TOP 30 Agile Myths BUSTED!

Top 30 Agile Myths - BUSTED!
is brought to you by Telerik TeamPulse

Author

Philip Japikse

Microsoft MVP, Agile Practices Evangelist at Telerik

Editors

Joel Semeniuk

Microsoft Regional Director, MVP Microsoft ALM,
and Executive VP of Agile Project Management at Telerik

Steve Porter

Agile Expert and Coach,
TeamPulse Product Owner at Telerik

Preface

In the 11 years since the Agile Manifesto was created, the adoption of agile concepts has continued to grow. The number of agile user groups, conferences, and books is a strong indication that more and more people are at least exploring the idea, if not working to improve their adoption and execution. This growth has been so significant that Gartner has declared that "agile is now mainstream"[1].

Along with this growth in adoption and exploration of agile, the number of myths surrounding agile has grown as well.

In order to objectively look at some of the most common myths it's important to examine the Agile Manifesto itself. There are two parts of the Manifesto – four value propositions and twelve principles. The value propositions are a great place to start debunking some of the most common myths.

"Manifesto for Agile Software Development

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."

agilemanifesto.org

[1] "Agile – The End of the Beginning" by David Norton on Gartner's blog.

Project Management/ Process Myths

Myths #1-4: Agile has no Process, doesn't have Documentation, doesn't believe in Contracts, doesn't follow a Plan

These four common myths most likely come straight from a misunderstanding of the Manifesto's value propositions. In order to fully understand the value propositions, you have to take all three parts into account, not just the middle section.

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

Nowhere does the Manifesto say "Stop writing documentation" or "Ban all contracts". There is indeed value in following a process, writing documentation, and having a plan. And contracts are absolutely necessary when dealing with external parties. However, they should not become the focus of the software project, nor should they prevent the users, customers, and development team from collaborating.

Myth #5: Agile Projects don't provide Budget Estimates

When projects are scope-boxed (as they are in traditional development methodologies like waterfall), given a list of requirements, and estimates for those requirements, the budget and schedule are derived completely from the estimates. Estimation is very difficult, and more often than not, they are wrong (either too high or too low).

Agile processes like Scrum and Extreme Programming use the concept of time-boxing. This embraces the fact that estimates aren't reliable, and instead they have teams work for a predefined fixed period of time (somewhere between one and four weeks), getting as much done in that period of time as they can. The goal of each sprint, as the time boxed iterations are called in Scrum, is potentially releasable software. This keeps the team focused on delivering completed features each sprint.

Time-boxing allows us to know exactly how much each iteration/sprint costs as agile projects discourage overtime and other variable swings in costs (largely to prevent "Death Marches"). Estimates can be used to rough out how many sprints will be needed to develop all of the requirements, but each sprint is truly a fixed cost. This makes budgeting much cleaner.

Agile teams fix the cost, fix the date, but are honest and understand that the scope can and will change.

Myth #6: Agile is not suitable for fixed bid projects

Before looking at this myth, it's important to define "Fixed bid projects". Does this mean fixed scope, schedule, and cost? History has proven that software projects of any significance in size or complexity cannot be accurately estimated well. What is more, according to CHAOS reports 24% of all projects fail and 44% are significantly challenged[2]. Regardless of the project management methodology, attempting to fix all three does not have a high probability of success. Something has to give – whether it's adding resources, cutting scope, or moving the delivery date.

As described in the myth "Agile Projects don't provide Budget Estimates", agile teams fix the cost, fix the date, and vary the scope. The onus is on the Product Owner to make sure the requirements are not only prioritized correct, but ***ordered*** correctly. The team will deliver the highest priority items in the order that they are specified in the product backlog. The lowest priority (and last in the order) items usually turn out to be superfluous, and never need to be developed anyway.

Myth #7: Agile means No Commitment

A popular myth is that agile teams don't make commitments – it's just 6 to 8 developers working until someone declares "We're Done!" On the contrary, successful agile teams are extremely realistic and transparent in what they promise to deliver. Scrum and Extreme Programming (XP) have the concept of ***a Sprint Backlog*** – these are the items that will be developed during a sprint or iteration. For some Scrum teams, the items on the Sprint Backlog are treated like a contract – the team is making a commitment that those items selected ***will be completed*** in that sprint (the newest version of scrum backs off on the strength of the commitment). XP teams sometimes allow change if all parties agree and the item being replaced hasn't been started yet. In both cases, deferment (not completing an item that was promised) is treated as an exceptional occurrence, and not treated lightly. This commitment and regular cadence of delivering software in small increments builds a high degree of trust between the team and the other interested parties, like stakeholders, users, and customers.

