What is Agile? Agile is a philosophy of product development that uses organizational models based on people, collaboration, and shared values. The phrase “Agile” was coined in February 2001 at a summit held in Snowbird, Utah. Attending were people who had been practicing ways of developing software that took advantage of the collaborative and creative energies of teams and the quality that comes from building learning in as one progresses. During the summit these folks defined their commonalities and created an Agile Manifesto, which details their philosophy of product development and the principles that drive Agile teams. If you haven’t yet read the Agile Manifesto that came out of this gathering, you can find it [here](http://www.agilemanifesto.org/).

While the term “Agile” originally denoted a way to deliver software products, it has in more recent years been broadened to refer to a way to deliver any product, not just software. The philosophy around collaborative work in iterative environments where teams continuously reflect and refine is suitable for many types of product delivery.

Being agile means that teams are working in ways that allow for change in order to better work together and provide a more useful and meaningful product to the customer. It’s the ability to inspect and adapt, reflect and refine, and efficiently and effectively manage the changes that inevitably occur.

What Agile Is Not

Agile is NOT an excuse to stop producing documentation. It IS a reason to examine why you are producing the document to determine if it’s truly useful and valuable or if it’s simply what you’ve always done. In Agile we question the need for the document, and eliminate producing anything that doesn’t make sense, isn’t valuable, or isn’t useful.

Agile is NOT an opportunity to eliminate planning. It IS an opportunity to institute rolling wave planning, a practice that’s documented in the PMBOK. In Agile projects, we don’t stop planning —instead we plan all the time, with the appropriate amount of detail for the time horizon that we’re planning for. Our project plan is a high-level vision with several key features, our release plans are focused on more detailed product features, our iteration plans get down to the tasks required to implement these features, and our daily plans coordinate activities, raise issues, and identify roadblocks. We can and do provide projected dates of completion and cost, using top-down planning and gross-level estimation techniques.

Agile is NOT open season on scope creep. It IS an invitation to the customer to collaborate with the team. It IS an opening for the customer to have a way to painlessly change the requirements and for the team to react accordingly. It IS a way to prioritize valuable features and work them through to completion, because we realize that eventually we’ll run out of time or money, or both.

Agile is NOT about blindly following a set of “best” practices, whether or not they’re best for your project. Agile IS about doing what makes sense, based on the agile philosophy and the given situa

# Agile Myths

Over the years several myths have formed around Agile delivery. Here are some of the more popular ones.



* [Agile is a silver bullet.](http://www.agilenutshell.com/agile_myths#silverbullet)
* [Agile is anti-documentation.](http://www.agilenutshell.com/agile_myths#antidocumentation)
* [Agile is anti-planning.](http://www.agilenutshell.com/agile_myths#antiplanning)
* [Agile is undisciplined.](http://www.agilenutshell.com/agile_myths#agileundisciplined)
* [Agile requires a lot of rework.](http://www.agilenutshell.com/agile_myths#agilerework)
* [Agile is anti-architecture.](http://www.agilenutshell.com/agile_myths#antiarchitecture)
* [Agile doesn't scale.](http://www.agilenutshell.com/agile_myths#agiledoesnotscale)

### Agile is a silver bullet

I wish this were true - but it isn't. You can fail just as spectacularly on an Agile project as you can using any other traditional method. You'll fail faster using Agile (due to the transparency and visibility it brings) but unfortunately it's not a silver bullet or an excuse to stop thinking.

There's nothing inherently magical about Agile. It basically says

Bring your development team and customer as close together as you can, give them what they need, and then get out of the way.

Now if you don't have people that like being empowered, taking initiative, and getting things done, that's a different problem. Agile just gives them permission to do their best work and be accountable for the results.

### Agile is anti-documentation

Agile isn't anti-documentation. A more accurate way to say it would be Agile doesn't do documentation for documentation's sake.

Documentation gets treated like any other deliverable on an Agile project. It gets estimated, sized, and prioritized like any other [user story](http://www.agilenutshell.com/user_stories).

Where Agile pushes back on documentation is as a means of communication. Agile prefers face-to-face communication over relying on the written word.

### Agile is anti-planning

Not sure where this one comes from. There's actually a lot of planning that goes on in Agile projects.

You've got your:

1. Daily planning with the 10 minute daily standups.
2. Bi-weekly planning with the Iteration/Sprint Planning Meetings
3. Release planning where team's decide what to ship every three to four months.

But it wouldn't be fair to say Agile is anti-planning. If anything it is anti-static planning. Meaning Agilist's expect their plans to change and use tools like burndown charts to track and make these changes visible.

### Agile is undisciplined

When Agile started gaining popularity, its reputation suffered a bit from some teams taking the easy parts of Agile (like attending daily standups) but leaving out the hard (like upfront testing and regularly shipping production ready working software).

