SHOWCASING THOUGHT LEADERSHIP AND ADVANCES IN SOFTWARE TESTING

LogiGear MAGAZINE

Integrated Test Platforms

Test Architect for Visual Studio

Keyword testing for the popular ALM platform

Agile ALM and the Tools to Practice It

By Eric Landes

Tools for Integrated Test Platforms

By Michael Hackett

The Cure for Continuous Integration

By Bryce Johnson

September 2012 | Volume VI | Issue 4

LETTER FROM THE EDITOR

Editor in Chief
Michael Hackett

Managing Editor
Brian Letwin

Deputy Editor
Joe Luthy

Worldwide Offices

United States Headquarters
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +01 650 572 1400
Fax +01 650 572 2822

Viet Nam Headquarters
1A Phan Xich Long, Ward 2
Phu Nhuan District
Ho Chi Minh City
Tel +84 8 3995 4072
Fax +84 8 3995 4076

Viet Nam, Da Nang
7th Floor, Dana Book building
76-78 Bach Dang
Hai Chau District
Tel +84 511 3655 33
Fax +84 511 3655 336

www.logigear.com
www.logigear.vn
www.logigearmagazine.com

Submission guidelines are located at http://www.logigear.com/magazine/issue/news/editorial-calendar-and-submission-guidelines/

There has been a tectonic shift in software development tools in just the past few years. Agile practices and increasingly distributed teams have been significant factors but, in my opinion, the main reason is a new and more intense focus on tools for testing driven by more complex software and shorter development cycles.

There have always been developer tools and platforms (IDEs) and they are getting better. There have always been project management tools and now they are extending further across teams and becoming more integrated.

For a long while testing stood alone. Most of the focus was on integrating bug trackers into team tools, but that's typically where it stopped. The first generation of integrated tools were cumbersome and poorly documented, and their lack of sophistication failed to provide significant benefits to the whole team or to make product development faster.

A lot has changed. Stand-alone tools are on the road to obsolescence. The direction today is full integration and transparency. Rapid development and deployment demands greater automation, communication, focus, and information transfer. To achieve this, integrated test platforms at a minimum need to include test case managers, automation tools, bug tracking, and automatic reporting that can fully integrate with and leverage development platform tools and reporting.

New tool technology is contributing to improved development practices. XP's Continuous Integration (CI) practice is based on autobuilds from source control using an integrated bot or agent, automated re-running of unit, integration and regression test, and automatic notification of code changes, bug fixes, and automated test failures. Continuous Integration enabled by better tools is now separating successful teams from *Scrumbuts* or more aptly—slow, less than efficient teams.

We have often stressed testers being integrated by co-locating in the same work area as developers to better integrate in the planning and estimation process, and have better access to product owners and customers. Ideally, this paves the way to becoming more technically integrated and valuable. This leads to teams that are cross-/multi-functional and have a bigger variety of quality work. The result of this type of integration is more gray box, and earlier test execution, and ultimately better product quality.

For your career growth and to better support your product teams, learning more about integrated test platforms, ALMs, CIs and the varieties of tools and practices that go along with them will be a great benefit. This issue focuses on the tools and practices that will help and support all of us in providing rapid and higher quality information to the rest of the product development team.

In this issue, Eric Landes looks at the benefits of using Agile ALM; LogiGear Marketing Director, Joe Luthy presents a keyword based testing tool for Visual Studio; an article from Software Consortium examines recent changes to the ALM toolbox; I talk about the importance of integrated test platforms; Bryce Johnson shows us how to speed up automated testing and finally, Tad Anderson reviews the book *Continuous Integration in .NET.*

I hope you find this issue full of information about the importance, growth and use of integrated test platforms, and that you learn a few tips about Continuous Integration.

Next issue? Mobile Test Automation!

Michael Hackett
Senior Vice President
Editor in Chief

IN THIS ISSUE

IN THE NEWS

VISTACON 2013 ANNOUNCED

In 2011, LogiGear partnered with software testing experts from companies such as Microsoft and EA to create a unique conference - VISTACON. The conference featured numerous speakers, panels and even hands-on lab sessions with experts.

The conference was so successful that we're bringing it back again for 2013. Next year's conference will take place on April 18 & 19 in Ho Chi Minh City, Vietnam. We'll announce the final speaker list and venue in the upcoming months so be sure to check out www.VISTACON.vn for updates.

Don't miss a great chance to learn, share and network with other software professionals.

Stephen Copp Video Interview on Integrated Test Platforms

At VISTACON 2011, Michael Hackett sat down with EA's Stephen Copp to discuss the world of integrated test platforms.

Please click here to watch the interview:
http://www.youtube.com/watch?v=BnenVA-V52w

Stephen has over a decade of experience in test automation, unit testing, load and performance testing. He has spent several years in learning, fine tuning and taking advantage of open-source automation frameworks such as FitNesse. He has successfully implemented and used the framework with the latest technologies available for web testing for many major customer-facing applications. Along with his automation expertise, Stephen has also worked as QA project lead and QA Linux administrator. With that background, his passion has been about how to apply technologies to testing software projects in the real world. Stephen holds a B.S. in Computer Engineering with a minor in Mathematics from Santa Clara University.

Test Architect for Visual Studio Released

In collaboration with Microsoft, LogiGear developed TestArchitect for Visual Studio, a keyword-driven automation platform exclusively as a Visual Studio extension for efficient large-scale test automation. TestArchitect for Visual Studio uses module-based keyword test authoring to automate coded UI tests in Visual Studio 2012. Keyword-based test automation allows more team members to be involved in test design and execution. This dovetails with the direction of Visual Studio as a team collaboration and ALM tool. By encapsulating elements of the automation in keywords, the technique offers a natural way to eliminate unneeded details from test design to simplify test creation and makes large-scale tests long-term maintainable.

TestArchitect for Visual Studio is now available for anyone using the Ultimate or Premium versions of Visual Studio 2012. You can request a trial of TestArchitect for Visual Studio by going to www.testarchitect.com.

BLOGGER OF THE MONTH

Agile ALM and the Tools to Practice It

Armed with the right tool or set of tools, a development team can incorporate ALM into its Agile process and start reaping the benefits of Agile ALM.

By Eric Landes

As the software development industry matures, it is devising methods for ushering products from inception to completion—a process that has come to be known by the buzzword ALM (Application Lifecycle Management). Of course, development teams having been practicing ALM since software development began. The fancy acronym is the new thing!

For its part, Agile software development methods support all aspects of ALM. In fact, Agile puts ALM practices at the forefront of software engineering. This article explains how ALM can help organizations in general and then discusses how Agile practices utilize ALM.

The Elements of Application Lifecycle Management

Practices in ALM can be grouped into three areas:

- Governance
- Development
- Production

Within each area, the following practices can be applied to any development team's ALM strategy:

1. Requirements gathering.
2. Project management tools.
3. Source control.
4. Defect tracking.
5. Automated testing and more.

As software teams have grown, tools to manage these parts of the lifecycle have become mainstays for many organizations and an important consideration in the ALM space. But what makes an ALM tool or practice Agile?

Agile ALM

Because of the discipline that Agile methods have imposed on development teams, many tools have emerged to assist with those methods. As the Agile Manifesto states, "Individuals and interactions OVER processes and tools (that is, while we VALUE the items on the right, we VALUE the items on the left MORE)."

> *To ensure that interactions are valued more than tools, do not choose the tool and then make your process fit what the tool does. Have your team familiarize themselves with the Agile methods first.*

Clearly, Agile does not discourage the use of tools. What makes an organization's ALM practice(s) Agile is not the tools or which ALM practices it uses. Organizations need to utilize ALM practices only when it helps them to deliver value. Agile is not a silver bullet. It is a better way of developing software. So, making sure that individuals on the team interact frequently and do not rely on the tools over the interactions is one way to ensure that your ALM practice is Agile.

