Chapter 2

Agile History and the Agile Manifesto

Agile Early History

THE EARLY HISTORY OF AGILE was influenced by a number of different trends, including:

- The evolution of new concepts in manufacturing, such as just-in-time and lean manufacturing processes
- New approaches to quality management, including the total quality management movement developed by Dr. W. Edwards Deming

Those fundamental trends will be discussed in Chapter 11, “Understanding Agile on a Deeper Level”; however, the strongest factor that has driven the agile movement is the unique risks and challenges associated with large, complex software development projects. I can remember a time when a computer that had 16 KB (kilobytes) of memory and a 5-MB (megabyte) disk was a lot. Even before that, I can remember the old days of developing programs on punched cards and paper tape, or even toggling in some binary, assembly language code into the switch register of an early Digital Equipment Corporation (DEC) mini-computer.

In those early days, software development was very limited and primitive; however, computer technology has changed rapidly and significantly since that time. Today, my iPhone 5 Smartphone has thousands of times more power than some of those early computers I worked with in the 1980s, and that has enabled the development of much more powerful and complex software to utilize that additional processing power. As software development has become much more widespread and has grown into much larger and more complex applications, it has created a number of new challenges for managing large, complex software development projects and a number of different software development methodologies have evolved over the years to meet these new challenges.

Note

While software development has been a key driving force behind agile and is still heavily associated with agile today, it should be clearly understood that agile is not limited to software development and is rapidly spreading to other areas. Any area that has a high level of uncertainty that cannot be easily resolved and/or high levels of complexity is probably a good candidate for an agile approach.

Dr. Winston Royce and the Waterfall model (1970)

In 1970, Dr. Winston Royce published a famous paper on "Managing the Development of Large Software Systems."[1] That paper is widely associated with the waterfall approach as we know it today and outlined an approach for breaking up a large complex software project into sequential phases to manage the effort. The model Dr. Royce proposed in his original 1970 paper is shown in Figure 2.1:

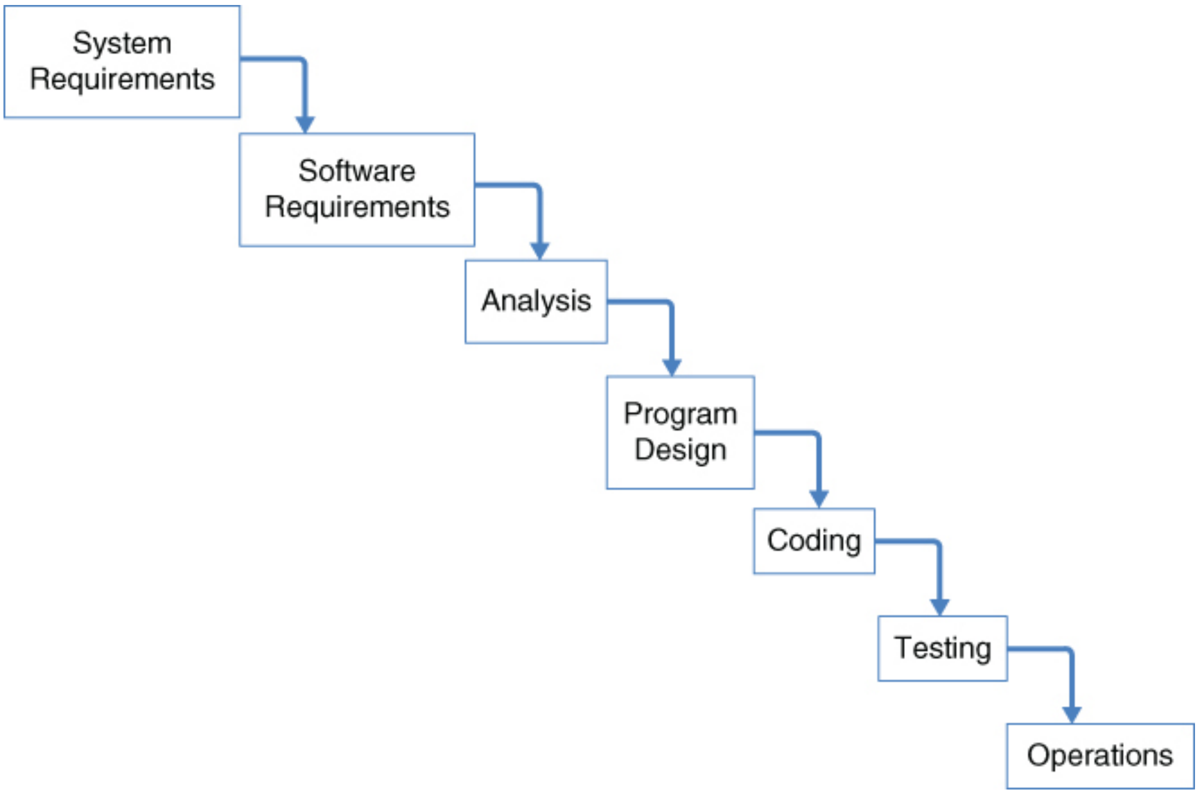

Figure 2.1 Original Waterfall model

However, even Dr. Royce recognized the weaknesses in this approach at the time:

> I believe in this concept, but the implementation described above is risky and invites failure. The problem is illustrated in Figure [2.1]. The testing phase which occurs at the end of the development cycle is the first event for which timing, storage, input/output transfers, etc., are experienced as distinguished from analyzed. These phenomena are not precisely analyzable. They are not the solutions to the standard partial differential equations of mathematical physics for instance. Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required. A simple octal patch or redo of some isolated code will not fix these kinds of difficulties. The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated. Either the requirements must be modified, or a substantial change in the design is required. In effect the development process has returned to the origin and one can expect up to a l00-percent overrun in schedule and/or costs.[2]

One of the fundamental problems with the waterfall approach that Dr. Royce recognized was that the entire process was sequential and an error or omission made in one of the very early phases of the project may not have been discovered until the very end of the project which might require huge amounts of rework to the work that had already been done in the early phases of the project. For that reason, more iterative approaches began to evolve that featured more incremental and evolutionary development approaches.

Early iterative and incremental development methods (early 1970s)

Walker Royce, the son of Dr. Winston Royce and a contributor to the development of iterative and incremental development (IID) methods in the 1990s, made the following comment regarding his father's thinking:

> He was always a proponent of iterative, incremental, evolutionary development. His paper described the waterfall as the simplest description, but that it would not work for all but the most straightforward projects. The rest of his paper describes [iterative practices] within the context of the 60s/70s government contracting models (a serious set of constraints).[3]

Many of the earliest efforts to break up large projects into incremental and iterative development efforts focused primarily on breaking up the development phase into iterations.

> The next earliest reference in 1970 comes from Harlan Mills, a 1970s software engineering thought leader who worked at the IBM FSD. In his well-known "Top-Down Programming in Large Systems" Mills promoted iterative development. In addition to his advice to begin developing from top-level control structures downward, perhaps less appreciated was the related life-cycle advice Mills gave for building the system via iterated expansions….
>
> Clearly, Mills suggested iterative refinement for the development phase, but he did not mention avoiding a large up-front specification step, did not specify iteration length, and did not emphasize feedback and adaptation-driven development from each iteration. He did, however, raise these points later in the decade.[4]

Further evolution of iterative and incremental development (mid- to late 1970s)

Support for iterative and incremental development continued to grow in the 1970s. In 1976, Mills wrote: "Software development should be done incrementally, in stages with continuous user participation and replanning, and with design-to-cost programming within each stage."[5]

Larman and Basili write that in the context of a three-year inventory control system, Mills went on to challenge the idea and value of up-front requirements or design specifications. Mills said:

> There are dangers, too, particularly in the conduct of these [waterfall] stages in sequence, and not in iteration—i.e., that development is done in an open loop, rather than a closed loop with user feedback between iterations. The danger in the sequence [waterfall approach] is that the project moves from being grand to being grandiose, and exceeds our human intellectual capabilities for management and control.[6]

Early agile development methods (1980s and 1990s)

During the 1980s and early 1990s, there was a proliferation of approaches designed to further improve the methodologies for large, complex software development projects:

Agile methods were direct spinoffs of software methods from the 1980s, namely Joint Application Design (1986), Rapid Systems Development (1987), and Rapid Application Development (1991). However, they were rooted in earlier paradigms, such as Total Quality Management (1984), New Product Development Game (1986), Agile Leadership (1989), Agile Manufacturing (1994), and Agile Organizations (1996)….”[7]

Agile methods formally began in the 1990s with Crystal (1991), Scrum (1993), Dynamic Systems Development (1994), Synch-n-Stabilize (1995), Feature Driven Development (1996), Judo Strategy (1997), and Internet Time (1998). Other agile methods included New Development Rhythm (1989), Adaptive Software Development (1999), Open Source Software Development (1999), Lean Development (2003), and Agile Unified Process (2005). However, the popularity of Extreme Programming (1999) was the singular event leading to the unprecedented success of agile methods by the early 2000s.”[8]

Another major influence during this same period of time was the evolution of the Rational Unified Process (RUP), which became another widely used iterative approach. (Variations later emerged from that, including the Enterprise Unified Process (EUP) in 2003 and the Agile Unified Process (AUP) in 2005).[9]

By the early 2000s, there was a broad and confusing proliferation of different methodologies. Ultimately, however, they evolved into a much more cohesive agile approach.

Agile Manifesto (2001)

Many people point to the *Manifesto for Agile Software Development*, or the Agile Manifesto, published in 2001, as the true beginning of the agile movement today. The Agile Manifesto clearly condensed all of the earlier agile methodologies into a set of clearly-defined values and principles that are still valid today. “The whole fuss started with the meeting in Feb. 2001 at Snowbird Ski Resort. 17 of us met to discuss whether there was a common, underlying basis for our work in the 1990s around what had been referred to as ‘light-weight processes.’ None of us liked the term ‘light-weight,’ feeling it was a reaction against something, instead of something to stand for.”[10]

One of the most important things to recognize about the Agile Manifesto and the values and principles behind it is that they must be interpreted in the context of whatever project they're applied to. The Agile Manifesto consists of four values and a number of principles that back up those values. The value statements are not meant to be absolute statements at all, and the Agile Manifesto clearly states that; however, in spite of that, many people have made the mistake of dogmatically treating these statements as absolutes. Think about these statements in a broad sense and interpret them in the context of a particular project. If you take that approach, most of these values and principles make sense to be applied to almost any project, not just "agile" projects.

Agile Manifesto values

The Agile Manifesto values that were published originally in 2001 are:

> 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.[11]

Individuals and Interactions over Process and Tools

The first statement indicates a preference for "individuals and interactions over process and tools." This statement is essentially a response to "command-and-control" project management practices that had been perceived as very impersonal and insensitive to people and rigidly defined processes where the "process manages you." From a project management perspective, it calls for a softer leadership approach with an emphasis on empowering people to do their jobs as well as flexible and adaptive processes, rather than a rigid, control-oriented management approach that is highly directive. Agile processes generally depend very heavily on empowered people making intelligent decisions and the power of collaborative teamwork.

The last part of this statement regarding "process and tools" might imply that there is no place in an agile project for process and tools, but that is certainly not the case:

- Agile uses well-defined processes like Scrum (see Chapter 3), but they are meant to be interpreted and implemented intelligently rather than followed rigidly and mechanically, and they are also designed to be adaptive rather than prescriptive.
- Tools can also play a supporting role, but the important thing is to keep the tools in the right context—they should be there to leverage and facilitate human interactions, not to replace them.

