Solving Technology. Together.

Hylaine - ALM Systems - Whitepaper

Cohesive Work Tracking

Ryan McElroy

Table of Contents

Document Version	Date	Notes
1.0	2018/07/16	Initial Authoring
1.1	2020/01/17	Updated with additional section on 2019 updates and Renaming of TFS

1. Introduction

While the exact terms and steps in an Application Lifecycle vary by organization, the core concepts and steps remain very similar. Hylaine defines a typical Application Lifecycle as the following:

- Requirements Gathering
- Development
- Testing
- Deployment
- Optimization

While these steps are taken with every software product, one or more may be neglected. An organization's process is tightly tied to the tools (or lack thereof) that they use to manage each of these steps.

ALM (Application Lifecycle Management) tools are those that attempt to manage the entire lifecycle in a one-stop solution. For organizations looking to simplify, or those early in their decision-making process who are passionate about making the right selection the first time, we would like to offer our experiences with two major players in the Enterprise ALM space: Atlassian's suite of core products (BitBucket, Jira, and Confluence) and Microsoft's ALM tool (Azure DevOps).

2. About Hylaine

Hylaine is a technology-agnostic software consultancy firm based in Charlotte, NC. Hylaine's consultants are industry veterans who have delivered hundreds of projects across 4 core service lines: Application Development, Quality Assurance, Business Intelligence, and Process Consulting.

3. A note on Naming

Microsoft has created what is, at best, a confusing situation when it comes to their core ALM product. Throughout this document, we attempt to be as accurate as possible when mentioning a specific product (for instance, if we completed a project with TFS we will say so, however the same lessons will largely apply to AzureDevOps). The table below shows how naming has changed over the past few years. We have excluded years with no significant name changes.

Year	On-Premises	Cloud
- 2012	TFS 2012	N/A
2013		Visual Studio Online
2015		Visual Studio Team Services
2018		Azure DevOps
2019 +	AzureDevOps Server 2019	

4. Key Benefits

Regardless of tooling choice, using ALM software provides major benefits to teams large and small.

4.1. Consistent Data

Even when an organization has tools for each step of the software lifecycle, organizing it across multiple applications is an intensive process. Integrations are deferred or never considered, leading either to duplicated information or, in the worst cases, to completely different versions of the software development story.

4.2. Telling the Story

Tracking a piece of functionality from conception to completion to maintenance is a critical part of a software engineer's job. Endless amounts of productive cycles are wasted emailing or instant messaging multiple members of a development team to track down clues about historical context. Employee turnover creates brick walls where even the most dedicated sleuth will be unable to answer crucial questions.

Both the JIRA and AzureDevOps ALM tools support the ability to dynamically update information about an application depending on the work that's been released. For stakeholders who may not be intimately involved in the day-to-day construction of a solution, this is a huge boon.

4.3. Streamlining

As a software development shop changes its processes or purchases tools, management of the lifecycle can often become cumbersome as organizations must develop or purchase integrations between them or incur the overhead of manual processes for its personnel to "Swivel Seat" across disparate applications. Training new employees becomes much simpler when there is a cohesive user interface and information flows seamlessly across the application lifecycle. Consolidating cost management to just one vendor is another benefit of using a full ALM solution.

5. Atlassian & Azure DevOps – Head to Head

Both Microsoft and Atlassian have occasionally provided a checklist of pros and cons against each other's tool suite on their websites. Both tools also cover many of the same use cases, but it's useful to point out differences in capabilities at each stage of the Application Lifecycle.

5.1. Requirements Gathering

Requirements Gathering is a core competency of both Jira and Azure DevOps. While they take different approaches to writing down requirements, both are excellent at telling the story of a requirement throughout the lifecycle.

One difference to note is that the Requirements Gathering process may differ from organization to organization. The ability to customize entities used by each ALM tool is important for certain use cases. While

both tools support customization, Azure DevOps' customization framework is more flexible. Organizations that require a vast amount of customization should consider utilizing the on-premises versions of these suites rather than their cloud-based offerings as the latter are generally more limited.

