Microsoft Visual Studio's Journey to Continuous Delivery

Vibhor Agarwal - Microsoft India Pvt. Ltd.

30th Dec 2013

Abstract

Microsoft Visual Studio is now releasing at a much faster pace and the Team Foundation Service has a deployment every sprint. We adopted agile practices across the board and made significant improvements to engineering processes and systems down the way. What are the impediments to agile and how you overcome them. How do you reimagine the role of developers and testers in this new era. What kind of tools do you need to make this transition a success for your team ?
Come and learn what it takes to adopt modern processes to ship complex products like Visual Studio with engineering teams spread across the globe to ship at cloud cadence.

In this case study, I talk about the journey that Microsoft Visual Studio has taken to transform itself in this modern world of continuous delivery and embraced the rapid cadence for both its on-premises and cloud-based product. In this report, I discuss the three waves of improvements that we undertook, each of approximately two years. The first big change in our practices was the transition from the release of Visual Studio 2005 with .NET 3.0 and Visual Studio 2008 with .NET 3.5 , where we focussed on the reduction of waste and trustworthy transparency. Second came from Visual Studio 2008 to Visual Studio 2010 where we emphasized flow of value. And most recently from Visual Studio 2010 to Visual Studio 2012, where we shortened cycle time and embraced cloud cadence.

Visual Studio 2003 – Visual Studio 2005:

Developer Division within Microsoft provides tools and frameworks that support many different Microsoft product lines. Many of the Developer Division's products, such as .NET Framework, Internet Explorer's F12 tools, and Visual Studio Express are free. They exist not to make money , but to make it easier for customers to develop software to target Windows, Windows Azure, Office, Phone and other Microsoft technologies. Other products such as the rest of Visual Studio product line and MSDN (Microsoft Developer Network) are commercial and typically licensed by subscription. An obvious tension exists among the business goals and based on our customer interactions, these sort of tensions among conflicting priorities are quite common elsewhere as well.

In 2003 , prior to availability of Team Foundation Server (TFS) , the division had no way to look at its investments as a single backlog or portfolio. The primary portfolio management technique was head-count allocation. This resulted in "dysfunctional tribalism" characterized by five behaviors :-

1. **Don't ask, don't tell**: There was an implicit convention that no manager would push on another's assertions, in order not to be questioned on his own.

2. **Schedule chicken**: Scheduling was a game of who blinked first. Each Product Unit keptan invisible assumption that it would be able to catch up during the other team's slippage.

3. **Metrics are for others**: No Product Unit saw the need for itself to be accountable to anymetrics although accountability was clearly a good idea for other guys because they were slipping the schedule first.

4. **Our customers are different**: Because Developer Division has such a broad product line, with many million users of Visual Studio and .NET, it was very easy for Product Unit to claim different sources of customer evidence from another and to argue for its own agenda.

5. **Our tribe is better**: Individuals took pride in their individual Product Units, and Product Unit Managers went to great lengths to reinforce Product Unit's morale.

This resulted in a significant "waste" in terms of high bug backlog. The uneven product quality implied by high bug-count created its own moral hazards :

- It makes the Beta ineffective as a feedback mechanism . Customers see too many broken things to comment on the feature experiences

- Internal teams see others' bug backlogs and take comfort on each others' numbers, thus driving a wrong behavior.

- Teams end up building newer features without fixing the fundamentals and completing the existing features.

- The end dates of the product is very hard to predict. No one knows how much of the quality issues still remain in the system and any kind of projections are misleading. The result is that the features get shipped based on dates rather than when they are completely done.

Not surprisingly, we experienced significant schedule slippage in 2005 release cycle and by the time we shipped, had very uneven morale.

Improvements after 2005 – Reduce waste and increase transparency

So what did we do differently the next time? Broadly we put seven changes for the next release cycle:

1. **Get clean, stay clean**: Prior to start of any product work, we put a new milestone for quality (MQ). The two main areas of debt we addressed here were bugs and tests reliability. There was a regular drive to fix more bugs and triage out the lower priority issues. At the same time, there was a drive to increase test automation and fix the reliability issues in test frameworks.

2. **Integration and Isolation**: When developing a complex product like Visual Studio, a constant tension exists between the need of feature crews to be isolated from other teams' changes and the need to have full source base integrated so that all teams can work on the latest code. To solve this, we allowed feature crews to work in isolated branches and then to integrate with closely related crews and then to integrate to main. We defined automatic gates or criteria to promote the code from one branch to another. This was challenging initially but once set-up the process worked quite effectively.

3. **Product Backlog**: We took a holistic and consistent approach to product planning. We introduced a structure of functional product definition that covered value propositions, experiences and features. For each level, we used a canonical question to frame the granularity.

4. **Iteration backlog**: Well-defined features were coarse grained enough to be visible to a customer or consumed by another feature and fine grained enough to be delivered in a sprint.

5. **Engineering principles**: We reduced technical debt and put in place rules and automation to prevent deferral of work. Small feature crews and short time boxes kept work in process low. A consistent definition of 'done' coupled to the correct branching and automation kept codebase potentially shippable. Automated testing was used widely and exploratory testing was used selectively where scenarios were not ready for automation.

