Vol. 1 Issue 2 – February 2013

ALM Mag.com

Empowering Rapid Delivery of Quality Software

The ALM Architect – Balancing freedom with control

Testing in the modern application lifecycle

Enterprise Application Delivery and the Cloud

Continuous Delivery: Delivering IT to the Always Up-to-date User

Deployment Automation Basics Implementing Consistent Deployments

Enacting Scrum and Agile with VS TFS 2012

* Why Small Companies Need ALM More

BONUS Video

Pro Scrum Dev with VS 2012

Table Of Contents

Featured Event

Features

Welcome

Keith Denham
Publisher

Treasure Cove Publishing
Volume 1 – Issue 2
February 2013

Subscriptions:
almmag.com/subscribe

Email Address Changes:
support@almmag.com

Mission Statement:
"Our mission is to promote, support and empower rapid delivery of quality software."

We're back from a very successful and enjoyable ALM Summit held in the Microsoft Conference Center on the campus in Redmond, WA. Five days and nights of terrific presentations, workshops, social events and networking with many of the brightest minds in our space.

We will be looking to bring you some commentary on the Summit in the next issue.

This February issue of ALM Mag has a definite DevOps flavor with articles from Andrew Stevens of XebiaLabs, Dr. Jacob Ukelson of Nolio and Eric Minick of UrbanCode all presenting articles on deployment strategies and processes.

The March issue is shaping up to have an even stronger Testing focus but in true Agile spirit we'll see how that plays out.

Several alliances were formed during the Summit that, while allowing ALM Mag to retain its independent position, will bring a continuity of more of the great content that we trust will be of value to your and your organizations in the months and years ahead.

For our international readers know that following the success at the Summit we have already begun to talk to conference organizers in Europe, the South Pacific and Asia about sponsoring their events in an ongoing effort to bring ALM Mag closer to you.

Our membership is growing rapidly and already in these early stages and while strong in the US, it has taken on a marked international reach.

For those of you reading this issue and who are not yet members it is not too late to take advantage of our early bird Founding Member Offer… click this link for more details.

http://www.member.almmag.com/goto/foundingmember

Enjoy!

Warmest Regards,
Keith Denham

Keith Denham

P.S. Don't forget, this is an interactive magazine, tap on the audios and videos and tap on the highlighted links to see additional related content.
Tap on the Index Page Button at the foot of any page to return to the Index.

The ALM Architect – balancing freedom with control

Dave West
Chief Product Officer
Tasktop

In his industry shaping comment, Marc Andreessen declared that software is eating the world. http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html Never before have so many industries, markets and services depended on software to be competitive, efficient or even useful. Software, in traditional and high-tech organizations alike, has gone from an ancillary service to a key business process.

But for many organizations, the support for this 'key' business process is fragmented with process, practice and tool decisions being made by development teams, departments or even individual developers. Developer tech populism is a growing trend where software developers (http://blogs.forrester.com/application_development/2010/01/forrester-databyte-developer-scm-tool-adoption-and-use.html) are increasingly bringing in their own tools and making decisions on process and even platform. Empowered developers and teams is a key tenant of Agile methods, allowing teams to respond to the situation, selecting the best tools, process and practices for the situation[1]. But as software increases in importance to people and departments outside of IT, its status becomes worthy of management attention and has to adapt to integrate with a broader set of business processes. And there lies the rub: to be effective, software teams need freedom, but businesses who are now relying on software for competitive advantage need control, visibility and integration.

ALM is the foundation

Application Lifecycle Management (ALM) is the application of business discipline to the practice of software engineering. It is enabled by the integration of the software development tools, processes and practices in the context of management reporting. At its heart, ALM is about automating the process of software engineering, enabling everyone on a software delivery team to work effectively together. For many organizations, ALM is a dream rather than a reality, an aspiration to their tool purchasing decisions. At best, organizations have connected their Source Code Management (SCM) with their Change Management (CM), allowing change information to provide context to file changes.

Requirements, test, design and planning are still separate artifacts with limited automated integration. Management still relies on spreadsheets

[1] The Agile manifesto can be found at http://agilemanifesto.org/

and email to provide oversight. Control and traceability between disciplines is almost impossible with costly manual processes. Lack of ALM success can be attributed to many things, including cost, variety of delivery platforms, legacy, quality of vendor solutions, lack of flexibility, and inertia of people willing to change. All of these issues hinder adoption but are systemic of a more fundamental problem of ownership, visibility and architecture. It is time to empower an ALM architect to provide a roadmap for ALM and start treating ALM as a traditional key business process like sales, procurement, and other traditional business processes.

The role of the ALM architect

A quick search on indeed.com resulted in: "The search 'alm architect' did not match any jobs". So why are we talking about a job that doesn't exist? Simply put, the position ALM architect *should* exist. The focus of the role is to provide a holistic perspective to how ALM is implemented within an organization. To do this, ALM architects must work with a variety of stakeholders involved in software delivery by understanding their needs, constraints and objectives. The ALM architect will spend a great deal of time understanding the existing tools, processes, workflows, and management landscapes. The role also requires balancing the needs of development with those of management, which in part is understanding what information is required by management and then communicating those requirements in a tactful way to those other groups. The relationship between development and management is often broken, and the architect needs to act as broker to ensure both camps' objectives can be satisfied without putting unfair burdens on the other.

For example, time reporting is a management requirement that could be better integrated and automated into the developer's normal working practices, but without a clearly communicated set of objectives and softer considerations, the integration of time management into say change management would be met with distrust and undermined. The ALM architect should focus on:-

- **Understanding the objectives of ALM** – It is easy to think that ALM, the business process of software delivery, has a clear set of objectives. Unfortunately, for many organizations, understanding the contribution software has to the organization is difficult to quantify. Instead of focusing on very high level and often aspirational objectives, the ALM architect should focus on more tactical objectives such as decreasing time to deploy, improving quality by X%, or better managing outsource development.
- **Getting an appreciation of stakeholders and what they want** – To better understand the objectives of ALM, the ALM architect must be connected to the organization's software delivery community and their customers. And not all stakeholders need the same thing. For many organizations, the applications supported vary greatly, so it is important that any architecture consider these diverse groups. For example, marketing systems change at a much greater cadence than account processing, but they also require much less oversight. The ALM architect will have to build an architecture that supports all stakeholders.
- **Understanding existing tools**– This can be a daunting task, but a combination of tool surveys and deep dives into teams working practices will provide visibility of existing tools and their usage. For some organizations, the number of tools can run into the hundreds, for others the tool stacks are linked to the number of run-time platforms.

- **Examine tool schemas and workflows** – Within each tool usage is a workflow, described in the schema of that tool. Some workflows are simply the required field for a status field, others describe complex dependencies and sign offs. Many of these workflows have evolved, rather than being designed, but the ALM architect should still try and uncover the reason why a schema and workflow works in such a way.
- **Identify the key artifacts** – The management of the software delivery process is actually the management of the artifacts that are used to define, plan, develop, test and deploy that software. Some artifacts have very strong semantics, such as source code and deployment scripts, while as others are much less defined in their structure, such as requirements and design documents. The most important artifacts are those that traverse tool boundaries and departmental walls.
- **Building a data / object model** – The artifacts not only have key information held within them, but also have relationships between each other. The ALM data model will describe those relationships and also the associated state model for the artifacts. The model will be implemented in the development and reporting tools and as such will be influenced by those tools. Because it is unlikely that a physical, separate database will be created, this model can also be logical in nature.
- **Expose the simple process model** – There is a clear relationship between ALM and a number of processes, including management, release and the SDLC; instead of focusing on the activities of the process, the ALM architect should instead consider the flow of work and the key transitions of the artifacts. Thus, the process is much simpler than traditional documented processes within an organization.

Building an ALM Architecture

The artifacts, data and process models provide a great definition of the ALM architecture, but they alone do not describe the overall architecture of ALM. Management aspects, such as reporting and planning, need to be added to the process of software delivery and maintenance. For many organizations, time sheets and financial management are also important and should be included in any ALM model. In addition to the engineering and management disciplines, the architecture needs a glue that holds the processes together and supports the interchange of data between disciplines, workflow and unstructured connections. Figure 1 brings this together into an overall ALM conceptual model.

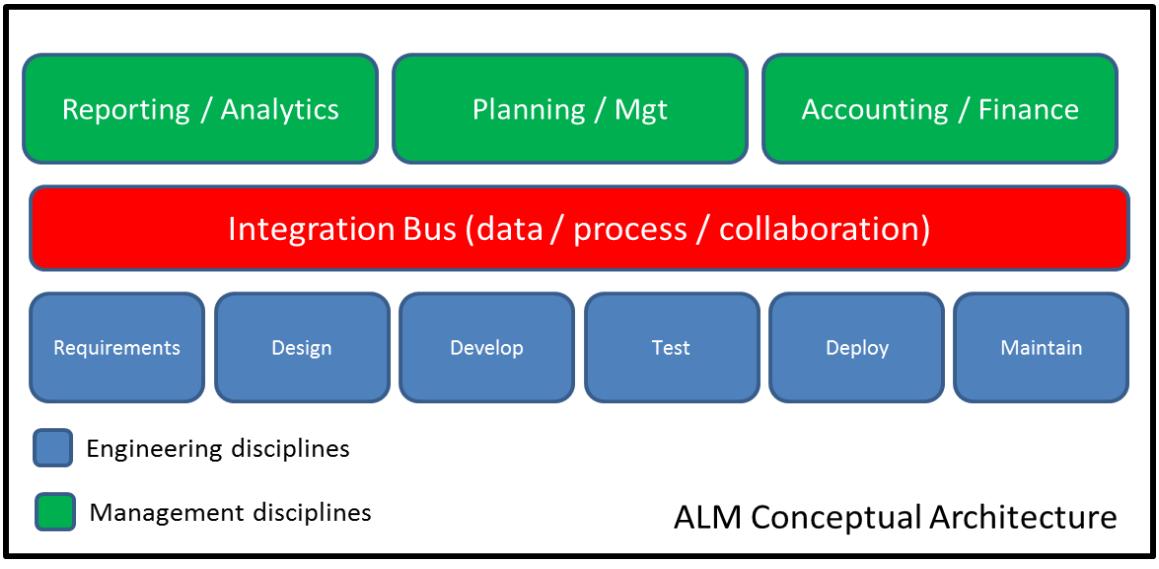


Figure 1

The ALM architect needs to review each area, looking to standardize a subset of tools and define standards such reports, methods of estimation, defined schemas, and the data interchange and workflow supporting the overall process. The ALM architect should concentrate on:-

- **Management disciplines** – Look to the overall processes being used by the organization to gain understanding of the reporting and management needs. By following the same route as other business processes, such as sales and procurement, many organizations attempt to create a standardized data warehouse for software delivery. This warehouse supports the capturing of historic data over time, allowing analytics to be performed.
- **Integration** – Automating the connection between engineering and management disciplines and between the different tools within each is important for streamlining the business process. Traditional manual processes that connect disciplines, such as a spreadsheet between testing and development or documents being emailed from requirements to development, do not scale or enable management reporting to be created. ALM tool platforms, such as IBM Jazz and MS Visual Studio, go some of the way to provide this interchange platform, but for the majority of organizations with heterogeneous tools, it will require looking into lifecycle integration technology.
- **Engineering discipline** – It would be true to say that not all disciplines are created equal, that certain engineering disciplines are more ready for an ALM approach. In particular, focus on the biggest areas of pain or areas where change is already happening because of other initiatives, such as continuous delivery or Agile development.

Conclusion

If your organization thinks that software delivery and maintenance is a key business process, it is crucial that they invest in the role of the ALM architect. That role provides ownership of the practical implementation and long term view of ALM within the organization. The architect should be measured by the success of ALM in delivering both management discipline to the practice of software delivery and maintenance and by realizing faster time to market or reduced cost.

By applying classical architectural skills, the ALM architect can build a roadmap and implementation plan that provides short-term benefits whilst moving the organization onto a more formal, integrated, transparent delivery and maintenance organization. This would allow executive management to make decisions in real time and plan and respond to market opportunities faster. For many organizations, software delivery slows down business change. By improving the business process of software, organizations will be better equipped to deliver and compete in the 21st century, which for many is the age of software…

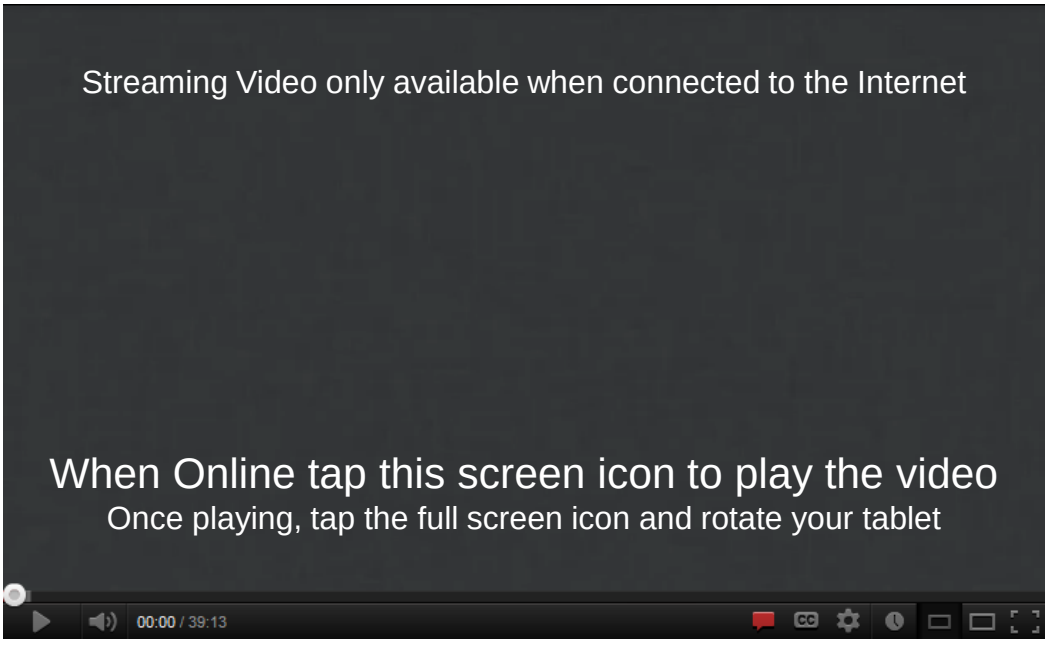
Dave West is the Chief Product Officer at Tasktop, where he engages with customers and partners to drive Tasktop's product roadmap and market positioning. Recently David served four years as vice president, research director at Forrester Research. Previously David led the development of the Rational Unified Process (RUP) for IBM/Rational. David authored the acclaimed book: Head First Object-Oriented Analysis and Design. On Twitter follow Tasktop @Tasktop or follow Dave West @DavidJWest

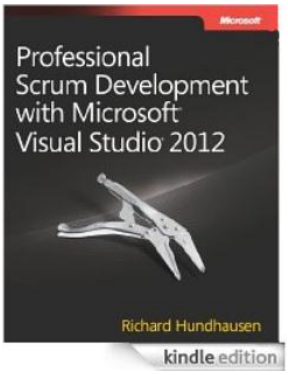
Professional Scrum Development with Visual Studio 2012

Scrum and Visual Studio 2012 make for a powerful combination. Organizations that can leverage both will see an increase in the quality of their products as well as the velocity of their teams. In this webcast we will explore the Scrum framework, the Visual Studio Scrum process template, and the planning and tracking tools in Team Foundation Server 2012.

Richard Hundhausen
President
Accentient

Recommended Audiences
Business Decision Makers, Developers, Technical Decision Makers





Richard Hundhausen is the president of Accentient, a company that helps software development teams understand and leverage Microsoft ALM tools and Scrum. He has over 30 years of software development experience and over 20 years of training experience. Richard is a Microsoft Regional Director, a Visual Studio ALM MVP, and author of Professional Scrum Development with Microsoft Visual Studio 2012 by Microsoft Press.

Why Small Companies Need ALM More

Andrew Clear
ALM Consultant
Northwest Cadence

We hear it a lot. "Oh, we're too small to need all that." But the truth of the matter is that, in all likelihood, you do *need that. You see, modern ALM best practices are all about one thing: keeping your project from failing.*

Let me run a couple scenarios by you.

You're a project manager at a Fortune 500 company worth a couple hundred million dollars. You're about to budget $200,000 on a six month software project and you're going to use a typical waterfall methodology. When that project falls flat on its face and gets scrapped without ever delivering anything of actual value you'll hem and haw, point out that this or that made the project unfeasible to begin with and point out that the other three projects you're managing are going great. No one is really happy about the project failing at the moment, but a few months from now no one will remember it anyways.

Now, consider you're a member of a small startup. You only have five developers (in fact the startup only has six employees), and you're about to budget that same $200,000 on a six month software project. Of course, your company doesn't actually have a net value at the moment, and the money you're spending is directly from one of your primary investors who also has an interest in the project itself. What happens if this project fails without delivering anything of value? The investor backs out, the company goes belly up, and you all get to play the employment game.

These scenarios, while obviously exaggerated, are not all that far fetched. A project failing without delivering any value can be *devastating* to a small to mid-sized company. The statistics tell a grisly tale: most software projects fail. Most of those that fail do so because of communication and requirements gathering issues. Most of the budget is spent fixing "self inflicted" problems, and most bugs are found by the end users. Stakeholder feedback comes too late to do anything about it.

These are *exactly the issues that ALM addresses*. Work item tracking increases communication and visibility. A prioritized backlog allows stakeholders to provide feedback *before* requirements are implemented. Unit testing, automated coded UI testing, repeatable test cases, and continuous integration help find bugs before they get into production. A tool like Visual Studio's Feedback Client allow stakeholder feedback during development.

Perhaps most importantly, the "best practices" associated with modern ALM ensure that we aren't *creating legacy code.* Maintenance is a huge cost that is rarely adequately planned for. Taking short cuts, and writing code that "just works" almost ensures that whatever you budgeted for maintenance won't be enough, because skipping things like unit tests and code reviews practically ensures that you are actively creating legacy code.

But all of those little bits are easy to skip when you're part of a small company. Creating that initial product is faster, and in most cases "easier" when you skip them. Landing that venture capital cash is all important, so you create content as quickly as possible. You think "we need to land this VC quick – then *after* we get the funding we'll be able to rewrite it properly and fix all those problems". In that situation you're almost *worse* off if you *do* get the funding. No VC is going to give you $1M to rewrite the same thing you just showed them. You can't go to your investors and admit that the thing you just sold them doesn't actually work. They expect that you're going to *build upon* what they've seen, not *rebuild* it.