The truth is Agile is a very disciplined way of delivering software.

* You have to test.
* You have to get feedback.
* You have to regularly ship software.
* You have to change and update the plan.
* You have to deliver bad news early.

This isn't easy stuff. It's not for the faint of heart and requires a lot of hard work, courage, and discipline.

### Agile requires a lot of rework

Rework comes in two forms on an Agile project. You've got the rework of requirements - customers discovering what they really want. And you've got the rework of the software - development teams discover better ways to design the software.

Both need to be balanced and tempered. Just as business can't indefinitely keep changing their mind, development teams can't forever keep redesigning the software. At some point we have to ship.

Agile deals with this tension by empowering both sides with the power to iterate, so long as they work within the project's means.

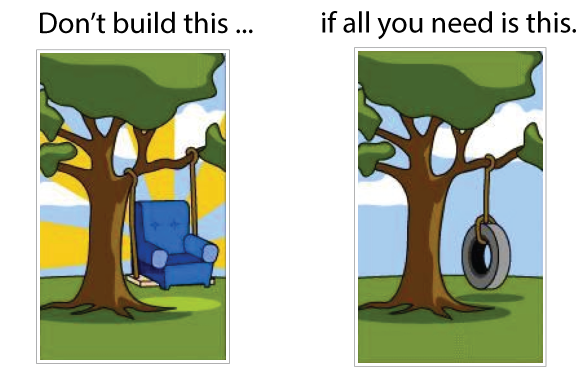
Burndown charts play in big role in tracking how Agile project are doing. Just as tools like the [Agile Inception Deck](http://agilewarrior.wordpress.com/2010/11/06/the-agile-inception-deck/) make sure everyone is on the same page with regards to time and money.

It's a balancing act not unique to software delivery. Any creative work with a deadline (i.e. plays, movies making, or the publishing of daily papers) faces the same challenges.

The trick is to do the best work you can, with the time and resources you've got.

### Agile is anti-architecture

Something we got really good at as an industry in the 1990's was building big, complex, expensive, hard to maintain systems.



Agile pushed back on this over engineering by creating terms like [YAGNI (You Aint Gonna Need It)](http://en.wikipedia.org/wiki/You_aren't_gonna_need_it) to remind teams to keep things simple until proven otherwise.

That doesn't mean Agile teams stop thinking, or don't leverage previous experiences.

It's more an attitude that the best way to build systems is to keep things simple, and only add the complexity when you need it.

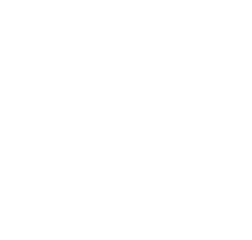
### Agile doesn't scale

Agile scales like any other software delivery process. Not that well.

Look - scaling is hard. There is no easy way to magically coordinate, communicate, and keep large groups of people all moving in the same direction towards the same cause. It's hard work.

The one thing Agile does bring to the conversation, is instead of looking for ways to scale up your project, look for ways to scale things down.

In other words, if we know we are really good at delivering with small, nimble, agile teams of ten, why don't we structure our work that way. More on this[here](http://martinfowler.com/articles/canScaling.html).

[](http://www.agilenutshell.com/)[](http://www.agilenutshell.com/about)

[Getting Started](http://www.agilenutshell.com/) [Videos](http://www.agilenutshell.com/videos) [Courses](http://www.agilenutshell.com/bootcamp)

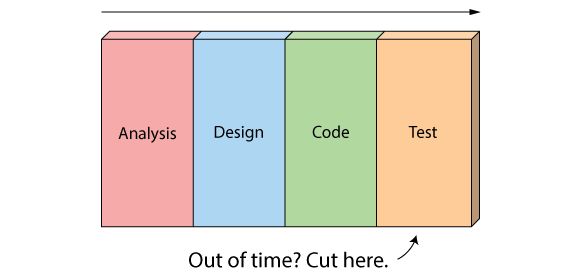
* [Learn More](http://www.agilenutshell.com/agile_vs_waterfall)
* [About](http://www.agilenutshell.com/about)
* [Feedback](http://www.agilenutshell.com/feedback)
* [Newsletter](http://www.agilenutshell.com/newsletter)
* [Blog](http://agilewarrior.wordpress.com/)

# Agile vs Waterfall

### Waterfall challenges

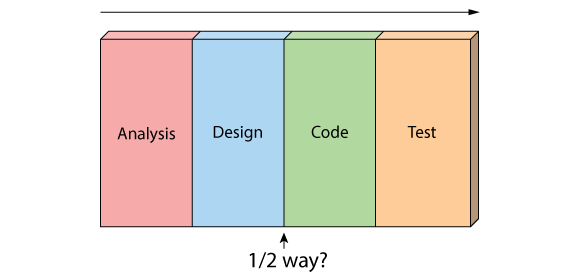
Traditional Waterfall treats analysis, design, coding, and testing as discrete phases in a software project. This worked OK when the cost of change was high. But now that it's low it hurts us in a couple of ways.

