# Agile Culture

10/02/2017 • 4 minutes to read

**In this article**



By: Aaron Bjork

If there's anything we've all learned in the last decade of "agile transformations" it's that there is no one-size-fits-all approach to adopting or implementing an agile approach. Every organization has different needs, constraints, and requirements. Blindly following prescription won't lead to success.

At the end of the day, the agile movement is about continually finding ways to improve the approach and practice of building software. It's not about a perfect daily standup, or retrospective. Instead, it's about creating a culture where the right thing happens more often than not. Those activities (standup, retrospective, etc.) have their place, but they won't change your culture.

This article details foundational elements that every organization needs to create an agile mindset and culture. They should not be followed blindly, but rather apply what makes sense to your environment.

## Schedule and Rhythm

First off, there is no perfect sprint length. Teams have been successful with lots of different sprint lengths ranging from 1-4 weeks. What matters is consistency.

Pick a sprint length that works for your culture, your product, and your desire to provide updates. For example, the Developer Tools division at Microsoft (roughly 6,000 people) works in a 3-week sprint. The choice of a 3-week sprint wasn't made by the leadership team or managers "on high"; instead it came from direct feedback from the engineering teams. The entire division operates on this 3-week sprint schedule. The sprints have become the "heartbeat" of the organization. Every team is marching to the beat of the same drum.

Pick a sprint length and stick with it., If you have multiple agile teams, use the same sprint length across all teams. Then, listen for feedback from your teams and see if it needs to be adjusted. You'll know when you've got it right!

## A Culture of Shipping

Peter Provost said "You can't cheat shipping". The simplicity and truth of that statement is a cornerstone of agile culture. What Peter means is that shipping your software will teach you things that you can't and won't understand... unless you're actually shipping your software.

Human nature is to delay or avoid doing things until absolutely necessary. This couldn't be more true when it comes to software development – teams punt bugs to the end of the cycle, don't think about setup & upgrade until their forced to, and typically avoid things like localization and accessibility wherever possible. When this pattern emerges, teams are building up technical debt that will need to be paid down at a later time. But shipping demands all debt be paid. You can't cheat shipping. To establish an agile culture, start by trying to ship your product at the end of every sprint. It won't be easy at first, but when you do this you quickly discover all the things that should be happening, but aren't.

## Healthy Teams

There is no recipe for the perfect agile team – however, there are a few key characteristics that make success for an agile team much easier to achieve:

#### Whenever possible, co-locate your teams

Can a team find success with people spread out across different geographies? Of course. But let's be honest... it's more difficult. When people are co-located and sitting in the same room, the right conversations just tend to happen. You can have teams located across globe and different time zones. However, do everything you can to ensure a team is co-located together.

#### Keep your teams intact for a reasonable length of time

Allow them to master the art of building software together. When you adjust the makeup of a team, you disrupt the team's chemistry. There are times when it's appropriate to reorganize teams. However, teams typically work better when they're given time to learn how to work together. As a guideline, try to keep your teams intact for at least 12 months.

#### Load balance work, not people

What happens if a team falls behind? When this happens it's best to load balance work to another team, rather than load-balancing people between teams. When you pull a person off a team to help a team falling behind, you disrupt both teams and the person being moved feels jerked around. All of this impacts team productivity and, more likely than not, negatively impacts your ability to get back on schedule.

When you load balance the work to another team you allow a team that is already established to step in and help out. It becomes a priority conversation, instead of a people conversation.

## Teams should own feature areas, not layers of architecture

Strive to build vertical teams that own feature areas. These teams are responsible for all the work required to

add features to their area, from database to user interface changes. The team is empowered to deliver and own an end-to-end experience.

Horizontal teams that own a layers of architecture means no single team is responsible for the end-to-end experience. Adding a feature requires multiple teams to coordinate and requires a higher level of dependency management. Resolving bugs requires multiple teams to investigate whether they own the code required to fix the bug. Bugs are batted around as teams determine its “not their bug” and assign it to another team.

Feature teams don’t have these issues. Ownership and accountability is clear. There may be a place for some architectural based teams. However, the more you can strive to build vertical teams, the more effective they will be.

## Start your journey...

As you embark on your own agile transformation, keep these foundational principles in mind. And remember, there is no single recipe that will work for your organization. Agile transformations are a journey. Make changes and learn from them. Only then can you develop an agile culture that fits your organization.

Get started with free agile tools in Azure Boards.

Aaron is a Principal Group Program Manager at Microsoft where he drives investments in work management, agile project management, reporting, and collaboration for Microsoft’s Azure DevOps Services and Team Foundation Server (TFS) products.