Let me return to a point I glossed over previously. Software projects that fail *without delivering any value* are devastating to small companies. The resources just aren't there to scrap an entire project and start over. But modern ALM espouses *continuous value delivery;* implementing requirements incrementally, the completion of which creates a unit of value. This means you can start seeing returns on your project before it is "complete." Then if (when) you need to pivot, you don't have to start back at square one, and all that work wasn't a complete waste.

One final aspect of ALM utopia to discuss. Tech startups quite often become victims of their own success. Someone has a great idea, a team of development ninjas throw it together as quickly as possible. Cash starts flowing in, as do the customers. But then the customers start clamoring for new functionality. And they want it now. So your dev team grows, throwing legacy code on top of legacy code to keep that feature deployment rate high. The original ninjas start whispering that the infrastructure can't support the weight, but the process is more or less still working. Right up until it stops working. And in another month you end up another MySpace because you are unable to pivot. Unable to control the exponential growth of technical debt.

The easiest time to implement the complete set of ALM best practices is *before* you start your project. Or as close to the beginning as possible. The longer you've been piling legacy code on top of legacy code the harder it is to implement process changes. So the next time you hear yourself saying, "we're too small for that" take a step back and reconsider. No matter where you are in your development life, *there won't ever be a "better" time to modernize you practices and processes.* It won't get any easier, it will only get harder. Of course, wait too long and it won't be an issue at all.

Andrew Clear, ALM Consultant at Northwest Cadence, is dedicated to bringing agile to the agile-less.

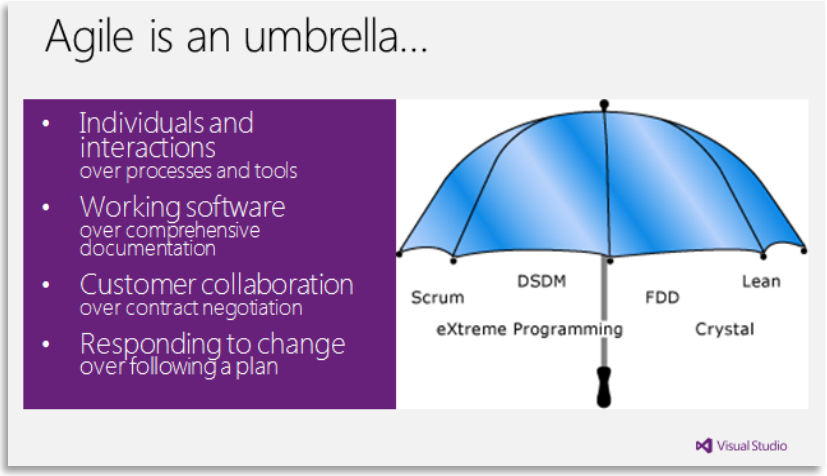
Northwest Cadence, Inc. *provides training and consulting services across the United States. The exclusive focus on Software Process and Microsoft ALM tools ensures clients benefit from deep technical expertise, sound process acumen, and hundreds of practical experiences on ALM projects. They partner with software development teams to improve processes and business results.*

Enacting Scrum and Agile with Visual Studio 2012

Clementino de Mendonca
National ALM Architect
Neudesic

In any conversation about Agile, it's always a good idea to set the ground by first revisiting the Agile Manifesto:

> *We are uncovering better ways of developing software by doing it and helping others to do it. Through this work we have come to value:*
>
> - *Individuals and interactions over processes and tools*
> - *Working software over comprehensive documentation*
> - *Customer collaboration over contract negotiation*
> - *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more.*



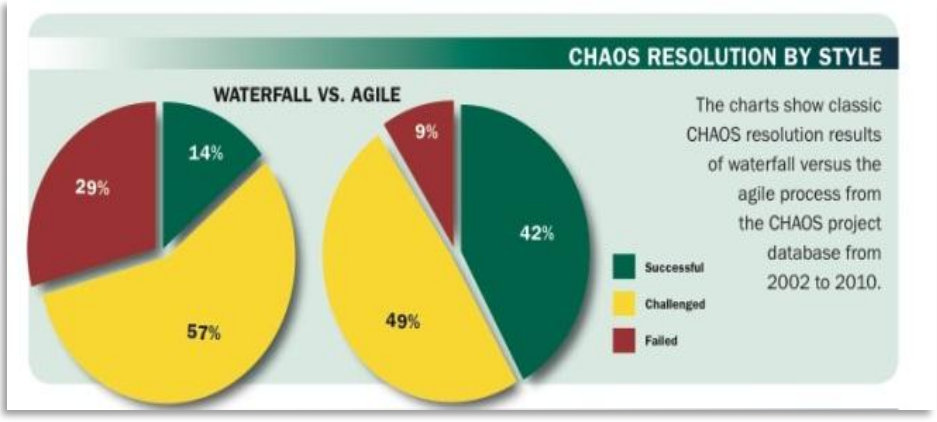
I want to emphasize the last phrase: "That is, while there is value in the items on the right, we value the items on the left more." So there is value on the items on the right. I have seen in several conversations with my Agile peers that they tend to sometimes forget that there is value in processes and tools, in doing some documentation, in negotiating a contract and in following a plan. It is just that when push comes to shove, if you have to make a decision, you have to value more the input that you get from individuals and interactions, from "kicking the tires" of working software, from collaborating with a customer and finding out face to face what they want exactly, and finally being flexible to respond to change when the time comes.

But how does TFS enable us to follow the Manifesto, to being Agile?

Agile is an umbrella. When we talk about Agile, we are talking about the work being done by the dozen participants that took part in the 2001 meeting where the Agile Manifesto was written. There were several software development methods represented: Scrum, DSDM, FDD (Feature Driven Development), eXtreme Programming, Crystal, Lean and others. All these methodologies come together and express the same set of fundamentals: Individuals and interaction, Working Software, Customer Collaboration, and Responding to Change. Whenever I talk about a specific technique I will be referring to Scrum as I will be using the Scrum template do illustrate the features in TFS. However, those apply to all Agile methods in the large. Sometimes it's just a matter of nomenclature, other times it's a difference in processes and how some of the deliverables are structured, but the essential ideas are there.

That said, when I mentioned that TFS can manage all kinds of projects I also meant Waterfall projects. You can create a process template from scratch or you can adapt an existing one and work it out in Waterfall manner.

Waterfall has been going into a decline, at least from a mindshare perspective. As you can see here in one of the latest CHAOS Manifesto reports, there is already a marked difference on the stats for success between Waterfall and Agile: 14% vs. 42%. If you factor in that the slice of failed Waterfall projects has been in the order of magnitude (around 30%) since the early 90's, it's easy to see that Agile has made in just 10 years more impact for the success of Agile projects than all work previously done in trying to fine tune waterfall. As a consequence the adoption rate of Agile has been unprecedented. Nowadays more



TFS is useful to manage any kind of project based on an Agile method, and all the process templates that come with TFS are Agile (Sometimes you will hear that MSF for CMMI Software Development is not Agile. If you talk to David Anderson – its original creator - he will eloquently defend the fact that MSF CMMI was developed based on Agile principles to fulfill an Agile need in the CMMI/SEI space. David explains the rationale for designing the process template in his report which can be found here).

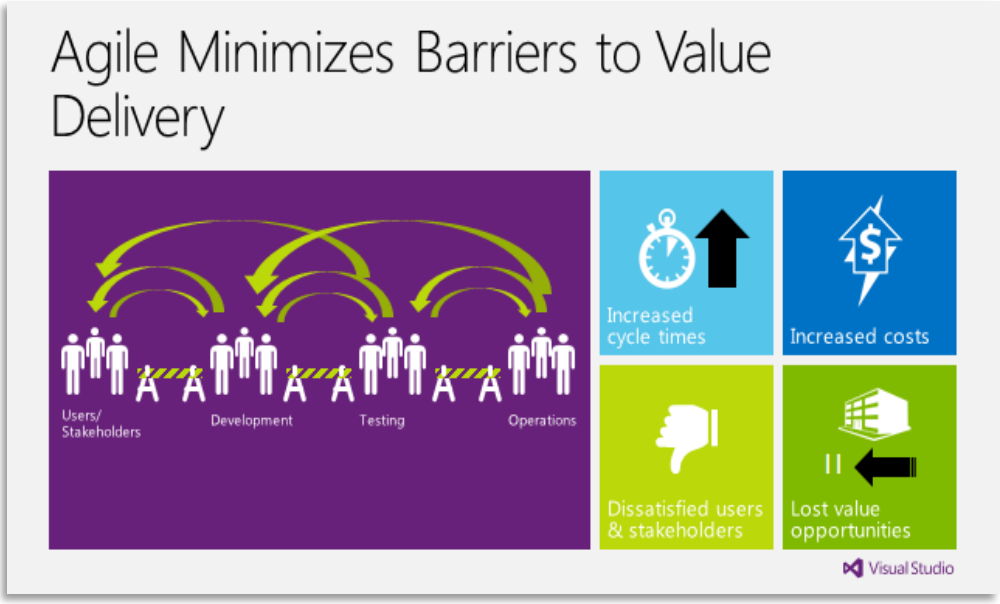
teams identify themselves with using Agile than any other development method, and most of those declare themselves as using Scrum. Waterfall is being left out as legacy (although I have yet to see statistics that take into account the number of people and dollars attached to those projects).

Without going too much into the nuances of the differences between Waterfall and Agile, there is also the "Governance tier" aspect. Sometimes you have very big programs that

behave in a waterfall manner from a product management perspective at a very high level, but allow the teams to behave in the fine grain level in an Agile manner, using either XP, Scrum or any other Agile way of doing things. This set of high level milestones that define a program are not necessarily waterfall – they comprise a governance tier that allows multiple teams to coordinate efforts at set dates.

What we are calling Waterfall here is the traditional cycle of capturing all requirements first, then moving to detailed planning next, then doing detailed design, followed by a single pass cycle of development, testing and deployment, never doing even a minimum of a 2nd pass, as recommended by Walkers Royce original paper. This paper, badly cited and misunderstood, gave him the vibe of being the father of waterfall when in fact the paper was showing how it did not work.

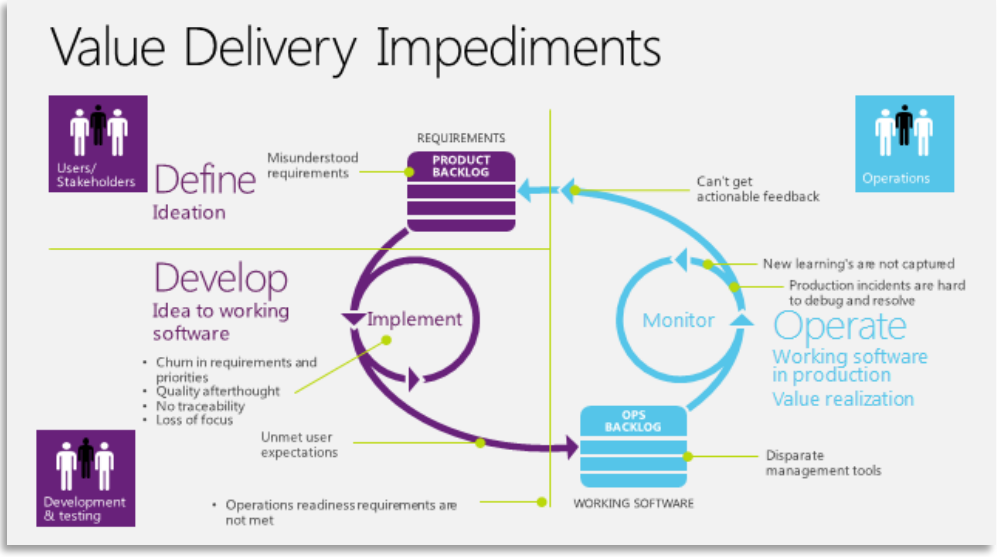
As Royce showed in his paper, this can only work in very specific and limited situations. For instance, sometimes when you have very well defined requirements, reused from an existing application, you can apply waterfall, but the applicability has become less and less nowadays as people recognize software systems as complex systems. Agile in general, and Scrum in particular are the best tailored to tackle those sort of application development projects.



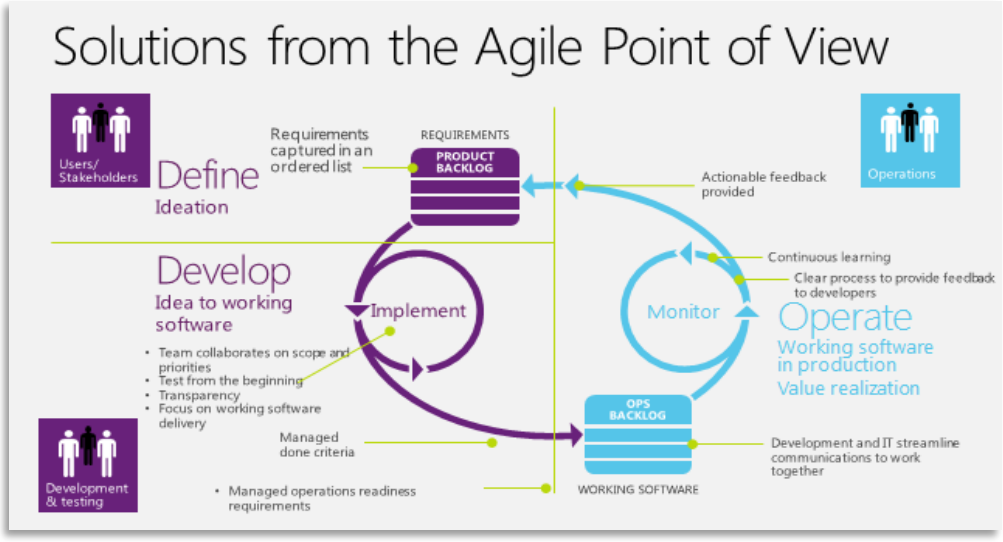
Agile Minimizes Barriers to Value Delivery. Most of those barriers are well-known to traditional development shops as they are a residue of “Waterfall to the max”: when it got entrenched into the Enterprise, each Waterfall “phase” became owned by a separate department that took whatever input they got and threw their contribution over – almost literally – the fence. And as communication barriers mounted, it is taking years to disassemble the traditional way of doing things, because it is just safe to continue doing what has worked so far, even though the numbers tell us that it hasn’t worked at all.

Most likely you have been attending a lot of presentations on TFS, or reading articles based on those. This article, based on a presentation I have given several times online and face to face, sort of reverses the picture: we are looking at the tool but first from an Agile perspective.

Agile, as an enabler, minimizes barriers to value delivery. You might see this same picture stating that TFS minimizes barriers to delivery. And that is fine because TFS embodies the principles of Agile. However we are talking about the way that you use the tool. TFS in itself can be an enabler or a constraint depending on how you use it, the same way a hammer or screwdriver can be useful or useless, depending if you have a nail or a screw to work with.



Here is a picture that has become a classic in the TFS world: Value Delivery Impediments in the lifecycle, or as I like to call it, the “owl cycle” (it does remind of the owl ship from Watchmen). Notice the impediments in the cycle: badly understood requirements, churn in requirements and priorities leading into code churn as developers have to redo and mend bad implementations of half-baked concepts. And in the end the unmet user expectations, if finally fixed and implemented, most likely lead to operational issues and support escalations, as time dedicated to testing tends to be squeezed out of the picture when the deadlines are close.



I changed the traditional picture that follows the Value Delivery Impediments so I could emphasize the solutions from an Agile method perspective. By no means an exact match, I wanted to just show how before you enact (that is, implement) a process with a tool set, you must design the solution from the process perspective.

Visual Studio from an Agile and Scrum perspective

Let's now take a look at the Scrum pillars and how to "enact" them (bring them to reality) with Visual Studio 2012. Tools should allow teams to work unimpeded, transparently capturing along the way the data needed to weave the actions of the team into a coherent set of artifacts that can be used by the team to self-organize. Microsoft has done a remarkable job: it started with a tool focused only on developers, Visual Studio, and through a process of versioned releases has delivered more and more value to allow teams to enact improved development processes. According to the latest ALM report by Gartner, Microsoft is the leader in this space with Visual Studio and TFS 2012.

However even a hammer takes learning to use it with skill. There have been many teams installing tools such as TFS without thinking about the processes that are being intrinsically modified, leading to many half-baked implementations that account for the 49% challenged projects in the CHAOS report, and result in a bad name for Agile and TFS.

Usually this happens because TFS was brought it to solve a specific problem instead of part of a concerted effort to reengineer and improve the work of software development as a whole. For instance, I have worked with more than one company that ascribed to "build and deploy" all issues that went bad with product delivery, without looking holistically at all aspects of the SDLC. It's as if you brought in SAP to solve just the tracking of packages of your product being delivered, and keeping the rest of the process untouched: a waste of time and money.

The Scrum pillars provide a simple guide to this holistic view of what an Agile process needs to drive the implementation of an ALM tool**.** They are the following, extracted directly from the guide:

- The team
- Events
- Artifacts
- Being "Done"

Now let's see how each of those pillars gets enacted in TFS.

