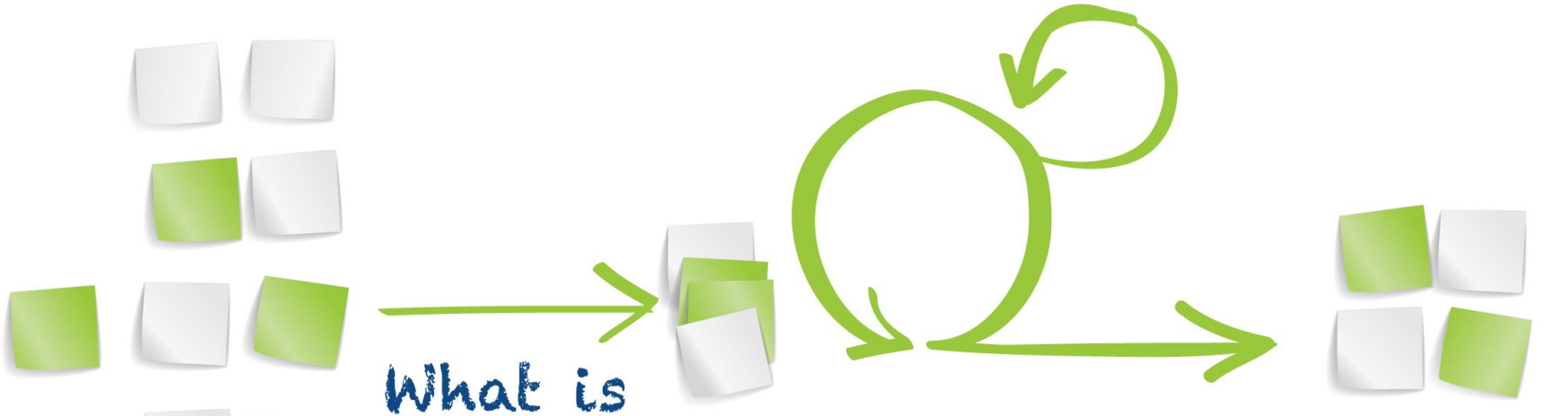
intelliware
software development

Agile



What is Agile Development?

What is Agile Development?

Once primarily the domain of early adopters, Agile flavoured methodologies have steadily gained acceptance as a mainstream approach to software development.[1,2,3] Although Agile has come to mean many different things to different people, at its core, it is a philosophy that guides a set of technically rigorous development practices. It has been adopted by a growing number of organizations to lower the risk that is inherent in software development and to deliver better software as defined by the end user. As more teams adopt Agile practices and more companies transform to Agile cultures, we continue to learn together about the best practices and benefits of developing software in an Agile manner.

This introductory paper is designed to provide a basic understanding of Agile software development.

The Foundations of Agile

A quick lesson in software development history provides context for both how and why Agile is important. Agile is typically seen as an evolutionary step forward from the “Waterfall” method, which is the traditional – and still most prevalent – way of developing software.

Top 10 Agile Links

1. agilemanifesto.org
2. en.wikipedia.org/wiki/Agile_software_development
3. agilealliance.org
4. codeproject.com/Articles/604417/Agile-software-development-methodologies-and-how-t
5. objectmentor.com/omSolutions/agile_what.html
6. agilemethodology.org/
7. mountaingoatsoftware.com/agile/new-to-agile-or-scrum
8. allaboutagile.com/what-is-agile-10-key-principles/
9. versionone.com/Agile101/Agile-Development-Overview/
10. i-proving.com

The Waterfall Method

Waterfall is a sequential approach to software development. As Figure 1 illustrates, it is a multi-step process with phases executed in a stepwise manner. First, software requirements are defined. This is followed by software design, then development and finally, integration and testing. Often steps are defined by gates, at which point some sort of approval, often involving a document, is required before the project can proceed to the next step. Hence, the approach is also often referred to as, “document driven development”.

Waterfall has obvious appeal. It provides an easily understandable process that seems to progress in a logical sequence. Moreover, it’s a familiar process because it is similar to how we build other things, such as buildings and cars. The gates provide a form of risk management, in that the process cannot proceed to the next step without an authorization of the preceding step.