The results were impressive. Unlike VS2005, there is no overhang of deferral and the reduction in debt was greater than 15x. At the same time schedule from beginning of the release work to general availability was half as long. And the transparency of the process allowed reasonable engagement of stakeholders all along the way. Many of the practices that we applied internally became product scenarios, especially for Team Foundation Server product. There was also a wider acceptance for the changes that were introduced by the team members.

Improvements in Visual Studio 2010 – Increase Flow of Value

Having reduced technical debt, we could now focus on flow of value. As we entered Visual Studio 2010 product cycle, we laid out several ambitious scenarios based on customers' requirements and what we discovered through our own usage.
Developer Division recovered and in the end Visual Studio 2010 has been the best release of the Visual Studio product line ever. The progress was not linear. We learned several engineering lessons from the sloppy start in 2008 and skipping MQ in particular.

1. **Planning and Grooming the product backlog cannot be skipped**: If you don't have a backlog that provides a clear line of sight to customer value, all prioritization decisions seem arbitrary.

2. **The Backlog needs to ensure Qualities of Service**: As particular oversight was the lack of suitable requirements in the backlog around the fundamentals such as performance and reliability and lack of clear product ownership of these. Fortunately we recovered by the time of release to manufacturing (RTM) but had we set the fundamentals early, established the ownership and put in place the system of instrumentation and transparent reporting, we would not have had to pay for the recovery.

3. **One team's enhancement is Another breaking change**: We had put a clear definition of "done" in place for previous release but our integration tests were not acting as safety net. As a result we had significant friction around code integration.

4. **Test Automation needs Maintenance**: We had not automated our integration tests fully and they were not finding the important problems fast.

The overriding management lesson was to celebrate success but not declare victory. We forgot the pain of Visual Studio 2005 release and after the success of Visual Studio 2008, we decided to skip MQ, neglect our backlog and underinvest in our engineering processes. We have since recovered but with the reminder that we have to stay vigilant and self-critical.

The Path to Visual Studio 2012 – Continuous Delivery

Visual Studio 2012 wave represents the third major phase of improvement for Developer Division. We had worked on reducing waste, increasing trustworthy transparency and expanding flow of value. Once you have mastered these three principles of Agile Consensus, the next challenge is to get better at expanding continuous flow. We believe there are two key actionable metrics here – cycle time, how long it takes to convert an idea from product backlog into working software in customer's hands and mean time to repair (MTTR), the interval from an unwanted event in production to the root cause being fixed and the service re-deployed and in use by the customer.

During the development of Visual Studio 2012, the teams worked in synchronous sprints of three weeks. At the end of each sprint, we would update our "pioneer" server of TFS and new builds of the client IDE (Integrated Development Environment) were available daily intra-sprint. Because of the earlier practice improvements builds were just expected to work.

We co-ordinated across geographies by recording the sprint review demos as videos, individually accessible by Product Backlog Item. We would also expand out sprint review to include customers. We created customer advisory councils and would have regular feedback from our customer channels.

Visual Studio 2012 wave coincided with a shift in the industry from on-premises deployment of software to the easy use of the public cloud. Microsoft's public cloud, Windows Azure, is one of the major "Platforms as a Service (PaaS), and we used that to pioneer a new form of TFS, Visual Studio Online, which became available as a developer preview in September 2011 and commercial preview in November 2013.

Visual Studio Online raised the bar on our expectation once again. In the past at the end of every 3 week sprint, code was ready for internal "dogfooding". On the Service, every three week it is deployed live for customer use. We built exhaustive monitoring through instrumentation, synthetic transactions and points of presence. One of the key metrics we measure for services is service availability. We have made some significant improvements in how we measure availability of our service based on the incidents reported, existing availability models and the real metric that we were interested in. Please read this awesome blog post by Brian Harry which describes this in detail. The fundamental learnings were the following :-

- The primary measure of availability should be based on real customer experience ratherthan on synthetic transactions.
- Any measure of "availability", if you want it to be a meaningful should be a measure of customer experience and needs to represent both reliability and performance. Just counting failed requests leaves a major gap. If your users have to wait too long, the system can be just as unusable as if it's not responding at all.

- Lastly, any measure of availability should reflect the overall system health and not just the health of a given component. You may feel good that a component is running well but if a user needs to interact with 3 components, to get anything done, only one of them has to have a problem to cause the user to fail.

Since then we have a better feel of the service availability across our Visual Studio Online system and continue to apply the learnings as we move forward.

Conclusion :

The changes that we introduced across these 3 releases are now getting more ingrained in the system and there are less people questioning the value of the changes rather there are discussions about how to take those to the next level. We have been able to institutionalize the Agile practices across Developer Division and the teams are constantly working on smaller chunks of work (user stories), have a consistent definition of "done" , have early customer adoption programs to get feedback and quickly adapt and learn from the feedback. The resounding success of Visual Studio Updates (for client based product) and tremendous uptake of the sprintly release of Visual Studio Online is enabling the product teams to 'Build, Measure, Learn' in a true form. The constant feedback channel and a ship-vehicle to incorporate that feedback has enabled us to provide continuous value with continuous delivery.

References :

1. Visual Studio Team Foundation Server 2012 Adopting Agile Software Practices From Backlog to Continuous Feedback – Sam Guckenheimer & Neno Loje