The team

From the Scrum guide we have the following essential guidelines for a team:

- **Self-organizing teams.** No one (not even the Scrum Master) tells the Development Team how to turn Product Backlog into Increments of potentially releasable functionality
- **Development Teams are cross-functional.** The team has all of the skills as a team necessary to create a product Increment
- **No titles for Development Team members other than Developer**. This regardless of the work being performed by the person; there are no exceptions to this rule
- **Individual Development Team members may have specialized skills and areas of focus.** However accountability belongs to the Development Team as a whole
- **Development Teams do not contain sub-teams,** such as dedicated teams to particular domains like testing or business analysis

Essentially in a Scrum team you have two kinds of participation: those who are committed to the project ("pigs")[1], and those who are interested ("chickens")[1]. Let's take a look at how to enact those concepts with Visual Studio. Out of the box you get the following Team Project Groups if you are using

[1] For the story of how "Chickens and Pigs" originated and became applied to Scrum see ALM Mag Vol 1 Issue 1

the Microsoft Scrum 2.0 process template:

- **Contributors** – those correspond to the team members who are committed to the project, meaning they "have their skin in the game" (those would be the "pigs"). Members of this group have read and write access to all artifacts;
- **Readers** – those correspond to the team members who are interested but follow the project from afar (those would be the "chickens" in the metaphor), and therefore they do not contribute to the project artifacts (nevertheless their indirect opinion might still matter for a lot of reasons; for instance, they might be funding the project);

There are two other ancillary groups to help with the team project setup logistics:

- Build Administrators
- Project Administrators

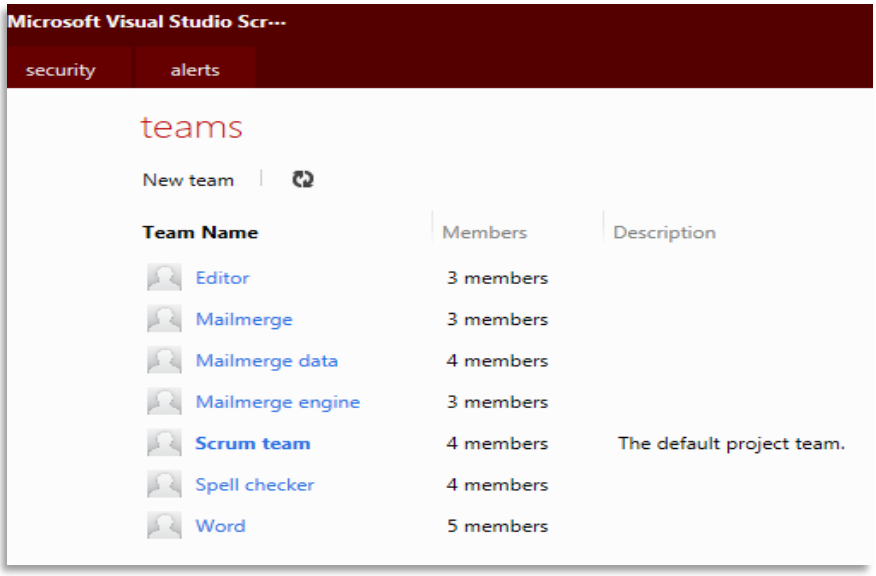
Those however do not correspond to a Scrum team member category as they do not contribute to any Scrum artifact, only to the team project configuration.

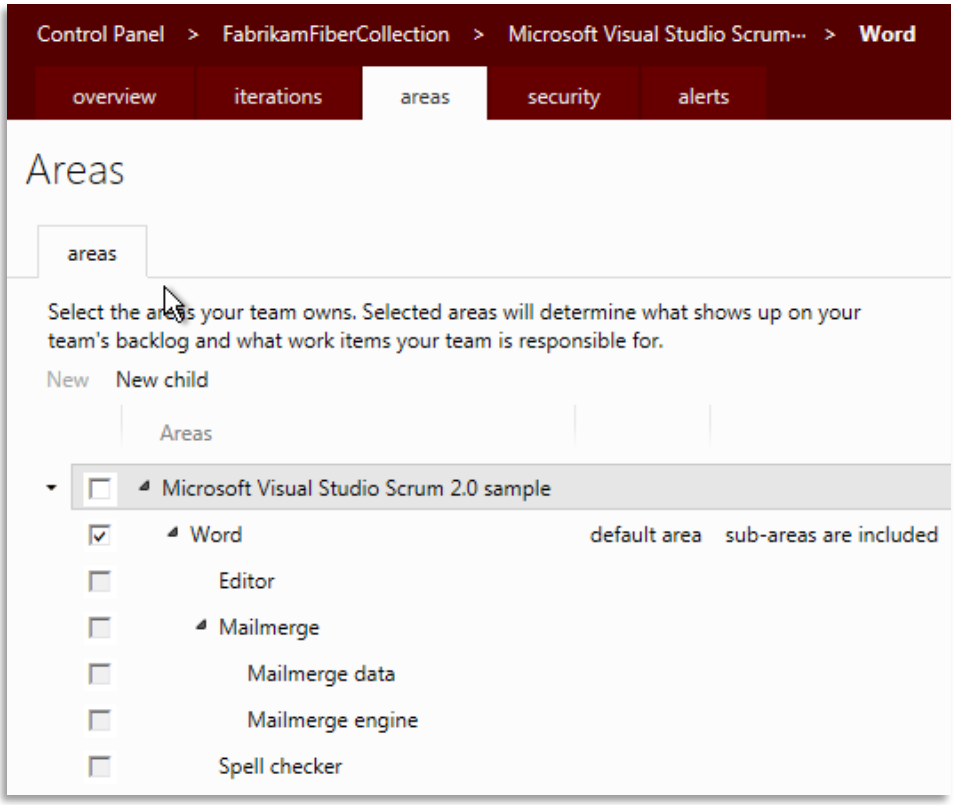
If you are using TFS Service, teams in TFS also help by allowing heterogeneous teams from different organizations, for instance, a Product Owner from the customer side, and the Scrum Master and Developers from the contracting side. Both would be able to communicate through TFS by using Windows Live accounts (and in the future there is the possibility that using federated Active Directory accounts might be possible).

TFS also now has the concept of Team. It's a way to structure team members into sub-teams within a team project. It would seem that this goes against the last principle I had mentioned ("Development teams do not contain sub-teams"), but in fact it doesn't as long as pay attention to what the Scrum guide is referring to: it does not condone specialized sub-teams on a functional role basis, such as a sub-team of testers or business analysts.

This in fact allows you to apply TFS to scale up Scrum: you can have sub-teams also containing heterogeneous teams focused on a subset of the functionality being developed. For example, you could have a team and sub-teams structured as displayed in the following picture.

To do that we created six teams as example areas of work within the Microsoft Word product (this is just an example, it does not map to the actual areas currently used). The teams by themselves are a flat hierarchy – you have no sub-teams in this view.





You can however map each team to an Area, and then structure those areas hierarchically. I will go in more about his in another article, so for now just take into account that there are plenty of ways of structure your areas and teams. The pattern I just followed here, in mapping a feature area to a team is famously known as *Conway's law* – that there is a correspondence of the software structure to team structure.

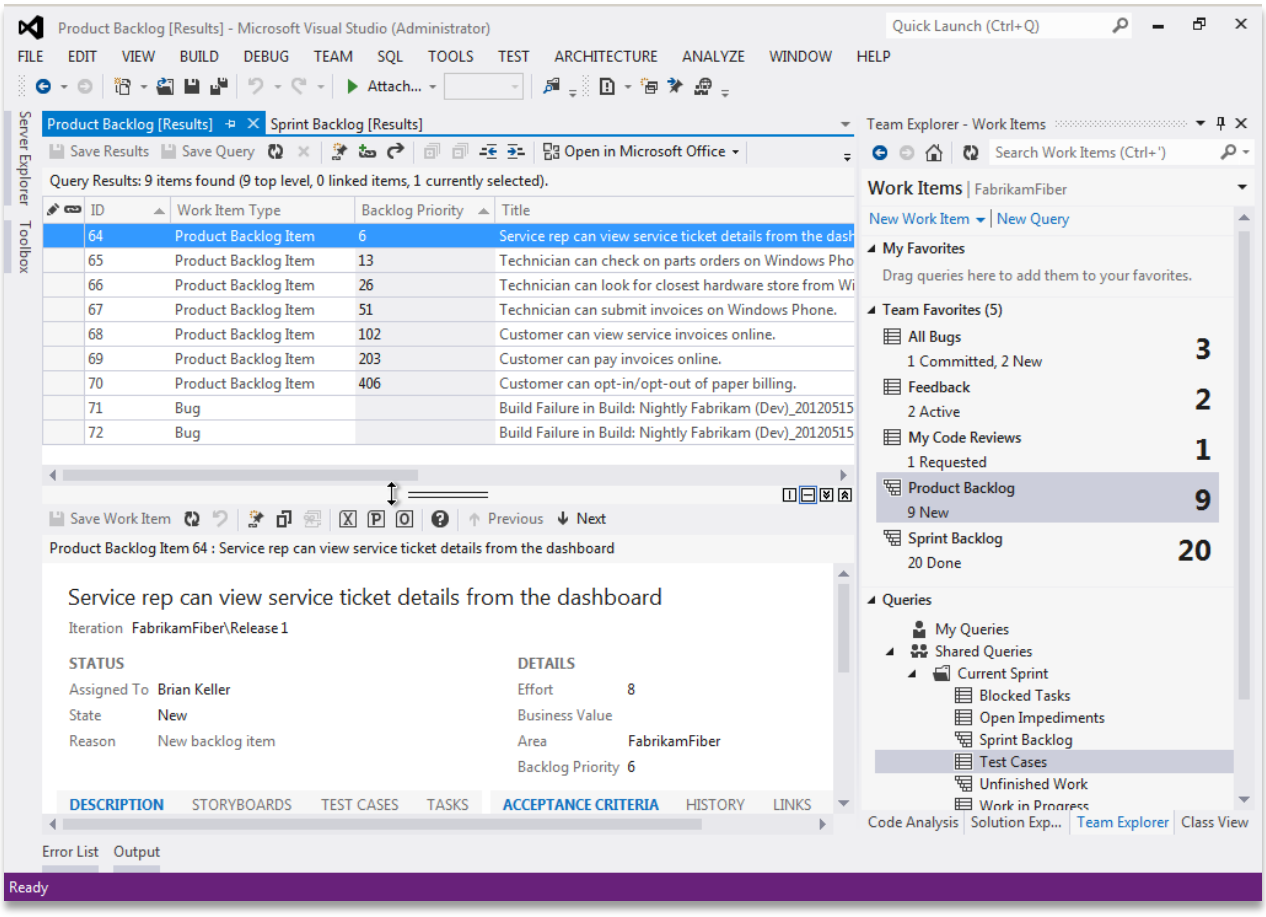
So we have seen how TFS can be used to enact or implement a Scrum team structure.

We are following the Scrum Guide document structure to go through this topic. At this point however we would take a look at how TFS has supports the events that the team goes through during the lifecycle of a project. However, to make it easier to see how this is being facilitated by TFS, let's take a look at the Scrum Artifacts first.

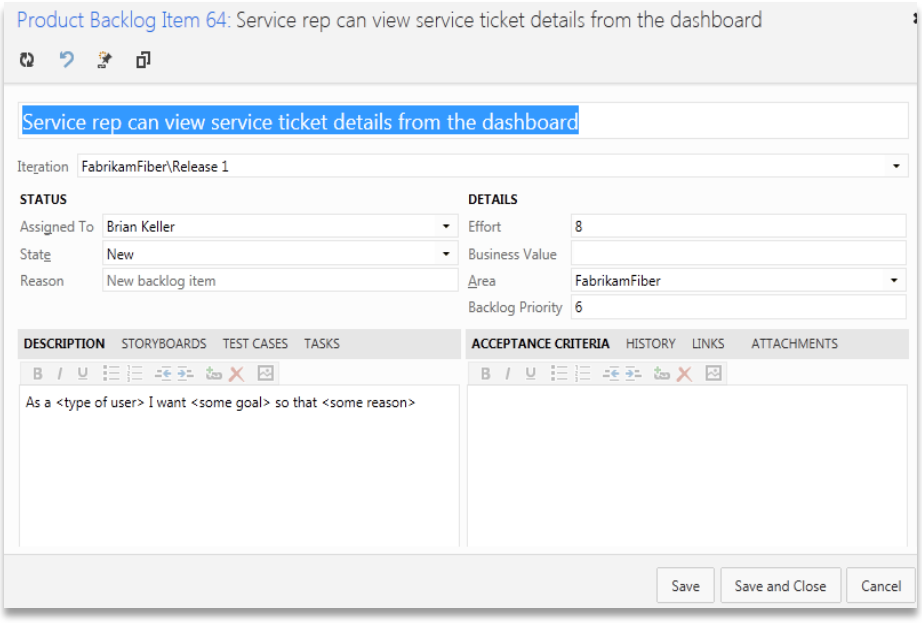
Scrum Artifacts

- **"The Product Backlog** is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product."
- "**The Sprint Backlog** is the set of Product Backlog items selected for the Sprint plus a plan for delivering the product Increment and realizing the Sprint Goal".
- "**The Increment** is the sum of all the Product Backlog items completed during a Sprint and all previous Sprints."

The Product and Sprint backlogs are the most visible aspects of the Scrum implementation in Visual Studio 2012. In the next picture, notice that the Product Backlog query is selected on Team Explorer and it shows in the left hand panel, with one Product Backlog Item showing in the detail panel below it.

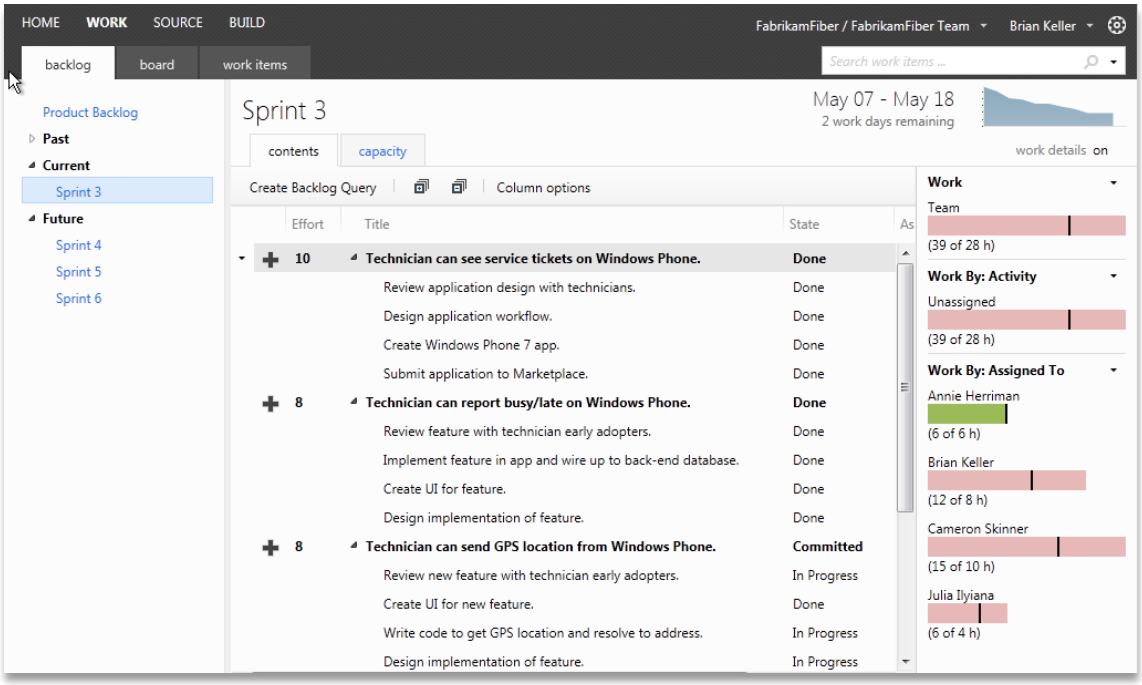


Next take a look at a PBI work item. It is important to understand that in Scrum a PBI can be anything: requirements, issues, bugs, technical debt items. Therefore the Product Backlog Item *work item type* represents a subset of the idea behind a PBI. This distinction is important to avoid fruitless discussions on what should be in the *tool* product backlog – the answer is “anything that might be needed in the product”, and if the tool does not have a type we need, we just use the existing PBI work item with notes to supplement the extra information needed.



Index Page

The Sprint Backlog in TFS is a similar rendering of the ideas suggested by the Scrum Guide. It allows you to easily map those PBIs selected for delivery within a sprint into tasks that will deliver the increment, therefore providing the delivery plan.



The idea of an Increment is enacted in Visual Studio in several interconnected ways. It starts by the sprint task breakdown itself. Since an increment can also be understood as a vertical slice of working software that goes across all tiers of an application, from that perspective you could represent it by tasks that correspond to work across all tiers, listed under the PBI they are related to. As each task is delivered as working code in source control, the build report becomes the ultimate representation of the increment: the PBIs, the tasks, the code, the tests that together correspond to a working piece of software that will incrementally be adding to an already working piece of software.

There are finer points about this that I will leave for another time, but one worth mentioning is that an increment can be mapped to a single PBI or to multiple PBIs. If you do it at the PBI or Story level, as in the picture, you can also release at that level. But for the purposes of delivering within a timebox you usually group those as described in the Increment definition by the Scrum Guide.

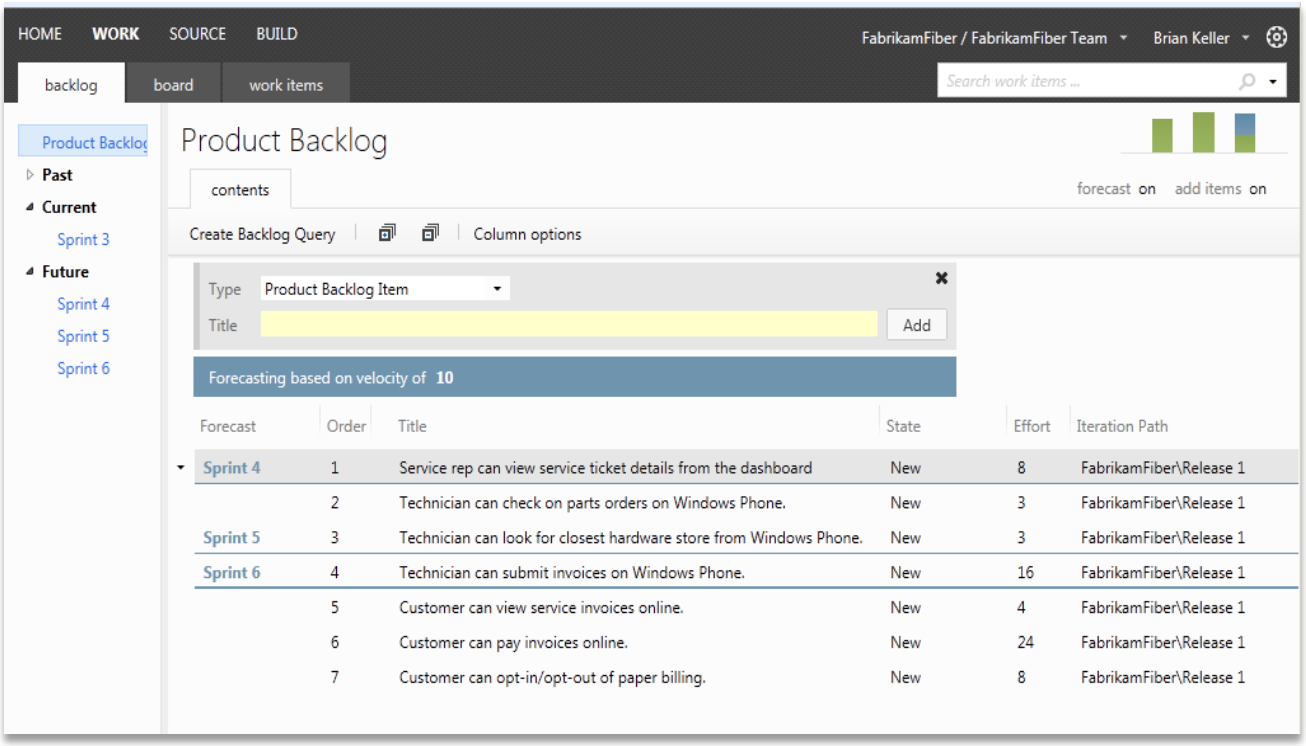
Scrum Events

The Scrum Events are the following, as per the Scrum Guide:

- **The Sprint**, a time-box of one month or less during which a “Done”, useable, and potentially releasable product Increment is created.
- **The Sprint Planning Meeting**, where he work to be performed in the Sprint is planned, is time-boxed to eight hours for a one-month Sprint.
- **The Daily Scrum** is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours.