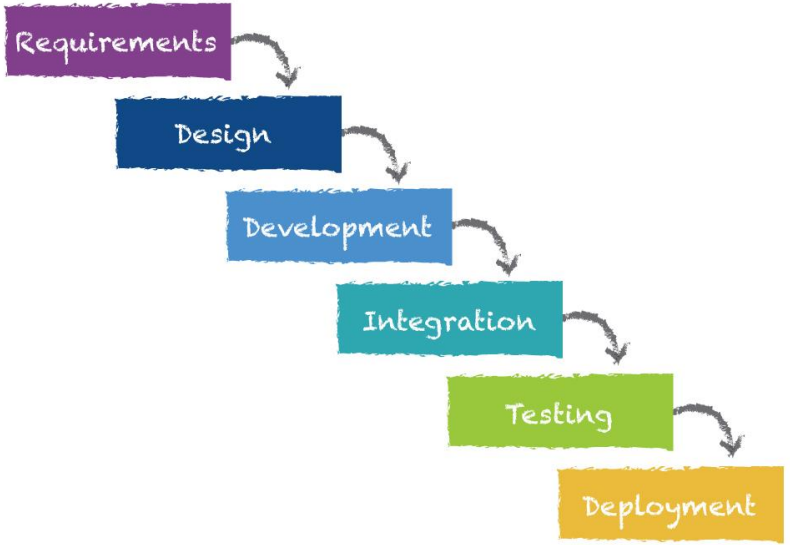


Figure 1: The Waterfall Process of Software Development

The Problem With Waterfall

The Waterfall approach works well if everything is straightforward, simple and predictable. But therein lies the problem: software development is rarely straightforward, simple and predictable. Waterfall poses the following risks:

1. **Startup risk:** Many projects fail to get off the ground because of the "analysis paralysis" that is symptomatic of the Big Design Up Front (BDUF) approach. Projects get stuck at a gate while various stakeholders hesitate to commit to a sign off.

2. **Integration risk:** The sequential approach of Waterfall development leaves integration until near the end of development. Consequently, there is significant risk that the software will have greater integration issues since it will be done all at once.

3. **Cost risk:** The cost of change in Waterfall projects is relatively high (see Figure 2) because it tries to minimize change by using BDUF (Big Design Up Front).

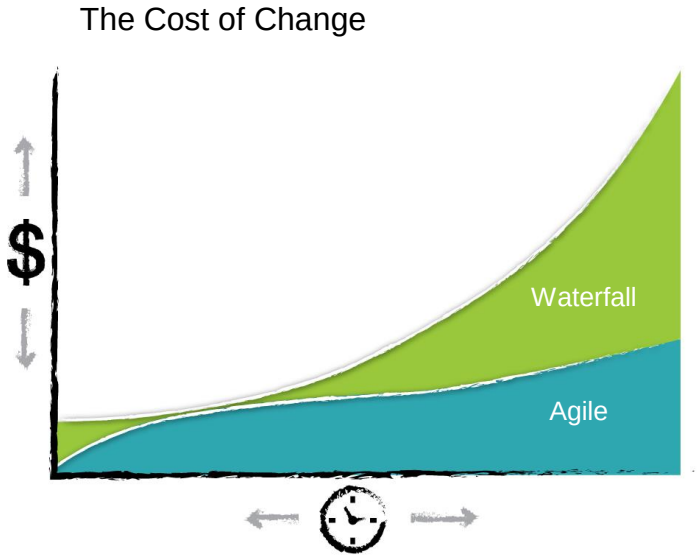


Figure 2: The Cost of Change: Traditional (Waterfall) vs. Agile[4]

The Problem With Waterfall (continued)

4. **Delivery risk:** Waterfall has a relatively high risk of failing to meet deadlines or accommodate customer needs. Business needs inevitably change throughout the development process (which can range from a few months to years) and the Waterfall approach, by nature, does not take this into account. Also, in a document driven development environment, a development team may place more focus on delivering documentation than on customer features.

5. **Technical risk:** Waterfall lacks rigorous practices of more modern methodologies. These practices include unit testing, collaborative code-base and frequent builds, all of which help ensure technical excellence.

To understand how these risks are avoided in modern approaches, it is helpful to review a few key historical milestones in software development methodologies.