To ensure that interactions are valued more than tools, do not choose the tool and then make your process fit what the tool does. Have your team members familiarize themselves with the Agile methods first, using a team room if possible to conduct lots of high-bandwidth communication. Then use low-tech options such as cards on a wall and spreadsheets for your ALM requirements and planning tools until you need something more advanced. All the while, hone the interactions between your team and the business. When it becomes too painful to continue with the low-tech approaches, research the tool that allows you to eliminate the pain.

How Agile ALM Improves Planning

For some folks, planning their software development project means putting the plan in Microsoft Project (or Primavera or some other project management system). This involves a lot of initial planning, followed by deciding what gets done first, and then writing a plan that estimates far into the future what will get done.

For Agile teams, while comprehensive project plans can help to coordinate large initiatives, most planning takes the form of release plans with stories. The focus is on team collaboration. The planning happens within planning sessions that allow for changes to scope, resources or dates quickly. Using reporting mechanisms like burn down charts (release burn down charts show the progress of the team), velocity charts, and cumulative flow diagrams helps teams plan. Task boards also make the team's planning more transparent.

Planning also happens in a more collaborative way using Agile methods. Scrum, XP, Sprint, or iteration planning sessions allow the team to plan and commit to what they will get done within a fixed amount of time. This also allows them to adjust their plan quickly when things change (as they always do).

Here are some examples of tools that can be utilized for planning activities. This list is by no means complete, but these tools support Agile methods:

- Pivotal Tracker
- Version One
- Rally
- Lean Kit Kanban
- Microsoft ALM
- IBM Rational

How Agile ALM Improves Requirements Gathering

Agile teams rely on smaller scoped work items in their requirements gathering. Most teams utilize user stories to convey features that need to be built.

ALM tools must have a way to capture stories and use these stories in planning sessions. The easiest ALM tool one could use for basic requirements gathering is index cards on a big board. Using a board and index cards is a great way for a team to begin a project. It is also a great way for co-located teams to show requirements and track them (planning). With large distributed teams (large being more than 10 people), however, it can become difficult to use this method.

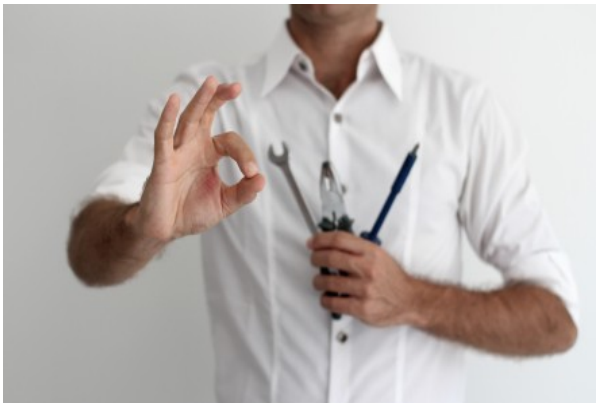
Other types of requirements include acceptance criteria. In Agile teams -- as in most teams – these take the form of wireframes, mockups, and some written format. Another twist that Agile teams use is automated acceptance criteria, tests that act as acceptance criteria for the team to develop against. Depending on team maturity, using an electronic tool to capture and keep user stories, mock ups, wireframes and Acceptance Criteria can help large distributed teams in requirements gathering. Teams would use the same tools for requirements gathering as they would for planning.

The Role of Version Control in Agile ALM

When writing code as a team you must have some type of version control system. This is a given nowadays for most software developers, but I still hear tales of teams that are using a file system as their version control. And I do not mean Visual Source Safe!

ALM practices – Agile in particular -- inform your version control practice. They require that your version control system allows you to get back to a particular version of software easily. Your version control system also should allow your team to work on new features while still allowing you to perform maintenance work if necessary. So you should establish a branching and merging strategy for your team.

This is not necessarily an Agile-only ALM practice; it is necessary for any ALM implementation. Your version control should not be an impediment to delivering frequently. Agile teams must have a way of doing continuous integration when a team member checks in code. This can be done with an integrated ALM solution like a Visual Studio ALM, or using an open source product like Jenkins.



Version Control Tools

Below are some examples of tools that can be utilized for version control. Again, this is only a partial list of (version control) tools that support Agile methods.

- Git
- Subversion
- Perforce
- Microsoft ALM
- IBM Rational

Quality Assurance in Agile ALM

An ALM strategy always contains an organization's quality plan. In an Agile ALM solution, this includes Test Driven Design or Test Driven Development (TDD). So automated unit tests are necessary, along with the ability to have some metrics on those automated tests. Let me emphasize that the metrics are not meant to be punitive; they are meant as information for the team. For example, using a metric like code coverage can tell a team that all the code they have developed has a coverage level of 86% against a goal of 95%. In that case, the team should examine the reasons why their coverage is at 86%.

However, the tool cannot directly tell the team whether those tests are good unit tests. Teams must perform code reviews through pair programming or some other mechanism to determine that. As previously mentioned, automated acceptance criteria can be used for quality checks as well. Some teams use tools for Behavior Driven Development (BDD), which can help in this regard too.

Most tools also include support for automating GUI testing. Since GUI testing is very fragile during some phases of development, use this feature with care. When your product has maintenance releases with new features, GUI testing can help reduce regression testing costs greatly. But if your team has good TDD and BDD tests in place there is not as much need for automated GUI testing. Most tools allow you to track metrics on these automated tests as well.

Whatever methodology or tool your team uses in your quality process, the process itself is another pillar in your ALM strategy.

QA Tools

Below is a partial list of tools that support Agile methods that can be utilized for quality assurance:

- Cucumber
- RSpec
- NUnit
- JUnit
- Raconteur

DevOps in Agile ALM

One aspect of development that gets overlooked is the involvement of infrastructure groups with a product. The DevOps approach encourages greater collaboration between the development team and the infrastructure departments, in particular the operations group. ALM practices work well with the DevOps concept because they prescribe stable, consistent development environments that help assure product quality and seamless deployment.

Different environments need to be set up depending on the product (Web facing, desktop, or mobile). For instance, a Web product team may need integration, QA, staging and production environments. If these environments are set up

in a haphazard way, your risk of introducing defects into production increases. The assumption is that the environments are set up in the same way. Keeping your environment stable and having a consistent, easily deployable build go a long way to eliminating variables between environments. This is critical to the DevOps concept, which includes a build system that takes the source control and builds and deploys it in a consistent manner on each environment.

Some Agile ALM practices in DevOps include continuous integration practices, where a team deploys to an environment once a day or sometimes every time they check in code. Some teams are using continuous deployment to quickly deploy to production, following all the good practices they want.

Picking ALM Tools

Choosing Agile ALM tools is a team decision. Development teams have different experiences and may be more productive when allowed to use the tools they select. I urge organizations to listen to their development teams and allow multiple ALM tools to be used. Also, allowing teams to pick their own tools (within some parameters) talks to the self-organizing nature of Agile teams. Organizations should give development teams some power in selecting the tools and making the ultimate decisions.

Armed with the right tool or set of tools, a development team can incorporate ALM into its Agile process and start reaping the benefits of Agile ALM. ■

About Eric Landes

Eric has been working in the software field since 1986. He is currently a solution architect at Agile Thought where he engages in Agile coaching, consulting on ALM practices, good engineering practices and leading software teams.

