Infosys®
Navigate your next

WHITEPAPER

AGILE EVOLUTION — 20 YEARS LATER

Agile has come a long way since its infancy as a software development method in the 1980s. Here, we chart its emergence and maturation into a tool and philosophy that allows large corporations to move fast, remain nimble, and deliver superior business outcomes.

Infosys® | Knowledge Institute

From humble beginnings, the Agile methodology has become the 21st-century's panacea for all things digital, innovative, and fast moving. But there are fears that its potency is being lost along the way as its usage spreads. Twenty years later, it's time to reassess the principles and practices of Agile for today's world.

Agile began as a lean software development method that delivered higher-quality code in less time. However, in the past decade, its concepts have increasingly been adopted by nontechnical business teams, from creative agencies to innovation labs, and from marketing groups to human resources departments.

The simplicity and flexibility of Agile is what made it so compelling. However, this is also a potential weakness. Among those who use it today, there is ambiguity about what Agile is and should be. While many now apply the term to their own work, few are fluent in the fundamentals.

This leads to the common complaint that many projects are Agile in name only — either running these practices in a waterfall method or focusing on rituals rather than actual outcomes.

Meanwhile, COVID-19 has led organizations to fully embrace cloud architecture and remote working. This phenomenon is simultaneously increasing the need for flexible and rapid development, while challenging some of the core precepts of the Agile manifesto regarding how teams collaborate.

This paper, therefore, acts as both a primer for those unfamiliar with Agile as well as a refresher and reappraisal of its history and evolution for those who feel it's losing its way.

Practices versus principles

Agile is based on a simple and direct set of values and principles that make it at once compelling, broad ranging, and ambiguous. In order to turn the philosophy into reality, dozens of Agile practices and toolkits have sprung up over the years.

Over time, however, the use and ritualization of these practices have caused some Agile projects to overly rely on method over purpose — one of the key problems that Agile was supposed to fix.

Today, business leaders often complain that Agile practitioners are overzealous in their application of these methods. This often alienates other teams within large enterprises rather than acting as a force for change.

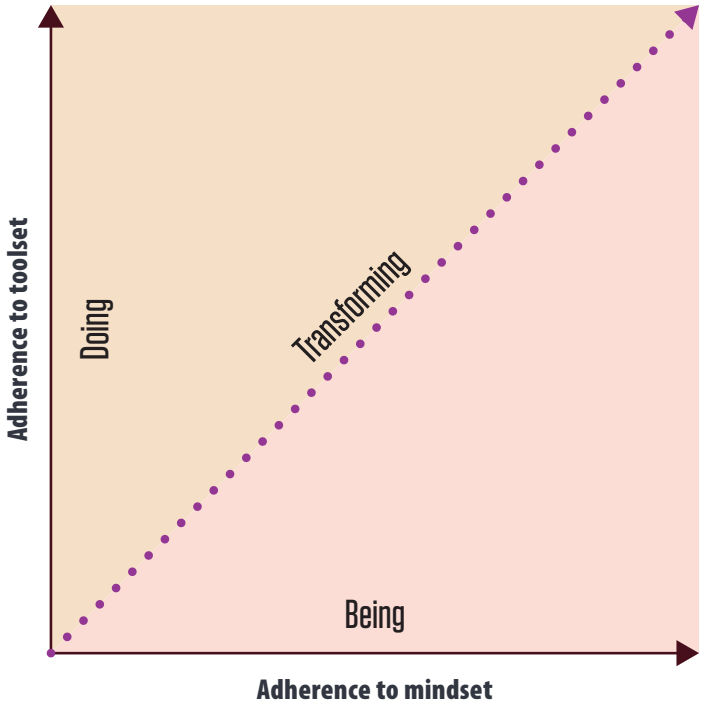
On the other hand, projects and teams that adopt the spirit of Agile thinking — focusing on agility, responsiveness, and flexibility — often struggle to scale across an organization. These types of teams tend to rely on specific members or leaders that imbue the right culture but often do not embrace enough documented methodology for Agile to be replicable.