Scrum teams that are following the most recent teachings on Scrum don't commit on the scope of the sprint. This more closely fits the idea of not scope-boxing ***anything***, since scope-boxing is a dangerously inaccurate game to play. Instead, the team has renewed their commitment on all of the tenants that make Scrum successful – rapid feedback loops, collaboration, and constant communication and transparency. Often, when teams are that communicative, there is less of a demand for an iron clad commitment. The trust and collaboration that results from teams working in this manner overrides the outdated requirement for commitments.

In contrast to typical waterfall projects, since everything is based on estimates, teams seldom deliver everything promised. It's not that they intentionally under deliver, and it's not a reflection of skills or work ethic. Estimates are seldom correct and fixing scope and schedule based on inaccurate estimates quite often leads to features being dropped from the release.

[2] "The Curious Case of the CHAOS Report 2009" by Jorge Dominguez

Myth #8: Agile Development is Not Predictable

This myth is closely related to the Agility Means No Commitments.
What is more predictable?

Code that is tested and delivered every two weeks, **teams** that are communicating their status every day (through standups, burn down charts, and various other mechanisms), and **requirements** that are constantly being groomed and updated.

- or -

Code that doesn't get tested until the end of a 6/9/12 month project cycle, **teams** that go dark for months at a time, not reporting status until the end of the construction phase, a **requirements** document that never gets updated because the change request process is too difficult to manage.

Successful Agile teams bring predictability to software development because every step of the way they are communicating, deploying real code, and adapting to change (and keeping the documentation current). Every two weeks, they will release a set of features that they stated two weeks prior that they were going to work on. They met the expectations and delivered on their predictions.

Myth #9: Agile Projects don't use Project Managers

It is true Scrum doesn't discuss Project Managers – it focuses on the Product Owner, Scrum Master, and the Development Team. Scrum is only one implementation of the agile principles (albeit probably the most well-known). Even though Scrum doesn't call out the role of Project Management in its literature, it never states that you can't have one.

Project Managers typically have different responsibilities, such as budgeting, reporting, and portfolio management. These are all extremely important, especially in larger organizations, and experience project managers excel at these tasks. Project Managers are also a vital link in the communication and collaboration goals set forth in the agile values and principles. These project management tasks still need to be completed, and individuals who are trained and experienced in performing these activities best handle these tasks. In projects using Scrum, these tasks are often split up between the product owner and scrum master. Often in these projects, professionally trained project managers will take the scrum master role and then train the product owners on how to perform the appropriate activities.

BUSTED!

Myth #10: Agile Projects don't use Business Analysts

Scrum doesn't use the term Business Analyst, and instead refers to the groomer of the requirements (typically called the backlog in agile projects) as the Product Owner. This allows for a single point of contact for the team when they have questions or concerns about any of the requirements. This does not mean that Business Analysts are excluded from agile projects. On the contrary, the deep domain knowledge that business analysts possess is a vital asset to the Product Owner. This cadre of domain experts helps craft the backlog items, as well as assist in determining order and priority.

Extreme Programming doesn't have a single role (like Scrum's Product Owner), but considers the customers and business analysts (along with development) as a collaborative team that determines the items on the backlog and the priority of those items.

As agile projects scale, teams can be broken up into feature teams. These feature teams will still be reliant on the customers, users, and business analysts to make the backlog for each of these features, as well as the overall backlog for the project.

Myth #11: Agile Projects are out of control

There is a common misconception that agile teams are free to do what they want, when they want, how they want. This can't be farther from the truth. Successful Agile teams are self-managed, which is not the same as un-managed. They are in constant communication with the other team members, as well as those outside the team, and therefore completely transparent. Daily communication (such as daily standups), pair programming, and other team practices bring teams together into a cohesive unit. Couple with reviews of each sprint with stakeholders and the business makes the development team accountable for their actions.

Additionally, when teams are told what needs to be done (as opposed to what ***and how***), successful agile teams will more often than not discover the best way to get done what needs to be done, finding efficiencies and optimizing resources.

