Creating a culture of modern engineering within Microsoft IT

To shorten feature and service development cycles and to respond to internal customer needs faster, Microsoft IT has started on a journey to modern engineering. We’re becoming more agile as an organization, adopting a DevOps culture of close cooperation and joint ownership of services. We’re shifting our mindset, changing our organizational structure, and starting to merge team roles. We’re also learning how to better focus on the customer experience. These changes support our new agile engineering practices and make us more responsive to our customers.

Shifting our mindset

Our development teams—sprint teams—consist of engineers, product managers, and business customers. They work together to define and develop solutions for internal business customers at Microsoft. Modern engineering requires great teamwork and close collaboration. Creating the organizational culture needed to support this teamwork isn’t done all at once. It takes time to change the way people typically think and function. The mindset of people within development teams and at the leadership level must undergo a major shift. We’re in the process of making this shift.

Changing from a waterfall development model to agile development is more than simply replacing one set of processes for another. Being agile is as much about a new mindset and continuous critical thinking as it is about new processes. As customer needs shift, a team must change what it delivers. In this environment, results are more important than the process.

Teams no longer spend weeks or months designing and planning a monolithic solution, and then even more months developing it. Instead, new feature requests are prioritized in a backlog and are coded during two-week sprints. New code is integrated into production builds on a regular basis (the technical requirements for this are covered in Adopting modern engineering processes and tools at Microsoft IT).

In this environment, both team members and management must become comfortable with ambiguity and making—and quickly fixing—mistakes. We must quickly adapt to new requirements and shifting targets. Each team must be autonomous and improve steadily. What was a best practice on one project becomes a starting point for developing a new best practice in the next project. Nothing is ever static, even goals.

Also, each one of us must share equal responsibility for quality. Waiting for a particular person to resolve an issue wastes valuable time. The first available person should be able to take care of it. Ideally, each team member has the background and skills to handle any issue that comes up. We haven’t reached this desired state yet, but we’re working steadily toward it.

Realigning the organization

To more agilely assign people to projects, we aligned the organization to business processes. Team members are now organized according to their function, such as software engineering, product management, and testing. Leaders manage their respective disciplines and assign people to portfolios and projects as needed to meet demand for their skills.

Evolving engineering roles

We’re in the process of evolving engineering roles so that team members share tasks and responsibilities. The goal is to make each engineer aware of the issues that other engineers face so that they can cooperatively resolve problems. When the evolution is complete, there will be no difference between software engineering (development) and service engineering/IT (operations) roles. Ideally, any team member will be able to function in any role on their sprint team, in a team structure that’s commonly called DevOps. There are four levels in the DevOps evolution, each of which represents a new state of maturity as operations and development/test functions gradually merge.
Level 1. Develop trust and understanding
Before beginning the evolution to DevOps, service engineers (SEs) and software engineers (SWEs) don’t understand one another’s jobs. In Level 1, SEs and SWEs begin to learn about each other’s worlds through job shadowing and cross-training. They gain understanding of one another’s roles and develop empathy for the problems that the others face.

Figure 1. Develop trust and understanding

Level 2. Shared goals and priorities
At the beginning of Level 2, SE and SWE roles are still differentiated, and there isn’t much cooperation yet. The task of Level 2 is for SEs and SWEs to learn more about each other’s roles and how their skill sets differ. They learn to step in to help each other wherever they can and consult together to make decisions. They also share a common backlog.

Figure 2. Shared goals and priorities
Level 3. More DevOps-like roles

In Level 3, SEs and SWEs begin to move into one another’s roles. They proactively own and drive common decisions and cultivate new skills. They move in and out of roles seamlessly and are accountable for results. Teams invest in common systems and tools to be proactive and manage services.

![Figure 3. More DevOps-like roles, less service engineer/software engineer](image)

Level 4. Merged roles

In Level 4, there are no separate operations and development roles. Everyone is part of the same team. Everyone is accountable for service delivery; a bug is no longer a software engineering problem, and a server that’s down is no longer an operations problem. Whomever is available does the work. The team has a single goal, and the customer is their primary focus. Individuals step in and out of functions seamlessly and accept accountability. There are no hand-offs, only common tools, and all team members have full visibility into processes and projects.

![Figure 4. Merged roles](image)

The extra time required for a team member to learn a different role, in addition to their regular duties, can slow the process of merging roles. Different teams are currently operating at different levels, generally between Levels 2 and 3.
Connecting with customers
A relentless focus on customers and partners is an important goal in our journey to modern engineering. We’re learning to prioritize the needs of our customers ahead of delivery processes, technology, and project mechanics. We’re starting to interact more with our customers and gain a deeper understanding of their needs. We’re trying to use our technical expertise to point out opportunities for increasing the value of the services we offer, even if it means stopping or redirecting a software development project. We’re also working to make our projects advance the overall capabilities and maturity of services and measurably increase their value to customers.

The new, customer-focused culture is making us curious about how our software performs and if it meets customer needs. This attitude helps us make improvements that increase their satisfaction. Also, our work is having more impact on the business as it more closely matches customer requirements. We’re responding to requests from customers more quickly and giving them the features they want, faster. And software issues are more quickly resolved because different talents within the team are more easily brought to bear on them.

Best practices
Moving to a modern engineering culture requires a broad effort across multiple disciplines involving major shifts in mindset and traditional leadership approaches. While making this shift, we’ve learned some valuable lessons.

Build cooperation within teams
The traditional way of managing a team by measuring the performance of each member tends to make team members compete with each other. For agile development to work well, it’s essential for team members to build upon each other’s work rather than compete. To help build cooperation, evaluate each team as a whole according to its success at achieving team goals.

Encourage teams to solve their own problems
Teams should be empowered to resolve any problems on their own. If escalation is necessary, then team members should do their best to present a solution as well as the problem. Team members should be encouraged to use data to develop insights and make sound decisions. Their goal should be to overcome roadblocks and deliver value faster.

Use a phased approach to culture change
It’s unlikely that an organization will be able to change to a modern engineering culture it in one big push. A phased approach that moves through a series of intermediate goals toward the ideal, agile culture is more likely to succeed.

For more information
Two additional case studies describe this change. Moving to modern engineering is an overview. Adopting modern engineering processes and tools at Microsoft IT gives details about the agile development processes and infrastructure that we’ve set up.

Microsoft IT Showcase
microsoft.com/itshowcase

© 2016 Microsoft Corporation. All rights reserved. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.