Eric specializes in Agile Project Management (XP/Scrum variant), Lean Software Development, Kanban, EPM, Enterprise Project Management implementation (Microsoft Project Server) and Software Project Management. He is a Microsoft MVP Visual Developer and Certified Scrum Master (CSM). You can read more from Eric at: www.ericlandes.com

Cover Story

Modern Module Based Keyword Test Automaton for Visual Studio

TestArchitect for Visual Studio is a keyword authoring platform extension designed specifically to enhance coded UI test automation in Visual Studio 2012.

By Joe Luthy, Marketing Director, LogiGear

Microsoft's Visual Studio's ALM solution helps organizations manage the entire lifespan of application development and reduce cycle times. With the introduction of coded UI as an integrated component of Visual Studio® in 2010,Microsoft offered a built-in alternative to third party testing tools.

The coded UI framework in Visual Studio has record and playback functionality for test case creation as well as the flexibility to write tests in code. Incorporating a coded UI test platform in Visual Studio helped expand the ability for team collaboration but the burden of maintaining coded UI tests creates limitations for continuous builds and large-scale testing needs.

Coded UI tests are fragile by nature and are easily broken by frequent UI changes, and are also slower to run than unit tests or mocked tests. In many cases, developers have to determine at what point and how much testing built on coded UI is appropriate. Testing early in the development cycle when the application's user interface is changing frequently can result in a high test maintenance cost, while waiting until later in the cycle can result increased development time.

The fragile nature of coded UI tests prevent reaching critical mass for test automation. After automating many tests, teams find themselves spending more time maintaining old tests than creating new ones.

TestArchitect for Visual Studio is a keyword authoring platform extension designed specifically to enhance coded UI test automation in Visual Studio 2012 and make it possible to increase test coverage and execute large-scale test automation. As an extension, it fully integrates with Visual Studio and allows keyword-driven tests to be created for all development platforms that are supported by coded UI in Visual Studio 2012, native windows WPF applications and Windows Internet Explorer browsers.

By using TestArchitect for test authoring, test creation is faster and more importantly, tests are long-term maintainable and highly reusable. The improvement in test creation and automation enhances ALM within the Visual Studio environment.

> *"LogiGear's TestArchitect being integrated into Visual Studio 2012 is a shining example of how automated, scalable testing within a development environment results in faster deployment of high-quality software by enabling test engineers and developers to expedite comprehensive testing," Tom Lindeman, Director of the Visual Studio Industry Partner program at Microsoft, Inc.*

Modern keyword-based testing

The TestArchitect test authoring platform gets its roots from the Action Based Testing (ABT) approach pioneered by Hans Buwalda over a decade ago to address the challenges of testing increasingly complex software.

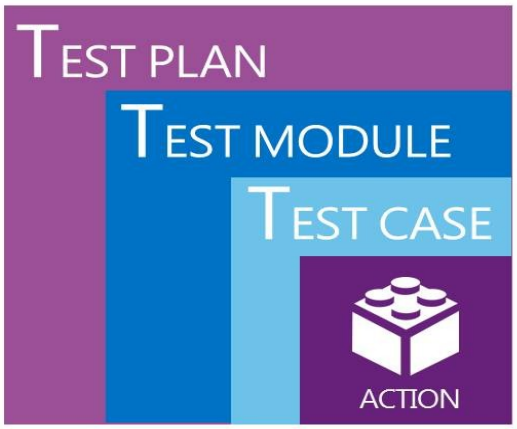
ABT uses actions (defined by keywords) that serve as building blocks for test cases. Actions can be used to create test cases, and combined to create more complex actions. Both actions and test cases are reusable so test development becomes faster with every test.

While there are similarities in the technical implementation of keywords, ABT embodies a test design methodology and a production process that focus on the three essential pillars:

- Test system design: Effective break down of the tests.
- Test module design: Right approach per test module.
- Test case design: Right level of test abstraction.

This framework makes it possible to create tests that are far less fragile when applications change. The benefits are long-term maintainability and high reusability of tests.

A significant advantage of TestArchitect's modular building block approach to test design is that actions, interfaces and test lines are all separate. When requirements change or a new platform is added, only minimal modifications are needed to maintain the tests. This also makes it possible to design and write tests before functionality is complete. For teams that have embraced Agile practices or operate in rapid development environments, the ability to create tests early adds efficiency.



Modular test organization with TestArchitect for Visual Studio

Test cases are created from a series of keywords with arguments. The automation focuses not on automating test cases, but on automating the keywords. Since there will be fewer keywords than test cases, and keyword implementations tend to be shorter than scripted test case implementations, the automation effort is faster and much more manageable.

In a well-designed keyword based test suite, only a limited number of keywords have to be maintained when an application changes. Apart from the more manageable and maintainable automation, a keyword test is more readable and becomes self-documenting compared to a verbose format where test steps are described in natural language.

All test development and execution in TestArchitect is based on test modules. Test modules are work files—very specialized spreadsheets—where tests are created. Test modules contain the following elements:

- Objectives - A statement of what must be tested in the software under test.
- Test cases - One or more actions followed by one or more checks, usually simulating end-user interaction with the software under test.
- Actions- A series of operations and checks.

Test modules act as containers for a group of test cases with a similar, narrowly defined scope. Proper organization of test modules is an essential factor in realizing efficiency gains.

Test modules should be well defined and differentiated from each other. Test cases within a test module can have dependencies among themselves, but test modules should be designed to run independently so that all actions, checks, etc. directly relate to the scope of the test module. This makes it easy to keep abreast of large tests and assure functional changes of a system under test only have a localized effect on the test set.

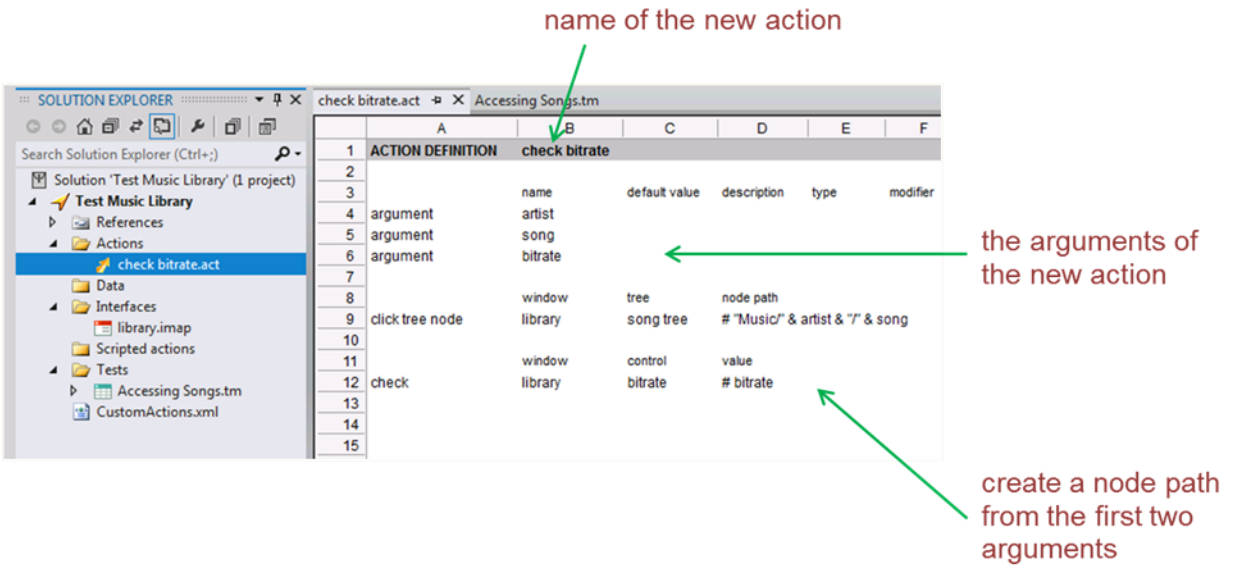
Each test module contains a list of test objectives - one for each test case. Test objectives serve to further refine the scope of the test module. These allow a reader to understand why test cases are designed the way they are, and give a quick insight in correctness and, more importantly, completeness of a test.