Working Software over Comprehensive Documentation

The second value statement indicates a preference for “working software over comprehensive documentation.” This statement was essentially a response to typical phase-gate project management processes that called for extensive documentation deliverables at the end of each phase. There was entirely too much emphasis on producing documentation, to the extent that the documentation took on a life of its own, and there was insufficient emphasis on producing working software.

Note: Although the Agile Manifesto was written around software development, there's really no reason why most of the values and principles can't be extended to other products if they are used intelligently in the context of whatever products they are applied to. For example, this value could read “working products over comprehensive documentation.”

One of the problems with documentation is that it can inhibit normal communication. The typical waterfall project was heavily based on documentation. The project team would develop an elaborately detailed requirements specification, and the software would be tested against meeting that specification. In many waterfall projects, the end-user didn't even see what was being developed until the final user acceptance testing at the end of the project. This approach presents several opportunities for problems:

- Many users have a difficult time defining detailed requirements up front in a project, especially in a very uncertain and changing environment.
- Relying too heavily on documentation can lead to significant miscommunications and misunderstandings about the intent of the requirements.

It's important to note that this statement doesn't imply that there should be no documentation at all in an agile project. The key thing to recognize is that any documentation should provide value in some way. *Documentation should never be an end in itself*.

Customer Collaboration over Contract Negotiation

The third value statement indicates a preference for "customer collaboration over contract negotiation." The typical project prior to agile has been sometimes based on an "arm's-length" contracting approach. Project managers for many years have been measured on controlling costs and schedules, and doing that has required some form of contract to deliver something based on a defined specification. Of course, that also requires some form of change control to limit changes in the requirements as the project progresses.

Agile recognizes that, particularly in an uncertain environment, a more collaborative approach can be much more effective. Instead of having an ironclad contract to deliver something based on some predefined requirements, it is better in some cases to create a general agreement based on some high-level requirements and work out the details as the project progresses. Naturally, that approach requires a spirit of trust and partnership between the project team and the end-customer that the team will ultimately deliver what is required within a reasonable time and budget.

Again, it is important to recognize that these values are all relative, and you have to fit this statement to the situation. At one extreme, I have applied an agile approach to a government-contracting environment. Naturally, in that environment we had to have a contract that called for deliverables and milestones and expected costs, but even in that environment, the government agency understood the value of collaboration over having a rigidly defined, arm's-length kind of contract, and we were able to find the right balance. At the other extreme, you might have a project where the requirements are very uncertain and a much more adaptive approach is needed to work collaboratively with the customer to define the requirements as the project progresses, without much of a contract at all.

Responding to Change over Following a Plan

The final value statement indicates a preference for "responding to change over following a plan." This statement is in response to many projects that have been oriented toward controlling costs and schedules. They made it difficult for the customer to change the requirements in order to control the scope and, hence, the cost and schedule of the project.

The problem in applying that approach to environments where the requirements for the project are uncertain and difficult to specify is that it forces the user to totally define the requirements for a project upfront without even envisioning what the final result is going to look like, and that's just not a very realistic approach in many cases.

In many situations, it is more effective to recognize that, at some level, the requirements are going to change as the project progresses and design the project approach around that kind of change. It's important to recognize that this is not an all-or-nothing decision…to have either completely undefined requirements or highly detailed requirements. There are a lot of alternatives between those two extremes, and you have to choose the right approach based on the nature of the project.

All of these statements are somewhat interrelated, and you have to consider them in concert to design an approach that is appropriate for a particular project. From a project management perspective, this calls for some skill. Instead of force-fitting a project to some kind of canned and well-defined methodology like waterfall, the project manager needs to intelligently develop an approach that is well-suited to the project—that applies to any approach, not just agile.

Agile Manifesto principles

The four value statements form the foundation of the Agile Manifesto. The principles in the Agile Manifesto expand on the values and provide more detail. Again, as with the value statements, these statements should also be considered as relative preferences and not absolutes. The principles are summarized as follows and are discussed in more detail in the following sections:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a project team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.[12]

Early and Continuous Delivery of Valuable Software

"*Our highest priority is to satisfy the customer through early and continuous delivery of valuable software*."