- **A Sprint Review** is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed.
- **The Sprint Retrospective** is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

Notice that the Scrum Guide does not list Product or Release Planning as most Scrum diagrams do. The best description of this activity contained in the guide is **Backlog Grooming**, and we will start at that to talk about how Scrum events get enacted with Visual Studio.



Index Page

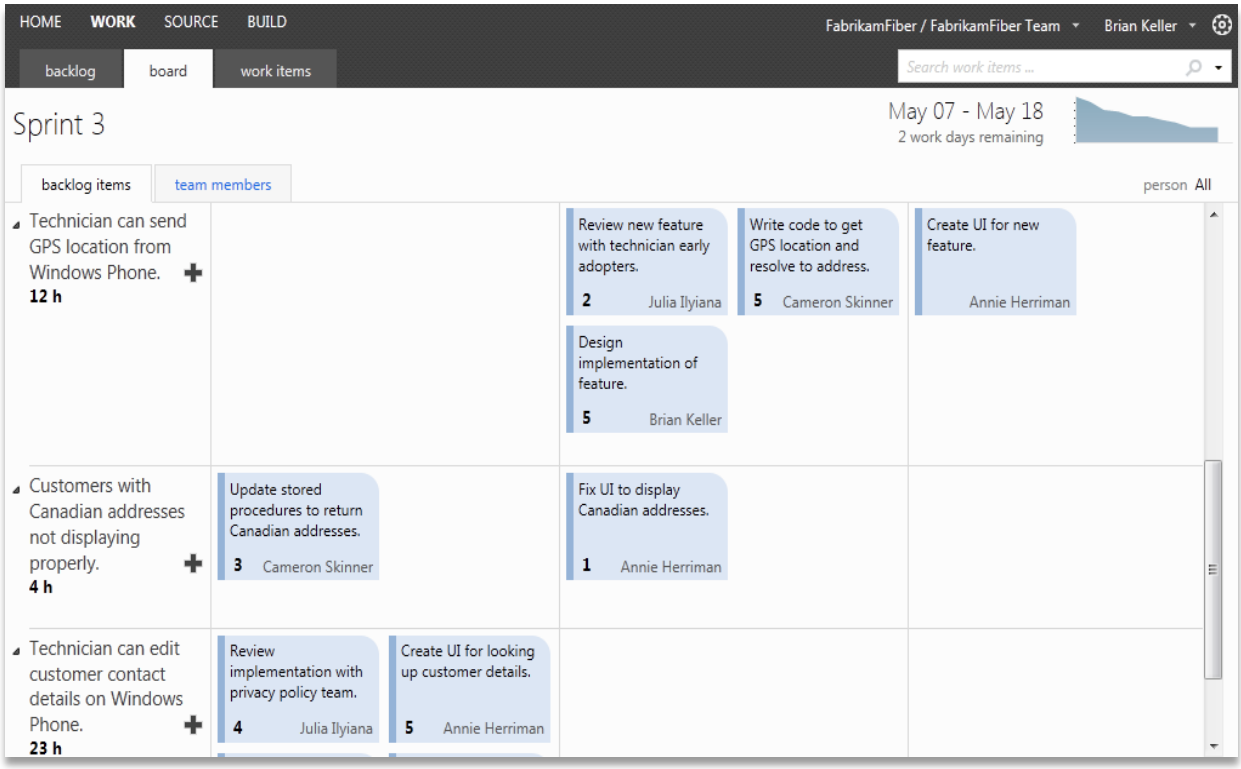
When talking about events the contrast between *process enactment* and *process automation* becomes clearer: you can't "automate" a team event as you could with team of robots working in an assembly line. Software development is creative work, and as such the idea of automating the production of intellectual artifacts smells of reductionist thinking backed by a superficial understanding of the nature of complex systems.

Even the enactment piece here draws more on convention vs. configuration: to enact a Scrum event you need among other things a calendar and a watch, and Visual Studio does not provide those. It does however provide a great aid for the **Grooming** meetings in the form of a web Product Backlog. With the option "Forecast on" it automatically takes hints from the available data to distribute the PBIs into sprints, creating a tentative longer term Release plan. The interactiveness and fluidity of this product backlog view may seem an "iffyness" to those used to deterministic planning tools. It however expresses the following idea from the Scrum guide:

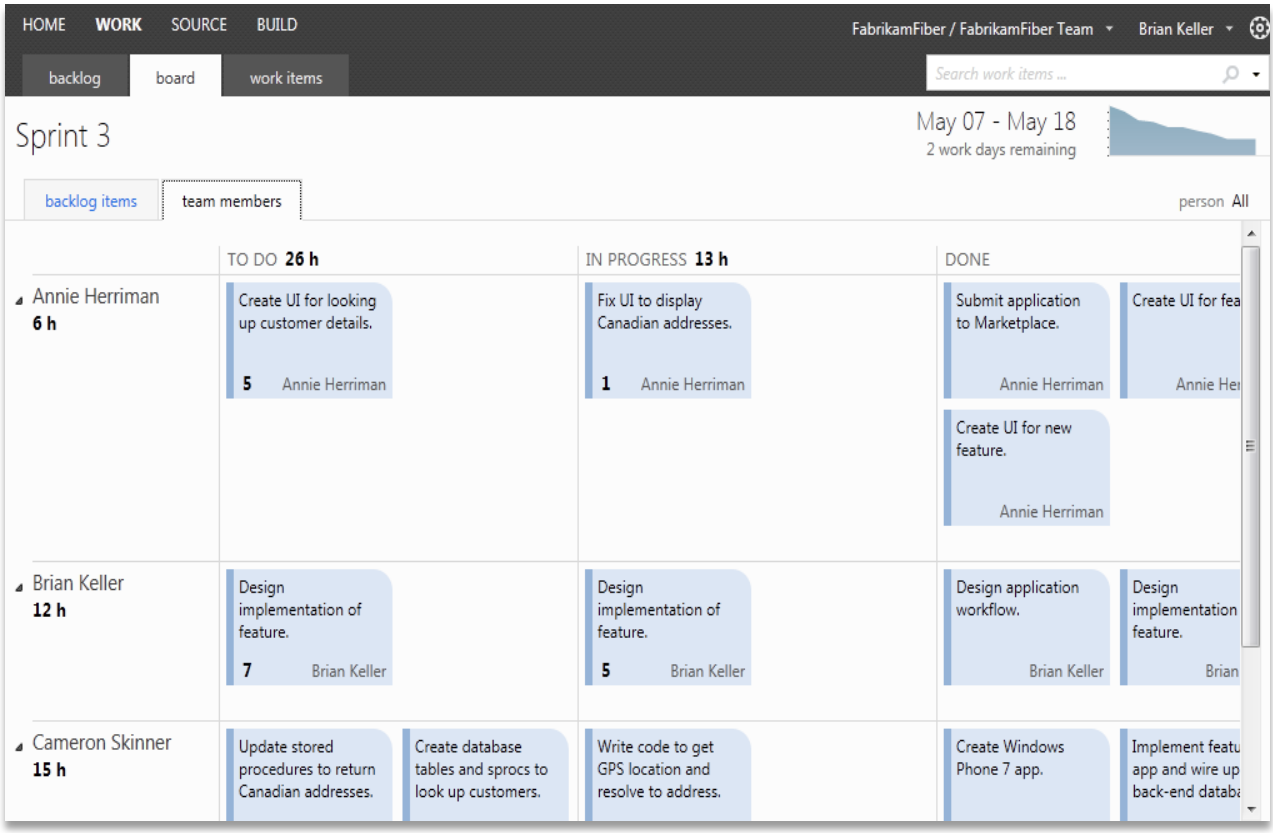
> "Various trend burndown, burnup and other projective practices have been used to forecast progress. These have proven useful. However, these do not replace the importance of empiricism. **In complex environments, what will happen is unknown**. Only what has happened may be used for forward-looking decision-making."

That is, release plans are supposed to be modified as the product backlog is modified. For those who are still clinging to their waterfall thinking ways, welcome to the real world.

So the Product Backlog view with "Forecast on" helps the team not only to work through Backlog Grooming meetings, but to work through the first part of the Sprint planning meeting as well in order to define the "What will be done this Sprint?"



The second part of the Sprint Planning Meeting can be done using the Board view of Visual Studio as a key aid to enabling the central idea of interactiveness and pragmatism contained in the Scrum Guide: during Spring Planning Meetings you can make real time adjustments to tasks and self-assignments, and the team can self-organize by using the “Team Members” view to balance work amongst team members.



The latter view also is very useful during Daily Scrums, especially if the team works beforehand by creating linked Impediment work items prior to the meeting if needed. The Scrum Master could also do it on the fly during the meeting, but that might take time from the short 15 minute time slot. The board obviously does not replace the need for a verbal report to the team on what has been done, what will be done and the impediments for the day, but it aids in enacting this meeting concept by providing contextual/background information so the meeting might flow more easily.

The Sprint Review is also facilitated by using the Backlog Items view of the Board. The Product Owner would go through each of the PBIs listed on the left side, open the PBI work item and look at the Done criteria listed by himself. The list of tasks associated with the PBI makes it clear whether work in it is still going on, but knowing whether it is done or not is about reviewing working software as a “primary measure of progress.” You can get some automation on this by having Tests associated with the PBI be done as part of an automated build. However the final measure of Done comes from the Product Owner, and that can’t be automated – yet another difference between process enactment vs. process automation.

Being "Done"
That's a segue to talk a bit about the definition of Done:

- It applies to a Product Backlog item or an Increment
- Team members must have a shared understanding of what it means for work to be complete, to ensure transparency
- This "Definition of Done" guides the Development Team in knowing how many Product Backlog items it can select during a Sprint Planning Meeting
- It is also used to assess when work is complete on the product Increment

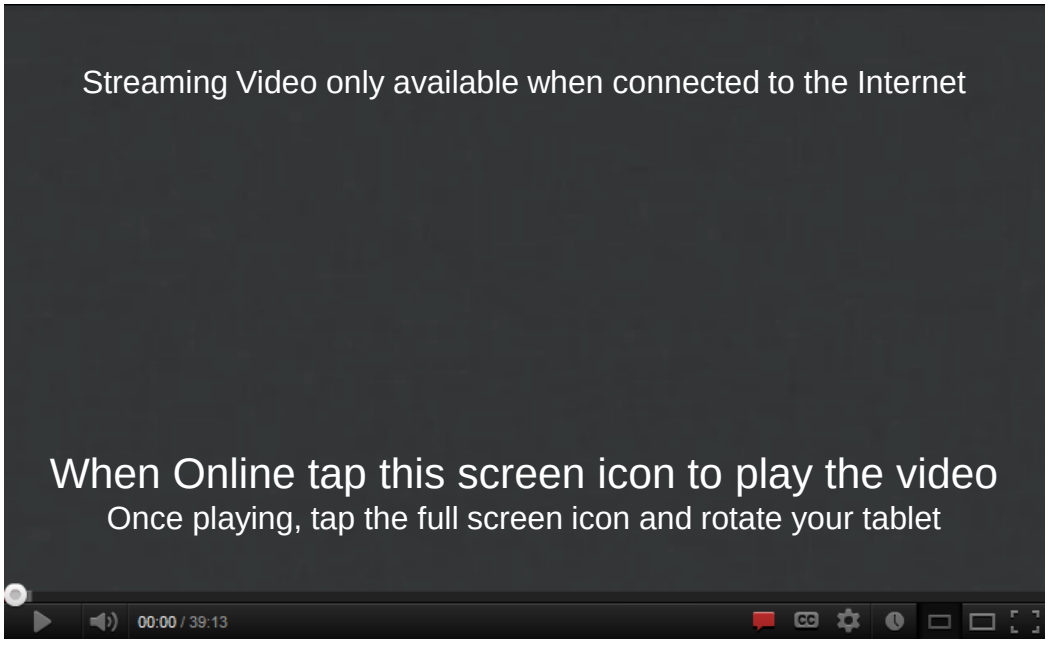
The discussion so far has also covered how Visual Studio enacts the definition of Done through:

- Acceptance Criteria;
- Task completion status;
- Test case run status;
- Bug status;
- Etcetera

The "Etcetera" piece relates to the fact that there is no predetermined list of Done criteria: "as Scrum Teams mature, their Definition of "Done" will expand to include more stringent criteria for higher quality". Therefore Visual Studio provides a great way to enact Done criteria, but some of those will most likely be conventions between the Product Owner and the team members, which can't be automated but can at least be documented as part of the Acceptance criteria.

I had skipped the enactment of the Sprint Review event. This one I have not yet found a clear-cut way to enact. One idea I have been using is to document the result of the meeting, that is, the Improvement Plan, as a set of tasks under a "Process Improvement" PBI, and use that as the starting point for the next meeting. I am open to suggestions on this topic.

Conclusion
So we have seen how the Agile and Scrum ideas are enacted, that is, enabled in reality,



A Webcast of this article: Scrum and Agile Management with VS 2012

through the help of tools such as Visual Studio 2012, which is currently the best ALM tool suite available.

Tools cannot make your team work in an Agile manner. Only after the ideas contained in Agile and Scrum are understood and learned is that your team will achieve proficiency, and will be able to consciously use the tools for their intended purpose. Until then tools will help you get started through some of the built-in processes they enable, but their best use will be as canvasses to your team's creativity and ongoing improvement efforts, not as silver bullets that would somehow take your team out of the morass of waterfall.

Acknowledgement:
My thanks to William Salazar, Wayne McDonald and Gary Pedretti for conversations which influenced this article.

Clementino de Mendonça is a Professional Scrum Developer Trainer, a Neudesic National ALM Architect, a former National Lead for ALM and Testing with Capgemini/Sogeti, and was on the Microsoft Services National ALM Team for 4 years. On the Visual Studio Process Team in 2007, he helped design the Visual Studio 2010 process templates (MSF Agile 5.0 and MSF CMMI 5.0).

Interviews with Agile Evangelists

Derek Neighbors
Integrum Technologies

Agile Weekly PodCast

Episode #95 – Moving to a Team Room

Clayton Lengel-Zigich, Roy van de Water, Isaac, Deepa & Ryan discuss:

Moving from cubes to team rooms

Derek Neighbors is a serial entrepreneur who helps people bring ideas to reality. Derek co-founded Gangplank, a collaborative workspace, in 2008 to encourage local "creatives" to explore innovative ideas. A partner at Integrum Technologies, a consulting services firm helping companies build high performing teams, Derek teaches entrepreneurship at Arizona State University.

Enterprise Application Delivery and the Cloud

Dr. Jacob Ukelson
VP Product Strategy
Nolio

I have been spending a lot of time lately looking at the cloud from an operational perspective with respect to applications – I guess it would fall under the general banner that some analysts would call DevOps and others would call AppOps. I see the difference between the two as either looking at applications from the perspective of everything that needs to be done before you can deploy an app, the other looks more at everything that needs to be done to deploy an app and monitor it afterwards. The line between them is really fuzzy – and as the cloud becomes more production oriented it will become even fuzzier.

What I found is that actual enterprise production applications in the cloud (not SaaS applications) are few and far between - so not a lot of attention has been paid to the lifecycle issues of managing enterprise production applications in the cloud. Dev and QA are the kings of cloud in the enterprise at the moment. I also found that as opposed to the NIST definition of cloud computing – IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service) which seem to describe a nice progression of functionality for the cloud – the real world is much messier. SaaS came first and most SaaS providers didn't build their applications on IaaS or PaaS - they built their own homebrew "Private PaaS" tailored to their specific application using a mixture of bespoke and off-the-shelf tooling. I think that enterprise production applications will look very similar - just that they will use off-the-shelf IaaS for infrastructure provisioning and off-the-shelf PaaS for specific components in their application stack.

As I was learning all this I think I finally understood why the cloud matters – way beyond its value as a cheaper delivery model, or a way to save on infrastructure costs. Cloud will enable IT to provide application delivery as an agile production line from dev to delivery. I use the term production line, but the cloud actually holds the promise of being able to provide much more than a physical production line – application update lifecycles of days or hours, not months or years. I think as this picture becomes whole – it will drastically change the way we think about applications.

In my depiction below, I clumped together Infrastructure and Platforms, not because they aren't important but because I wanted to focus on what

most people are ignoring at the moment – what happens after the app is assumed to be ready for release (i.e. a release candidate). Using “classic” application delivery metaphors, that means understanding what happens after dev has finished and the app has moved into the realm of operations.

In figure 1, the image has a sense\respond loop between the APM (Application Performance Monitor) and the rest of the system. This is the frontal lobe of cloud operations - its job is to analyze the information coming in from the APM and translate it into an appropriate action. This is one key area that still needs a lot of work (and invention) - but an autonomic response capability to changes in usage will be one key differentiator between cloud based applications and traditional applications.

transformed into a panacea. Sure you can go and allocate another dozen web\app servers if the current systems aren't keeping up with demand, but once you do that you'll need to pay for the extra capacity (and it may end up deferring the problem, not fixing it). It becomes an immediate additional expense, so just because you can do it doesn't mean you should – or that you will be allowed to. These cost aware decisions will be a new role for operations, and will require a new type of SLA management - "cost aware SLA management". Currently most SLAs focus on downtime (e.g. 99.9), and some focus on performance (x second response time) - but ignore the costs associated with maintaining the SLA. Once costs become more immediate and visible someone is going to have to manage them, and operations will be tagged with the job.