It's helpful to think of these two challenges as separate but interrelated capabilities that organizations need to hone — and balance — in order to deliver truly transformative outcomes across the entire enterprise.

Our Agile framework pits "adherence to toolset" against "adherence to mindset" (Figure 1). Those who are strong on the methods and practices that support Agile projects are doing Agile, whereas those who are strong on the mindset and cultural elements are being Agile. Those who are able to balance both are truly transforming their enterprises through scaling Agile practices.

We use "mindset" to describe the cultural and organizational traits and values that are most aligned with the principles and values of the Agile Manifesto. We use the term "toolset" to describe those teams that adhere closely to the recognized tools and practices associated with Agile — the most prominent of which we list later in this paper.

Figure 1. A framework to understand the balance between toolset and mindset in Agile



Source: Infosys

The roots of Agile

Agile practices and principles developed from three different roots: manufacturing, software development, and project team management.

Manufacturing

Most people see Agile as an alternative to the waterfall software-development model. However, many of its ideas originated with manufacturing principles. After World War II, W. Edwards Denning used principles from Bell Labs studies conducted by his mentor to improve products and processes in Japanese manufacturing. Toyota hired Denning to train their managers in these techniques, which led to the development of the Toyota Production System. This is the main source of today's Lean methodologies.

Software development

Kent Beck developed Extreme Programming in 1996, while leading a rewrite of Chrysler's payroll application. Extreme Programming is responsible for introducing the ideas of small early code releases, automated testing, pair programming, refactoring, and team ownership of code as standards for all projects. Though it predates Agile, it has since incorporated Agile values and principles.

Team management

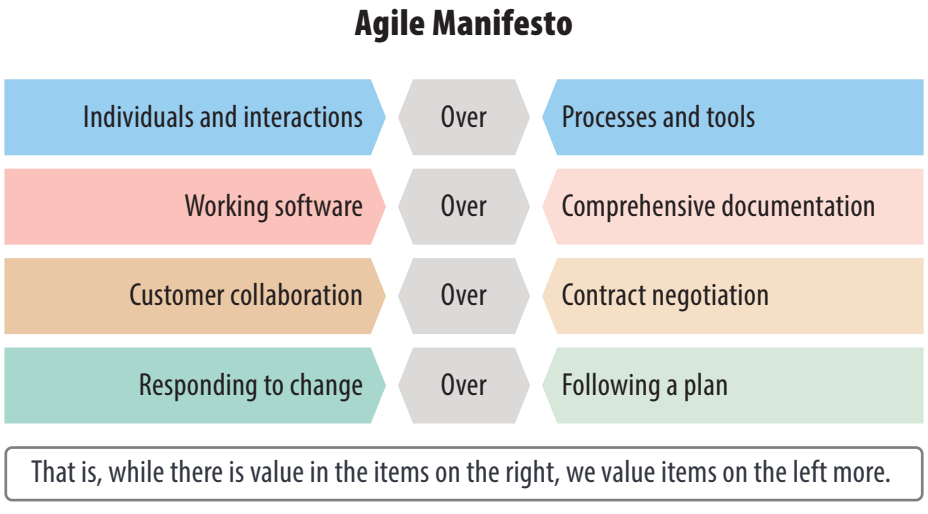
Scrum is the most popular Agile variant. It originates with a 1986 Harvard Business Review article titled "The New New Product Development Game." Authors Hirotaka Takeuchi and Ikujiro Nonaka describe a new approach to product development that eschews the standard notion of completing project phases that are then passed to new teams (a relay race). Instead, it favors a method in which a single team produces a product from start to finish (a rugby approach). By organizing teams for entire product life cycles instead of specific phases, technical expertise is not lost. Additionally, early choices made by product teams have to be managed by the same team, instead of becoming others' problems.

Mindset — the foundations of Agile

In 2001, 17 programmers met at a ski resort in the Wasatch Mountains of Utah to share best practices in software development. Each came from a different programming background, but all were keen on developing a new lightweight approach to their craft. The meeting resulted in the Agile Manifesto, a statement of four values (Figure 2), backed by 12 principles.