From Waterfall to Agile

The Waterfall method has been prevalent in software development since the mid ’50s. It wasn’t until the late ‘60s and early ‘70s that references to iterative software development were made.[5] In 1970, Winston W. Royce was credited as one of the first people to describe the Waterfall approach and also be one of the first to highlight its flaws, arguing for an iterative, Agile-like approach to software development. [6]

In the mid ‘80s, after decades of experience with software developed using the Waterfall methodology, change was needed. Software projects were earning the reputation of often being over budget, behind schedule, and failing to meet the needs of end users. Overly complex software was the norm (Figure 3 pokes fun at how this tends to happen when using BDUF). Then, in 1986, Frederick P. Brooks stated the following in his landmark paper, *“No Silver Bullet-Essence and Accident in Software Engineering”:*

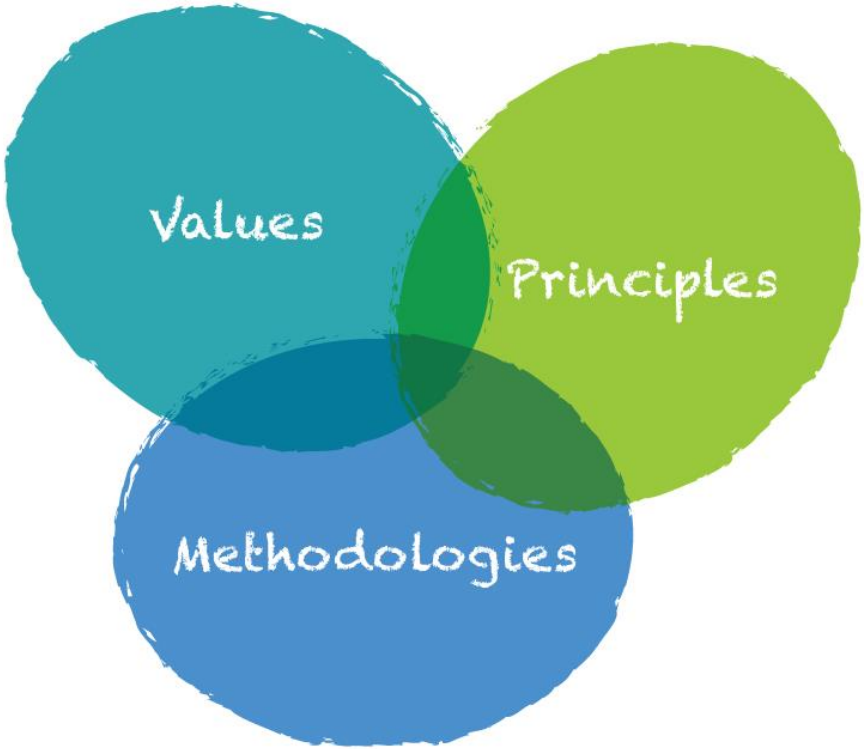
> *…it is really impossible for clients, even those working with software engineers, to specify completely, precisely, and correctly the exact working requirements of a modern software product before having built and tried some versions of the product they are specifying.*
>
> *Therefore one of the most promising of the current technological efforts…is the development of approaches and tools for rapid prototyping of systems as part of the iterative specification of requirements.*[7] *– Frederick P. Brooks*

Over the next fifteen years, several “lightweight methodologies” were developed. In 2001, a group of software leaders, in a field known at the time as lightweight methods, met to bring these new methodologies together to find a better way to build software. The result was the Agile Manifesto.

Figure 3: The problem with guessing customer requirements

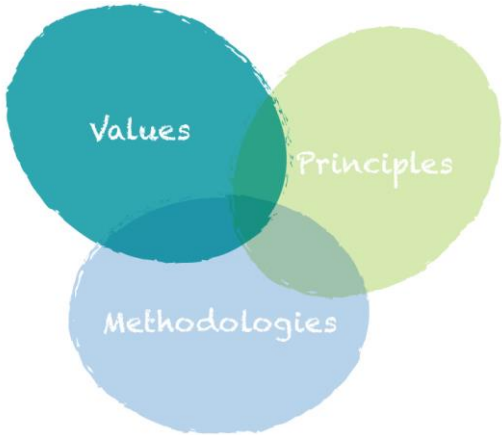
The Basics of Agile

Agile software development is comprised of Values, Principles and Methodologies. The 4 Agile Values serve as the foundation of Agile philosophy. The 12 Agile Principles embody the Values and provide more concrete examples of what Agile means at a lower level. They form the Agile philosophy that guides Agile Methodologies.



