Case Study - Microsoft's 16 Keys To Being Agile At Scale

Steve Denning, Senior Contributor Leadership Strategy, Forbes Magazine, 2015

Microsoft

Doing Agile and Scrum in a single team, or even a couple of teams, is one thing. Doing Agile and Scrum in hundreds of teams in a coordinated way is a whole different ballgame. How are dependencies handled? How do the teams keep in sync? How does management even know what's happening, let alone stay in control, if hundreds of teams are autonomous? Many general managers doubt whether Agile at scale is even possible. The site visit to Microsoft's Visual Studio group in the Developer Division of some 4,000 developers, as part of the Learning Consortium for the Creative Economy, shows that it is. Microsoft is not a giant warship, but more like a flotilla of speedboats operating in sync.

The Learning Consortium is a group of firms, sponsored by Scrum Alliance, that have committed themselves to learning from each other about the management practices of the Creative Economy, or, as Fortune has called it, "the new industrial revolution." Scrum Alliance itself is a membership association of more than 400,000 members with the mission of transforming the world of work. The host for our site visit to Microsoft was Aaron Bjork, a group program manager in Visual Studio Online.

1. Pursue "Agile at Scale," not "Scaling Agile" : The question addressed at Microsoft is: "How do we make the whole organisation agile?" not "How do we scale Agile or Scrum?" There is a tight focus on delivering continuous value to customers, not merely generating quarterly profits or boosting the current stock price. It also rests on a deep respect for the talents and capacities of those doing the work, and the teams in which they work, not treating workers as "resources" that are assignable, optimizable and ultimately disposable. A key aspect of the management mindset is an explicit self-awareness of the potential of managerial authority to inspire and uplift as well as to discourage and dispirit. There is an intentional, even delicate, use of managerial authority. There is a continuing effort to get the right balance between alignment and autonomy

2. Take Care of Planning and Coordination : Planning begins with an overall vision for the product. They develop a "scenario," the goal for the product for the next 18 months. It's a story of where the program wants to be 18 months from now. The group has about a 60% confidence in its ability to predict what the customers want to and to deliver that. Every six months, the group reviews progress and examine where they want to go next. There are three questions: what have we learned over the last six months based on what we built? What do our customers tell us? And what's changed in the marketplace? It's both planning and learning. Every team has the authority to make changes. If the team sees something that was missed, they don't have to ask permission to make a change. They just keep the leadership team informed. The group of teams executes for six months and then they take another look at the plan. At all times, each team maintains and owns a thoughtful and detailed three-sprint plan, each sprint comprising three weeks.

3. Get the Right Balance of Alignment and Autonomy : The goal at Microsoft in Agile at scale is to have alignment at the top and autonomy at the bottom. The teams need autonomy. That's what drives them to come to work and deliver great stuff. But at the same time, their work has to be aligned with the business. If there is too much control, nothing gets done: no one wants to work there. It's not a fun environment. In fact, it's a disaster. If there is too little control, it's chaos. Everyone is building whatever they want. There are no end-to-end scenarios. Customers are frustrated. Nothing makes business sense. So the managers are always striving for the right balance. The managers are responsible for the rules of the road. This include clarifying the roles, the teams, the cadence, the vocabulary and the limits on the number of bugs a team can have ("the bug cap."). The team has autonomy in terms of how they go about doing the work in both planning and practices. Within the overall framework, each can take a different approach. The specific engineering practices are up to the team. For instance, whether the team decides to do pair programming is up to them.

4. Master The New Role of the Manager : What happens when a team misses a sprint? A manager doesn't monitor a teams' burn-down charts. The burn-down charts are for the teams. If they get behind, guess what they do? They talk about what to do. That's the behavior the manager wants. The manager gets that behavior, because the culture supports it. If the manager yells at the team or monitors their burn-down chart, guess what the manager gets? Perfect burn-down charts. So does the manager want perfect burn-down charts or the right conversation? In the end, it has to be the latter. If the daily stand-ups aren't working, then be a grownup. Make a change! That's where the autonomy comes in. "You are in control! You are responsible!"

5. Handle Dependencies At The Team Level : In Microsoft's Developer Division, the dependencies are handled to the extent possible by the teams themselves. The teams all know what the other teams are doing. It's expected that the three-sprint plans include all dependencies. And the same process is happening in the six other groups of teams. The managers are talking about this and sorting it out on a continuous basis. Every three months, there's a standing meeting across all of the teams. It's called "a feature team chat." Every team comes in and shares their plan. Aaron has 90 minutes and his teams come in and they each have 15 minutes to share their plan. And his peers come in with their teams and they also share. This takes place across all the teams, so that everyone is aware. And the leadership team of the Developer Division also gets a chance to get sync with what's going on.