Finally, Agile methodologies introduce cadences to the development routine to ensure that projects stay *in*-control. Every day, teams are verbally giving a state of the union in the standup, as well as updating the burndown charts and/or Kanban boards. Each iteration is capped off with a review where the team demonstrates the current state of the project to interested parties. As software is prepared for release, all of the variables that traditionally cause software projects to fail or be perceived to be a failure (such as developers that went dark, unrealistic expectations, poor communication, and many others) have been met head on long before the software gets deployed. There is an incredible level of control, since every step of the project has been communicated at a steady cadence to all interested parties.

Contrast that with traditional waterfall, where individual developers and teams can go weeks or even months without having to show any results. It's these situations that can cause end of project surprises that can destroy a team's reputation or even kill a project.

Myth #12: Agile has no discipline

To successfully execute agile development, the team needs a very high level of discipline. This includes engineering practices like Test Driven Development and/or Behavior Driven Development, Pair Programming, and Continuous Integration to name a few, as well as process practices like backlog grooming, rapid feedback loops, and constant communication and collaboration. All of these techniques require a highly dedicated and disciplined team of professionals to work in sync.

Myth #13: Agile is the latest hype

Iterative software development processes were first documented as early as 1974. Lightweight methods further evolved in the 1990s as a reaction to the heaviness of waterfall and big design up front. What is now referred to as "Agile" is simply an evolution of ideas that have been growing for quite some time[3].

Myth #14: Agile is just an iterative and incremental process

It is true that agile is iterative and incremental. In fact, many feel that agile is the evolution of the iterative styles that have been in existence since the 1970's. With a focus on collaboration and rapid feedback loops, agile brings the customer into the process during the entire length of the project, not just at the very beginning and the very end. Successful agile teams are also very focused on continuous improvement through the constant communication, identifying and elimination waste, and striving to deliver more business value in progressively shorter periods of time.

Agile also addresses many human conditions by recognizing that humans aren't computers. The best means of communication for people is in-person – the best being face to face, voice at a minimum. There are many mechanisms for this, from the daily standup, sprint planning meetings, and regular retrospectives. Whether it's communication between team members or the team and customers and/or users, the goal is rapid feedback, transparency, and constant communication and collaboration.

These factors go above and beyond the length of the project plan, and create a significant difference in the core philosophies of software development.

BUSTED!

[3] "Hype Cycles" by Gartner

Myth #15: Agile is just a series of waterfalls

There's a lot more to being agile than just shortening the cycle. The waterfall process is defined by stages (or tollgates) that happen in serial - Requirements, Design, Construction, Verification, and then Maintenance. Agile incorporates all of these tasks in parallel, with a focus on collaboration, rapid feedback loops and the embracing of change. Each story (or item of value) moving through an agile queue can be labeled with similar states from waterfall. This does not make the sprint mini-waterfalls. Since the transitions are at the feature/story level (and not at the project level), it just shows the natural progression of an idea. And these state transitions are much shorter in length, as the goal is to produce a minimal marketable feature, and not every potential variation. This can cause a single feature to move through the state transitions multiple times as the value of the feature is expanded after a release through additional requirements and potential use cases.

Myth #16: Agile is a silver bullet solution!

Proponents of agile will sometimes claim that moving to agile will "fix all of your problems". That is not the case. It's important to stress that the Agile Manifesto is a set of Values and Principles that define a core attitude for software development. These values point to Collaboration, Rapid Feedback Loops, and Quality. In this way, agile ***exposes*** your problems - before any issue can be addressed, it needs to be surfaced. Once exposed, one can work to eliminate the friction and blocking issues.

Myth #17: Agility helps in any case

Just as agile is not a silver bullet, there are cases where the popular agile software development methodologies do not make a good fit. Developing systems that put human life at risk (such as the NASA space program) require a much higher level of requirements and design, and can be a good argument for a more waterfall approach. However, the values of collaboration, high bandwidth communication and rapid feedback loops, as well as a focus on quality certainly apply. And the NASA space program didn't go from zero to a man on the moon.. it iterated by starting with the X-15 rocket, and then the Mercury and Gemini programs before the eleventh Apollo mission actually landed humans on the moon.

Myth #18: There's only one way to do Agile

The Agile Manifesto consists of four values and 12 principles. It does not document implementation details. There are many interpretations of agile, including Scrum, Extreme Programming, Kanban, and Feature Driven Development, to name a few. Each style has benefits, as well as weaknesses, and one must evaluate their own specific situation to determine which interpretation is the best match. As long as you are adhering to the Agile Manifesto's values and principles, you should be considered agile.