### Poor quality



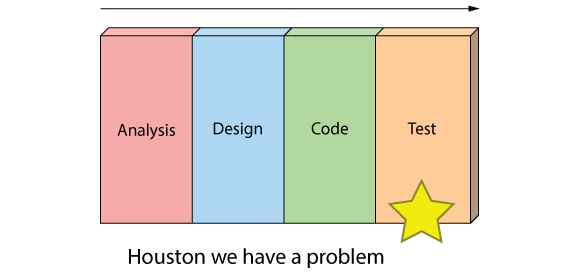
First off, when the project starts to run out of time and money, testing is the only phase left. This means good projects are forced to cut testing short and quality suffers.

### Poor visibility



Secondly, because working software isn't produced until the end of the project, you never really know where you are on a Waterfall project. That last 20% of the project always seems to take 80% of the time.

### Too risky



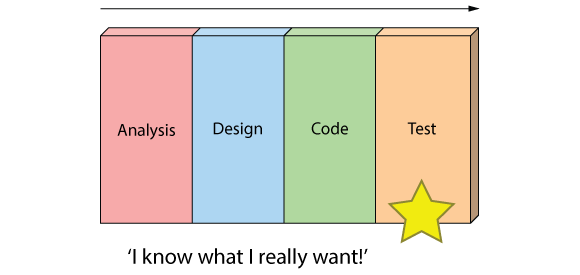
Thirdly you've got schedule risk because you never know if you are going to make it until the end.

You've got technical risk because you don't actually get to test your design or architecture until late in the project.

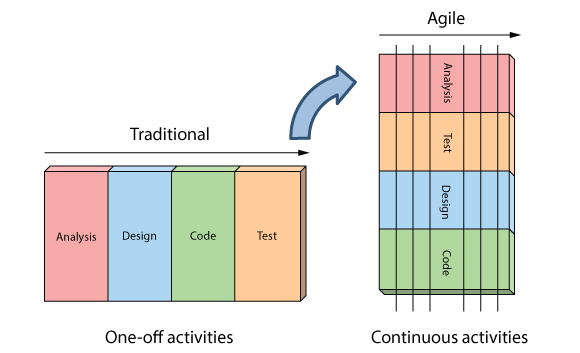
And you've got product risk because don't even know if you are building the right until it's too late to make any changes.

### Can't handle change

And finally, most importantly, it's just not a great way for handling change.



## The Agile Approach



Instead of treating these fixed stages Agilists believe these are continuous activities.

By doing them continuously:

* Quality improves because testing starts from day one.
* Visibility improves because you are 1/2 way through the project when you have built 1/2 the features.
* Risk is reduced because you are getting feedback early, and
* Customers are happy because they can make changes without paying exorbitant costs.

The six Scrum principles are:

1. **Empirical Process Control**- This principle emphasizes the core philosophy of Scrum based on the three main ideas of transparency, inspection, and adaptation. [More](http://www.scrumstudy.com/scrum-empirical-process-control.asp)
2. **Self-organization** - This principle focuses on today's workers, who deliver significantly greater value when self-organized and this results in better team buy-in and shared ownership; and an innovative and creative environment which is more conducive for growth. [More](http://www.scrumstudy.com/scrum-self-organization.asp)
3. **Collaboration**- This principle focuses on the three core dimensions related to collaborative work: awareness, articulation, and appropriation. It also advocates project management as a shared value-creation process with teams working and interacting together to deliver the greatest value. [More](http://www.scrumstudy.com/scrum-collaboration.asp)
4. **Value Based Prioritization** - This principle highlights the focus of Scrum to deliver maximum business value, from beginning early in the project and continuing throughout. [More](http://www.scrumstudy.com/scrum-value-based-priotirization.asp)
5. **Time-boxing** - This principle describes how time is considered a limiting constraint in Scrum, and used to help effectively manage project planning and execution. Time-boxed elements in Scrum include Sprints, Daily Standup Meetings, Sprint Planning Meetings, and Sprint Review Meetings. [More](http://www.scrumstudy.com/scrum-time-boxing.asp)
6. **Iterative Development** - This principle defines iterative development and emphasizes how to better manage changes and build products that satisfy customer needs. It also delineates the Product Owner's and organization's responsibilities related to iterative development. [More](http://www.scrumstudy.com/scrum-iterative-development.asp)

- See more at: <http://www.scrumstudy.com/scrum-principles.asp#sthash.8qsH1cc4.dpuf>