB, ObjectStore, ODABA, Realm, OpenLink Virtuoso, Versant Object Database
Document store	Azure Cosmos DB, ArangoDB, BaseX, Clusterpoint, Couchbase, CouchDB, DocumentDB, eXist-db, Google Cloud Firestore, IBM Domino, MarkLogic, MongoDB, RavenDB, Qizx, RethinkDB, Elasticsearch, OrientDB
Wide-column store	Azure Cosmos DB, Amazon DynamoDB, Bigtable, Cassandra, Google Cloud Datastore, HBase, Hypertable, ScyllaDB
Native multi-model database	ArangoDB, Azure Cosmos DB, OrientDB, MarkLogic, Apache Ignite,[23][24] Couchbase, FoundationDB, Oracle Database
Graph database	Azure Cosmos DB, AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso
Multivalue database	D3 Pick database, Extensible Storage Engine (ESE/NT), InfinityDB, InterSystems Caché, jBASE Pick database, mvBase Rocket Software, mvEnterprise Rocket Software, Northgate Information Solutions Reality (the original Pick/MV Database), OpenQM, Revelation Software's OpenInsight (Windows) and Advanced Revelation (DOS), UniData Rocket U2, UniVerse Rocket U2

Key–value store

Key–value (KV) stores use the associative array (also called a map or dictionary) as their fundamental data model. In this model, data is represented as a collection of key–value pairs, such that each possible key appears at most once in the collection.[25][26]

The key–value model is one of the simplest non-trivial data models, and richer data models are often implemented as an extension of it. The key–value model can be extended to a discretely ordered model that maintains keys in lexicographic order. This extension is computationally powerful, in that it can efficiently retrieve selective key *ranges*.[27]

Key–value stores can use consistency models ranging from eventual consistency to serializability. Some databases support ordering of keys. There are various hardware implementations, and some users store data in memory (RAM), while others on solid-state drives (SSD) or rotating disks (aka hard disk drive (HDD)).

Document store

The central concept of a document store is that of a "document". While the details of this definition differ among document-oriented databases, they all assume that documents encapsulate and encode data (or information) in some standard formats or encodings. Encodings in use include XML, YAML, and JSON and binary forms like BSON. Documents are addressed in the database via a unique *key* that represents that document. Another defining characteristic of a document-oriented database is an API or query language to retrieve documents based on their contents.

Different implementations offer different ways of organizing and/or grouping documents:

- Collections
- Tags
- Non-visible metadata
- Directory hierarchies

Compared to relational databases, collections could be considered analogous to tables and documents analogous to records. But they are different – every record in a table has the same sequence of fields, while documents in a collection may have fields that are completely different.

Graph

Graph databases are designed for data whose relations are well represented as a graph consisting of elements connected by a finite number of relations. Examples of data include social relations, public transport links, road maps, network topologies, etc.

Graph databases and their query language

Name	Language(s)	Notes
AllegroGraph	SPARQL	RDF triple store
Amazon Neptune	Gremlin, SPARQL	Graph database
ArangoDB	AQL, JavaScript, GraphQL	Multi-model DBMS Document, Graph database and Key-value store
Azure Cosmos DB	Gremlin	Graph database
DEX/Sparksee	C++, Java, C#, Python	Graph database
FlockDB	Scala	Graph database
IBM Db2	SPARQL	RDF triple store added in DB2 10
InfiniteGraph	Java	Graph database
JanusGraph	Java	Graph database
MarkLogic	Java, JavaScript, SPARQL, XQuery	Multi-model document database and RDF triple store
Neo4j	Cypher	Graph database
OpenLink Virtuoso	C++, C#, Java, SPARQL	Middleware and database engine hybrid
Oracle	SPARQL 1.1	RDF triple store added in 11g
OrientDB	Java, SQL	Multi-model document and graph database
OWLIM	Java, SPARQL 1.1	RDF triple store
Profium Sense	Java, SPARQL	RDF triple store
RedisGraph	Cypher	Graph database
Sqrrl Enterprise	Java	Graph database
TerminusDB	JavaScript, Python, datalog	Open source RDF triple-store and document store[28]

Performance

The performance of NoSQL databases is usually evaluated using the metric of throughput, which is measured as operations/second. Performance evaluation must pay attention to the right benchmarks such as production configurations, parameters of the databases, anticipated data volume, and concurrent user workloads.

Ben Scofield rated different categories of NoSQL databases as follows:[29]

Data model	Performance	Scalability	Flexibility	Complexity	Data Integrity	Functionality
Key–value store	high	high	high	none	low	variable (none)
Column-oriented store	high	high	moderate	low	low	minimal
Document-oriented store	high	variable (high)	high	low	low	variable (low)
Graph database	variable	variable	high	high	low-med	graph theory
Relational database	variable	variable	low	moderate	high	relational algebra

Performance and scalability comparisons are most commonly done using the YCSB benchmark.

Handling relational data

Since most NoSQL databases lack ability for joins in queries, the database schema generally needs to be designed differently. There are three main techniques for handling relational data in a NoSQL database. (See table Join and ACID Support for NoSQL databases that support joins.)

Multiple queries

Instead of retrieving all the data with one query, it is common to do several queries to get the desired data. NoSQL queries are often faster than traditional SQL queries so the cost of additional queries may be acceptable. If an excessive number of queries would be necessary, one of the other two approaches is more appropriate.

Caching, replication and non-normalized data

Instead of only storing foreign keys, it is common to store actual foreign values along with the model's data. For example, each blog comment might include the username in addition to a user id, thus providing easy access to the username without requiring another lookup. When a username changes however, this will now need to be changed in many places in the database. Thus this approach works better when reads are much more common than writes.[30]

Nesting data

With document databases like MongoDB it is common to put more data in a smaller number of collections. For example, in a blogging application, one might choose to store comments within the blog post document so that with a single retrieval one gets all the comments. Thus in this approach a single document contains all the data you need for a specific task.

ACID and join support

A database is marked as supporting ACID properties (Atomicity, Consistency, Isolation, Durability) or join operations if the documentation for the database makes that claim. However, this doesn't necessarily mean that the capability is fully supported in a manner similar to most SQL databases.

Database	ACID	Joins
Aerospike	Yes	No
Apache Ignite	Yes	Yes
ArangoDB	Yes	Yes
Amazon DynamoDB	Yes	No
Couchbase	Yes	Yes
CouchDB	Yes	Yes
IBM Db2	Yes	Yes
InfinityDB	Yes	No
LMDB	Yes	No
MarkLogic	Yes	Yes[nb 1]
MongoDB	Yes	Yes[nb 2]
OrientDB	Yes	Yes[nb 3]

1. Joins do not necessarily apply to document databases, but MarkLogic can do joins using semantics.[31]
2. MongoDB did not support joining from a sharded collection until version 5.1.[32]
3. OrientDB can resolve 1:1 joins using links by storing direct links to foreign records.[33]

See also

- CAP theorem
- Comparison of object database management systems
- Comparison of structured storage software
- C++
- Database scalability
- Distributed cache
- Faceted search
- MultiValue database
- Multi-model database
- Schema-agnostic databases
- Triplestore
- Vector database

References

1. http://nosql-database.org/ "NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points : being non-relational, distributed, open-source and