The first principle emphasizes "early and continuous delivery of valuable software." In many traditional plan-driven projects prior to agile, the end-user customer doesn't see anything until the final user acceptance test phase of the project, and by that time it is very difficult and expensive to make any changes that might be needed.

Emphasizing early delivery of the software accomplishes two major goals

1. It provides an opportunity for the customer to see the software early in the development cycle and provide feedback and input so that corrections can be made quickly and easily.
2. Working software is a good measure of progress. It's much more accurate and effective to measure progress in terms of incremental software functionality that has actually been completed, tested, and delivered to the user's satisfaction rather that attempting to measure the percentage of completion of a very large development project that is incomplete.

It is very difficult to accurately measure progress of a large software development project as a whole without breaking it up into pieces. That can be a very subjective judgment with some amount of guesswork. Breaking up the effort into well-defined pieces that each have clearly defined criteria for being considered "done" provides a much more factual and objective way of measuring progress.

Welcome Changing Requirements

> “*Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage*.”

The next principle emphasizes creating an environment where change is expected and welcomed rather than rigidly controlled and limited; but, of course, that doesn't mean that the project is totally uncontrolled. There are lots of ways to manage change effectively and collaboratively based on a partnership with the customer. The important thing is that the project team and the customer should have a mutual understanding upfront of how change will be managed.

Deliver Working Software Frequently

> “*Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale*.”

The next principle emphasizes using an iterative approach to break up a project into very small increments called *sprints* or *iterations*, which are typically in the range of two to four weeks. There are a couple of reasons why this makes a lot of sense:

1. All agile development processes such as Scrum are based on continuous improvement. Instead of having a rigidly defined process that never changes, the team is expected to take an empirical approach to learn what works and what doesn't work as the project progresses, and make adjustments as necessary. If the project is broken up into very short increments and learning takes place at the end of each increment, learning and continuous improvement can happen much more rapidly. A popular agile mantra is, “Fail early, fail often.” In other words, it's better in many cases to try something quickly and learn from it and make adjustments, rather than taking all the time that may be needed to try to design an approach that is going to work flawlessly the first time.
2. People work more productively given short time-boxes to get things done. If it is done correctly, the team develops a cadence and a tempo that is very efficient for producing defined increments of work quickly and efficiently, like a manufacturing assembly line.

Business People and Developers Must Work Together

"*Business people and developers must work together daily throughout the project*."

The next principle emphasizes a partnership approach between the project team and the business sponsors. This is very consistent with the Agile Manifesto value of "collaboration over contracts." To implement this principle, both the business sponsors and the project team need to feel joint responsibility for the successful completion of the project. This calls for a much higher level of engagement of the business sponsors than is commonly found in many traditional projects where the implementation of the project might be almost totally delegated to the project team.

The degree of engagement, of course, should be appropriate to the nature of the project and how the business sponsors get engaged might be different depending on the circumstances. For example, Scrum has a role called the *product owner* that provides the day-to-day business direction for the project, but the direction might not be limited to that. In a large, enterprise-level project, there might be a number of other stakeholders that need to provide input and need to be engaged somehow.

Designing an approach that gets the right people engaged at the right time is very important for making the project successful.

Build Projects around Highly Motivated Individuals

"*Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done*."

The next principle emphasizes the importance of properly motivated individuals on a project. Too often in the past, some project managers have used high-pressure, command-and-control tactics to pressure project teams into delivering results faster. Many of us have been involved in "death march" projects in our careers where people are given an absolute deadline for getting something done, and have to work nights and weekends if necessary to get it done. When you're in an environment that requires high levels of creativity and innovation, that approach just doesn't work very well.