5.2. Development

There are two core components to the development portion of the lifecycle to consider for each tool, source control and work tracking.

5.2.1. Source Control

Source Control is a non-negotiable component of modern software development. Behind the scenes, both Azure DevOps and BitBucket utilize Git for source control. This means any repository using Git will be trivial to import/export and allow extremely lightweight branching to match any development model. Azure DevOps also allows the use of TFVC – a centralized source control offering that was in widespread use at one time.

However, for modern teams TFVC has little to no value. It is tightly coupled with certain Azure DevOps concepts that discourage the use of best practices and has fewer capabilities than Git.

In addition, Atlassian offers free Git-specific GUI tools such as SourceTree that are generally richer than those on offer from Azure DevOps. This gives it a slight edge over Azure DevOps.

5.2.2. Work Tracking

Work tracking includes allowing estimates of future work and measuring progress against current work. Another critical part of work tracking is *connectedness*, or the ability to track effort against coding, building, and testing software.

Jira and Azure DevOps approach work tracking differently, with the former taking an overall simpler approach. This is clear in the number of fields available in the default templates for each tool, and the number of reports and report creation tools available in Jira.

Both tools support tracking work against code and building/deploying. However, Jira does not provide out-of-the-box solutions for integrating manual QA testing efforts with work tracking. While bugs can be created and tracked, additional tools or custom plugins are required to track test case execution against requirements. Azure DevOps offers integrated tracking of manual QA without addition plugins.

For teams engaging in greenfield, truly agile development with great automated testing coverage, Jira's simplicity can help reduce the amount of time spent tracking work. For teams that utilize manual QA, or work in legacy environments that need detailed reporting, Azure DevOps has a distinct edge.

5.3. Testing

Testing includes both automated testing and manual testing.

5.3.1. Automated Testing

Both tools support a dizzying array of automated testing tools and reporting methodologies. Out of the box, Azure DevOps's automated testing reports and supported tasks are a bit more intuitive. With the addition of plugins such as Xray, Jira can also provide reports of automated testing defects.

5.3.2. Manual Testing

Jira contains little to no support out of the box for manual testing. There are approaches that can utilize Atlassian's Confluence wiki product, but these approaches are weakly tied to Jira, and they are only marginally better than archaic test case tracking using spreadsheets.

There are several products that integrate much better with Jira, which available directly from the Atlassian Marketplace such as Xray and Zephyr at an additional cost.

5.4. Deployment

Atlassian's Bitbucket uses industry-standard YAML build and deployment scripts. While out of the box there are not very many examples, YAML scripts are easy to find for most tools that you'll include in a solution and committing or creating scripts is easy.

Modern versions of Azure DevOps (TFS 2015+) use Microsoft's "Build vNext" engine, which is mostly Powershell-based. There are an enormous number of build tasks that support most mainstream tools and deployment pipelines, including first-party support for deployment to either Azure or AWS. For special cases, several scripting shells are supported beyond Powershell, including the Windows command prompt, Linux's Bash shell, and Python out-of-the-box. In addition, Azure DevOps provides the ability to separate build and release pipelines, which is a critical part for correctly managing these as distinct steps of the ALM lifecycle.

Previous versions of Azure DevOps (TFS) use Microsoft's XAML based framework, which is much less robust, and not currently recommended.

In 2019, AzureDevOps began supporting industry-standard YAML scripts as well, and is the default for new build and release pipelines. However, this feature is still iteratively improving, or put another way is not as good as the vNext engine.

Both tools' build systems are extremely powerful and should be able to deploy software and database artifacts to virtually any platform or environment. Organizations should also consider their DevOps personnel's core competencies when evaluating the two build systems. In addition, the pros and cons of YAML vs GUI based pipelines should be evaluated (for instance, YAML definitions can be controlled with a solution).

5.5. Optimization

Optimization refers to general software maintenance and other post-deployment tasks. This can include creating new features, providing documentation, and gradually improving a software product.