A key benefit of well-designed test modules is readability. Stakeholders with relatively little knowledge of technology or of testing details can easily understand a test. The details needed to execute the test (like which menu item to choose to open a dialog) are not visible in the test module unless they are important to the scope of that test module. Test details are handled by actions, which themselves are maintained in an action library.

The other essential elements are actions. Actions are re-usable containers of a series of operations and checks. Instead of specifying operations and checks in detail in the test module, the tester only has to enter the action and its arguments on a line, Arguments define input values and expected outcomes.

Actions are best regarded as a byproduct of the test design. Testers essentially define the actions and their arguments. While this makes it possible for testers to create both the test and automation, it's best to collaborate with the automation engineer when developing tests for larger scale automation.

Actions need to be organized and managed well to avoid having more than one action for more or less the same operation. Actions should be clear and unambiguous and should be well documented (meaning, arguments and their default values, etc.).



Action definition in TestArchitect

Actions are characterized in the terms of levels:

- Low-level, UI related, like “click”, “select” (many of these are predefined in TestArchitect).
- High-level, business related, like “enter customer” and “check balance”.
- Intermediate levels, like “enter name and address”, “enter car details”.

Many low-level actions, like “click”, “enter” and “check” come preprogrammed and are located in the action library in TestArchitect for Visual Studio. Low-level actions are application agnostic and apply to a wide range of UI objects. For example the action “click” can activate a button in a window, or it can also activate a hyperlink in a web page. Intermediate actions typically address one screen in the application under test, and are the building blocks for higher level actions like “enter customer”.

There are a couple of ways to create higher level actions. Actions can be created by programming in C#, but the need to program actions has been minimized. The “action definition” feature allows defining how the new action will work by using existing actions. The action definition feature is the preferred method to create actions that don’t contain complicated technical functionality since it’s much simpler than coding. Replacing automation scripts with keyword-driven actions makes it possible to develop the majority of tests without programming so only the few highly technical test routines need to be coded.

In addition to simplifying the development, execution, and maintenance of tests, TestArchitect for Visual Studio enhances the testing and test-automation development lifecycle. It seamlessly integrates with Microsoft Team Foundation Server (TFS) to define testing based on the same team projects that other areas of the organization use. Test cases created in TestArchitect can be run as part of a test plan in Microsoft Test Manager (MTM) by associating automation with the test cases.

Test automation for today’s software

The sophisticated functionality of today’s software combined with fast-paced agile development environments, compressed release cycles, and platform proliferation makes large-scale software testing extremely complex. TestArchitect for Visual Studio simplifies coded UI testing by replacing coded UI scripts with keyword-driven preprogrammed actions. By speeding the development, execution, and updating of coded UI, programmers and automation engineers have more time to focus on critical tasks making the team as a whole more productive. Increased test coverage, decreased testing time, and the ability to release software that will deliver the expected customer experience are the benefits that TestArchitect for Visual Studio delivers. ■

About TestArchitect

TestArchitect for Visual Studio is now available for anyone using the Ultimate or Premium versions of Visual Studio 2012. You can request a trial of TestArchitect for Visual Studio by going to www.testarchitect.com.

About Joe Luthy

Joe Luthy is the Marketing Director for LogiGear. Joe has over 15 years product marketing experience in technology and Internet services for major companies including AT&T and Verisign.

Feature

A Look Inside The Agile Toolbox: 10 Top ALM Tools

The pervasive use of Agile methodologies has changed the way that development teams work. With this change, new tools and vendors have entered the traditional application life-cycle arena.

From Software Consortium

A recent evaluation of this trend was published by Forrester Research, the "Agile Development Managements Tools Forrester Wave" report.

Key observations of the report:

Tools have changed to be task, management and report oriented. There has been a shift from traditional ALM tools, which focused on development artifacts and their relationship, to tools which bring change and task management to the top of the feature list, support the development team in their daily tasks, and enable work to be reported and measured.

Project management becomes integrated. Traditionally, planning and reporting in development projects has been disconnected from the actual work being done with project management tools, and only limited integration with ALM tools. As Agile methods are adopted, project management requires more complete integration to allow teams to re-plan frequently and optimize flow within the team. More integrated solutions allow task, change (defect or story), and resources to be linked and reported on, which provides a more streamlined planning and reporting approach.

Scrum is popular; "Scrum but" is the implementation. Many teams use the Scrum framework for the basis of their approach, yet increasingly the report found teams following Scrum but adding other practices to this base flow. These hybrids include other Agile processes such as Extreme Programming XP and Feature Driven Development (FDD) as well as traditional approaches such as Project Management , Body of Knowledge (PMBOK) and Unified Process.

Integration with other tools is key. ALM tool users typically have unique collections of practitioner tools, so it is unrealistic to expect those customers to move to one vendor for products. As the practitioner market becomes both specialized and commoditized, it is important for an ALM tool to provide good integration, allowing data to be harvested from those tools and work to be driven to them. Agile encourages dashboards to be created to show build, test and work status, so tool integration must be able to provide that information.

Gone are the days when application development and delivery teams could cavalierly ask the business to pick two: cost, time, or quality. The business needs all three. Delivery teams have responded by adopting Agile development practices, choosing fit-to-purpose development platforms, and designing application architectures for faster change. Now it is time for quality management and testing to respond to this faster-moving environment. Functional testing tools are not enough. Quality must move beyond the purview of just the testing organization and must become an integrated part of the entire software development life cycle (SDLC) to reduce schedule-killing rework, improve user satisfaction, and reduce the risks of untested nonfunctional requirements.

Forrester

Ten Agile development management (ADM) tool vendors were evaluated: MKS, IBM, CollabNet, Rally Software, Atlassian, HP, Serena, Microsoft, VersionOne, and Micro Focus.

A summary of the Forrester vendor evaluation concluded that IBM and MKS led the pack with the best overall current feature sets. Atlassian, CollabNet, and Microsoft are also leaders with capable products and aggressive strategies that will result in significant product improvements. Rally Software Development is also a category leader—it offers the best current balance of product capability and strategic outlook. HP, Serena Software, and VersionOne are strong performers offering competitive options. In the case of HP and Serena, their products are recent introductions to the market and are expected to improve as the vendors mature and gain customers. VersionOne is a stalwart in the Agile space and offers excellent planning capabilities but is less flexible than other products when it comes to reporting and integration with application life-cycle management (ALM) tools. The solution acquired by Micro Focus appeals to client-server and legacy developers, but Micro Focus must clarify its future strategy.

The authors recommend three take-aways for application development professionals:

- Look for their current tools to provide better support for Agile teams.
- Integrate planning into ALM.
- Use ALM tool as the change hub for software delivery.■

About Software Consortium

Software Consortium is an award winning provider of software technology solutions for private and public-sector clients. SCI teams focus on helping clients achieve strategic, as well as tactical goals at high returns on investment using the wisest proportion of teaming, people, processes and technology. For more information visit: www.softwareconsortium.com

Feature

Tools for Integrated Test Platforms

Regardless of your current state of tools, building an effective Continuous Integration suite of significant automated regression tests is the key to moving to a higher level of confidence in today's development world.

By Michael Hackett

In the evolution timeline of software development tools, new tools have recently proliferated. We have all been sold on collaboration, transparency and traceability - but that is not enough. Today, test tools are more often taking center stage. Integrated test platforms can push us deeper technically, help test faster, add more tools for our use and help us provide better, faster appraisals of the product under test.

In this article, I will discuss the introduction and integration of test tools into project wide tools, called ALM (application lifecycle management) and go into greater details about eXtreme Programming's important *Continuous Integration* practice.