Agile Software Engineering Myths

Myth #19: Agile has No Architecture

Because agile projects are developed in short iterations, it's a common belief that teams can't (or won't) develop architecturally sound software. This is due to the misconception that everything gets done at the last ***possible*** moment. Successful agile teams actually will wait to the last ***responsible*** moment. There are architectural decisions that are made all of the time, and they need to get done at the correct moment. Larger architectural decisions are made prior to writing any code. Finer grained decisions can wait until later in the project life cycle. But in the end, all decisions should be based on the correct architectural patterns and guidelines.

Myth #20: Agile doesn't need up front design

Just like the myth "Agile has No Architecture", there is a misconception that agile teams do all of their design work "on the fly". Design needs to be done at the last ***responsible*** moment. Some elements of design will be done long before any code is written. Other elements of design will be completed a sprint (or more) prior to when it's needed. Some design will be done as the particular code is being written. This is commonly referred to as emergent architecture and design. It emerges as it is needed – not before and not after.

Big Up Front Design tries to answer every single question prior to the questions even being surfaced. Agile teams design the big rocks first, then the medium rocks, and finally the small rocks. This process correctly identifies the questions as they become clear, therefore providing the correct answers when they are needed.

Myth #21: Agile Projects don't use Quality Assurance/Quality Engineers

Quality is one of the driving goals of all successful agile teams. The best teams incorporate Quality Engineers as an integral part of their processes. Ping Pong Testing where code is "thrown" over the wall to QA and bugs are thrown back at developers) is time consuming and inefficient. Having code tested sooner enables developers to correct bugs in code they recently developed, which is significantly more productive than waiting several weeks or even months. The ninth principle from the Agile Manifesto sums it up very nicely:

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

Myth #22: We're doing Scrum so we don't need to do TDD, Pair Programming, etc.

Scrum is an agile framework for completing complex projects. It is an excellent way to manage the processes around software development. However, Scrum only focuses on the processes and interactions. Nowhere in the teachings of Scrum does it mention software engineering practices. This doesn't mean that the founders of Scrum didn't think they were important; it was because there were other references that already dealt with these issues, such as Extreme Programming (XP). The best agile teams that are using Scrum also use the engineering practices detailed in XP. It's almost impossible to be successful in software development (regardless of methodology) without executing the core software engineering disciplines such as testing and continuous integration. After all, it doesn't matter how well a team communicates if the software doesn't work!

Myth #23: Using TDD (writing tests first) doubles the amount of work.

All too often, people think of Test Driven Development as all about the "Tests". Indeed, it is driven by unit tests, but it's really about driving the design through exercising individual units of work. It just happens that the best way to do that is through Unit Tests. A better name would be Test Driven ***Design***, since that is the true goal.

When developing in a TDD manner (commonly referred to as Red-Green-Refactor), you write a unit test that exposes a failure of your code – this could be an undeveloped requirement, or a bug that has been reported. This part is the "Red" – taken from most TDD Graphical User Interfaces that show red for failing tests. The test is written in such a way that once the feature is developed, or the bug is corrected, the test will pass – or go "Green".

Then, just enough code to make the test pass is written. This does several things, but mostly if keeps developers from writing things that you aren't going to need (or YAGNI). All too often, as developers, we add in features that we just "know they are going to ask for next". This adds unnecessary code and possible bugs into your software that are truly unnecessary, as they were never requested by the business.

During the refactor phase, additional use cases are added to the tests, as well as cleaning up the code to remove things like hard coded results, repetitive code, or difficult to read sections, for example.

It's important to consider how much of development is "typing" vs. "thinking". A significant more time is spent thinking with bursts of typing. Unit Tests capture those thoughts in code, enabling us to spend more time on the solution, and less time on the problem.

BUSTED!

Myth #24: Test Driven Development (TDD) satisfies all of your testing needs

Test Driven Development is an important part of software engineering, regardless of your chosen project management methodology. But it's only one part of the testing story. Acceptance tests, surface tests, and integration tests are all vital aspects of quality assurance. The different testing paradigms need to be coordinated, with a degree of overlap that is mutually agreed upon.

Myth #25: Pair programming means double the people to do the same amount of work

Pair programming is one of the core software engineering principles for agile teams. Pair programming isn't merely two people and one computer. There are several different ways to execute the pattern, but a common one is referred to as ping pong programming. One developer writes a unit test, and the other developer implements the code to make the test pass. Then, the second developer either refactors the unit test to add use cases or writes a new unit test, then control goes back to the first developer.