Agile Values

Agile is a somewhat loose term. If it had a motto, it would be embrace change.[8] The Agile Manifesto is accepted as the official definition of Agile and expresses the four foundational Values of the philosophy:



Manifesto for Agile Software Development[9]

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

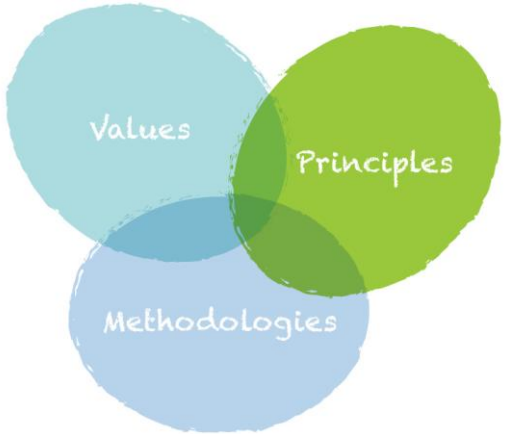
That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas.

Agile Principles

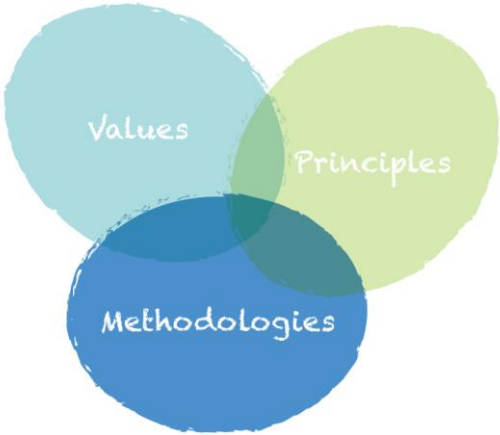
The 4 main Values of the Agile Manifesto guide the twelve Principles of Agile development, which can be summarized as follows:

1. Satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently.
4. Bring business people and developers together to work throughout the project.
5. Build projects around motivated individuals.
6. Communicate face-to-face when possible.
7. Measure progress primarily through working software.
8. Develop at a sustainable pace.
9. Focus on technical excellence and good design.
10. Simplicity, maximizing the amount of work not done, is essential.
11. Enable teams to self-organize.
12. Reflect at regular intervals on how the team can become more effective.



Agile Methodologies

Agile software development is perhaps best understood as a philosophy that guides several groups of practices known as Methodologies. These Methodologies, which are like recipes to follow to help you be Agile, include Scrum, Kanban, XP / Extreme Programming, Lean Programming and Agile Modeling.[10] Collectively, they comprise only a selection of the "lightweight methodologies" that can be used to be Agile. While each Methodology has its nuances, the important thing to understand is there is more than one way to practice Agile development.



While ground-breaking at the time, the Agile Manifesto has led to misconceptions about Agile as a method-less, process-less approach. These claims are false. Many are based upon straw man fallacies – oversimplifications of arguments to make the Agile Manifesto easier to attack. People claim that "Agile has no documentation". It's a fallacious argument. Agile has documentation but places a greater value on working software. As Agile co-creator 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."*[11] *- Jim Highsmith, for the Agile Alliance*

In addition to the Values, Principles and Methodologies, there are additional characteristics that are common to organizations that have adopted Agile philosophies.

Agile Process

The Agile process takes a large software project and divides it into many smaller pieces to be developed incrementally and iteratively. Studies have found that project size and success are negatively correlated, i.e. the shorter the project the higher the rate of success.[12]

The Agile approach helps reduce project size by effectively making several mini-projects. It is the iterative approach that distinguishes Agile from other methods.

Unlike traditional approaches, Agile has multiple planning and development phases, known as iterations. Each iteration tends to be a week or so of work. The development team and the customer team work together to prioritize what will be included in each iteration. The end result of the relatively short sprint is working software delivered to a production-like environment where it can be tested by the customer. This iterative process is repeated until the software project is completed. Along the way, depending on the customer's needs, there can be a series of releases – software released to the end-user.

Agile Practices

There are dozens of Agile practices. Not all are used by Agile practitioners, but those looking to be Agile should be aware of these key practices. Here are some examples to help illustrate how the Agile values have been applied into more concrete practices.