The philosophy of agile is based on a high level of empowerment and individual initiative by the people on the project. Instead of being told specifically what to do and being pressured into doing it to meet deadlines, agile teams are given general direction and are expected to figure out how to get it done most effectively and efficiently themselves. Making that kind of approach work requires a people-oriented leadership style. However, it doesn't mean that there is no need for leadership whatsoever.

An agile project manager needs to adapt his or her leadership style to fit the situation and that will typically depend on several factors including the nature of the project and the level of maturity and experience of the team.

Face-to-Face Conversation

“*The most efficient and effective method of conveying information to and within a project team is face-to-face conversation*.”

The next principle emphasizes face-to-face conversation. This is another statement that you have to not take as an absolute but think of it as relative. It is not always possible with distributed teams to have face-to-face communications, but it is certainly desirable *if it is* possible.

This statement also doesn't mean that the *only* form of communication is direct, face-to-face communications. It is a reaction to the history of waterfall projects that heavily relied on documented requirements as a way of communication. There are many ways to communicate information in various forms, and you need to choose the optimum mix to fit a given situation. The right mix will depend on a number of factors, including the scope and complexity of the project and the distribution of the team working on the project.

Working Software Is the Primary Measure of Progress

“*Working software is the primary measure of progress*.”

Measuring progress on a software development project can be difficult and problematic. The traditional method is to break a project into tasks and track percent completion of those tasks as a way to measure progress; however, that can be very misleading, because often the list of tasks is incomplete and the level of completion often requires some subjective judgment, which is difficult to make and often inaccurate.

Testing is another factor in this—very often in the past, the entire development process and the testing process might have been sequential. The result is that even though the development of the software might have seemed to be complete, *you don't know how complete it really is until it has been tested and validated to be complete*. An agile approach emphasizes doing testing much more concurrently as the software is developed. There is a concept in agile called the *definition of done* that you will hear quite often. The team should clearly define what *done* means—it generally means that the software has been tested and accepted by the user. In other environments, the definition of *done* might be a lot more ambiguous and subject to interpretation. If you don't have a clear definition of done, any estimate of percent complete is likely to be suspect.

A more accurate measure of progress is to break up a software project into chunks of functionality where each chunk of software has a clear definition of done and can be demonstrated to the user for feedback and acceptance.

Promote Sustainable Development

> "*Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely*."

Many of the underpinnings of agile come from lean manufacturing and total quality management (TQM). In a manufacturing environment, companies learned many years ago that running a manufacturing plant like a sweatshop and forcing workers to work an excessive number of hours under poor conditions does not often result in high-quality products.

A similar thing is especially true in an agile environment, because the success of the effort is so critically dependent on the creativity and motivation of the team. In that kind of situation, it is even more important to create an environment where work is sustainable over a long period of time.

Continuous Attention to Technical Excellence and Good Design

> "*Continuous attention to technical excellence and good design enhances agility*."

This next statement is an interesting one. Many people might have the image of an agile software project team as a bunch of "cowboys" that just get together and hammer out code without much design planning and without any coding standards. That is not usually the case.

Agile recognizes the need for doing things the right way to avoid unnecessary rework later. However, an agile approach should not result in overdesigning or "gold-plating" a product. A comment you will hear often in an agile environment is the concept of "just barely good enough." In other words, the work should be done to a sufficient level of completeness and quality to fulfill the purpose it was intended to fill, and nothing more. Going beyond that level of "just barely good enough" is considered waste.

Simplicity Is Essential

"*Simplicity—the art of maximizing the amount of work not done—is essential*."

This next statement emphasizes *simplicity*. How many times have we seen projects go out of control because the requirements become much too complex and very difficult to implement and the requirements become overdesigned to try to satisfy every possible need you can imagine?

This is also related to the concept of "just barely good enough"—don't overdesign something; keep it as simple as possible. In some cases, it might make sense to start with something really simple, see if it fills the need, and then expand the functionality later only if necessary. Another concept in agile is called the *minimum viable product*, which defines the minimum set of functional features a product has to have to be viable at all in the marketplace.

