Visual Studio Team Services: DevOps and agile methods in practice

To support our customers and partners, Core Services Engineering (CSE, formerly Microsoft IT) must respond more quickly to evolving business needs. Taking six months to deliver an application or an update is no longer an option. We need to deliver value faster and more efficiently. To increase our responsiveness, we started the journey to modern engineering. Our goal is to release new functionality every day in a process called continuous integration, continuous delivery.

Today, it doesn’t take us three months to release a new feature or sales and marketing program. With the Visual Studio Team Services and 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 for our solutions using Visual Studio Team Services on a continuous basis—to release every day.

Moving to modern engineering

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.

![Modern Engineering](image)

**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. Our four levels are:

- **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.

**Applying the Engineering Fundamentals**

Each sprint team applies the Engineering Fundamentals to each project. The modern engineering approach is a fundamental cultural change to the development process, rather than a new set of items on a checklist. This encourages engineers to think not only about what they’re delivering, but also about how they’re delivering it, and how it will best serve the customer.

Engineering Fundamentals serve as requirements, and each fundamental has acceptance criteria that features inherit. Sprint teams make sure that the service they deliver adheres to the fundamentals. New employees use them as guiding principles to learn how to do engineering in our organization. And the Engineering Fundamentals are embedded in Visual Studio Team Services—where the engineers do their daily work—so they’re easy to find and use.

**Continuous delivery**

*Focus on minimal viable product.* Our engineers adopt a start-up mindset. They focus on developing a minimal viable product so that new code features are delivered in a way that we can collect the maximum amount of learning based on customer feedback.

*Dynamic and on-demand environment provisioning.* Engineers can, at any time, provision an environment that includes all of the prerequisite components and dependency services, so that when code is deployed, it can run the designated functions.

*Continuous integration.* Engineers can, at any time, start an unattended build for a check-in that produces a functional build in minutes, which is deployable to any environment without competing for resources. The integration environment provides a live view of the solution that can be used to demonstrate the new features and drive subsequent code iterations and feature development.

*Continuous validation.* Engineers can, at any time, start an automated validation process for a deployment that increases release readiness without manual effort, in minutes.

*Continuous service deployment.* Engineers can, at any time, start an unattended deployment process for a build that takes minutes in a functional environment without competing for resources.

**Customer focus**

*Safe testing in production.* Engineers can, at any time, start tests to learn from experimentation or prove service health.

*Know the stakeholders deriving value from the service.* Engineers keep a relentless focus on stakeholders: users, customers, and partners. This helps them communicate requirements and translate them into features, user stories, and tasks that produce an output that delights the customer.
Core engineering practices

**Ready to code.** Engineers can, at any time, check out code to any development environment (local or remote) to compile, run, debug—all within minutes.

**Ready to build.** Engineers can, at any time, provision a development environment with all of the prerequisite components, so that code can be checked out, compiled, and run.

**Componentization.** Engineers use componentization to make the code base easier to build, compose, and deploy. Use feature flighting to enable new features and code changes to be turned on or off independently from the rest of the solution.

**Security and privacy compliance.** All services are highly secure and comply with security and privacy standards. Engineers integrate security infrastructure and tools into the continuous delivery process/pipeline.

**Service health, analytics, and monitoring.** Engineers use telemetry and data to form insights about the user experience, system health, the business value of the service, and to support automation.

Modern engineering in action

![Figure 2. Modern engineering in action](image)

**Development process**

An important part of the modern engineering process is iterative design. Instead of using the traditional process of creating detailed plans up front, the iterative design process uses rapid prototyping to validate and refine design choices.

Engineering 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.
• **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 (platform as a service) solution and/or virtualized environments using Azure IaaS (infrastructure as a service). 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.** Visual Studio Team Services Release Management provides a pipeline that will let us release production-ready code every day.

### Using Visual Studio Team Services for modern engineering

Visual Studio Team Services (VSTS) is our default solution development environment. Hosted in the cloud, VSTS gives us a globally available and resilient shared platform for solution creation and production. We use four separate environment scenarios in a typical solution build. Our environments each represent an important part of the development, test, and release process. Our scenarios are:

• **Development.** The development environment varies from solution to solution, but this is the initial environment in which our engineers create their solution. When we host our development environment in Visual Studio Online, we benefit from many built-in features:
  - **Backlog.** We can track features, user stories, and work items with backlogs. They correspond to the overall project plan, and provide a prioritized view of the entire project.
  - **Kanban boards.** We use Kanban boards to track requirements across sprints and allow us to monitor development workflow and how out team works. Using Kanban boards, we can transform our backlogs into a visualization of the project as a whole.
  - **Linked code.** With VSTS, any code changes can be linked to a user story, bug, or task. It gives us a traceable path through our code base and visibility into how the code base for a solution is evolving.
  - **Git repository.** Git stores our code repository, which enables code sharing and collaboration. This increases code reuse, consistency, and transparency.
  - **Visual Studio gated check in.** We use the Visual Studio gated check-in model to enforce a check-in policy to ensure that code is validated by at least two code reviews and a successful build before the code change is checked-in.
  - **Visual Studio build system.** We use the Visual Studio Team Services build system to enable continuous integration by automating build integration using hosted build agents.
  - **Visual Studio Release Manager.** We use Visual Studio Release Manager to automate the testing and deployment process to enable continuous deployment.

• **Integration.** The integration environment is critical to the agile development method. It gives our developers a live environment where they push a sprint release to the next stage. In integration, our engineers place a minimum viable product: a functional yet potentially incomplete version of their solution that can be demonstrated to stakeholders and shared with team members for feature testing, functional reviews, or readiness for pre-production.

• **Pre-production.** We use the pre-production environment primarily for beta testing and feature pilot release. In this environment, we can provide a solution experience that mirrors production, but that is released only to select users or groups for final testing.

• **Production.** The production environment is just that: our solution made available for consumption by all users, supporting our business. Getting releases to production from pre-production is a straightforward process that further enables our sprint releases and continuous delivery goals of daily standups and a customer-first mindset.
Realizing the benefits of modern engineering

Putting the modern engineering model into action with Visual Studio Team Services has already yielded significant benefits to CSE. 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 whether 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

http://www.Microsoft.com/ITShowcase

Team Services Continuous Integration
Team Services Agile Tools
Team Services Complete DevOps Solution

© 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.