Remember: not too long ago, test teams had bug tracking tools. That was it. Then we added automation tools that pushed us to get test case managers. For years, that was where we stood. Now, imagine your user stories, test cases, automation, source control, unit test automation, API test automation, UI test automation and bug tracker all in the same tool. For many software development teams, this is already taken for granted. If you have not moved to this situation yet - get ready.

Team tools often get brought into a company where the main purpose is communication and transparency. Just as important to testers are - speed, ease, notification and access to information. Transparency and centralized communication are great for management and project progress. Using integrated test platforms, especially those inside ALM, give test teams access to information and should make your job easier.

Stand-alone tools no longer serve us in today's fast-paced world. I have too much experience with frustrated test teams using a stand-alone tool like Quality Center where the requirements section is empty because the marketing team or business analysts would not use "the test tool." And the bug data was piped out to Bugzilla or another tool because developers would not go into "the test tool."

This won't happen with the test tools in an ALM platform, for example, like TFS/Visual Studio, Rally or a custom built suite of tools like Jira, Bamboo, Bugzilla, nUnit, Subversion, etc. The tasks, storyboards, user stories, test cases, bugs, code repository and automation are all visible and accessible from one interface. The platform belongs to the product owner, designer, developer, tester, release engineer, database admin - everyone. The test team's tool is everyone's tool. Transparency and visibility are obvious, but there is no withholding of information due to "it's not my tool" or lack of access. It's a big step forward.

ALM—Application Lifecycle Management: A Brief History

The main forerunner of this ALM, all-in-one, tool explosion is Rational Software's suite, now Rational/IBM: Requisite Pro (requirements), Rational Rose (code design), ClearCase (source control), Rational Test manager, Rational Robot (test automation), and ClearQuest (bug tracking). We now have a wide variety of tools - from enterprise wide tools to open source individual tools - so you can put all the pieces together to have the same impact.

The use of these tools has grown in the past couple of years as teams have become more distributed, faster and more dynamic while at the same time, coming under increased pressure from executives to be more measurable and manageable. When Agile, as a development framework, grew explosively, the *anti-tool* development framework got inundated by tools. Now every tool professes to support Agile practices and be Lean.

ALM tool suites can generally manage user stories/requirements, call center/support tickets and also help track a sprint's planning game, store test cases associated with user stories, include source control, facilitate Continuous Integration, coverage analysis and track bugs.

There are many tools that do this, but the big leap forward with current ALM tools is the ability to be linked to source control, unit testing frameworks, and GUI test automation tools. Lite ALM tools satisfy the need for speed required by the rapid nature of software development – it's a communication tool.

There are great reasons for using ALM tools, just as there are great reasons not to use them! Let's start with great reasons to use them:

- If you're working on products with many interfaces to other teams and products, ALM tools can be very beneficial in communicating what your team is doing and where you are in the project.
- ALM tools are essential if you're working with distributed teams.
- If you are working on projects with large product backlogs, or with dynamic backlogs often changing value, a management tool can be very helpful.
- ALM tools are useful if you have teams offshore or separated by big time zone differences.
- You are scaling Agile - for highly integrated teams, enterprise communication tools may be essential.
- If, for whatever reason, the test team is still required to produce too many artifacts - like test cases, bug reports, build reports, automation result reports and status reports (ugh!) – most ALM tools will have all the information you need to report readily available.

Some reasons *not* to use them:

- The Agile manifesto states people and their interactions are more important than processes and tools. Clearly, tools cannot dictate process - that is a mistake. People and their interactions must be supported by tool use or do not use one.
- Bad tool use is not Lean. Tools have to be easy, fast and not create waste.
- The tools can be expensive or the open source ones have a high build/integrate/maintain cost. I have to say, if this is the problem, the cost to the team of not using an ALM will be greater.

Development and Test Tools

In the old days, developers had their IDEs (Integrated Development Environment). These tools have progressed to be more than text editors and compilers. They've grown into task tracking, build management, debuggers, coverage analyzers, syntax checkers, unit test harnesses, dependency analyzers, etc. These tools now make programming easier, give access to many useful built-in tools — some with a variety of libraries - lots of "stuff" to help programmers work faster, more efficiently and produce a better product faster. An IDE is now a unified platform for development, production and white box testing of code.

Testing tools developed more slowly. Test teams started to have access to sets of integrated tools in the mid-1990s mainly because teams began using multiple automation tools. The early integrated tools included a GUI/UI test tool, a performance test tool and perhaps an API test tool. Teams also needed a single bug database, maybe automatic bug creation, and, to reduce the maintenance headache, a unified interface for test case creation, execution and management. These tools tended to be "owned" by test teams. Rarely, if ever, were they used by other roles on the project team and there was no avenue to integrate the test team into the wider project.

The days of separate platforms are going the way of the dinosaur. It can be hugely beneficial for test teams to have easy access to source code, developer tools, coverage analyzers, debuggers and make their own builds as needed. It can also be convenient for developers to run test automation suites or execute test cases specifically linked to pieces of code. In addition to traceability for compliance or coverage measurement, ALM tools support tracing user stories to chunks of code to test cases to bugs. The benefits of linking or tracing items this way is a big benefit for targeted regression, gated builds and resolving issues. These are all the benefits that can be realized from a unified platform.

Continuous Integration- *Not* just autobuild!

The XP practice of Continuous Integration (CI) is revolutionizing software development. With the need for speed, rapid releases and very compressed development cycles, an automated build process is a no-brainer. This is not rocket science and not specific to Agile/XP. Continuous Integration tools have been around for years; there are many of them and they are very straightforward to use. It is also common for test teams to take over the build process. Implementing an automatic build process by itself is a step forward, but a team will realize more significant gains if they add to automated builds with Continuous Integration.

Continuous Integration includes the following:

- Automated build process.
- Re-running of unit tests.
- Automated smoke test/build verification.
- Automated regression test suite.
- Build report.

The ability to have unit tests continually re-run has significant advantages:

- It can help find integration bugs faster.
- Qualifying builds faster will free-up tester time.
- Testing on the latest and greatest build will save everyone time.

The positives for Continuous Integration far outweigh any resistance to their implementation. Test teams can take on more responsibility here: they can be more in control of the testing process — on how many builds they take and when they take them.

There's no reason why a test team can't take over making a build. There is also no reason a test team does not have someone already technically skilled enough to manage Continuous Integration. With straightforward Continuous Integration tools such as Bamboo, Cruise/CruiseControl, Git, Jenkins, Hudson — the list goes on — test teams can do it. Integrated platforms make this possible with easy access, easy notification, sharing information, linking requirements or stories to code, to test cases, to automation to bugs as needed.

I suggest you visit Wikipedia to see a very large list of Continuous Integration tools for integration into various platforms at this here.

Test teams can use Continuous Integration tools to make a very effective build validation process:

- Scheduling new builds.
- Automatic re-running of the unit tests. Note: this does not mean write the unit tests, it means re-run — for example, the JUnit harness of tests.
- Re-running automated user story acceptance tests and whatever automated smoke tests are built.

Test teams taking over the Continuous Integration process with high volume automation is, from my experience, what separates consistently successful teams from teams that enjoy only occasional success.

Significant Automation: Automation is The Rule, Not Optional

It's a forgone fact that significant, sophisticated, yet simple automation is mandatory. If you do not have significant, high volume automation, you are doing the product team a disservice. All the integrated development/test platforms are including significantly more test tools. Many have GUI test recorders, limited performance tools and database test tools. To have these automation tools integrated into the test and development platform is essential.

High-volume test automation is the only path to rapid development and confident deployment. Making it happen can still be a challenge. It requires perseverance, support, cooperation, expertise and more! But at the same time, having an integrated test platform will make it more effective than ever.