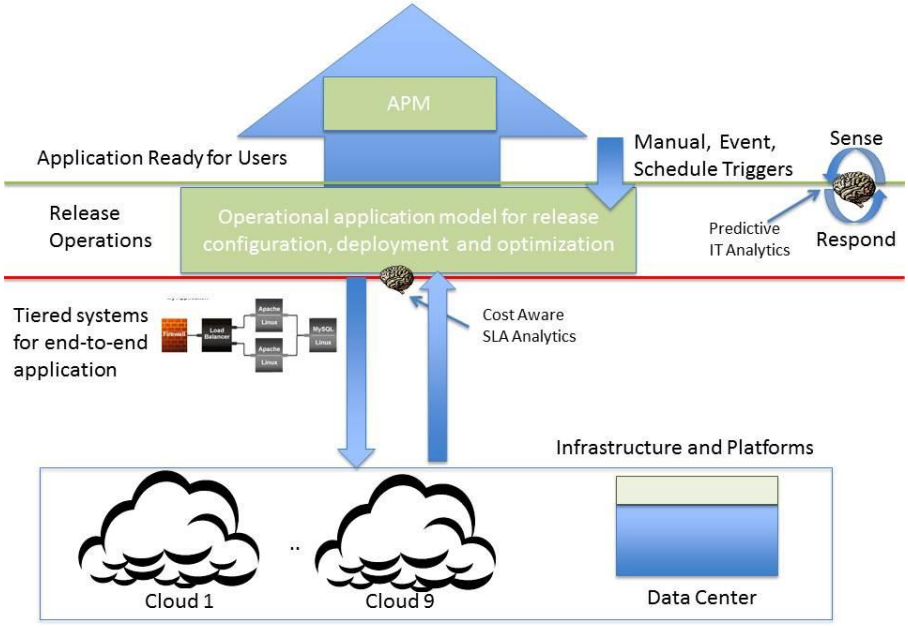


Figure 1: Release Operations for Enterprise Cloud Applications

The reason is the inherent elasticity of the cloud. You can always get more - more capacity, more storage - but it will cost you. If you have ever been part of an IT performance war-room then you know that capacity is a magic elixir that fixes everything. The cloud makes that elixir so simple to obtain, it can get

The problem is that APMs provide just too much information for humans to manage. There will need to be some sort of intelligent analysis distilling the information coming from the various APM systems and distilling the raw data into actionable information. That is the only way to obtain the benefits that the cloud

can provide, through intelligent systems that can make some decisions on their own (e.g. increase the number of servers to meet demand, within a predefined policy), and provide distilled, actionable information for operations when they can't.

Cloud Operations 2 – The Parietal Lobe

So let's assume that you have a smart monitor that notifies when an application is misbehaving – what should happen?

In the fast paced world of cloud operations you essentially have one of two high level decisions to make - incrementally deploy more infrastructure to solve the problem or rollback to a previous (hopefully working) version of the application. Either decision has impact on the business - rolling back means that your customers will lose functionality or features, and deploying more infrastructure means extra costs for the business.

There needs to be an additional "brain" that can both synthesize information from different systems, to make (and more importantly - act on) the decision about rollback vs. additional resources. This is part of "SLA cost awareness" that I mentioned above - it needs to weigh the cost of rollback vs. extra infrastructure, and also needs to make decisions about the efficacy of either course of action - whether to initiate a "flight" or "fight" response.

Once the response is decided, there needs to be a mechanism for implementing the decision. This where release operations comes into play. If the decision is "flight" (aka rollback) - there needs to be a well-defined process that enables rollback in a timely, orderly and non-disruptive fashion. If the decision is "fight" (aka deploy additional resources) - there needs to be a way to define exactly which resources need to be applied, where they should be applied and how to apply them.

This is harder than it sounds, since most enterprise applications have complex dependencies between compute resources, data resources, network resources and application components. For many applications just deploying a few extra web servers doesn't cut it – you may also need to add some additional app servers, maybe reconfigure your message queue and DB, as well as change configurations in multiple places.

Actually this additional brain isn't only for emergency situations. It needs to provide the same type of capability in any stressful situation - whether caused by a problem caused by an application anomaly found by your APM, or because a new feature is being released. New feature release and upgrades are mundane, but apply a whole set of new operational pressures on IT operations at the application layer. Businesses are betting more and more on their applications, users with always available platforms (i.e. mobile) mean that applications really do need to work 24×7, virtualization is making the underlying infrastructure elastic and easily available, while agile development enables features to be developed faster and in smaller increments. All this comes together so that the ability to flawlessly release new application features will become a key measure of enterprise IT value. This is on top of all the operational work needed to ensure that the current release is working well and serving customers as expected.

DevOps and AppOps

All these are putting new pressures on IT operations at the application layer. DevOps is a growing methodology that is starting to

address these issues. DevOps is related to AppOps but it isn't AppOps – nor does it replace the need for AppOps. DevOps is the process of streamlining the dev to ops lifecycle for applications, but AppOps is specifically the operational side of application management. I think that we'll be seeing a lot more enterprises starting to use the term AppOps, as way to describe their IT operations at the application layer – since there doesn't seem to any better term around. AppOps has two separate, but related, components:

- **Application Release Operations** – all the operational work needed to make sure production applications and features are deployed in a timely and robust fashion. This goes way beyond just release automation – since the automation component is just a small part of the operations surrounding a release. Release Operations also includes all the corrective and preventative operations for making sure imperfect applications and unexpected events don't cause catastrophic application failure.
- **Application Monitoring Operations** – the monitoring needed to make sure that application issues (especially ones that affect end users) are discovered as quickly as possible.

In many cases it is up to the AppOps folks to notice a problem and then (working with dev) figure out a quick workaround to make sure things continue to run. It is then up to dev to come up with a longer term fix which gets deployed in the next release.

There has been very little focus on Application Services CAPA (Corrective Action/ Preventative Action) – or maybe a better term would be COPO (Corrective Operations/ Preventative Operations). I am not sure why Application Services CAPA gets so little attention – maybe because it is the unglamorous daily work of ensuring everything works correctly, as opposed to getting a new release deployed.

Application Release Operations, Application Monitoring Operations and Application Services COPO – these are all areas that will be changing drastically as the cloud moves closer to production in enterprise IT.

Dr. Jacob Ukelson is an expert in developing innovative solutions to real-world customer challenges. He is VP of Product Strategy at Nolio - the Zero Touch Deployment™ company. Nolio's Application Release Operations platform reduces time-to-market and makes enterprise operations cloud ready. Click here to read Nolio's Blog.

Bridging Language Gaps and Creating Executive Summaries Using VS Test Professional 2012

Tara Charter
Sr. Agile Business Analyst

I feel like I'm speaking two different languages when I deal with the company itself and the developers working within the company. I also think developers become frustrated with me when they can't immediately determine the origin of a requirement.

In *Seven Steps to Mastering Business Analysis*, one developer reveals frustration with the business analysis process:

> I began my career as a developer and was successful. I found that not only did I enjoy solving problems but enjoyed finding them and learning about business processes from the business experts. I remember the first time that I was programming based on requests and specifications from a BA. I felt frustrated because I wasn't tasked with talking with the users directly. I wanted to know exactly what went on in the business and why they had requested this software functionality. (Carkenord, 2009, B2T Training)

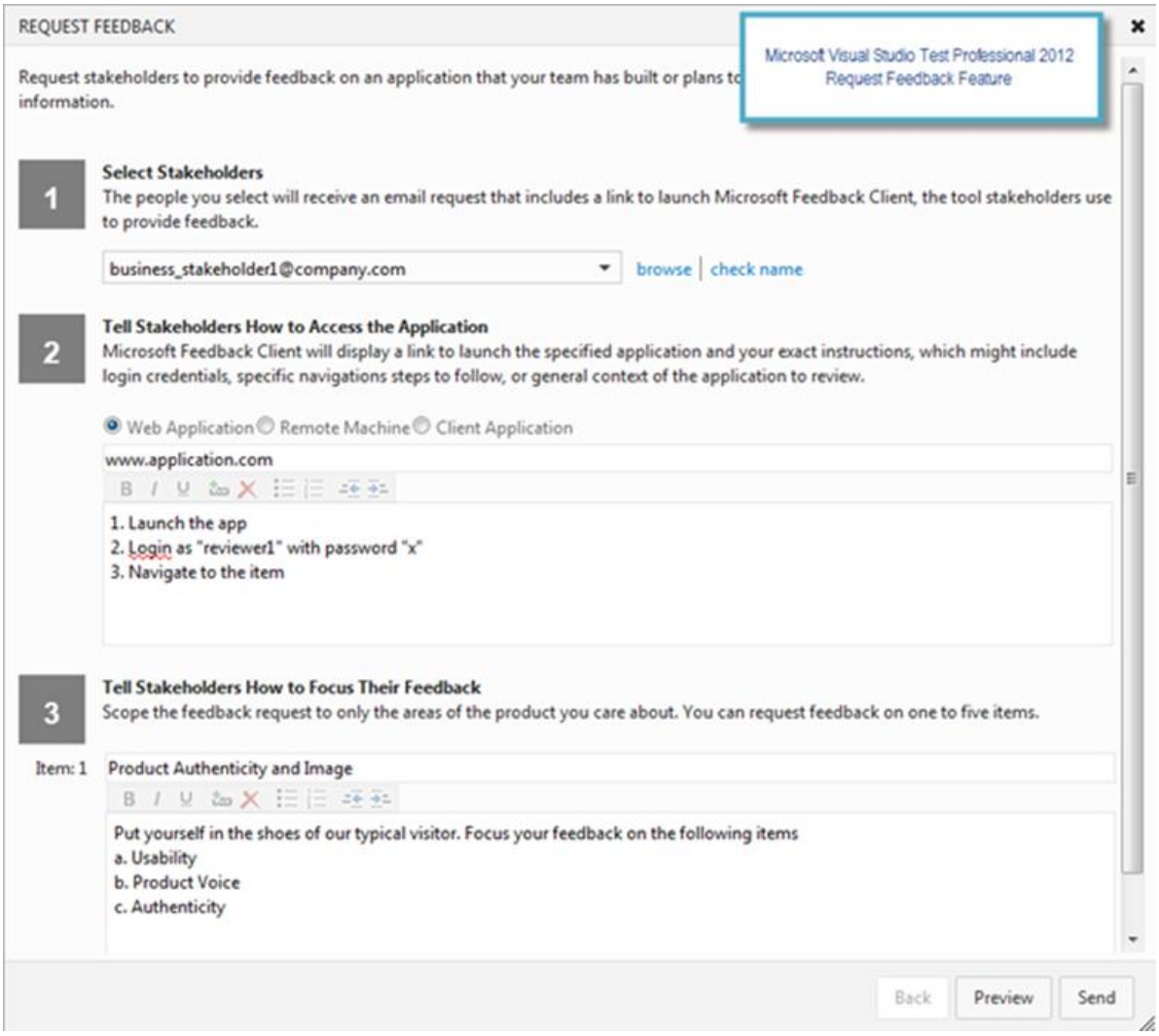
One of the most common communication challenges and often a barrier to delivering excellence on a project is the ability to speak multiple languages. As a business analyst or product owner, I must speak several languages throughout the day depending on my audience.

Each time I communicate, I must tailor my communication to my audience to avoid communicating ineffectively and wasting valuable time. And the language for each needs to be appropriate to their levels of expertise and understanding. Communicating with the developers must be crystal clear thus enabling the developers to create the system or application required by the company (Project Community, Louise Ledbrook, November 18, 2011).

Microsoft Visual Studio Team Foundation Server (TFS) 2012 provides built-in features that allow me to speak several languages without having to use any other tool. For example, when speaking "Executive Summary" – the preferred language of my project sponsor – I quickly gather executive-summary-type details from the customizable task board, check

the forecast in the product backlog, and then generate a current status report from a work item query. The sponsor can easily drill down for more detail. Find out more.

For the business stakeholder, I can keep them in the loop by creating and sharing prototypes, storyboards, and requesting feedback:



Because these Feedback Requests become work items, I can easily keep track of my communications by Status (who the feedback is assigned to, whether or not the feedback is active) and iteration. Here is a good example of a home page in Team Web Access displaying an overview of a project.

For co-located or remote agile teams, the developer/analyst/stakeholder/business owner/sponsor/manager needs to be able to advocate consistently for the other team members without a language gap. With a tool like TFS 2012, I no longer have to say I am from the IT world or the business world. I can focus on speaking the language of “business problem resolution”.

> *Tara Charter, a Sr. Agile Business Analyst, greets every new business challenge as an opportunity to make a difference. Tara knows the only way to truly solve a business’ unique problems is to really understand what the customer really wants.*

Testing in the modern application lifecycle

Martin Hinshelwood
ALM Consultant
Northwest Cadence

Manual Testing in this new age of the modern application lifecycle has taken on new complexities that make it even more difficult to track and identify which tests are passing, which are failing and to which environment that data should be associated.

As we move into, or you might already be there, the era of agile and the delivery timetable shrinks to only a few weeks we can no longer do all of our testing in isolation. We need to work with the Engineers, Analysts and Operations on Development Teams and effectively swarm to solve the problems. Whether that problem is adding new features, fixing a bug or deploying to an environment it is now in the scope of all the departments and this new world does not recognize even the idea of the silo.

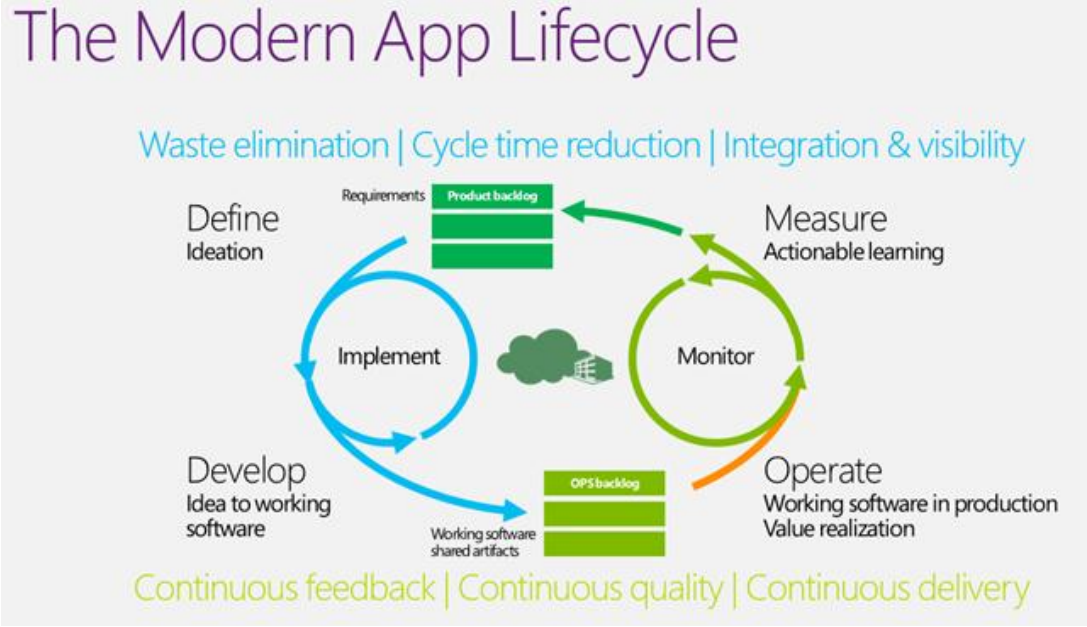


Figure: Manual Test makes sure that we build what we need and no more

It is no longer acceptable to "*test quality In*" as this is far too expensive and we need to move to a model of iteratively "*building quality in*" so that each and every iteration outputs working software that is of releasable quality. The only way to do this is to be able to test, measure and learn with very tight feedback loops…

The Problem

There are a number of key elements that when one tries to deliver fully tested software at least every 30 days become acute problems and blockers. Many teams fail because they do not realize that all of their processes will need to change in order to achieve agility of purpose. The three main stumbling blocks are:

- **Where am I?** – What has been tested and what has not
- **Actionable Bugs** – Collecting enough information to reproduce
- **Automation Generation** – Code generation from the recorded data

This things however are not insurmountable, but do need to be defined.

Where am I?

It is hard for testing teams in this new era to even know where they are in the testing cycle. To be honest they found it hard even in the world of linear testing as the shear amount of data collected can be monstrous. The traditional method would be to have a document that contains the test itself and some sort of spreadsheet that allows them to track progress.

But if you have a multitude of flat data sheets to record your test status it is at most impossible and at least incredibly time consuming to formulate any sort of reporting from the desperate data sources. It is hard enough to know which tester is taking which tests let alone which tests have been run against which version of the software under test. This can make testing incredibly opaque and more of an art than a science.

Actionable Bugs

When a bug is found it is eminently more difficult to figure out where it is and what to fix, if you lack even the most basic information on how it came to pass. If we do not have the steps that were followed then reproducing all but the simplest of issues can be impossible, resulting in wasted time from both the developer and the tester as they go back and forth in a kind of bug ping-pong that gets nothing done.

Reducing the cycle time of bugs is equally hard and the more information that we collect during the lead up to the failure, greatly increases the likelihood that we will be able to find and fix the issue.

Automation Generation

Regardless of finding a test while proving a bug exists or does not, or if that test verifies that a feature now correctly exists, if we are to attain agility with quality we need to be able to always run this test. This poses a problem as our tests multiply iteration on iteration and very quickly to a point where we can only pick and choose which tests need to be run.

To combat this, it would be awesome if we could take all of the data collected and generate an automation that can be run time and again to verify that we have not broken anything. This automation needs to be easily regenerated so that we do not spend large amounts of time maintaining it in the scope of a continuously evolving application.

Achieving quality product without automation generation can make testing very expensive and can even put the road to “testing quality in” beyond the means of many organizations.

The Integration

There are a number of points within which you can integrate your tools with Microsoft test Manager and give users a seamless experience across a multitude of technologies.