Pair programming in this way slightly reduces productivity, but as explained in the myth "Using TDD doubles the amount of work", programming isn't just about typing. Pairing helps with think time by providing a built in code review, therefore significantly increasing quality.

Even though pairing slight reduces productivity, quality is increased significantly. The silly "I can't believe I did that!" mistakes are typically caught immediately when pairing, and never get checked into the code base. Pairs are not only continually code reviewing each other as they work, but also idea reviewing as well.

Finally, pairing isn't a full-time process. Checking email, updating documentation, other non-development tasks are not pairing activities. Each developer should have their own computer, and only share one when they are pairing on development tasks.

Myth #26: Agile means two people working together on a computer.

There are several myths that can be grouped together here, including "Agile means TDD/Daily Standups/Retrospectives/etc." While these are all aspects of successful agile teams, they don't define Agile. Pair programming (described in the myth "Pair Programming results in better code" is another tool that teams use to improve quality.

Agile Startup Myths

Myth #27: One can learn Agile from a book

Agile software development isn't a straightforward recipe or simple algorithm. Like the game of chess, it is often considered fairly simple to learn, but difficult to master. It's important to read up on the different agile methodologies to get a deeper understanding of the "why" behind the techniques, but no book can ever address every issue or challenge of your unique environment or situation. The baseline prescriptive manners that are typically outlined in the books available are a great place to start, but it's more important to learn by doing it. Only with the experience of failing fast and sustainable success will you be able to evolve and adapt to achieve continuous improvement. The very first line of the Agile Manifesto sums it up very succinctly:

"We are uncovering better ways of developing software by doing it and helping others do it."

Myth #28: Agility is for free

Transforming from a traditional waterfall team into a successful agile team is not an insignificant effort. There are significant growing pains involved with such a transformation – you will not simply completely change everything about your culture overnight. It will take study, practice, and courage to make the change. The goal is to make sure that you fail fast when you try new things so you can quickly try a new approach. If you can work your way through the learning curve (hopefully with the help of an experienced agilist), you will end up much better off in the end.

Myth #29: Agility just happens.

The belief that agile just "happens" is similar to the myth Agility is for free. Chaos just happens, but agile takes work. It is a discipline like any other development discipline. To be successful transforming teams into agile takes study, planning, and work.

Myth #30: You start with a 'backlog'

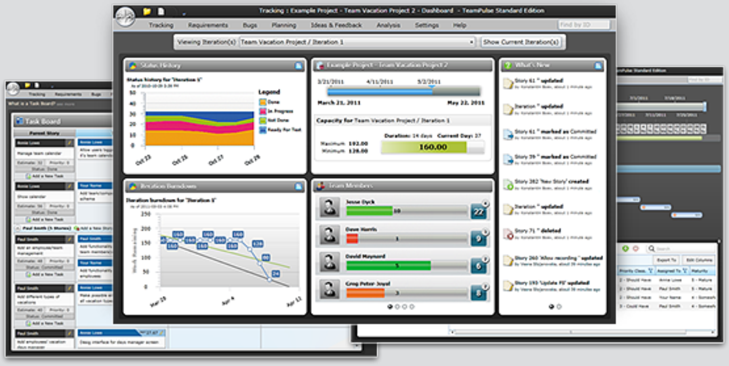
It is true that all projects must have requirements. However, there is more to becoming agile than just renaming your requirements the "backlog". Creating good requirements is hard, and there are a lot of tricks and techniques that have come out of the agile community that can assist with creating good requirements, also known as the Product Backlog. These include (but are not limited to) User Stories and Context Specification. Here are a few agile project management techniques you can actually start using today.

In addition to a well-defined backlog, you have to put the correct processes and procedures into place as well. These vary based on your chosen methodology (e.g. Scrum, XP, Kanban). But it is a holistic process, and taking a single aspect from agile and using it won't bring you all of the benefits of truly becoming an agile team.

Top 30 Agile Myths - BUSTED!
is brought to you by Telerik TeamPulse

Telerik TeamPulse is an agile project management software meant to help your team collaborate more effectively and deliver better software on time.

- Track progress across multiple projects
- Capture customer feedback with the Ideas & Feedback Portal
- Manage requirements, bugs, issues and risks
- Plan and schedule releases
- Collaborate with your entire team



Additional Resource:

Our ZeroToAgile podcast series discusses many facets of agile software development.

For a regular dose of agile information: SUBSCRIBE TODAY >>