Daily Standup (Stand-Up Meetings)

Also referred to as a Daily Scrum, Agile teams have daily meetings to briefly share need-to-know information. These meetings are held to keep the development team on the same page, not to provide a status update to the project manager. Brevity is the key to these meetings. In Daily Scrum meetings, participants answer three questions[13] :

1. What did I do yesterday?
2. What will I do today?
3. What problems are preventing me from making progress?

Agile Practices

User Stories

A User Story is a short description of a function that an end-user would want. There are three elements to a User Story:

- a written description for planning (card),
- a conversation about the story to better understand it (conversation), and,
- a series of tests to confirm the story (confirmation).[14]

Figure 4 provides an example of a User Story card. These are written from the perspective of the end-user in language that is understandable to them. Stories serve as a form of currency between customers and developers that both parties understand.

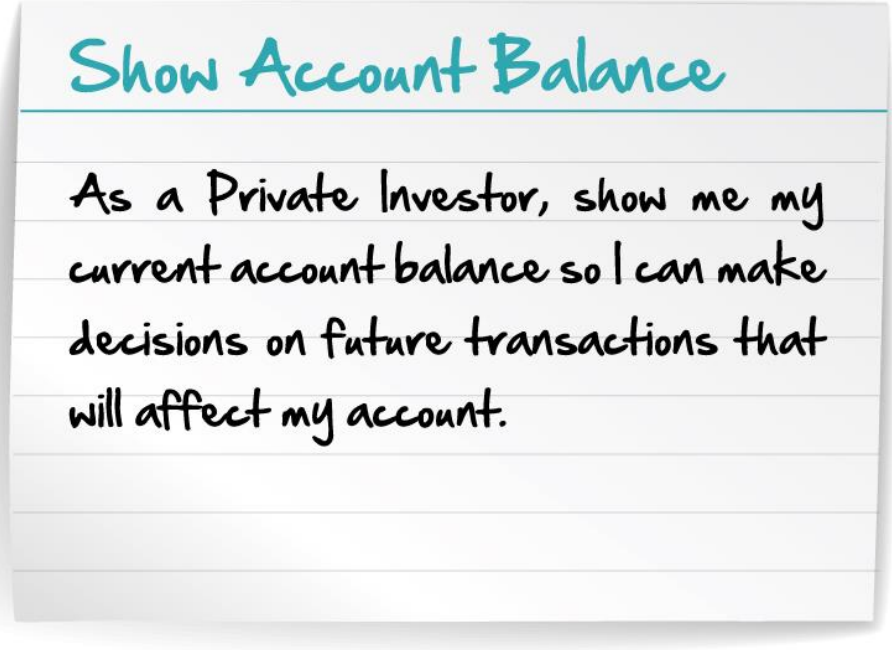


Figure 4: An example of a User Story for a Financial Services System

Agile Practices

Automated Testing

With Agile, a key aspect is the implementation of automated tests. Formal and thorough automated tests help ensure the delivery of working software throughout development and eliminate defects at the source. Developers write test code using any number of available frameworks (e.g., JUnit is the go-to Java framework) at the same time as they develop the code. In this manner they build a safety net for the working code that allows them to make changes and have confidence that other features haven't been broken. It also greatly reduces the length of time between the origin and the discovery of a bug.

Acceptance tests are functional tests for a User Story. In other words, they are programmed checks to ensure that existing features continue to work as defined as new features are added. Ideally, acceptance tests are defined by the customer and coded by developers. Running acceptance tests help prove to the customer that the Story is done. After acceptance tests are created, they are run automatically and repeatedly so that the newly developed code works.

Agile Practices

Automated Builds

A key principle for Agile methodologies is to have running software at all times. In practice, the only way to do this is by ensuring that all software development is regularly and automatically compiled, built, deployed and tested. This is usually done many times a day and at least once every time a developer “checks in” code as a main part of the development branch.

The advantages for automating builds are huge. They reduce troublesome deployment and integration problems and they provide a reliable environment for demonstrations and testing.

Agile Practices

Agile Planning: Release, Iteration and Task

In Agile development, planning is divided into three levels: release, iteration and task. At the start of the project, the developers and the customer come together to discuss the major User Stories (features) that are needed in the software. They focus on identifying must-have features and roughly prioritizing and estimating them, which helps facilitate Release, Iteration, and Task planning.

Release planning: A customer-driven planning session. The developers and the customer decide on a date for the first in a series of product releases. They decide what stories to incorporate in the release. The Developers drive the effort estimates for the stories while the customer drives the selection of the stories. Effort estimates take various forms depending on the preferences of the development and customer teams.