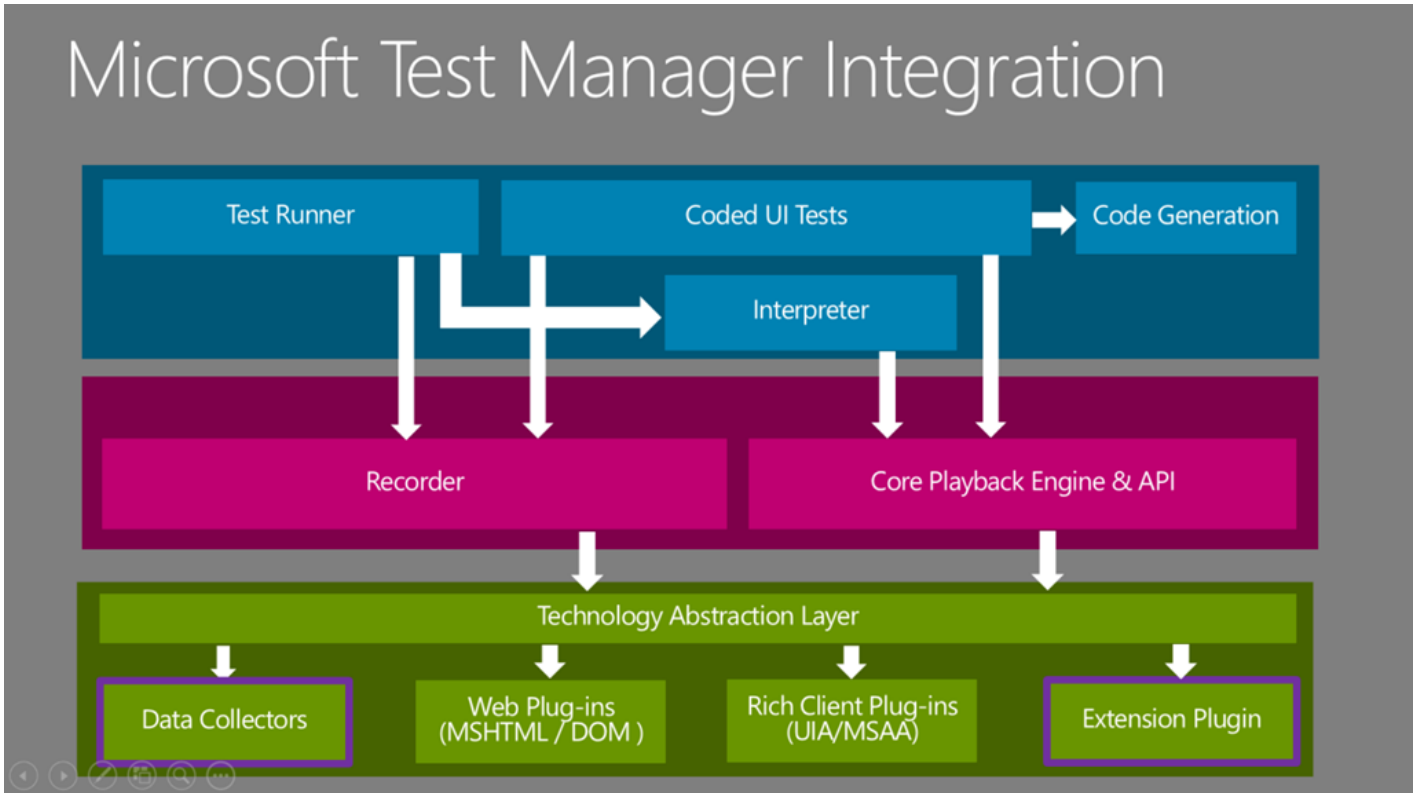


Figure: Microsoft Test Manager Architecture

These integration points currently cover two areas, collecting data and recording and playback.

- **Data Collector** – Extending a data recorder
- **Recorder /Playback** – Extend the recording and playback features
- **Test Runner** – Building a custom Test Runner

Integration: Data Collector

Microsoft Test Manager provides a bunch of data collectors out of the box for collecting information to both create actionable bugs and to enable the recording / playback functionality. You can create your own data collectors to capture whatever you want as part of the manual test run. These collectors can run on the client, or any of the machines under test, via the test agent.

Integration: Recorder / Playback

Being able to extend the recording / playback functionality means that applications might traditionally not work with the standard recorder. The standard recorder is based on the same technology that enables a screen reader to work and if those features are not present in the application under test it will fail. To make it work you would need to create an extension that would allow you to for example, test using Chrome or on a Java application.

Integration: Test Runner

There may indeed be the need to replicate a subset of, or the entire functionality of Microsoft Test Manager in order to run in environments and systems not currently supported by MTM. This can be done using the rich set of APIs that allow you to import test results and interact with any of the Test Case and Test Result data.

The Solutions

In order for testers to become first class citizens in the Development Team we need to be able to effectively turn the quagmire that is traditional test tracking into a well oiled repeatable machine that takes all of the mundane tasks away from testers and let them concentrate on validating the delivery of value to the customer.

Microsoft Test Manager provides a framework to achieve this by allowing you to easily:

1. Create and Manage your many Test Cases
2. Build Test Suites out of queries, requirements or manually
3. Execute those test cases and collect data on which tests have been executed and against which builds
4. Know and understand your test matrix
5. Collect enough data automatically to be able to reproduce issues
6. Collect enough data to create throw away automation

These things alone would be enough to make it a must have solution, but add to that the integration described above, mix in a few partners and you have the ambrosia of test management systems.

Solution: Microsoft Test Manager from Microsoft

Microsoft Test Manager provides support for some pretty advanced testing techniques and while it is limited to a Windows host it can be used against almost any application. It has some limitations on unmanaged and java applications, but even most old web applications can be fully tested. There are some features, like IntelliTrace and Test Impact Analysis that are only supported on managed code, but really those are value adds rather than the core reason to move to MTM.

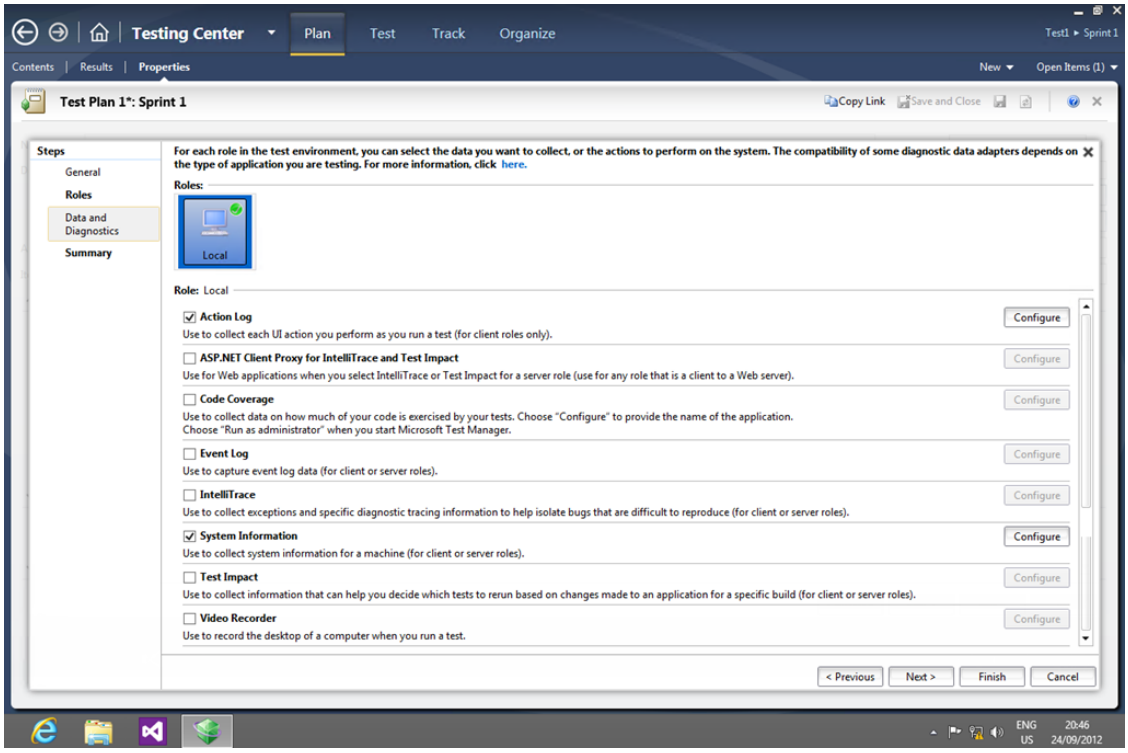


Figure: Manage your Microsoft Test Manager Data Collectors

That said some of the built in data collectors in there are pretty cool:

- **Video Recording** – Simply a video of the test session indexed by step
- **IntelliTrace** – like tivo for developers… debugging forward, backward and sideways
- **Code Coverage** – Even for manual test you collect code coverage data
- **System Information** – Windows version, running application
- **Event Log** – Scrape specified events
- **Test Impact Analysis** – log every method hit by your manual test so that we can match that to code paths changed and reduce our test matrix

These features are awesome additions to the platform but they pale in comparison to the coup de resistance which is the **Action Recorder**. The action test recorder allows you to fast-forward your manual tests and gives one the ability to generate Coded UI tests.

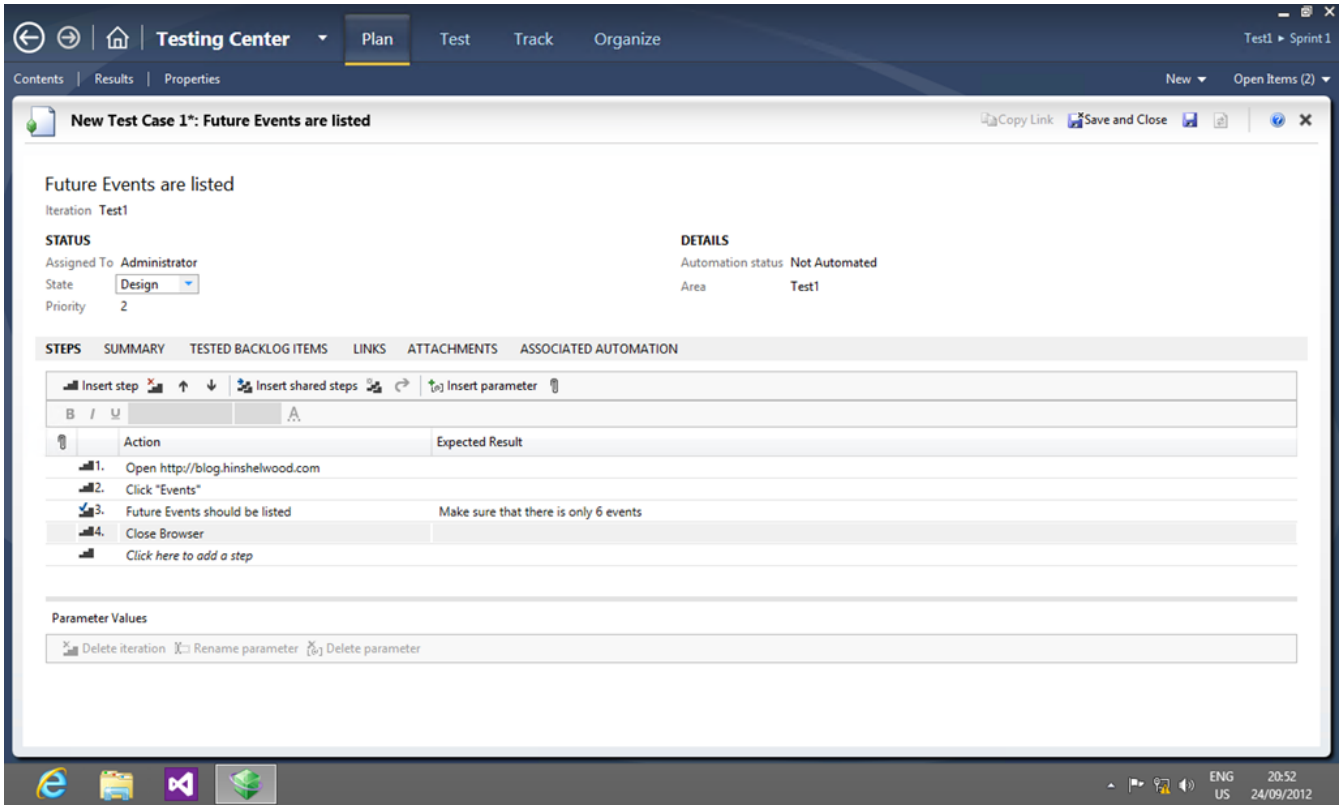


Figure: Test Manager allows you to manage the Test Steps

These abilities allow us to both manage our test cases and to record the data, and the test runner allows us to record the results and create actionable bugs. You see all the while that one is executing the tests, the data recorders are collecting all of that lovely data and storing it.

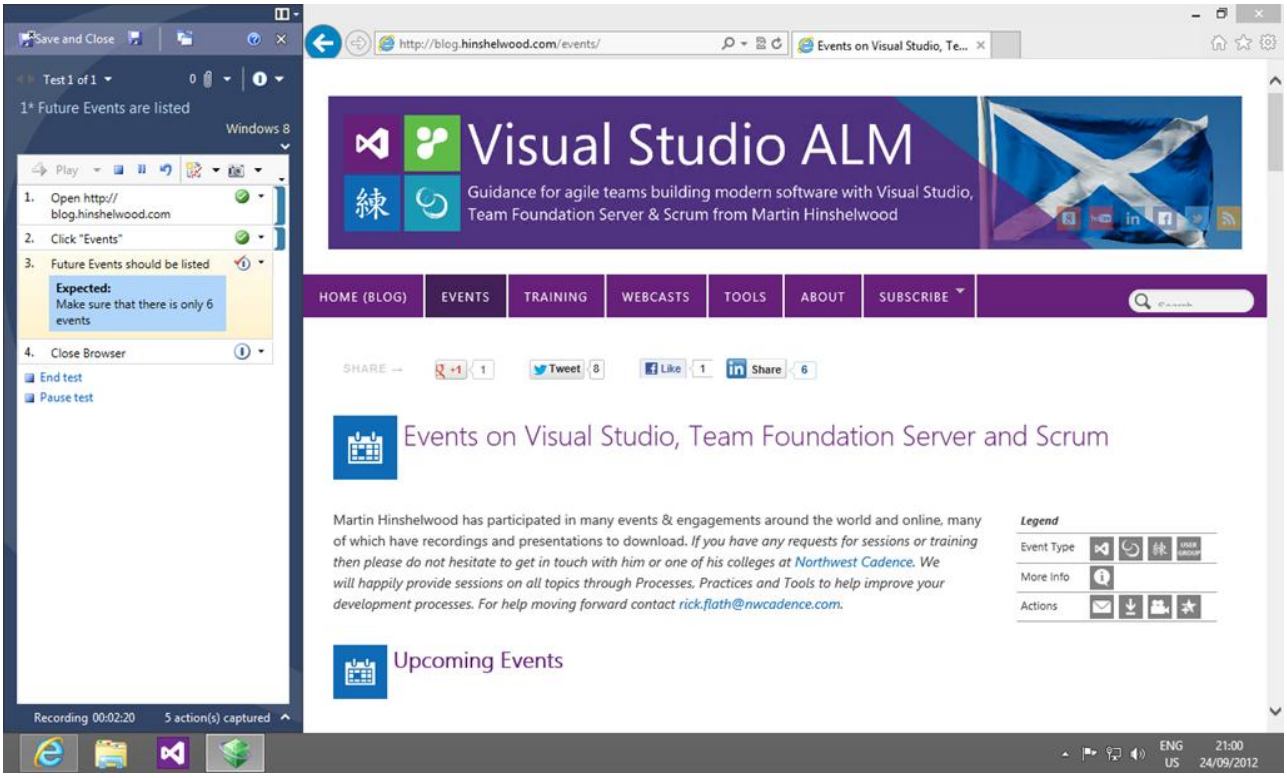


Figure: Microsoft Test Runner in action

And if while running one of your test cases you encounter a nasty little bug, you can have a rich actionable bug generated automatically making it easy for developers to isolate and find the problem.

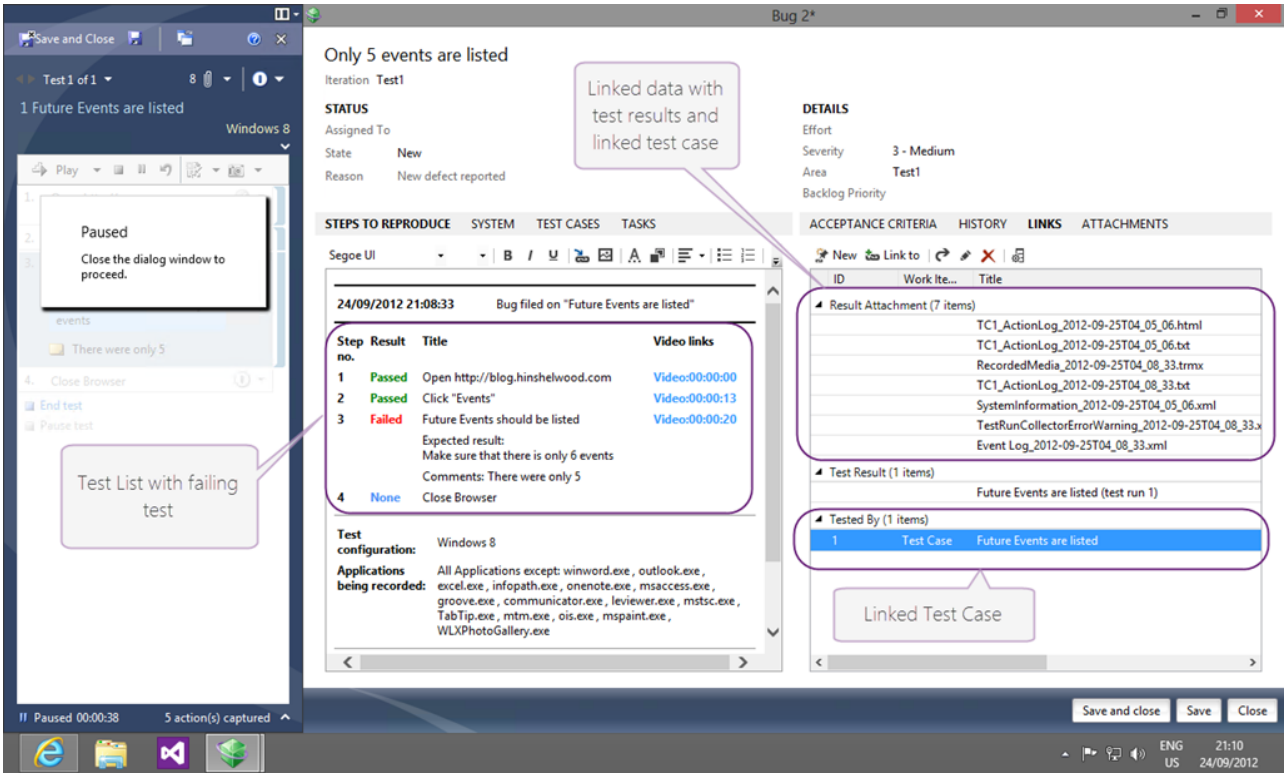


Figure: Microsoft Test Runner created Bugs from nowhere

The tool is in fact that good, as long as you are using managed code. The further you get from the modern development tools the less of these features will be available although you will always get the ability to run test cases and record the data.