This was a seminal moment for the software development community. The manifesto was revolutionary, and its hidden message marked a profound shift from the received business

Figure 2. Values in Agile Manifesto



Source: Agile Manifesto

wisdom of the 20th century. Agile was not about a one-size-fits-all, mass-produced product built on assembly lines. It was about freeing engineers to be creators who could explore their craft and respond to the needs of end users and customers flexibly and rapidly.

> Agile brought in the concept of engineer-as-creator, responding flexibly and rapidly to customer needs

The 12 principles of Agile

Though an inspirational starting point, the Agile values are still vague and not prescriptive of any particular practice or method. The 12 Agile principles help bring clarity and focus on the type of organizational culture needed to make Agile succeed.

We find it useful to organize them into four categories: priorities, culture, team organization, and work processes.

Priority principles focus on delivering quality and valuable software. Cultural principles ensure business alignment, a focus on continuous improvement, and the need for team members to meet in person. Team organization principles and work process principles focus on effectiveness, transparency, and accountability.

Priorities

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- We deliver working software frequently, from every couple of weeks to couple of months, with a preference for the shorter timescale.
- Working software is the primary measure of progress.

Culture

- We welcome new requirements, even late in development. Agile processes harness change for competitive advantage.
- Business staff and developers must work together daily throughout the project.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Team organization

- Projects are built around motivated individuals. Organizations should give them the environment and support they need, and trust them to get the job done.
- The best architectures, requirements, and designs emerge from self-organizing teams.

Work processes

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a consistent pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity — the art of maximizing the amount of unnecessary work not done — is essential.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Toolset — Agile's practices and techniques

As teams have applied Agile principles, specific practices have emerged as synonymous with this methodology. The most dominant are Scrum and Kanban, which can be applied to a wide range of technical and nontechnical projects. But as business demands have changed and understanding of Agile has deepened, other practices have emerged to take center stage. Notably, Scaled Agile Framework (SAFe) was designed to enable large software projects to be agile, and DevOps, which has become increasingly popular as a way to increase automation.

Yet, the popularity of these tools can overshadow others that might be better suited to today's enterprises, even if those tools didn't initially capture the public's imagination. The following list provides a brief introduction to many of these practices, with notes about usage derived from an internal survey of 25 senior Agile practitioners and their experience with more than 90 clients.

We break these down into three major project groups: planning, execution, and tracking.

Project planning

Project planning is a major feature of all software development. Agile, however, seeks to re-engineer that process in significant ways. With this methodology, requirements don't have to be exhaustively defined in the earliest stages. This important benefit lowers lead times and enables a quick release cycle.

CRC cards

Class responsibility collaborator (CRC) cards are among the earliest forms of software development planning. Introduced in 1988, CRC cards physically show the relationships among the different classes used in object-oriented programming. A team typically creates these cards as a group exercise to design how pieces of software interact and fit together.

Planning poker

Planning poker is one of the earliest elements of project planning. It was first described in 2002 as a way to estimate the length and difficulty of tasks without members influencing each other. In planning poker, team members receive cards with different point values (typically in Fibonacci sequence). After the product owner describes a user story, each team member silently chooses a card with the point value he or she thinks the user story should be assigned and places the card face down. Once all team members have voted, the cards are revealed, and the team members with the highest and lowest estimates explain their reasoning. The team then

tries to come to a consensus point value by playing additional rounds.

Magic estimation

Because some planning poker sessions would fill an entire day, magic estimation was developed in 2010 as a method to shorten the time spent assigning points to user stories. This approach is most useful for estimating entire backlogs and getting a rough idea of the size and complexity of a project.

With magic estimation, point values are placed on the floor or wall, and every user story or backlog item is written on a card. The product owner then briefly describes items in the backlog, and the team settles on a user story or backlog item that will serve as the baseline task. This task is assigned a total that is close to the middle of the range of points on the wall.