From TestArchitect for Visual Studio, to Selenium and Eclipse integrated with Git or Hudson, there are a wide variety of possibilities for having all your test tools integrated into a development platform beneficial to both dev and test teams. Integrating testing tools with development tools and project wide tools allows test teams to provide the maximum benefit to the whole product team.

Project Management

The project management aspects of software development are facilitated by having integrated tools. For test teams, the obvious reasons are traceability, reporting, communication and transparency. The most important reasons are: speed, ease of work, being aware of other team members' work, more automation, more technical testing, smarter testing, form more grey box knowledge, leveraging other team members' work, knowledge and artifacts.

Summary

Stand-alone test tools, although beneficial, are not leveraging easy technologies we have available today. To keep pace with current software development practices, efficient tools to support rapid release and confident deployment are essential.

Having a unified test platform is the first step. Integrating the test platform with the development platform and the whole ALM is the most effective infrastructure for software development. Most importantly, regardless of your current state of tools, building an effective *Continuous Integration* suite of significant automated regression tests is the key to moving to a higher level of confidence in today's development world. ■

Glossary: Integrated Test Platforms

ALM (Application Lifecycle Management):
A continuous process of managing the life of an application through governance, development and maintenance. ALM is the marriage of business management to software engineering made possible by tools that facilitate and integrate requirements management, architecture, coding, testing, tracking, and release management.

Source Control:
There are many source control tools, and they are all different. However, regardless of which tool you use, it is likely that your source control tool provides some or all of the following basic features:

- A place to store your source code.
- A historical record of what you have done over time.
- A way for developers to work on separate tasks in parallel, merging their efforts later.
- A way for developers to work together without getting in each others' way.

Build Automation:
The act of scripting or automating a wide variety of tasks that software developers do in their day-to-day activities including things like:

- Compiling computer source code into binary code.
- Packaging binary code.
- Running tests.
- Deployment to production systems.
- Creating documentation and/or release notes.

Continuous Integration:
The implementation of a *continuous* processes of applying quality control—small pieces of effort, applied frequently. Continuous Integration aims to improve the quality of software, and to reduce the time taken to deliver it, by replacing the traditional practice of applying quality control *after* completing all development.

Unit Testing:
Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing is often automated but it can also be done manually. This testing mode is a component of Extreme Programming (XP), a pragmatic method of software development that takes a meticulous approach to building a product by means of continual testing and revision.

Test Case Manager (TCM):
A tool designed for software test engineers to organize test cases for storage and execution logging. Test cases are written in a standard format and saved into the system. Test cases can be organized by level (Smoke, Critical Path, Acceptance Criteria, Suggested), by area (GUI breakdown, installation, data, etc.), by status (pass, fail, untested, etc.), or other breakdown criteria. Once test cases are built, testers use TCM to track and report success or failure of test cases. TCM provides an unlimited number of central, multi-user databases, each of which will support an entire test team. TCM is intended for use by small to midsize software development companies or organizations.

Traceability:
The ability to link product documentation requirements back to stakeholders' rationales and forward to corresponding design artifacts, code and test cases. Traceability supports numerous software engineering activities such as change impact analysis, compliance verification or traceback of code, regression test selection and requirements validation.

Automated Regression:
Any type of automated software testing that seeks to uncover new software bugs, or *regressions*, in existing functional and non-functional areas of a system after changes, such as enhancements, patches or configuration changes, have been made to them.

Smoke Test:
Smoke testing is non-exhaustive software testing, ascertaining that the most crucial functions of a program work, but not bothering with finer details. The term comes to software testing from a similarly basic type of hardware testing, in which the device passed the test if it didn't catch fire the first time it was turned on. A *daily build and smoke test* is among industry best practices advocated by the IEEE (Institute of Electrical and Electronics Engineers).

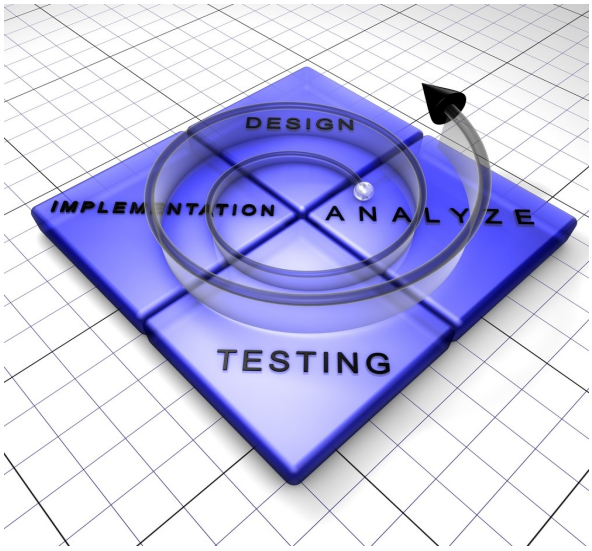
Sources: Wikipedia, testingfaqs.org, searchsofwarequality.techtarget.com

Feature

The Cure for Continuous Integration

Do you want to speed up your automated tests by a factor of 10 and deploy your application continuously? In this article we share how the JIRA development team at Atlassian has accomplished this using Stages in Bamboo.

By Bryce Johnson



Stages have allowed the JIRA Development team to take a week's worth of testing and condense it to one day.

Additionally, we have taken functional testing time from 8 hours to 40 minutes and use it as an "inner loop" continuous integration feedback build.

There are two primary feedback loops that the JIRA development team utilizes:

- Generic "CI" Build - developers get feedback on their code changes quickly.
- Tested Platform Matrix - feedback about how our release is validated against different types of distributions and the different environments they can run on, which include different operating systems and database configurations.

What Did We Do Before Bamboo Stages?

Running our tests in serial, the CI build would take 8 hours to run our functional test suite, which consisted of around 3000 tests. However, we only did this to deploy the JIRA artifacts to Maven because it is important to run all of our tests prior to deployment. This is key for JIRA artifact consumers to make sure they can pin point true integration failures instead of tracking down failures within JIRA. In addition to this build, in order to get faster feedback, we had 15 different build plans to run our functional tests but had no way to act upon those results. Finally, we also ran a unit test build as a fast CI loop that ran in 15 minutes, but only provided unit level testing. We had two builds for functional testing, one for validating deploying artifacts to Maven and the other to give us a faster feedback cycle with those tests. This meant we were potentially testing the same changeset twice and burdening our build hardware. For the Maven deployment functional test build that took 8 hours, we could deploy our artifacts once a day. Sometimes this deployment would be delayed two or three days if those tests stayed in a failed state. The Tested Platform Matrix build would take an entire week and might not pass within that week. For most, if not all development teams, those feedback times are unacceptable.

What are we trying to run that took 8 hours of serial execution?