This is one of the best test tools in the industry and one of the few that help the manual tester.

Solution: Web Test Manager from Sela

In an effort to allow users more flexibility in managing their test cases and to allow the test cases to be run against any environment Sela have created their Web Test Manager which takes the entire Test Runner into the web.

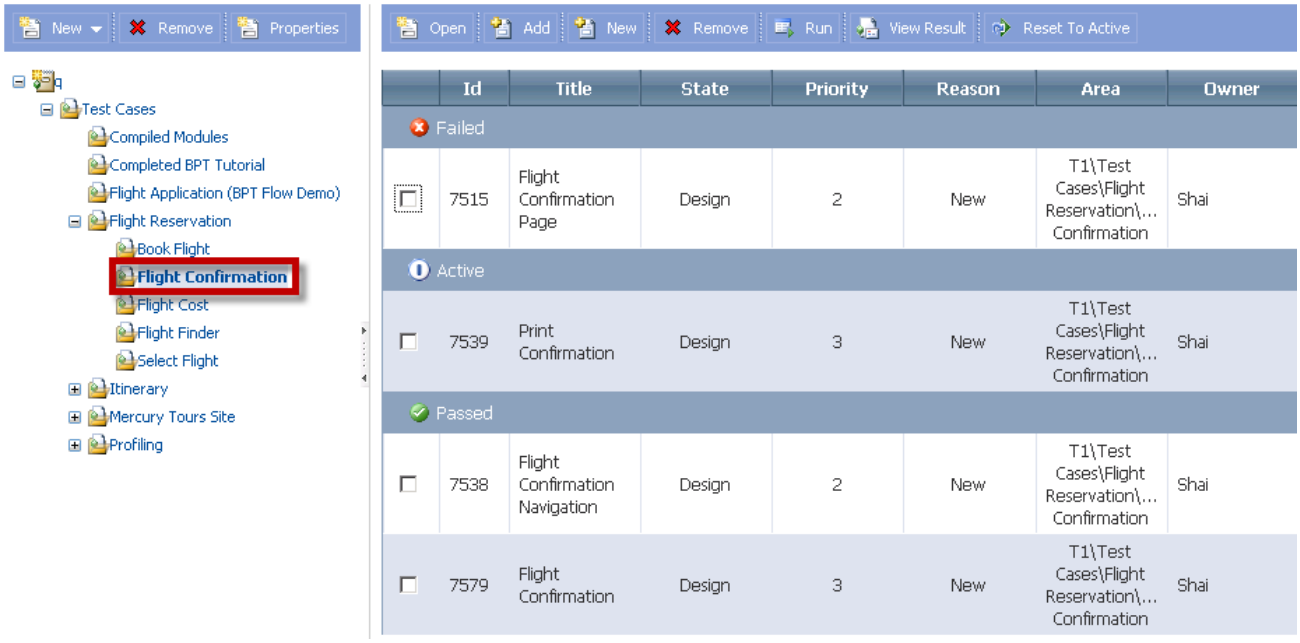


Figure: Managing your Plans and Suites from the web

Not only can you manage your plans and suites but they have also built a platform independent test runner so you can run your tests on Linux and against any browser.

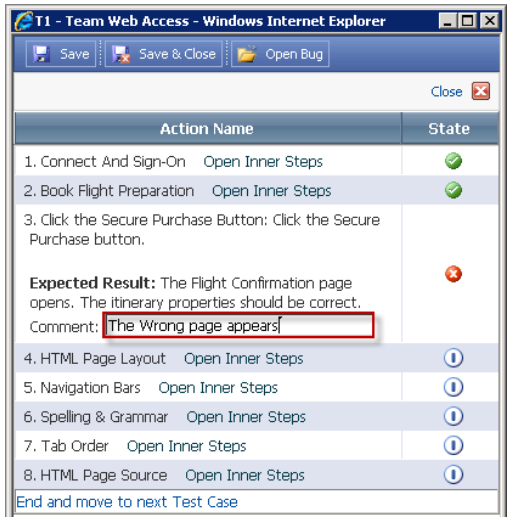


Figure: Edit and Run test cases from the web

And what solution would be complete without the ability to see which tests have been passing and failing.

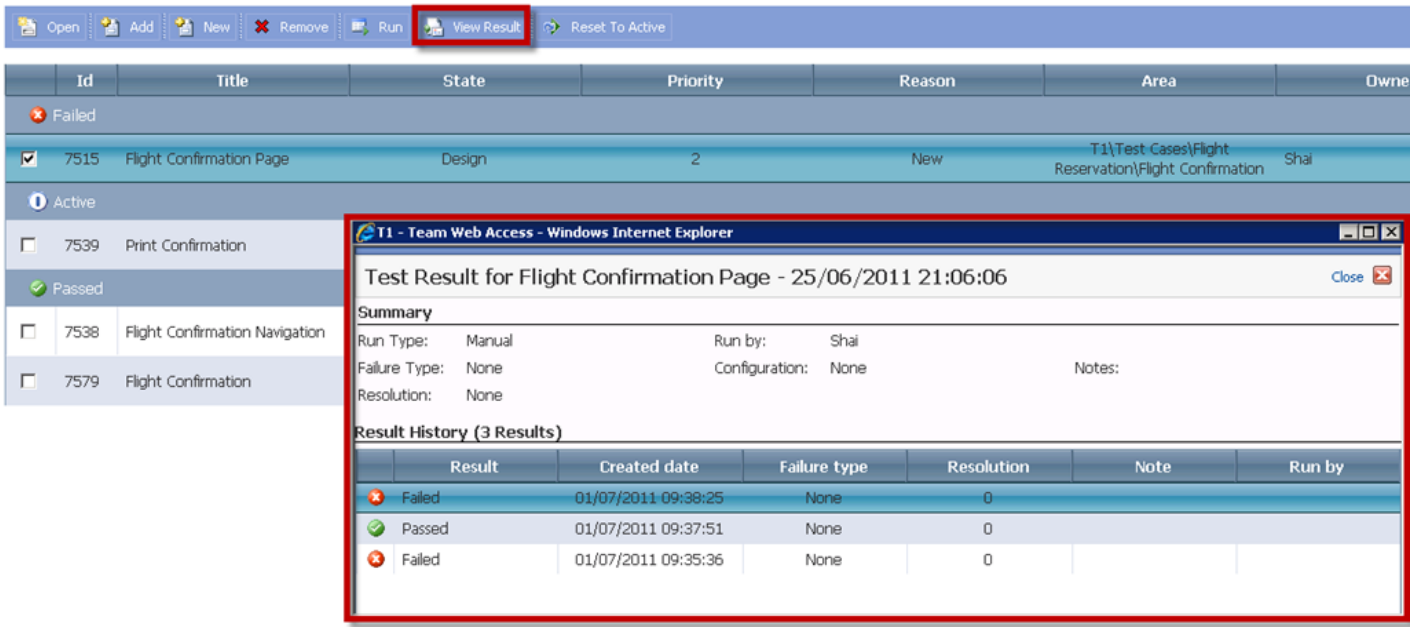


Figure: View test runs and results in the web

There are still some rough edges, but Sela shows what can be done for testers leveraging the rich data management system within Team Foundation Server that links our Test Runs to Builds and requirements.

Conclusion

Even if you are only interested in moving away from Excel and Word for your test management, Microsoft Test Manager is the best Manual Testing Management system. The integration that it has with Build and Work Item Tracking will make it invaluable for those teams seeking to increase their agility. Do not think that these are in any way silver bullets, and while the features are fantastic you may need to make subtle changes to your workflow or your software to take advantage of all of the awesomeness.

The ultimate goal is to try and reduce the all to common scenario, that we have not met the requirements or that we have not met the users expectations. The Manual Testing tools in Team Foundation Server can help you be providing a framework to have acceptance criteria begin to drive your engineering efforts and more frequently meet your customers expectations and reduce the waste inherent in software development.

Martin Hinshelwood is an ALM Consultant at Northwest Cadence, and Visual Studio ALM MVP (2013). He was named ALM MVP of the year 2011, ALM Ranger Champion for 2011 and was one of the first Professional Scrum Trainers with Scrum.org. Martin regularly delivers consulting and training while writing for his Visual Studio ALM blog, and is a sought-after speaker.

Continuous Delivery: Delivering IT to the Always Up-to-date User

Andrew Phillips
XebiaLabs

Trying to serve today's always connected, always updated IT user with anything less than a fully automated application delivery process makes no sense. Software Delivery is evolving from an effort-intensive, error-prone "big bang" activity to a seamless, incremental and continuous process that is proving to be faster, more reliable and more efficient.

With today's always-connected user, able to access your services all the time, everywhere, and to evaluate your competitors' offerings with a few taps or clicks, the business of service delivery is never "done". Your service is expected to be always available and continuously improving.

Businesses understand that they need to be "always-on". This requires constant, consistent application and service delivery to ensure a positive, seamless, rich customer experience.

Certain software delivery and integration strategies are central to allowing your organization's offering to be reliable and engaging, meeting and exceeding the high expectations of today's audience that demands its content and applications be constantly available and instantly gratifying.

Expanding on from the widespread adoption of Agile development practices in the enterprise, the leading strategy to achieve this today is *Continuous Delivery*.

Continuous vs. Traditional Manual Delivery

In most organizations, the software delivery process today involves many manual or semi-automated steps, requiring a significant investment in terms of scripting and troubleshooting. Further, the delivery and release process followed varies widely across departments, applications and technologies, leading to error-prone, non-transferable activities and costs that increase in direct proportion to the release volume.

Continuous Delivery is a fundamental shift in the software development and delivery process, based on a simplified release management practice that unifies development, production testing, QA, testing and operations and production in a risk-reduced process that delivers higher-quality software in a seamless and steady flow.

By embracing Continuous Delivery, organizations are creating symbiotic IT departments that work together more seamlessly, providing higher

output quality more consistently. Through Continuous Delivery, we move toward greater automation and fewer manual activities, resulting in fewer errors and significant productivity gains.

How does Continuous Delivery help?

Continuous Delivery is the essential, automated software delivery strategy for today's "always-on" business. Two key elements are required in order for the IT Department to achieve CD:

1. Work toward creating a standard process or "pipeline" of moving software from development to production
2. Move software through this pipeline on an iterative basis, in smaller chunks

As a result, IT reduces risk in the delivery process, gains the ability to be more reactive throughout the software delivery process and offers insight to help inform and guide the next feature set. Ultimately, with Continuous Delivery in place, organizations can create a fully automated software delivery process with new software "rolling off the assembly line" seamlessly and continuously.

Building Things People Need

A key benefit of Continuous Delivery (and a big budget saver!) is in helping ensure investment is focused on features with the greatest customer benefit.

Today, many companies rely on market research as the basis for change. Only after the full investment has been made in the completed product will the success or failure of your product become apparent.

With Continuous Delivery, ongoing investments are capitalized due to the regular release of new features. Customer response is gauged throughout the process, determining which features are more heavily utilized by users and which should not be developed further. Market demand is a driver for building a product that's better aligned with your audience, leading to a more enthusiastic customer response.

The quality level of your application is improved as well. With manual application delivery, tests are often not performed until the last development stages; bugs are difficult to track so late in the game, and there is a greater level of reluctance to invest more time and money into resolving any issues as release deadlines loom.

With an automated testing process in place, any coding issues become apparent right away, and are easily fixed while they're still small. Applications are deployed with a greater level of reliability and much more quickly.

Conquering Existing Challenges

The idea of continuing to practice manual deployment in the face of today's market pressure to deliver immediacy no longer makes sense. In fact, application delivery has become an increasingly stressful and time-consuming process, making new software development increasingly costly as well. Waiting until an application goes live to address any bugs is risky, leading to additional, expensive release cycles.

With automated releases, the high stress levels and unsustainable investment suddenly level out. New features can be added to your online service in hours rather than days or weeks, and the features that are added are better, faster and more reliable. They're also much less expensive to create and distribute.

Finally, and perhaps most importantly, with continuous application delivery, customers will have access to increased functionality almost instantly, and on a continual basis. With manual processes eliminated or reduced to the bare minimum, and automated methods eliminating a greater percentage of errors at a

faster pace, features will land in your customers' hands much more quickly, and with a far lower failure rate. This delivers not only the latest and greatest your company has to offer, but also aligns the continuous delivery of software with today's connected culture giving rise to a seamless customer experience and greater innovation.

Making the Transition

Beginning the process of changing over to Continuous Delivery begins with an honest assessment of your existing model and your future needs. This will help you determine the best order and approach for your implementation, and give rise to realistic expectations and deadlines.

Engaging your development team is an essential element to ensuring a positive shift into automated application delivery. Not only will their insight and experience be invaluable for understanding the challenges you have to overcome, their participation is required to accurately determine the best strategies for optimizing a new process.

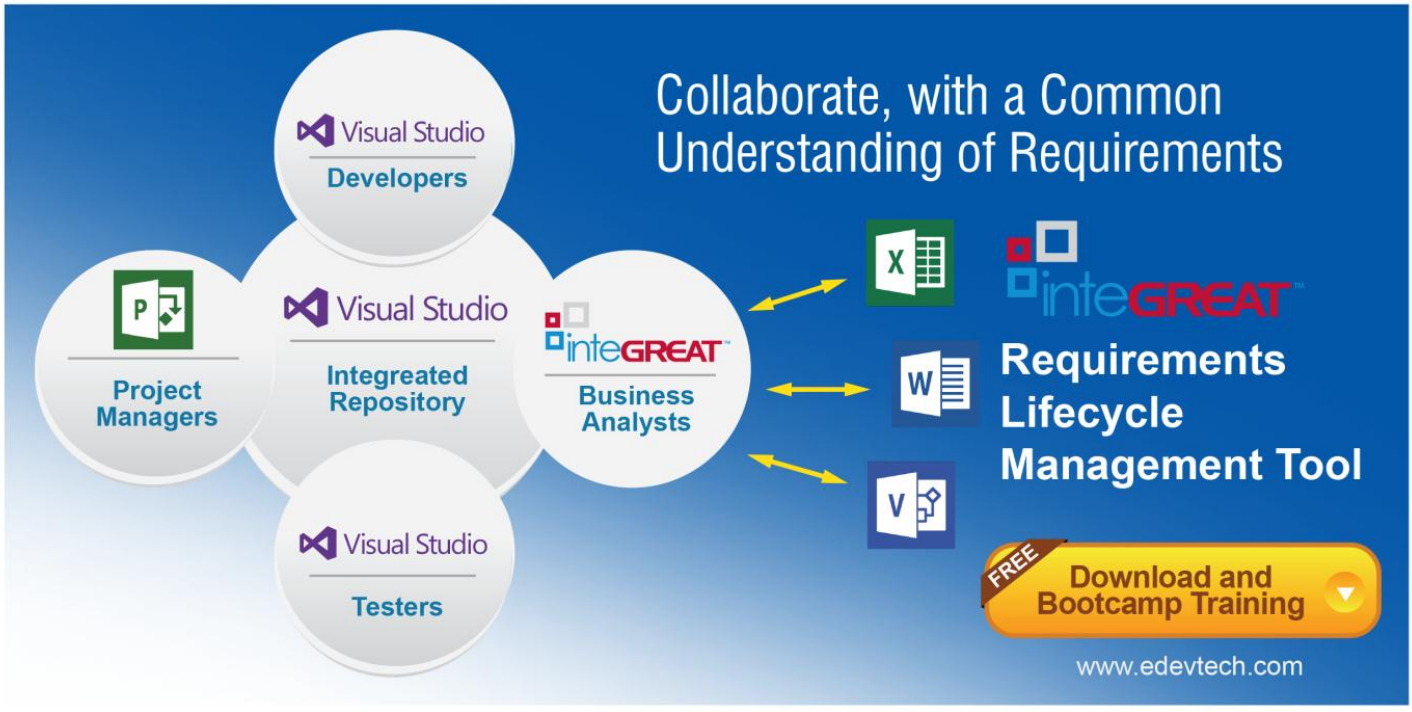
The next step is to review technologies that underpin a move to Continuous Delivery, including tools for Continuous Integration, automated testing, provisioning and automated deployment. Oftentimes, organizations are finding they have some of these tools in-house already, and it is a matter of adding some final pieces to make the move to Continuous Delivery.

Once the elements are all capable of working together in order to deliver improved functionality, continuous application delivery becomes the new standard.

Andrew Phillips is Vice President, Product Management at XebiaLabs. a global provider of Application Release Automation offering a Continuous Delivery platform to help enterprises improve and accelerate the application delivery process. To get started moving to Continuous Delivery, Read the new white paper.

Modern Requirements Management

inteGREAT™
Simply Comprehensive



inteGREAT & TFS share a single database

Key Features that extend TFS capabilities

Adaptive project methodology support	Completely configurable taxonomy	Project baselining	Automatic document generation
Automatic test case generation	Reusability across projects	Impact assessment	Author and review in WORD
Simulations and on demand Wireframes	Non-Functional requirement library	Automated Function Point estimation	Specification Checker to validate requirements
Wizard authoring for completeness	Legacy Word & Excel import	Conceptual data model creation	Drag & drop traceability

Index Page

Deployment Automation Basics

Implementing Consistent Deployments Across Environments

Eric Minick
Urbancode

IT organizations are finding it difficult to release more quickly without adding more people and incurring additional risk. Here we discuss the limitations of manual deployments, common failure patterns, the benefits of automation, and a base set of features an automated system should provide.

Key Findings

Manual deployments are inherently slow and error prone. Deployment automation used only in development or only in operations may help one silo, but leads to a hand-off where changes to the process may be insufficiently communicated.

Automated deployments provide superior audit trails.

An automated deployment infrastructure provides a framework to build upon. Additional activities such as automated functional testing can leverage the deployment infrastructure.

Deployments standardized across environments must still take into account environmental differences. The environment configuration is a key concern.

Introduction

In a continuing poll, over 75% of respondents identified their deployment process as either "Entirely Manual," Mostly Manual," or "Mostly Scripted." Relatively few indicated that they had push-button deployment capabilities. This result is not surprising: Until recently, deployment automation options have been sparse, with vendors offering "solutions" that do not scale to the enterprise; require considerable process reengineering to implement; and provide little visibility into the "who, what, when, where, why and how" of deployments.