Iteration planning: A joint effort between the customer and developers to do part of the release plan. As in release planning, the customer defines and prioritizes the User Stories and the developers estimate the effort it will take to develop them. Naturally, the timeline is shorter for an iteration than that of a release. It is often a matter of weeks, not months, for an iteration.

Task planning: After planning the iteration, the development team breaks down the stories for the iteration into a series of tasks. A list of the tasks is documented in the project room where it is highly visible; Post-it® notes and whiteboards are common tools to help with task planning. Developers sign up for tasks and assign estimates to them.

Agile Practices

Pair Programming

This practice has two developers working together on a programming task. Typically, one person takes the role of entering the code (the driver) while the other thinks about the next steps and how this code will fit into the bigger picture (the navigator).

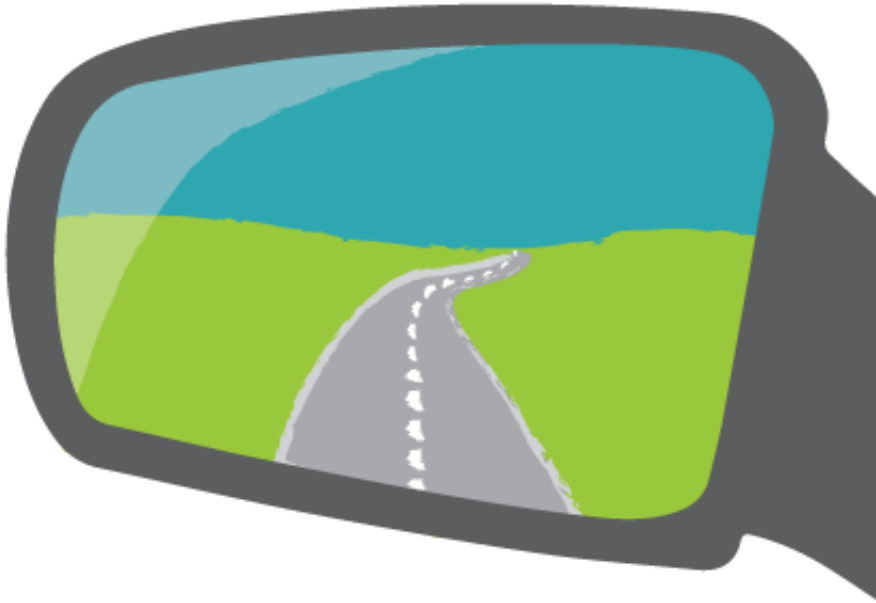
A common objection to pair programming is that it is wasteful to have two people doing the work of one. While it may take more developer time to program this way, the output often justifies it. As one study found, pairing takes 15% more effort than one person working alone but produces results more quickly and with 15% fewer defects.[16] Results will vary on a case-to-case basis, but when considering pair programming, ask whether a reduction in defects is worth added effort and resources. Also, pairing is not a full-time requirement. Teams often set their own working rules around when it is advantageous to pair.

Agile Practices

Continuous Integration

This practice has the development team integrate their code to the system several times per day. Before integrating completed code, the developer runs a series of tests to make sure the code to be integrated won't break any existing functionality or tests in the system. To do so, the developer runs all of the tests for the system and makes any necessary fixes. The more frequently code is integrated, the less time it takes to merge and find errors.

Agile Practices



Retrospectives

A retrospective is a meeting to look back over an iteration, release, or project, specifically to discuss what worked well, what could be improved, and most importantly, how to translate the lessons learned into actionable change. They are a forum for the team to improve upon their process.

The Business Benefits of Agile

Industry View

The appeal of an Agile approach to develop software incrementally and iteratively continues to grow. Agile helps lower risks associated with delivery, scope and budget that are common to every software development project.

> One *MIT Sloan Management Review* study on software development practices found that early delivery of a partially functioning system and frequent deliveries throughout development are both positively correlated with higher quality software.[17] Agile enables the collaboration between the development team and the customer, which offers the mutual benefit of mitigating the inherently high risk of software development.