Each team member then receives a unique set of backlog item cards. The team places those cards under point values as compared to the relative complexity of the baseline task. If a team member is uncertain, the card is placed to the side for further explanation by the product owner. In further rounds, cards can be moved from one point value to another by team members. But cards that are not moved or not moved very far have their point values written on them and are given to the product owner. The entire backlog can be estimated in an hour or less.

User stories

User stories divide features in a software project into functional increments. Their use originated with the practice of Extreme Programming in 1998, but was more formally described a few years later. User stories have become the standard for creating feasible units of work on Agile teams, and our survey respondents indicate that their use is almost universal today.

INVEST checklist

The INVEST checklist was promoted in 2003 as a way to codify useful user stories. According to the document, a good user story should be negotiable, valuable, estimable, small, testable, and independent of all other stories. If it fails to meet these criteria, the team should consider rewording or rewriting the story.

Story mapping

In 2008, the concept of story mapping was introduced to help teams consider whether a set of features will be useful for each type of user. This generally takes the form of plotting user stories against the independent dimensions of complexity and priority. The aim is to avoid a situation where valuable business features are deferred because they depend on other, lower priority features.

Domain-driven design

The core of domain driven design, created in 2003, is to have code be more "human readable" by naming functions and classes in the code using the nomenclature of the specific business in question.

Definition of done

Used in Scrum, the term "definition of done" was coined in 2005. This idea helps teams create consistent and well-understood acceptance criteria. It aids in limiting the cost of rework once a feature has been accepted as done and limits the risk of misunderstandings between the development team and the product owner. The definition of done is a shared understanding; obsessing over a list of criteria can be counterproductive.

Definition of ready

In 2008, the definition of done expanded into the "definition of ready," which encompassed the beginning of user stories. This helps determine whether a feature is ready to be developed. The aim is to avoid beginning work on features that do not have well-defined completion criteria. This helps avoid rework.

Project chartering

Project chartering was developed in 2006 to define the scope and objectives of a project. This is done so that projects do not fall into indefinite release cycles. This chartering approach fell out of favor as software developers began to choose more product-based approaches instead of project-based ones.

Project execution

While the elements of Agile that concern project planning and tracking are useful, the major goal was to change attitudes regarding project execution. Instead of insisting that all features be defined before the first line of code is written, Agile adopts the mentality that changes are inevitable and should be welcomed. Instead of treating the customer as an antagonist to be wrangled, teams should work daily with the business to ensure program features carry value. To execute these ideas, teams needed better defined examples of what this looked like.

Lean, Scrum, and Extreme Programming all predate Agile but have elements that are now considered components of Agile.

Continuous integration

One of the best-known features of Agile software development is continuous integration. The term had been in use for many years before Martin Fowler, an American software developer and public speaker, gave a more complete description in 2000. Earlier, the best practice was "scheduled" integration because of a lack of thorough testing in continuous integration. When tools were developed for automatic testing,

continuous integration became more popular. Integrating code automatically is one of the key drivers Agile uses to produce working code in a short time.

Continuous delivery

Automation of code integration catalyzed the idea of further software automation, resulting in the innovation of continuous delivery. This approach automates the next step of software development by automatically pushing code to the quality assurance or staging environment. Continuous delivery requires manual approval after regression tests are completed. These tests confirm that new functionality does not break existing systems and is ready for production.

Continuous deployment

In 2006, software authors Jez Humble, Chris Read, and Dan North published the first article describing continuous deployment. This further automates software development by automating — without human intervention — the deployment of code to production. The authors noted that this is only possible with fully automated unit, integration, and regression tests to minimize the chance of introducing error into production environments.

DevOps

DevOps is the combination of microservices, continuous integration and delivery, monitoring and logging, and infrastructure as code. This was developed in 2009 to solve friction between development and operations teams trying to implement Agile practices. Since then, DevOps has become one of the most popular flavors of Agile.