6. Ensure Continuous Integration: Continuous delivery has meant more modularity in design and a change in architecture. When they started, each team would work in a branch of the code during a three week sprint. At the end of the sprint, they would put it all together, and it was chaos. In fact, the teams had created a lot of "integration debt." That model didn't work. So to ship every sprint, they had to make a fundamental change. In principle, all the teams are now "working in the same branch.". What that means is that each of the teams is branching their changes in a program called Git. But the teams don't work in a silo by themselves for three weeks and then hope it all comes together. They come together all day, every day. So if a team breaks the build, it fixes the build immediately. They do whatever they have to do. The longer the team waits to put code together, the greater the risk of technical and integration debt—and disaster. At the end of every sprint, the team sends out an email to all 450 people in the Visual Studio Online group and the leadership team. They talk about what they accomplished that sprint and what their plan is for the next sprint. And they record a 3-5 minute video. (Warning: the videos can get fancy, if the teams have aspiring Hollywood directors.) The video replaces the sprint demo.

7. Keep On Top Of Technical Debt : "In the old days," says Aaron, "when the code had been written, the team had a party. They were celebrating. They felt they had accomplished something. But in reality, they were sitting on a mountain of bugs. In fact, they hadn't even found all the bugs. Now they had to go back and find them all. And fix them. And they were still months away from delivering software. It was a nightmare." "Now the bugs never grow. There 's a Key Performance Indicator (KPI) we call 'the bug cap.' It's the number of engineers on your team times four. So if you have ten engineers, your bug cap is 40. If you get to 40 bugs, the team needs to stop work on new features and the next sprint, get the bug count back down below 40. It's self-managing. Teams know this. It means that we can ship product all the time now, because we know we're always in a healthy state."

8. Embrace DevOps and Continuous Delivery : In this way of working, development and operations merge. The teams own the planning of each new feature. They own the execution of the feature. They own the delivery the feature. And they own the operation of the feature. So if the service goes down, then they have to stop everything and deal with it. If bugs are found or fixes are needed, they are the ones doing it. They used to have a separate support team doing that, but who wanted to spend their time fixing someone else's mistakes? The teams own the entire life of the feature. If the service is breaking down frequently, then it's a problem with the quality of the code. It's also a motivation to build great code. The teams are living it on a continuing basis. They own the features they create. No one else to blame. And they don't have the pressure of one big release. They can stage things and resolve problems as they go.

9. Continuously Monitor Progress : The teams do a great deal of monitoring how the features are being used. The results flow into the aspirational backlogs, which are called scenarios. Every month, the program manager reports out on metrics, on the accounts using different aspects of the service. So the group is learning to become a data-informed business. They don't call it "data driven" because that would run the risk of missing the big picture. They use their brain and their gut feel as well as being informed by the data. The data isn't an after-thought though. It's often the first part of the conversation. Part of the very definition of "done" is having the right telemetry. The teams see this data and monitor it both when they are testing it and as soon as it goes live. It's not something they do in the sprint after they ship it. It's part of the acceptance criteria to ship. As soon as the code ships the team asks: How are people using it? Is it driving people through our conversion funnel? Are they becoming a dedicated account? Or just a casual user? They use the metrics to drive the business forward.

10. Listen To Customer Wants, But Meet Their Needs : The program managers do all kinds of customer visits, at the highest level at the executive briefing center, which is all about strategy, in customer councils with folks who use the products. There is also a distribution list, called the Champs List. These are people who are writing to Microsoft all day long about things that they want. The program managers talk to them. The sales teams hooks the developers up with customers. The program managers talk to customers on Twitter all the time. The teams don't blindly follow what customers say. They have what they call "the cookie principle." If you have a plate of cookies and you ask people if they want one, they will say yes. No one turns down a cookie. If you go to a customer and ask them, "Do you want this feature?" guess what they say? Why wouldn't they? It's the innovator's dilemma. There's a whole bunch of good stuff out there that you could do. You need to listen, but not blindly follow. The program managers need to listen to what the customers say they want, but their job is to build them something they need. And something that the firm can sell. Otherwise the managers are working themselves out of a job.

11. Deal With Directions from Above: Aaron has never heard anyone above him say, "Stop this Agile stuff!" One reason of course is that the Agile teams have been very successful. Agile has grown and spread as a result of its success. Yet the teams do get directions from above. During our visit, one of the teams was being shifted to do something else. That's disruptive. But it's happening because there is a need for that team somewhere else. The team will stay together as a team. The mangers sit down with the team and talk about it. There is very little load balancing among teams. If a team gets behind, they don't break up the team or move individuals to the team to fix it. They ask the team itself to fix the problem. They try to keep the teams together for 12 or 18 months. That's what the teams themselves like. The goal is to let them get good at building software together. That's their job. If you're reshuffling them every three sprints, or even reshuffling the things they are working on, that makes it pretty hard to get really good. The firm is making an investment in the team for at least nine months or a year.