Companies have turned to internal solutions, and more scripting, to help their team members achieve more consistent deployments and keep pace with the increasing demands of competitive markets. However, as with many "home grown" solutions, the expense ratio, measured in both time and dollars, often proves high: Most organizations are not in the business of providing "process automation," and have to borrow

resources in order to design, create, implement and maintain an automation initiative. Often, even after initial success, the solution does not stand the test of time - changes in practices, such as the introduction of Agile, require a complete rewrite of the initiative.

Considering the options, companies attempt to improve human-driven deployments: it is difficult to invest in an automation initiative when perceived gains do not outweigh risks, when the options seem to provide only marginal returns. However, these incremental improvements do not sufficiently address the underlying problems of manual deployments: they still rely on a human to log into a machine, consistently run scripts in a precise order, and then notify interested parties.

This situation seems to lead to a single question: "Is automation really worth the time and effort?" If you ask teams with automated deployments, the short answer will most likely be similar to, "Yes ... but it depends on what you mean by automated." As it turns out, a scripted process is not really an automated process (as we will discuss below). For those in the business of providing automation, a much higher threshold is set: If the "automation" does not address the patterns of deployment failure, then the process is not automated.

Failure Patterns in Manual Deployments

Using the rough and ready definition of manual deployments, it is easy to see where they fail. It is safe to say that any deployment is manual if it is:

- characterized by operators logging into various machines and following written scripts;
- slow and inconsistent;
- prone to error and failure; and
- lacking in visibility and traceability.

The first three are rather straightforward. However, the final criterion requires a better understanding of what happens to a deployment once the process is "complete.“ This is because a deployment process, is not a separate and independent event in time: once the deployment is run, team members still need to know what was deployed where, why it was deployed, and who performed the deployment – especially if a failure occurred. For example, if a push-button deployment exists but the system does not provide visibility into the process, team members have to sift through logs on numerous machines to track down the error - which is not just manual but also laborious.

Keeping in mind this extended definition of a deployment may help as we identify a pattern of practices, described below, that show why manual deployments are slow, error prone, and "black boxes." In turn, these .. “bad" practices will provide insight into what is needed from a system that will operate quickly, be highly successful, and provide high traceability (i.e., an automated system).

Handoffs through Email or Drop-boxes

Because the person executing the deployment varies from environment to environment, email is often used to inform another team that they should deploy “something”. It is not uncommon for the files to be deployed to be attached to the email or placed in a designated location on the network (i.e., a drop box).

Lag, inherent to this process, is a main reason why manual deployments are slow. A considerable amount of time can pass between sending a request for deployment and the recipient actually reading the email. In a perfect world, the deployment will take place as soon as the email is read; however, in the real world the actual deployment will take place after an additional delay.

Because different people request and execute the deployments in each environment, inconsistency is built-in, raising the risk of failures in production deployments. Individuals will respond to requests and interpret installation instructions differently.

Traceability is limited. The core audit trail is the series of emails scattered between many inboxes, making complete visibility virtually impossible. Further, because file shares (or attachments) are used to deliver the artifacts for deployment, the organization must deal with the risk that those files are tampered with, changed between build and various deployments, or just don't make it into the correct deployment.

Development, Test and Operations Deploy Differently

Developers see manually deploying to a private test environment slowly and carefully as inefficient and burdensome, so they will often create short cuts or scripts to deploy. If the Operations team has scripts, they tend to be created by operations for operations. Test teams have the desire to move quickly like developers. Between their desire to avoid false bugs and test the deployment process, they also want to mirror the deliberate, controlled production deployments. Unfortunately, test teams have little pull on developers or production support and often invent a third way of deploying.

Because inconsistency across environments is built into the process, repeated failures, on top of the duplication of work, are inevitable.

Adjustments to Deployment Process Communicated on "Asbroken“ Basis

When each team is deploying differently, another nefarious pattern emerges: When minor adjustments are made to the deployment process, those tweaks are communicated to other teams only in response to a breakage. It is common for a test team to report that the application is not working, or to file a bug. Time is wasted due to a broken deployment. At some point, the development team eventually realizes that they changed a setting, or added a step to the deployment process. This is eventually communicated to test teams that hurriedly apply the change and try to catch up on their testing. The same pattern repeats itself when neither development nor test team remembers to tell operations of the change and the deployment fails in production.

Deployment Steps Contained in Large Document

Complicated, manual deployments require careful instructions - often in the form of large deployment documents. The documents take many forms: from a Microsoft Word manual, to a spreadsheet, to a wiki. Often, there will be dozens, hundreds, or even thousands of steps that must be assigned to someone on the team and then precisely executed.

It is very difficult to correctly document and record hundreds of steps. Likewise, executing those steps, correctly and exactly in the right order, is difficult and time consuming. The process inevitably becomes slow and prone to failure when the deployment instructions become lengthy.

Developers Attain Production Passwords

Many of these patterns have lead to production deployment failures, which leads to another failure. When there is trouble in production, the highest priority of the deployment team is to repair production. Developers who know the system best are often brought in to explore it, figure out which of the steps in the large document were skipped, or what tweak was not communicated. More often than expected, the

developers are simply given the production passwords so they can work more quickly. After all, fixing production is of the highest importance.

When we interview a diverse set of people from the same organization and ask if developers have passwords for production, usually almost everyone will say "No. That would be absurd and violate our audit requirements.“ And yet, one developer will say, "Yes, I have passwords to fix broken deployments.“ Because the passwords are handed over in a stressful time, and usually late at night, the deployment team often forgets they gave out the password. The result is a breakdown in control, audit capability and traceability.

If They Are so Bad, Why Are Manual Deployments Common?

Manual deployments are slow, painful and fail in production; yet, they are still extremely common in the IT industry. Why? A simple answer may be that until recently, powerful deployment automation tooling was not available.

However, even today, there is a great deal of resistance to automating deployments. Deployment teams remain comfortable with their existing practices. Because they are at the keyboard executing the manual steps, they feel in control. They immediately see all the output and can respond to problems appropriately. Operators' feeling of control, along with organizational inertia, prolongs the use of manual deployments even when those not performing the deployment see serious problems.

Also common is the notion that "discipline" and "consistency“ can remove the patterns of failure associated with manual deployments - but this is not the case. The problems that we see with manual deployments are often exposed by human error. If discipline and consistency were the answers, then there would be no need to communicate on an "as broken" basis if every change process change is properly documented; however, even the most disciplined organizations rely on this method – humans are not the best at performing rote activities. Likewise, executing dozens of steps correctly every time would not be a problem – if only proper attention were paid. The reality is that no matter how attentive a person is, they will eventually error. Finally, the notion that a manual deployment process would not be as slow if everyone involved executed each step immediately when the previous one completed is unrealistic: How many people sit around waiting for something to do? More often than not, when that deployment request comes in, the recipient is busy doing something else.

Unfortunately, and as many studies have shown, people struggle to be consistent over time. Instead, humans are better at utilizing their creativity than following a script. To demand that people perform their *tasks* mechanically, and absolutely consistently, is to demand something against their nature and an invitation to failure.

Unlike humans, machines have yet to perfect the art of creative thinking; however, they are excellent at acting, well, mechanically: they have proven their ability to perform repetitive tasks, to let us know when something has gone wrong. and have the "discipline and" "consistency" to perform those tasks that hamper human performance.

So, if a task requires consistency and discipline, why not delegate it to a machine? Why not free up people to do what they are good at, and gain positive returns in saved time and improved performance?

Elements of an Automated Solution

Manual deployments are broken and cannot be saved by more disciplined deployment engineers. The rote work of executing a deployment should be delegated to an automated system that can execute a process consistently; however, not all automated solutions are alike. Whether you are looking to buy or build a deployment automation system, there are some "must have" goals to keep in mind. A mature deployment system should be characterized by:

- *reliable, highly successful deployments - especially in production;*
- *easy deployments - encouraging teams to take new builds faster;*
- *fast deployments -- allowing early test environments to be on the newest build as quickly as possible;*
- *complete audit trail - spanning across all environments; and*
- *robust security and separation of duties - controlling who can do what, when and where.*

To achieve these goals, an automation solution should contain at least a few key elements:

- *the entire deployment should be automated;*
- *the deployment should target specific environments and automatically adapt itself to a target environment;*
- *the files being deployed should come from a controlled artifact repository; and*
- *security and visibility should underpin the entire system.*

One Complete Definition of Deployment Process –Automated

The first element of a successful solution is to translate the large document containing the deployment steps into an automated system. The same basic process for deployment must be used through all the test environments and production. Likewise, only the automated system should be used for deployments, and deployment errors should be corrected by fixing the automation and performing the deployment again.

Using the same process in all environments, and only fixing failures through redeployments ensures that when a change to the process is needed in an early test environment, the automation is updated. Tweaks are no longer communicated on an "as broken,. basis, but are instead immediately incorporated. The deployment plan is always up to date.

Because a (consistent) machine is executing that plan, it is always followed precisely and with no lag time between steps.

Formal Environment Definitions

To use the same process for all environments requires factoring out all environment specific information. This element is critical and a formal definition of an environment can capture key information including:

- *what servers does each component of the application get deployed to;*
- *preconditions for deployment such as approval requirements; and*
- *parameter variances (i.e., what's the database password in this environment?) --parameters, like passwords, should be handled securely while others are more visible.*

Combining a generic deployment plan with the environment to deploy to should yield the full deployment configuration.

Repository for Deployable Artifacts

Traceability is important and deploying "whatever happens to be on the file share" lacks traceability and increases risk. One needs to know that not only has the deployment process worked, it has worked with the files that are about to be deployed. One (or more) authoritative repositories should be employed to provide a definitive source of uniquely addressable file versions.

These repositories should be integrated into, or natively accessible from, the deployment automation system. Example repositories include better build servers, maven repositories, asset management systems, or even the more robust deployment tools.

Inventory and Reporting

When the deployment automation framework is the sole deployer of an application, it knows what versions of each application are in each environment This information can be leveraged to provide clear information about which build is where, as well as optimize system deployments, skipping deployments of components already present in the environment.

Access Control

In manual deployments, deployment access control has been equivalent to server access control. With an automated system in place, that is no longer true. Access to the automation provides deployment rights. The automation systems must themselves be secure and provide role-based security.

This approach provides new capabilities. An environment owner, who may only have approved manual deployments, but lacked skill or permissions to execute them, can now be provided self service deployment capabilities. Likewise, it may be appropriate to revoke permissions from some users who were granted broad permissions to servers in order to perform deployments but were instructed to not otherwise modify the servers.

Audit Trail

In many manual deployments, the audit trail consists only of what people think they did, or recorded. In some, extra logging is added in production. Others require those who execute deployments to take screenshots along the way. An automated system should provide full logging in all environments, automatically, every time. It should be clear what steps were executed as well as who triggered each deployment.

Maximizing Automation Investments

An automated deployment infrastructure is something the team can build upon. Smoke tests can be integrated so that rollbacks occur automatically on test failure. Likewise, application deployment can be coupled with automated updates to application and application server configuration. Deployments to test environments can be automatic based on standing schedules or events.

Once deployment automation is in place, these sort of additional practices can be layered on top. The deployment infrastructure provides the mechanics and traceability. People provide the creative insight to understand what would most benefit their organizations.

Eric Minick is a lead consultant at UrbanCode where he helps customers get the most out of their build, deploy and release processes. Eric has been at the forefront of Continuous Delivery for 8+ years in roles ranging from developer to tester to release consultant.

It's Time for Test Management in the Cloud

Mush Honda
QA Director
KMS Technology

When it comes to testing, what's up with the cloud?

Cloud services have really captured the imagination of the business world. According to a Gartner survey 71% of organizations have adopted SaaS (Software as a Service) within the last three years.

Quality Assurance teams however, seem to have missed the boat, barely leveraging the benefits of the cloud. While enterprises and their so-called "QA Verified" software may be cloud-based, Test Management itself generally isn't.

Advantages of Test Management in the cloud

The benefits a cloud-based approach brings are plentiful and are particularly pertinent for QA teams. Top of the list for most companies adopting SaaS is the cost-savings. The ability to eradicate setup and ongoing maintenance costs for your QA toolset and data storage is most welcome. Working via a browser means no special software/hardware overhead.

Consider the nature of most test projects and you'll find that they involve disparate groups working in remote locations, often around the clock. A central repository that remains accessible for everyone with solid SLAs (service level agreements) ensuring zero or minimal downtime is a vast improvement over the traditional approach. This increase in reliability comes alongside solid security encryption for peace of mind.

A snapshot of cloud-application benefits include:

- No on-going infrastructure management, operational or installation costs incurred. Backup and recovery is relatively simpler and cheaper than traditional methods, as service providers offer recovery services;
- Access to applications is location-independent (e.g., office, home, hotel, airport, customer site)
- Ability to always run on the latest application release from vendors; thus can quickly leverage new functionality;
- Subscription-based licensing model requires less upfront investment; no lock-in with any particular vendor.

What to look for in the cloud

First and foremost you want your cloud-based test management to enhance workflow and streamline processes for greater efficiency. One of

the first things worth considering is integration. Can you integrate your existing bug tracking software? Are there any plug-ins or browser based tools that can help generate logs and record screenshots to create clear and concise bug reports? Can you easily import and export documents, deliverables, log files, images and other files? Can you set permissions levels, make bug status changes, and see real-time updates? Does it support automated test scripts?

It's also important to think about versioning and tracking. Every action should be traceable and the ability to revert when something needs to be rolled back can prove to be a real time-saver.

Communication within the team is the key to success in software development. The greater the clarity between developers and testers, the greater the chance of a polished product at the end. It is vital that your test management application has a good commenting system and real-time messaging support. You'll also want to be able to set up email notifications based on specified triggers for different staff members or groups.

Since you're putting all this data (and effort) into a test repository, the ability to re-organize it is also critical. You want to be able to configure and customize reports, filter data, and create metrics to measure progress. Being able to generate a quick snapshot of your team's progress will help you to satisfy the vendor and your own management team.

Besides the obvious functional efficiencies, the following infrastructure reviews are also recommended:

- Review services offered by competitive cloud service providers hosting the tool;
- Review the security and data encryption measures in place, to prevent malicious attacks on your data;
- Review performance and availability guarantees;
- Review available Control Interface options offered by the provider, to further optimize and configure the tool's deployment.

Important questions to ask

If you are assessing a cloud-based test management solution, then a few questions need to be asked. Make sure you check the SLAs carefully to avoid any unexpected trouble down the line. If you encounter a problem it is the provider's responsibility to ensure that things don't grind to a halt because a server goes down.

Make sure that you ask about encryption standards and security protocols. You need a solution that you can trust, but you may also need to be able to detail those security details for your own clients.

If development is ongoing then ensure that you know what will happen in the event that the vendor rolls out a new version. You should be able to migrate your data easily, if the need ever arises. It's important not to get locked in.

What are you waiting for?

QA may be set in its ways, but the potential of cloud-based test management is clear to see. With benefits that range from cost savings to greater efficiency, isn't it about time that you took a closer look? A few cloud-based QA management solutions you may want to check out: qTest by QASymphony, Test Rails, SpiraTest, and Zephyr. Your budget and feature needs will dictate which one suits you best.

Mush Honda is QA Director at KMS Technology, a provider of IT services across the software development lifecycle with offices in Atlanta, GA and Ho Chi Minh City, Vietnam. He has previously played various QA roles at Ernst & Young, Nexidia, Colibrium Partners and Connecture. KMS services include application management, testing, support, professional services and staff augmentation.

Empowering Rapid Delivery of Quality Software

The e-Magazine for:

ALM - Application Lifecycle Management

DevOps - Development & Operations

Agile - Scrum, Kanban, etc.

TFS - Team Foundation Server

Click for more Details...

ALM Mag Brings You Each Month:

- **Articles, Whitepapers, Case Studies and Videos and Audio** providing:
  - Advance notice and Previews of new versions,
  - **Reviews** of new versions and updates,
  - **How To** and **Step-by-Step Guides**,
  - **Tips** highlighting **Primary Tools** and
  - **3rd Party Integrated or Bridging Tools**,
  - **Processes, Books and Events**
- **Surveys** to discover more about the types of content that interest you so we can serve you better
- **Industry Study and Survey Results** to provide you more insight into what others in your community are doing
- **Interviews with top ALM Leaders** to help you discover what's coming and how you can get there
- **Tutorials and Training Courses** to increase your knowledge and leverage your career
- **Event and Conference Information** with a spotlight on Details, Agenda and Calendar so you are aware of what you want to attend to increase your career potential
- ...and finally **IT Humor** (yes it does exist) including **Cartoons**

and more...

This Magazine For You If You Are:

- a CEO, CIO, COO
- an Application or Development or QA or DBA Manager or Architect
- a Development or QA Lead
- a Product Owner
- a Project Manager
- a Scrum Master
- a Business Analyst
- a Tester
- a Developer
- ...Or anyone involved in software Planning, Development, Testing, Infrastructure Provisioning or Deployment

Subscribe

Index Page

Scrum – It's STILL not a Silver Bullet!

Mike Vizdos
Certified Scrum Trainer
Agile Consultant & Coach
implementingscrum.com

Scrum is not a Silver Bullet that will fix everything in your organization. I've been saying — and writing — about this for many years (look at the copyright of this cartoon!). Take a look at some of the themes in this magazine (DevOps and more), along with the different tools and methodologies that offer possible promises to fix what hurts today (including TFS, Scrum, and others).

My suggestion is to continue reading, reviewing and digesting this information.

It is good stuff; however, don't think for a moment that anything will be a Silver Bullet in your organization.

There is no such thing as a Silver Bullet. Remember this.

If something sounds like it is to good to be true, it probably is. We all know these statements make logical sense… but in reality we do not always deal with logic. If you are looking for a quick fix, one does not exist. Change is hard. Deal with it. In reality.

For Reprints of this page
And previous strips

Click Here

Michael Vizdos is a world traveler, author, Scrum & Agile Consultant, speaker, and the person behind implementingscrum.com.

ALM Mag - Vol. 1, Issue 3 March 2013

ALM Mag.com

Empowering Rapid Delivery of Quality Software

Examples of What You'll See in Each Issue

TFS Migrations
2008 & 2010 to 2012

Audio Podcasts
Technology & Process with the Experts

ALM Consulting Update

Industry Case Studies

Industry Surveys

Interviews with Industry Leaders

Best NEW Features

Event Wrap Up & Review

More
BONUS
Video

***TFS - Beyond Source Control**

Index Page