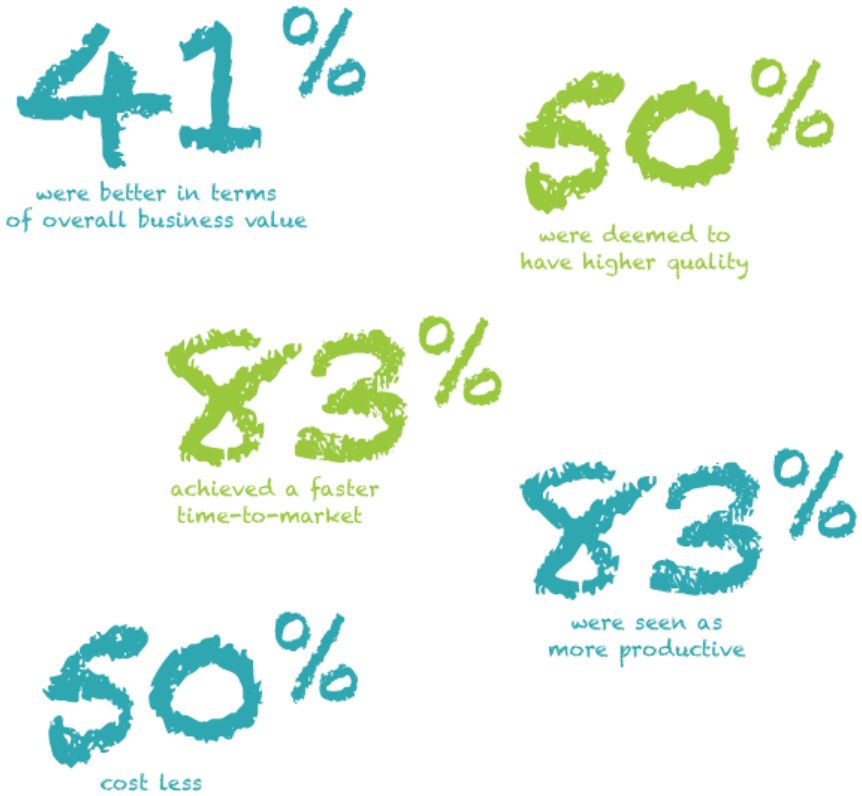
The Business Benefits of Agile

Industry View

Version One's 7th Annual State of Agile Development Survey found that 90% of those who implemented Agile reported an improved ability to manage changing priorities[18]. In addition, most (70%) believed that Agile projects have a faster time to completion.[19]

Finally, in 2009, Dr. David F. Rico's research and synthesis of available data on Agile versus traditional methods[20] identified many compelling business benefits. In an analysis of 23 Agile versus 7,500 traditional projects[21], Agile projects stacked up impressively well:

- **41%** were better in terms of overall business value
- **83%** achieved a faster time-to-market
- **50%** were deemed to have higher quality
- **50%** cost less
- **83%** were seen as more productive



The Business Benefits of Agile

Our View

Intelliware has been involved with Agile since the very early days of Kent Beck's articulation of his Extreme Programming principles in the late 1990's, at a time pre-dating the Agile Manifesto. For us, Beck and the rest of the Agile pioneers articulated and validated many of the frustrations and challenges we had experienced using Waterfall and BDUF. XP, and by extension, Agile, offered an alternative that made sense and put simply, worked.

Over the years, different projects have enjoyed a range of benefits as a result of the application of Agile philosophies. However, there are four key business benefits that our clients view as particularly important.

The Business Benefits of Agile

Our View

1. Get unstuck.

In all organizations, it is often very difficult to just get started. With Waterfall, there are unreasonable demands to “nail the requirements”, which fuels analysis paralysis. With Agile, a project can get started with high-level requirements and a simple gating of the first level of useful functionality. You need to have a high level map of where you are headed, but the nitty-gritty detail work of fine-grained requirements, design, coding and testing can be done more efficiently and with a higher degree of quality when they are done in parallel.

Agile helps you get going. If you don’t start, you won’t get to the business benefits of your project until it’s too late.

The Business Benefits of Agile

Our View

2. Change is built into the process.

The biggest fallacy of Waterfall is that you can “nail the requirements”. All projects have change, and the longer they run, the more changes your project will require. If all your software project plans are tied up in an elaborate Microsoft Project®-style plan with dependencies, you and your organization are in for a rough ride.

Agile software development builds change into the process from the get-go.