- 12,567 JUnit Tests
- 3020 Functional Tests
- 328 QUnit Javascript Tests
- 639 Selenium UI Web Browser Tests

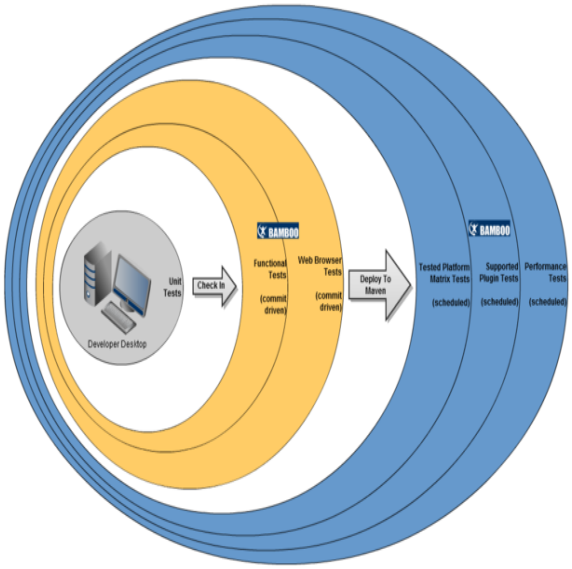
Staging a Revolution in JIRA's Testing Cycles

By relying on an 8 hour development testing cycle and a week release testing cycle, we added higher risk in our ability to release JIRA on time. A lot of time was spent figuring out what commit broke what test, and sometimes even what *day* it happened. We needed help with running our tests in parallel and we needed to do it on a massive scale in order to shorten these long feedback loops. Bamboo stages allow us to run more than one build "job" within a stage and more than one stage in a build plan. The JIRA development team has done work to break the test suite into smaller parts, called batches, to launch each of these batches as a job within a Stage. Our functional testing Stage, for example, now includes 24 batches, or "jobs" that run simultaneously. Of course, we need somewhere to execute all of these jobs in parallel. This is where another important capability of Bamboo helps us out: the ability to run multiple remote agents on different types of hardware.

But the build pipeline is not done yet. Once our CI Build passes and "goes green", what if we wanted to...

1. deploy these tested artifacts to Maven?
2. deploy our tested JIRA distribution to QA Environments for manual testing?
3. build other JIRA Plugins so they can consume and validate against these tested artifacts?
4. run more tests, specifically our Tested Platforms Matrix?

We'd like to ensure that we can build and promote through these phases in this diagram going from the center left in development to release validation on the outside of the circles. The two primary loops are signified in yellow for development testing in our CI build and blue for release testing in the Tested Platforms Matrix and other important consumers of JIRA.



How Stages Help Us Reach Our Goals

Like many other development teams, the JIRA team wants to do all of these great tasks and accomplish these goals. The focus is to have our development CI loop as the producer bridge to launch into other outer loop CI activities. However, we can't act until we can aggregate all of the jobs that run our functional tests into one result, to decide if we want to proceed further into greatness. This is another key feature in Bamboo Stages.

Previously there were 15 functional test build plans all running on their own after a dependent parent build triggered from commit detections. However, it was impossible to launch other build plans against the results of these 15 builds.

Because of this, we would not be able to deploy our tested Maven artifacts from those results. Now, with Stages, our CI build plan can not only have these same 15 jobs but we also get the ability to deploy the artifacts from that commit revision to Maven *if and only if* all of the functional test jobs pass. Additionally, Bamboo Stages also makes adding Jobs to each Stage an easy process. This is important because we plan on growing our functional test batches from our current set of 20 to 40 with a goal of having each job execute in less time.

Next we modified our other child build plans—that needed to consume JIRA artifacts—to depend upon the result of the CI build plan. For example, the final Stage in the CI plan is to deploy our JIRA artifacts to Maven only if all the previous Stages' jobs passed, then build and validate important plugins like the JIRA Fisheye, JIRA Workflow and Activity Streams plugins. The final step was to build against the JIRA artifacts that were just deployed in the parent CI plan only if all functional tests passed. All this was done with the use of Bamboo Stages and additionally, another child plan of our CI plan, then deploy the JIRA Standalone distribution to a clean test remote environment for QA manual testing. The best news of all is that the CI build now executes all functional and unit tests on average in 40 minutes and we have ambitions to make this a 10 to 20 minute build in its entirety.

Our Tested Platforms Matrix is the second set of longer running tests that will drive us closer to release readiness. We run exactly the same functional tests in this set of builds. However, the difference here is we run against each distribution that we allow customers to download against most of our supported environments.

What Does This Mean?

We have:

- 4 distributions - Standalone, War, Source, and a Windows Installer.
- 3020 functional tests that were previously defined
- MySQL, Postgres, MS SQL Server, and Oracle databases that, except SQL Server, run on two operating system platforms, Linux and Windows.

Adding up all of the cells in the Tested Platforms Matrix shows us that we execute 24,160 functional tests.

These 24,160 functional tests are executed against the same Maven artifacts that were deployed from any previously passing CI Build. We can use staging again to help organize the number of jobs launched against the matrix and also aggregate results. Because of our ability to deploy JIRA artifacts through our development CI build, we can be assured our Tested Platforms Matrix builds run at least once a day. By using Stages, the execution of the matrix can help us complete the entire set of build plans in 5 hours. This allows us to have faster assurance from the Tested Platforms Matrix that we are ready promote what we created from a passing build in CI to make these same artifacts be release candidate artifacts for customer consumption.

Stages For the Win!

Bamboo Stages have been the ultimate win in condensing the amount of time to execute our functional tests in JIRA. It has given us the ability to not only help us launch our functional tests in parallel but also to importantly aggregate the results of those tests into a single pass fail result. This result has enabled us to then, with confidence, deploy and publish our JIRA artifacts for further testing in validation in JIRA and other products that depend on JIRA.

	Postgres	MySQL	Oracle	MS SQL Server 2005	MS SQL Server 2008
Linux	3020	3020	3020	N/A	N/A
Windows	3020	3020	3020	3020	3020

There are other elements to what I have shown you. We use Bamboo's artifact passing to build once and re-use many times so we can pass the JIRA artifacts from one Stage to the next. This means that all of the functional test batches I described don't have to checkout JIRA source and compile it. They can re-use the passed artifacts and give us the purity of testing against the exact same artifacts that were built in the compile Stage. When we started using this feature were able to trim even more time from both our CI build and our Tested Platforms Matrix --nearly 45 minutes! ■

About Bryce

Bryce is a Lead Build Engineer at Decree in Sydney Australia. As a build engineer he is a quality gateway between development and QA/Release and provides services and processes to ensure that quality. He has been working in the software industry since 1996 and specializes in SVN, Mercurial, Bamboo, Maven2, Ant, Ivy, Java, Unix, Linux, Shell Scripting, Continuous Integration and Puppet. To read more from Bryce, visit his blog here: http://blogs.atlassian.com/author/bjohnson/.

To keep up-to-date with the latest features released into Bamboo, as well as get tips for tricking out your CI pipeline, check out the Bamboo blog at http://blogs.atlassian.com/blog-cat/bamboo/.

You can also follow Atlassian Dev Tools on Twitter: @AtlDevTools. Happy building, everyone!

Principles of CI

Continuous integration is the practice of frequently integrating new or changed code with the existing code repository. This should occur frequently enough that no intervening window remains between commit and build, and so that no errors can arise without developers noticing them and correcting them immediately.

Normal practice is to trigger the builds by every commit to a repository, rather than a periodically scheduled build. The practicalities of doing this in a multi-developer environment of rapid commits are that it's usual to trigger a short timer after each commit, then to start a build when either this timer expires, or after a rather longer interval since the last build. Automated tools such as CruiseControl, Jenkins, Hudson, Bamboo, BuildMaster, AnthillPro or Teamcity feature automatic scheduling.

Advantages

Continuous integration's advantages:

- Can revert the codebase to a bug-free state, without wasting time debugging.
- Detect and fix integration problems continuously.
- Early warning of broken/incompatible code.
- Early warning of conflicting changes.
- Immediate unit testing of all changes.
- Constant availability of a "current" build for testing, demo, or release purposes.
- Immediate feedback to developers on the quality, functionality, or system-wide impact of code they are writing.
- Frequent code check-in encourages modular, less complex code.
- Metrics generated from automated testing and CI (such as metrics for code coverage, code complexity, and features complete).

Disadvantages

- Initial setup time required.
- Well-developed test-suite required to achieve automated testing advantages.