Pair programming

Pair programming started in 1996 as an element of Extreme Programming. Coding is done in pairs, with one developer typing the code while the other helps direct how it should be written. This practice has always had a mixed reception. Some developers like the back-and-forth brainstorming. However, working with people has been antithetical to the persona of many programmers and is often rejected as a practice.

Ping-pong programming

First appearing in 2003, ping-pong programming was an evolution of pair programming. It combines pair programming and test-driven development. With this approach, developer A writes a test that fails, and developer B writes the code to make the test pass. Then developer B writes a test that fails, and developer A writes the code to make the test pass. This method is used to keep both programmers engaged in the problem so the risk of one developer zoning out is minimized.

Acceptance testing

Acceptance testing was developed in 2002 and uses automated tests to

determine whether a feature is ready for release. These tests are created with the customer and show whether the code resulting from a user story is producing the correct output.

Reflection workshops

Reflection workshops were developed in 2001 and became popular soon after. These were designed to help teams determine when to end a current release iteration or begin a new one. The purpose is to reflect, as a group, on what portions of the iteration or sprint went well and what portions did not succeed. Using the information from the workshops or retrospectives, the teams can make adjustments to their process.

Lean software development

Lean was first described in the book Lean Software Development: An Agile Toolkit in 2002. The principles are to eliminate waste, amplify learning, decide what code to build as late as possible, deliver as early as possible, empower the team, build integrity into the product, and optimize the whole.

FitNesse

FitNesse is a tool that was developed in 2003 to facilitate automated integration testing and to support acceptance testing, rather than just unit testing. FitNesse allows software end users to create tests that are then turned into code and can be run automatically to test new features.

Acceptance test driven development

Acceptance test driven development (ATDD) is the practice of having all team members, including customers and testers, collaboratively write tests before any code is created. It was initially dismissed as impractical by Kent Beck, Extreme Programming's founder. However, it became popular as a way to allow nontechnical people to contribute to test design.

Backlog grooming

Backlog grooming, also known as backlog refinement, was first mentioned in 2005 but not formally described until 2008. This involves the product owner and the rest of the team reviewing items in the backlog for importance and priority. The goal is to remove irrelevant user stories, add new user stories based on net information, estimate or re-estimate user stories, and refine any current user stories that are too large to fit into an iteration.

Behavior-driven development

Behavior driven development (BDD), also known as specification by example and first described by Dan North in 2006, is an adaptation of test-driven development (TDD). North's initial insight was to think about writing test functions as testing a behavior in one sentence. This limits the scope of a test because there is only so much that can be described. The method also codifies where to start in a process by asking the question "What is the next most important behavior that the system does not currently do?" Though not expressed in his initial article, the Agile Alliance explained that BDD asks team members to apply the "Five Why's" to each user story and ensure it is related to business outcomes.

SAFe

SAFe originated in 2007 as a solution to the problem of large software projects. It is a set of practices and organizational structures that guides enterprises in scaling lean and Agile. SAFe is a prescriptive framework that creates agile teams of Agile teams. SAFe generally fits well into established hierarchical structures. Practitioners sometimes criticize the framework as "not Agile" because of its prescriptive nature. But it has gained in popularity, perhaps because of those reasons rather than in spite of them. According to the State of Agile report released in 2020, SAFe is about twice as popular as the next most popular scaling framework, Scrum of Scrums.

Agile testing

Created in 2017, Agile testing employs collaborative practices that occur from inception to delivery, supporting frequent delivery of quality. Agile testing focuses on defect prevention rather than defect detection. It also includes asking questions to test ideas; automating tests; exploratory testing; and testing for reliability, security, and performance.

Project tracking

Tracking the progress of a sprint or project is essential in order to ensure that working software is delivered on time. This ties into the first principle of Agile — that the highest priority is to satisfy the customer with early and continuous delivery of valuable software.