The Business Benefits of Agile

Our View

3. Risk is managed, your project is delivered.

Because change is built right into the process, and Agile depends on robust technical practices such as automated testing and automated builds, you have a running production system as early as day one. As the system grows, and the project endures changes in requirements and design, the core application is kept running. The risks associated with change are dealt with daily, and so, at the end of the project, there isn't a big bang release with a mountain of hidden problems.

When steered properly, and with discipline, Agile projects get delivered.

The Business Benefits of Agile

Our View

4. Deliverable quality is higher.

Key drivers for better application quality include a number of the rigorous Agile technical practices such as automated tests and builds. But another, less talked about contributing factor to overall project quality is the effect Agile processes have on the people involved with the project. Because of the cross-functional nature of Agile teams, questions get answered quickly, roadblocks can be removed, and the team is motivated to work together towards a common end goal. It's not a year-long project; it's a series of one or two week sprints with real deliverables in a short period of time. It's a satisfying and energizing manner in which to work.

Stronger technical practices and greater overall team motivation all combine to result in higher quality project deliverables.

Is Agile right for you?

A transition to Agile is not a trivial effort, nor is it necessarily for everyone. A common pitfall is to try a few Agile practices that seem convenient and ignore those that seem difficult. This often results in a failed transition based on faulty assumptions and missing pieces. A truly sustainable Agile transformation requires a significant change in culture within your organization. Take time to really appreciate what Agile entails, understand the main barriers to adoption and consider its compatibility with your organization before starting implementation.

Sources

[1] Scott M. Ambler and Matthew Holitza. *Agile for Dummies.* Hoboken: John Wiley & Sons, 2012. Print.
[2] Krill, Paul. "Agile software development is now mainstream". *InfoWorld*. The IDG Network. 22 Jan. 2010. Web.
[3] Heusser, Matthew. "Has Agile Software Development Gone Mainstream?" *CIO.com*. CXO Media Inc. 12 Aug. 2013.
[4] Ambler, Scott. "Examining the Agile Cost of Change Curve". *Scott W. Ambler + Associates*. Ambysoft Inc. 2002. Web.
[5] Larman, Craig and Basili, Victor R. "Iterative and Incremental Development: A Brief History". C2.com. 21 Apr. 2011. Web.
[6] Winston, Royce, W. "Managing the Development of Large Software Systems", *Proceedings of IEEE WESCON 26* (1970). Web.
[7] Brooks, Frederick P. "Silver Bullet: Essence and Accidents of Software Engineering". Pgs. 13-14 *Computer In Computer*, Vol. 20, No. 4. (1987). Web.
[8] Craig Larman. "Agile & Iterative Development: A Manager's Guide". Boston: Pearson Education. 2004. Print.
[9] Beck, Kent et al. "Manifesto for Agile Software Development". *Agilemanifesto.org.* 2001. Web. 15 Sept, 2013.
[10] Scott M. Ambler and Matthew Holitza. *Agile for Dummies.* Hoboken: John Wiley & Sons, 2012. Print.
[11] Highsmith, Jim. "History: The Agile Manifesto". *Agilemanifesto.org.* 2001. Web.
[12] Craig Larman. "Agile & Iterative Development: A Manager's Guide". Boston: Pearson Education. 2004. Print.
[13] Shore, James, Warden, Shane. *The Art of Agile Development*. Sebastopol: O'Reilly Media, Inc., 2008. Print.
[14] Mike Cohn. "User Stories Applied: For Agile Software Development". Boston: Pearson. 2004. Print. Reference also to Ron Jeffries (2001)
[15] Wake, Bill. "INVEST in Good Stories, and SMART Tasks". *XP123.* Web.
[16] Shore, James, Warden, Shane. *The Art of Agile Development*. Sebastopol: O'Reilly Media, Inc., 2008. Print.
[17] MacCormack, Alan. "Product-Development Practices That Work: How Internet Companies Build Software". *MIT Sloan Management Review.* 2001. Web.
[18] VersionOne®, 7th Annual State of Agile Development Survey © 2013.
[19] VersionOne®, 7th Annual State of Agile Development Survey © 2013.
[20] Dr. David F. Rico, PMP, CSM, "*Business Value of Agile Methods – Cost and Benefit Analysis"* davidfrico.com/rico09e.pdf, 2009.
[21] Mah, M. (2008). Measuring agile in the enterprise: *Proceedings of the Agile 2008 Conference, Toronto, Canada*.