Scaling Agile across the enterprise: Case Study

A little history

Little less than a decade ago, the entire software industry could have been accused of being “slow.” By today's standards, waiting for projects to be completed could be likened to watching grass grow.

It was common to have release cycles of two to three years. It wasn't unheard of to spend six months to a year planning, a year or more coding, and then several months testing and packaging everything up for delivery. Today's environment requires a different approach. Today, we buy software daily—often with just the touch of a finger on the device we're carrying with us. Market opportunities come and go more quickly, and customers demand increasingly faster turnaround to meet their needs.

Everything we do around development — collaboration, testing, and customer feedback — has changed.

In the Developer Division of a major software organization, we needed to adjust our business to respond to these changes. The old way was no longer working. Agile methods and practices offered some hope, but how could we bring Agile to our entire division? We had hundreds of teams across thousands of engineers. You see, we've had Agile "pockets" for years throughout Microsoft, but nobody really knew whether Agile could be rolled out successfully at this scale.

We knew we needed to transform our business. The time had come to change... or be left behind. This story is about how the Developer Division at Microsoft made an Agile transformation that scaled to support business needs.

Today, we think about how fast we can translate an idea into reality, and get it into customers' hands.

Reinventing work

We started by re-thinking how we work. Our goal was to build software incrementally, deliver it to customers faster, and get feedback and input that helps make the end result better. Building software has always been a creative process and we needed to find a way to let teams "create" again. Teams needed more autonomy to think outside the box. They needed to be empowered to make decisions without being held back by the process. We needed to fundamentally change the way we work.

We needed to work more incrementally, deliver to customers faster, and let feedback make the product better.

Creating change is never easy, in fact, it's scary. We had to plan for change. And the reality is that the Developer Division isn't a battleship that we have to steer and move. It’s three thousand canoes that we have to orchestrate, turn, and maneuver.

The Agile Shift

We didn't decide we were going to be Agile starting tomorrow. There was gradual buy-in with teams and leadership.

Planning transformation

We started by taking our senior executives in the division through a bit of Agile training. We did a project where everyone had to design and build a game. At the end of every hour, participants had to play their game and demonstrate their progress. This repeated until the games were "done." It was a real eye-opener for the management team to see Agile in practice. They weren't reading about it or hearing about it from a colleague. They were doing it, and seeing the impact firsthand.

A new cadence

As an organization, we knew right away that if we were going to build in smaller increments, we needed to think in smaller increments. From that moment, our entire schedule shifted from multi-month milestones down to continuous three-week iterations. I'll never forget Sprint 1. It was our first three-week iteration, and I don't think a single team was successful. It was a train wreck. However, we took the learning from that Sprint, made a few adjustments, and got started on Sprint 2. At the writing of this article, we're halfway through Sprint 64; we've been using this approach ever since.

We did this one project where you had to design and build a game. And at the end of every hour, you had to play the game and sort of demonstrate how much progress you'd made, and then you'd go to your next Sprint and do the next hour. It was an eye-opener for the management team to see it in practice.

The new normal

A typical Sprint looks as you might expect. Day 1 is Sprint planning, where teams discuss what's next on their backlog. Teams gather to talk about each story and make sure they understand all the requirements. This can take some back and forth, but the goal is to build a plan for the three-week Sprint in a single day. The entire team works together on this.

Physical Transformation

We moved out of our individual offices, and put teams together into the same room.

Staying "shippable"

Next, we execute. Code is written and tested daily. In our daily stand-ups, teams gather to talk about what's been accomplished, what's next, and if anybody's blocked. The key here is that we're not just writing code during a Sprint. We're writing code, testing that code, and making sure the code is "shippable." At the end of the Sprint, each team holds a retrospective where it talks about what's working, and what could use some adjusting—reviewing and assessing so that the team is constantly improving.

Prior to this approach, we separated coding, testing, and upgrade tasks. Coding might happen months before testing. More than anything, the old way created a technical debt problem. The longer we wrote code in isolation, the more debt we created. The bug curve grew and grew. One of our new principles is that we don't carry technical debt forward. We pay it as we go every Sprint allowing us to remain in a "shippable" state at all times.

Celebrate

And the end of a Sprint, each team sends out an email listing all the work completed together with a quick demo video showing changes to the product. The video helps us celebrate the work and keeps everyone abreast of progress.

The New Normal

We need to make sure we're hearing our customers, and learning from them as we're building the software.

Staying Connected

The process worked well for teams, but we needed a new way to communicate with our leadership team as well. We need to stay connected and informed and our answer was something we call "Team Chats." Every three Sprints, we invite teams to sit down with leadership to answer three basic questions: Are there any issues in the way, any impediments? What's your technical debt situation? What's next on your three-Sprint roadmap? This approach allowed us to communicate across teams and make adjustments as the software came to life. It wasn't about following a plan laid out eighteen months earlier; instead, it was about learning as we built the software. Adapting it as we went.

Employee Response

We'll talk about what's working and what's not working... the idea is continuous improvement.