Velocity

The first defined Agile methods for project tracking were velocity, burndown, and burnup rates. All these methods became popular in 2002. Velocity is a measurement used in Scrum that shows the number of story points a team has accomplished in a given sprint. Velocity is a useful measure of the same team across sprints. But it cannot be used to compare teams since the number of story points assigned to a given task can vary greatly. Additionally, even within the same team, the velocity measure can be gamed by allowing the team to inflate the story points required for tasks in future sprints.

Burndown

The burndown rate, first described in 2000, is a way to see whether a team is on target for the current sprint or

project. It shows the quantity of work remaining versus the amount of time remaining, along with an expected position for any future point. The drawback to the burndown chart is that changes in the scope of the sprint or project are not captured and teams that are working as expected may look like they are falling behind.

Burnup

Burnup charts fix the issue of uncaptured scope changes by showing the ideal burnup rate relative to the team's performance and also the total scope.

Five-column task board

In 2003, the five-column task board was formally described by Mike Cohn, a programmer and one of the founders of the Scrum Alliance. The columns include story, to do, in process, to verify, and done.

> Tracking progress ensures software is delivered on time and at high quality, tying in with the first principle of Agile

Three-column task board

That previous task board was supplanted in 2007 by the three-column task board, which removed the "story" and "to verify" columns. Task boards feature very heavily in most Agile implementations since they focus the team on progress and obstacles during daily meetings. It is important to have a physical task board in a physical space as a constant reminder of team progress. Virtual boards can be too easily forgotten. The Kanban board was also introduced in 2007, and closely resembles the three-column board. The key difference of the Kanban board is that the team limits the number of items that are allowed to be in progress. This keeps teams on track to release working code at the end of a sprint by forcing them to complete a user story before starting a new one. Respondents to our survey indicated that they universally use Kanban boards.

Daily meetings

The idea of daily meetings was described in 1994 by computer science researcher James "Cope" Coplien in his observations of the Borland Quattro Pro team, wherein he described the effect of frequent meetings on the teams process, quality, and productivity. In 1997, Ken Schwaber, a software developer and industry consultant, described the "daily scrum." And by 1998, the daily "standup" meeting was a core practice of Extreme Programming. In 2004, daily meetings were generalized as a core

Agile practice. This practice has not changed, as daily meetings are a large part of Agile teams. Sometimes a lack of effectiveness in carrying out Agile lies with who is in the daily meetings. Less than half of our respondents indicated they work with the relevant businesspeople on a daily basis.

Kanban

Kanban emerged in 2007 as a way to manage projects by balancing capacity and removing workflow bottlenecks. Kanban's four principles — visualization of workflow, limiting the work in progress, improving workflow, and continuous improvement by reducing friction — show its practices are meant to be an iterative change process, not just in production but also in culture. This is important since cultural change is often one of the biggest challenges to Agile adoption.

Agile offers a vast number of tools and practices. However, to be successful, they must not be shoehorned into existing processes. Rather, Agile must work at scale across the organization, with employees free to use technologies and techniques that fit the specific context. To work holistically, all people, processes, and technology must work with the Agile Manifesto, which requires significant cultural change. Doing Agile well is, in fact, more about mindset than a rigorous attunement to its techniques. The need for a continuously evolving and learning culture is critical.

Agile is not just about software. In a post-COVID-19 world, with all the turbulence buffeting business and operating models, Agile must extend beyond technology and make itself felt in all business functions and across all industry segments. As we have underscored in other Agile papers in this portfolio, the zenith for any agile organization is one where the business and operating model is agile in both name and substance. Such an organization can react quickly to environmental shocks, and its leadership is able to rally the whole team around new ways of delivering the bottom line, at scale.

Author

Isaac LaBauve

Senior Consultant – Infosys Knowledge Institute
isaac.labauve@infosys.com

About Infosys Knowledge Institute

The Infosys Knowledge Institute helps industry leaders develop a deeper understanding of business and technology trends through compelling thought leadership. Our researchers and subject matter experts provide a fact base that aids decision making on critical business and technology issues.
To view our research, visit Infosys Knowledge Institute at infosys.com/IKI

For more information, contact askus@infosys.com

Infosys®
Navigate your next