Many teams report that the advantages of CI well outweigh the disadvantages. The result of finding and fixing integration bugs early in the development process can save both time and money over the lifespan of a project.

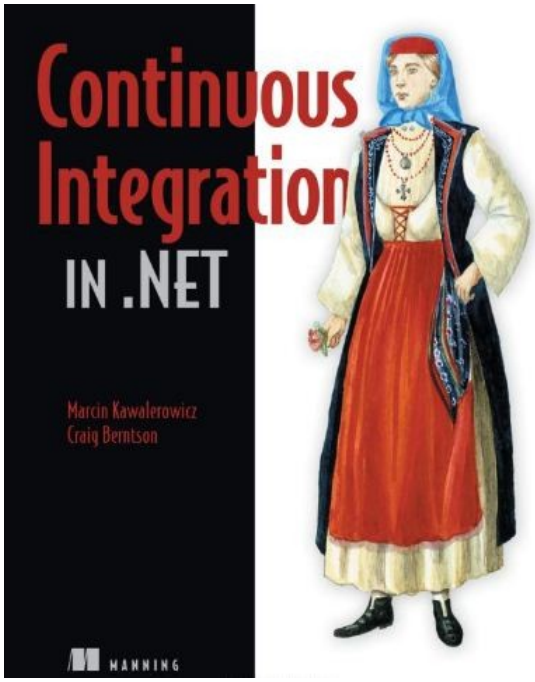
Source: Wikipedia

Book Review

Continuous Integration in .NET

Times have changed, the tools have improved, and with books like this available you have no reason to not give CI a go.

By Tad Anderson



I still remember the first time I was on a project that used NAnt and CruiseControl.NET. It was years ago and both were new tools with plenty of bugs. The project manager took one of the team's architects and dedicated him to getting CI up and running. I didn't work with him for another 9 months. It was a complete nightmare. Every morning was dedicated to finding out why the builds failed, fixing the issue, and then manually rerunning the builds until successful. Then it was off to show management the new build reports. It didn't take long for them to not want to be hassled with the process. A year after it all began the code base was removed from the CI process and went back to manual builds.

Not a very good story to start the review of a book on CI that I highly recommend you read. Times have changed, the tools have improved, and with books like this available you have no reason to not give CI a go. You may have plenty of excuses, but no reasons. Luckily this book contains a nice summary of excuses commonly used and does a nice job of debunking them.

The book starts off with a chapter titled "Understanding continuous integration" which gives a nice overview of CI and introduces the CI tools. They include source control, CI server, feedback mechanism, build manager, unit test framework, documenting, and code-analysis tools.
The book continues with chapters on Setting up a source control system, Automating the build process, Choosing the right CI server, Continuous feedback, Unit testing continuously integrated code, Performing integration, system, and acceptance testing, Analyzing the code, Generating documentation, Deployment and delivery, Continuous database integration, and Extending continuous integration.

One of the things I like about the book is that it is .NET centric and not Microsoft centric. It introduces the Microsoft tools, other vendor's tools, and open source tools. For example, Subversion, TFS, SourceSafe, Git, and Vault are introduced as source control options, Nant and MSBuild are introduced as build tools, and CruiseControl.NET, TFS 2010, and TeamCity as CI servers.

The authors do a great job of providing in-depth examples of the different tools and technologies. The example's accompanying downloadable code is very well organized and usable.

Although implementing CI can be a big change for a team, this book will definitely help educate you and your team on all the different tools available in the context of .NET projects. That give a big advantage when planning your team's path to CI.

All in all I highly recommend using CI on your projects, but I recommend reading this book first even more. ■

About Tad

Tad lives in Mount Joy, PA with his wonderful wife Wendy and his faithful companion Artimus (4 year old Maltese). He has been programming since 1992 and specializes in Software Architecture. He is currently serving in the role of Enterprise Architect at Penn National Insurance.

VIETNAM SCOPE

Landscapes of Vietnam

Vietnam is a country of contrasts due to its geography. This makes the country unique as it offers travelers the ability to experience a multitude of landscapes.

By Brian Letwin

From the rice-cultivating tropical lowlands, to the rich-soiled, coffee-producing highlands, Vietnam offers a unique and varied travel experience.

22% of Vietnam's 87 million people live in the Mekong Delta, the fertile rice-basket of the country. It's this region that grows the majority of Vietnam's rice and makes Vietnam the leader in global rice exports, producing more rice than South Korea and Japan combined.

Life in the Mekong Delta is deeply connected to water. The lowlands are continually sliced by rivers and streams filled with boats. Taking a trip of any length here often requires taking multiple ferries, and passing through floating towns where merchants travel house to house by boat selling their soups, clothes and household items.

Moving northeast, towards Da Lat, the landscape quickly changes as the seemingly never-ending plains of the Mekong are replaced by a rising topography. Da Lat, founded by the French as a vacation town, lies in the misty highlands. Here, mist often covers the peaks and valleys, earning it the name "City of Eternal Spring." The temperature here stays between 60 – 77 degrees year round, making it ideal for agriculture. Renowned for its orchids, roses, fruits and vegetables and surrounded by pine trees, it's somewhat akin to what one finds in Northern California. This mountain range will continue into China but moving toward the coast, one will find the gem, Ha Long Bay.

Ha Long Bay, a UNESCO World Heritage site, is perhaps the crown jewel of Vietnam's beauty. The bay features thousands of limestone islands which take various sizes and shapes, each topped with tropical vegetation. Many of these islands feature unique caves which house not only massive stalagmites and stalactites, but also a collection of 19th century graffiti from French tourists. Located within the maze of islands are 4 floating fishing villages where tourists can view marine life, buy crafts and most importantly see this community of floating houses, schools and shops.

With many distinct climates and cultures, Vietnam is a stunning place. It's easy to appreciate the country's rice paddies, rivers, misty peaks and wondrous caves. But when visiting, be sure to take note of the intimate relationships people have with their environment. Their ingenuity has harnessed the natural landscape to produce the staples necessary for Vietnam's growing economy and tourist sector. ■

Speed and Collaboration

The winning combination for large-scale test automation

TestArchitect™

Go to market with confidence

The modern module-based keyword test authoring platform that enables large teams to create, maintain, and execute enterprise-scale test automation with groundbreaking speed.

Exponentially increase automation.

Create one test for multiple platforms/versions.
Supports: Microsoft® Windows (up to Windows 8), Linux (Red Hat, Cent OS), Android™ (2.x, 3.x, 4.x); Native Windows, Microsoft .NET WinForm, WPF, Microsoft Silverlight, Java™ Swing, Java RCP, Java OSGI, Flash/Flex, QT, Android SDK, Android WebView

Test sophisticated functionality.

Easily test complex custom controls, 3D graphics and objects.

Accelerate test creation and maintenance.

Action-based Test Language creates stable, reusable tests without coding or fragile recording.

Scale with ease.

Spreadsheet style UI and dynamic action keywords make teams more productive.

Find out more and download a FREE trial at www.testarchitect.com or call +1 800 322 0333

LogiGear®
Software Testing

Microsoft Visual Studio PARTNER

TestArchitect™ for Visual Studio®
Modern keyword test automation for Coded UI

LOGIGEAR MAGAZINE
SEPTEMBER 2012 | VOL VI | ISSUE 4

United States
2015 Pioneer Ct., Suite B
San Mateo, CA 94403
Tel +1 650 572 1400
Fax +1 650 572 2822

Viet Nam, Da Nang
7th floor, Dana Book Building
76-78 Bach Dang
Hai Chau District
Tel: +84 511 3655 333
Fax: +84 511 3655 336

Viet Nam, Ho Chi Minh City
1A Phan Xich Long, Ward 2
Phu Nhuan District
Tel +84 8 3995 4072
Fax +84 8 3995 4076