Measuring results

This new way of working is very cyclical. We write something. We give it to the customers. We ask them what they think. It's a continuous and steady buildup toward the final product, instead of a big push all at once. The software is developed based on an ever-evolving understanding of what the customer wants. This helps us get to the end result faster and helps us get it right—thus eliminating waste. For the team, it's very rewarding.

A new mindset

We also adopted a mindset common in the Agile community—"fail fast." A Waterfall approach to software development gives you only one shot at getting things right. If you try and fail, there's very little time to make adjustments. Agile is different. It's expected that you'll get some things wrong, but it's also expected that you'll learn from your failure... and that the learning will lead you to a better end result. Teams can run an experiment—try something for a Sprint. See how it goes. If it doesn't work, well, it was just a Sprint.

Measuring Success

We need to know that what we're building scales for software teams around the world.

It’s fun

All these changes have helped us deliver a better product, but they've also directly contributed to a boost in morale. Employee satisfaction is a success measure. In the old world, a team might have to wait a year or more before a customer exercised the code it had written. With a more frequent delivery cadence, it's now just weeks between checking something in and having it actually show up in production. That's rewarding. Once we switched to this rapid cadence, teams never wanted to go back.

We need to self-host and "dog-food" what we're building so that we know it works for different types of teams.

New digs

In June of 2013, we took an even bigger step on our Agile journey by changing our physical environment to support the changes we were making to our process. We moved the majority of our division into a building completely built around team rooms. This new building didn't look like the other buildings most of us were used to. Individual offices were no more, replaced by "team rooms" designed for a cross-functional team.

Collaborating

In a traditional building, each team member has a private office. In some cases, the development team, program management team, and test team for the same project are all in different buildings. This isn't ideal, so we completely redesigned the building to reflect the Agile mindset. We wanted teams of 10 to 15 people to be physically together, working across disciplines—program management, development and test—all in one space. Standups happen in the room instead of having to gather in a hallway or a pre-scheduled conference room. Desks are even on wheels so teams can re-organize as needed without overhead. The space adapts to fit the team.

Each team room is also fitted with a conference room and focus rooms for private conversations. It took some time for everyone to acclimate to this new environment, but we've found that the physical changes contribute to helping us build better software. Teams love it.

Before you leap

It's important to understand that our Agile transformation has been a journey and not a leap. Many organizations want to be Agile tomorrow, but it doesn't happen overnight. Following are a few "rules of the road" we developed along our journey:

Takeaways

Your software development efforts have to aid your business. That’s the point of it.

Take it slow. We didn't decide one day to flip a switch and have everyone start doing things differently. We literally took three years and went team by team, onboarding them when it made sense to do so.

Business needs. In a large organization, whether it's our Developer Division or any Fortune 500 company, software development exists to aid the business. That's its purpose. Don't try to be Agile just to be Agile. If it's not aligned with a business need, it's not going to work.

Persevere. We've found that Agile practices are the best way to achieve maximum ROI. Don't give up after your first failure. Let that failure inform your next steps, and then get back to it. Even if your first few Sprints are disasters, hang in there because it’s a big shift.

Listen. Ultimately, customers are the ones who will tell you if you're doing it right. You might have a brilliant process, but if you can't sell what you're producing, then you've got a bigger problem. Listen to what your customers are saying. What we discovered on our journey is that the product we're producing is better. Our customers tell us that, and we can see it.

Questions:

1. Can you identify the major milestones in the three-year journey that is described for this Agile transformation?
2. How did the team align with the business needs? Specifically, which business needs were addressed by using Scrum?
3. Can you identify what the failures were in the journey and how they were overcome?
4. Can you describe the mechanisms and metrics that can be used to capture customer feedback?

Case 1 – Untrained Scrum Master

Notes from a Scrum coach observing a Scrum team go about the Daily Scrum.

Nine developers were working at their workstations—five clustered in the center of the room and a pair at each end of the room. From a structural point of view, this was an open work area for the entire team.

At this meeting, the Scrum Master kicked things off by pulling out a list. Reading from the list, he proceeded to go around the room, asking each person present whether he or she had completed the tasks from the list. He asked questions like, “Julia, did you finish designing the screen I gave you yesterday? Are you ready to start on the dialog boxes in it today?” Once he had exhausted his list and spoken to everyone in the room, he asked whether the team needed any help from him. The team members were all silent.

Questions:

1. Can you identify what was good (and not good) about the team, the setup, and the meeting?
2. How would you conduct this meeting differently?

Case 2– Uncommitted Scrum Master

An organization was going in for a Scrum implementation. The Scrum Masters were to attend a training and start on the implementation the very next week. A coach was appointed to help the teams get started and clarify any questions. The coach also had a detailed discussion with the Scrum Masters and senior executives about Scrum, where it was to be implemented and what the benefits would be.

The day the implementation was supposed to start, Chris (the Scrum Master for the team) sent a member of his staff (Mary) for the kick-off meeting. Chris had an offsite meeting, and he wanted the kick-off to happen as planned and had deputed Mary for the task. The coach had a word with the executives about this. They decided to promote Mary to the Scrum Masters role.

Questions:

1. Why was Mary pitch-forked into the role that Chris was supposed to play? Which expectation of the Scrum master role did Chris not put into practice?
2. What do you think are the other expectations from the Scrum master?