12. Use Self-forming Teams To Encourage Team Ownership : The managers let people choose which team to work on. People can reshuffle every 18-24 months. Around two thirds of the team members decide to stay where they are. As a result, there are not many brand new teams. But the team members have the choice. The result is a significant investment in persistent teams. Quite apart from team well-being, it leads to higher performance. The team owns the backlog. Of course, there is a lot of discussion about priorities. But a manager doesn't tell the team what should be next on the Kanban board. Nor does the team dictate to the manager. They talk about it all the time. It's a constant conversation. It's like, "Hey, what about this? Should be we doing that? Do you think we've already invested a lot there? Should we go over to here?" And the program manager is having the same kind of conversation with his manager. These conversation require a level of mutual trust. As a manager, you can't be involved in everything and you can't know everything. A manager says things to the team and the team listens but they don't blindly follow what the manager says. It's give and take. It's a data-informed conversation. That conversation goes on all the time.

13. Recognise that The Team is the Product : In software, the product life cycle is shortening. In traditional accounting, the business asset is the product. But more and more, in practice, the asset is the team that is capable of delivering products. The team has a longer life-time of generating value than the product itself. That's a big shift in focus. Microsoft has an advantage in that it had teams, long before they went Agile. There was already a strong team culture. It's more difficult for firms going Agile that don't have a history of teams. When you think about Agile, there's a lot of value, just from the fact that the work is being done in teams. So it's important to see what baseline you're starting from. Microsoft sees itself as making an investment in those people. Sometimes organisations don't recognise that investment. There's a risk that the top may think of people as just pluggable resources. "At Microsoft, it doesn't work that way," says Aaron. "The leadership team understands that." But change is sometimes necessary and it has a cost. For instance, when one team was moved, there was mourning in the other teams, because they had made an investment in working with these guys. They had worked together for the last year. It was an investment. When the team was pulled out, it was disruptive to everybody. Aaron explained the reasons for the decision and told the team: "You are not going to have the same velocity in that new space. You will have to ramp up and build expertise." But in this case, they are at least already a team, so they already have a level of trust. If it was a brand new group of people in a brand new space the cost would be much higher.

14. Build Quality From The Beginning : There was a lot of learning at the start of the Agile transformation at Microsoft. In the first sprints, there was agreement on 3 week sprints. The leadership signed off on the idea of Agile and Scrum, but they were a little worried as to how it was going to work. So they planned for "a stabilisation sprint" after five sprints. However that encouraged some teams to think: "No need to worry about bugs, because we have the stabilisation sprint!" So a whole lot of bugs were generated and all the teams had to pitch in to help fix them. In effect, they had told people to do one thing, but they created an environment that prompted some teams to do the opposite. Who could blame the teams? The teams told the managers: "Don't ever do that to us again!" It was a great example unintended consequences. Above all, the goal is to avoid the sequence: write code in the first sprint. Test it in the second sprint. Fix bugs in the third sprint. The rules of the road are: deliver finished product every sprint. It's part of the concept of autonomy. If the team can control its own quality, they're not surprised about what they're going to have to do in the future. Like working on weekends and stuff. They can take care of their own business, and not be subject to stuff they can't control.

15. Use Coaching Carefully : External coaches at Microsoft were noticeable in the site visit by their absence. There had been some coaching and basic training in Scrum at the outset. But after a while the group just started "doing it" themselves, figuring out what was working and doing more of that, and seeing what wasn't working and not doing that. Some of the Microsoft staff and managers in effect became Agile and Scrum coaches. But overall, the group itself just set out and went for it. More recently, there is a recognition that a lot of new people have come on board who didn't do the basic training. So thought is being given to doing more training. At the same time, there is a recognition that there is no "one size fits all" and that what works elsewhere may not fit the Microsoft culture.

16. Ensure Top Level Support : Microsoft's top management was cautious about the Agile transformation at the outset. But that has changed. "There is now a broad recognition," says Aaron, "that Agile is the modern way to build software. That's not too difficult at the team level. You grab ten people and the Scrum Guide, and you can do it. But how do you do it across four thousand people and stay in sync? That's the challenge. How do you do it at scale?" To achieve Agile at scale, the support of corporate vice-president, Brian Harry, has been central. Aaron has had the benefit of living in the Developer Division where Scrum and Agile practices now have a deep foothold. The Visual Studio group is leading the charge for Microsoft as a whole. It owns the "first party engineering system charter" (IES) and is driving that across the company. There are monthly scorecards on how the big divisions are doing in adopting it. It has to evolve. People need that time to let it evolve. Emotionally, it takes time. You can't make all that change at once."

Take-a-ways : When I asked Aaron for his advice to other big firms setting out on an Agile transformation, he offered the following:

1. Get good at the science of Agile and Scrum but don't be overly prescriptive
2.Don't copy others: learn from others
3. Build the culture you want ... and you'll get the behavior you're after
4. Stop trying to predict the future
5. Optimise around customer feedback