It's generally much more effective to take an incremental approach to start with something simple and then expand it as necessary, rather than starting with something overly complex that may be overkill for the requirement.

Self-Organizing Teams

"*The best architectures, requirements, and designs emerge from self-organizing teams*."

Agile is heavily based on the idea of self-organizing teams but that needs some interpretation. Sometimes, developers have used the idea of "self-organizing" as an excuse for anarchy, but that is not what was intended. The intent is that if you have the right people on a cross-functional team and the team is empowered to collectively use all the skills on the team in a collaborative manner, it will generally deliver a better result than a single individual could deliver acting alone.

Continuous Improvement

> *"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."*

Agile is adaptive in two respects

- The design is adaptive to uncertain and changing user requirements—it can start with a high-level view of requirements and progressively elaborate requirements after the project is initiated.
- The process itself is adaptive rather than highly prescriptive. Agile is based heavily on continuous improvement, using short intervals to reflect on what's working and what's not working and taking quick corrective action as necessary. In Scrum, this is called a *retrospective*, and it happens at the end of each sprint.

The team is expected to continuously improve and adapt the agile process as needed as the project progresses.

Summary of Key Points

1. Agile History

   Agile has evolved over a number of years from a proliferation of approaches going back to the 1980s and 1990s. The Agile Manifesto which was published in 2001 was a key turning point that defined some principles and values that were needed to simplify and synthesize that diverse array of different methodologies.

2. Agile Manifest Values and Principles

   The Agile Manifesto values and principles provide a strong foundation for understanding agile at a deeper level. These values and principles were not meant to be applied rigidly; they were intended to be interpreted and applied in the context of a given project and business environment.

   That is the essence of an adaptive approach; rather than rigidly applying a fixed set of rules to all projects, it is much better to focus on the values and principles and use good sense to apply them intelligently to fit a given project and business environment.

Many of these values and principles are applicable to almost any project in some way. It's not a matter of having one set of values and principles for agile projects and another set of completely different values and principles for other projects; it's a matter of applying these values and principles intelligently in the right context to fit the project.

Discussion Topics

Agile History

What do you think is most significant about the role of the Agile Manifesto in the overall history of the development of agile principles and practice up to that point?

Agile Manifesto Values

1. What are some of the trade-offs that you think might have to be made in applying the Agile Manifesto values to real-world projects? How would you go about resolving these trade-offs?

2. Provide an example of a project and discuss how you might have applied the Agile Manifesto values to that project.

Agile Manifesto Principles

3. Which two or three of the Agile Manifesto principles do you think are likely to have the biggest impact on the success or failure of a typical project, and why?

4. Are there any Agile Manifesto principles that you think would not be applicable to every project (agile or not)? Why not?

5. Provide an example of a project and discuss how you might have applied the Agile Manifesto principles to that project.

[1] Dr. William Royce, "Managing the Development of Large Software Systems," ***Proceedings, IEEE Wescon* (August 1970), pp. 1–9**.

[2] Ibid., p. 2.

[3] Vic Basili and Craig Larman, *Iterative and Incremental Development: A Brief History* (IEEE Computer Society Digital Library, June 2003), http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf, p. 2.

[4] Ibid., p. 3.

[5] Ibid., p. 4.

[6] Ibid., p. 4.

[7] David F. Rico, “The History, Evolution and Emergence of Agile Project Management Frameworks,” ProjectManagement.com (February 4, 2013), http://davidfrico.com/rico-apm-frame.pdf, p. 1.

[8] Ibid., p. 1.

[9] Scott W. Ambler, “History of the Unified Process,” Enterprise United Process (2013), http://www.enterpriseunifiedprocess.com/essays/history.html

[10] Blogroll, “10 Years of the Agile Manifesto—It started in 2001 with the Manifesto,” posted by Alistair on January 2, 2010, http://10yearsagile.org/it-started-in-2001-with-the-manifesto.

[11] http://agilemanifesto.org/.

[12] http://agilemanifesto.org/principles.html.