Moving to modern engineering

Microsoft IT develops and maintains partner solutions. For example, when Microsoft business managers want to create a new partner incentive program or add features to an existing one, we develop the new service or feature and put it into production.

We were using a traditional waterfall development process, and our business customers often waited three months for a new solution to be released or a bug to be fixed. Also, we couldn’t change focus and respond quickly to high-priority requests. With the quickening pace of change in the services, software, PC, and devices business, this release tempo wasn’t keeping up with business needs. The longer it took to fulfill a solution or feature request, the greater the lost market opportunity, and Microsoft was missing out on many potential sales.

To respond faster to customer and business needs, we moved to a modern engineering model. This model has two components. First, people and culture maturity increases efficiency by merging development and operations roles—DevOps. This way, any engineer can perform any task on the team. Second, service maturity uses agile development principles, methods, and tools to further shorten cycles.

Agile software development breaks large, complex projects into increments that can be delivered in weeks rather than months. It uses tools and infrastructure that speed development and empower cross-functional teams to quickly iterate solutions and solve problems based on monitoring and other real world data. It includes customer stakeholders on the team to ensure that they’re satisfied with the end result.

Today, it doesn’t take us three months to release a new feature or partner incentive program. With the flexibility and speed of the modern engineering approach, high-priority projects are released in as little as two weeks from inception. Ultimately we plan to deliver incremental updates on a continuous basis—to release every day.

Figure 1. The modern engineering maturity model

People and culture maturity

As a first step in the move to modern engineering, we realigned the organization to map to business processes. This removed organizational barriers so that the right resources could be assigned to each project. Then we started the process of merging software engineering (development and test) and service engineering (operations) roles into an agile DevOps team. The goal is to make each team member aware of the issues that other roles face so they can cooperatively resolve problems. This increases both the efficiency and effectiveness of the team. Ultimately, any sprint team member will be able to perform in any engineering role.

We identified four levels of people and culture maturity in the progression from traditional to modern engineering. Moving through the levels, operations and development/test functions gradually merge.

- **Level 1—Trust:** Software engineers (development and test) and service engineers (operations) learn more about one another’s roles. As they begin to understand each other better, they develop greater patience and trust.
• **Level 2—Shared goals:** Roles are still differentiated. Service engineers and software engineers continue to learn about each other’s roles and begin sharing a common backlog. They begin to step in and help each other where possible.

• **Level 3—Role sharing:** Roles begin to merge. Service engineers and software engineers start using common systems and tools and cultivate new skills so they can begin stepping into one another’s roles.

• **Level 4—Fully integrated roles:** Finally, there is no difference between service engineers and software engineers. Everyone is part of the same team with common goals. The customer is the focus. There are no more hand-offs, only common tools, processes, and projects.

Our engineering teams are progressing up these levels, albeit at different speeds. Most teams currently are in Level 2 or 3. One hurdle to moving up a level is the time it takes the team member to learn a new role while maintaining responsibility for their daily tasks.

### Services maturity

To keep quality high with fast delivery cycles, we began to adopt a second, necessary aspect of the modern engineering maturity model: services maturity. Services maturity results when appropriate practices have been followed to ensure that both business customers and end users have a great experience and get the results they want.

We are focusing on these eight pillars and associated principles that support services maturity. They provide a conceptual framework for a set of methods that put these principles into action during the development process.

### Pillars and principles

Each pillar represents an area of software development focus. The principles define goals for the focus area.

*Table 1. Engineering pillars and principles*

<table>
<thead>
<tr>
<th>Pillars</th>
<th>Principles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Services foundation</td>
<td>Rather than focus on the next generation, constantly look at how to improve the service and make sure that the current, live site is healthy.</td>
</tr>
<tr>
<td>Analytics, monitoring, diagnostics</td>
<td>Use telemetry and data to form insights on the user experience, system health, business value of the service, and to support and implement automation.</td>
</tr>
<tr>
<td>Availability</td>
<td>Build resilience and fault tolerance into each service.</td>
</tr>
<tr>
<td>Security, privacy, and standards</td>
<td>When embracing the new culture, integrate security and privacy into the development process.</td>
</tr>
<tr>
<td>Performance and scalability</td>
<td>Set targets for excellent performance, design to meet performance targets, and monitor performance in real time. Design each component of the service to scale dynamically and independently, on mixtures of old and new commodity hardware, accounting for geographical distribution of servers.</td>
</tr>
<tr>
<td>Deployment practices</td>
<td>For continuous integration, implement monitoring, controlled flighting, and controlled production tests. Put code into production quickly, but in a way that limits the impact of issues.</td>
</tr>
<tr>
<td>Operability</td>
<td>Deliver all service components at high quality, including peripheral systems. Respond quickly to customer needs and requests. In incident response, work with urgency to restore and maintain live site service health and continuity.</td>
</tr>
<tr>
<td>Core engineering</td>
<td>Focus on engineering fundamentals during development and deployment. The application portfolio may change, but engineering practices remain the same.</td>
</tr>
</tbody>
</table>
Processes and tools
To shorten release cycles, we adopted agile development processes and set up supporting infrastructure and tools.

Development process
An important part of the new, agile development process is iterative design. Instead of using the traditional, slow process of creating detailed plans up front, the iterative design process uses rapid prototyping to validate and refine design choices.

We also devised a set of agile development methods, called Engineering Fundamentals. These are principles that our engineers adhere to in their work. The fundamentals are embedded in Visual Studio Team Services as epics, benefits, and acceptance criteria that engineers inherit in their sprint backlog. The fundamentals help speed up development by defining practices like: enabling rapid, on-demand provisioning of development and test environments; unattended deployment; automated testing; and secure and compliant services.

Infrastructure and tools
We set up an infrastructure to run agile projects, which includes:

- **Moving to Visual Studio Team Services.** We moved existing work items and code to Visual Studio Team Services, and all new projects use this tool. We use the GIT repository for source-code management, so we can build solutions faster.

- **Reducing the Dev-Test environment footprint.** We were using static environments with physical servers that took a lot of time to maintain. Today, we deploy environments using a Microsoft Azure PaaS solution and/or virtualized environments using Azure IaaS. Now we can dynamically deploy an environment on demand.

- **Increasing build and test automation.** Automation cuts the time required for each build and test run.

- **Creating a pipeline for continuous integration and continuous deployment (CICD).** Visual Studio Team Services Release Management provides a pipeline that will let us release production-ready code every day.

Modern engineering in action

*Figure 2. Modern engineering in action*
Putting the modern engineering model into action has already yielded significant benefits to Microsoft. These benefits include:

- We meet business customer demand with faster release cycles—delivering new features in as little as two weeks.
- Issues are corrected quickly, rather than making customers wait for the next major release.
- Updates and enhancements to the application portfolio are delivered continuously, so the business value of development efforts is realized much faster than it was before.
- Breaking down releases into smaller chunks has reduced risk because features represent only two weeks of effort rather than several months.

In addition to faster release cycles, a cultural shift in the way engineers view and interact with customers has also led to improved customer satisfaction. Engineers are now more curious about how the software is performing and if customers are satisfied with the features. To improve software and make our customers’ lives easier, engineers are learning how to translate customer problems into the backlog and prioritize them.

For more information

**Microsoft IT Showcase**

[microsoft.com/ITShowcase](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.