Long-form documentation available to users, for instance in a Wiki, is occasionally a part of the Optimization phase of the application lifecycle. Atlassian's Confluence is a clear leader in wiki products. It's more mature,

allowing interaction directly with its user interface or via an API. Direct integration with Jira allows the tracking of issues and epics across wiki pages.

Azure DevOps' wiki product is much newer and less mature. It currently features only some integration with the rest of the Azure DevOps suite, and acts as a barebone replacement for SharePoint integration (which has been essentially deprecated for multiple releases now). However, it does have one significant advantages. The wiki product is controlled via a git repository full of markdown files under the hood. This means you can connect it directly with a software solution's source control or use any sort of solution that can interact with that repository for integrations.

Both tools are excellent at executing gradual improvement in an agile context and support the identification of potential architectural issues via static code analysis as part of their build or release pipelines. There's a robust selection of 1st and 3rd party steps for these build systems available.

In Azure DevOps Build and release pipelining also uses source control under the hood, meaning that you can always see if a build broke because of a change in the pipeline or bad code.

5.6. Administration

Generally, the workflow for user groupings and administration in the Confluence suite is weaker than what is offered in Azure DevOps. This is due to each part of the suite being technically its own product with its own permissions, but with an additional Atlassian user management system wrapped on top.

While permissions can be complicated within Azure DevOps when used incorrectly, in general tight Active Directory integration means most enterprises can use it out of the box. Additional controls outside the scope of Active Directory are possible as well.

5.7. Pricing

An important consideration with tooling is pricing. Both tools offer on-premises and cloud-based versions, with the benefits and drawbacks of those hosting models similar to many other SaaS (Software as a Service) products.

Microsoft prices Azure DevOps per seat regardless of model, with on-premises instances included with the top tier Visual Studio IDE licenses. There are also individual seat licenses. The cloud version of Azure DevOps is priced per user. There are two other important points about the Azure DevOps pricing model. The manual testing portion of Azure DevOps is a non-trivial additional fee per user, meaning that its testing capabilities should be weighed against this cost. Note that MSDN subscribers, or Visual Studio subscribers (Professional or Enterprise) also have complimentary Azure DevOps licenses. In addition, there is a free "Stakeholder" access tier that allows non-developers to interact with work tracking features of the tool for free.

The storage of a custom organizational package managers is also an additional cost for both platforms.

Jira's products that cover the functionality of Azure DevOps (Jira, BitBucket, and Confluence) are priced separately. They allow per-user pricing and discounted annual plans. Pricing below is only an example, as Atlassian has 3 pricing tiers as of 2019. Jira and Confluence can also be hosted on-premises. Costs are per-month.

Workflow	Cost (ADO)	Cost (Atlassian)
5 Developers (Work Tracking, Source Control, and Wiki)	$30	$125
5 Business Users, 5 Developers	$30	$220
5 Business Users, 5 Developers, 2 Testers *Note that Jira has no actual testing product. Instead, testing is governed by conventions and work organization in Jira and Confluence.*	$134	$277

6. Case Studies

As part of our Process Consulting service line, Hylaine consultants have had a unique window into the effectiveness of ALM tools across a wide variety of industries, team sizes, and maturity levels. Because of our long-term approach to partnerships, we've had the ability to track changes over time as ALM tool use has changed, even when those changes may not have come about directly as part of an engagement.

6.1. Insurance Brokerage

- Team Size: 40-45
- Development Process: Agile (Scrum)
- Maturity Level: Varied per Team/Product
- Language/Tooling: Primarily Microsoft, with other OSS supporting

This engagement began as part of an assessment of QA practices at the brokerage. This evolved into the creation of an Automated UI testing framework, and eventually a huge cross-team overhaul of how both QA and work tracking were handled within Azure DevOps.

The development team was split among various products at the brokerage, allowing different approaches to products that varied from entrenched legacy software to cutting-edge applications using the latest in front and middle-tier tooling.

The length of the engagement meant consultants were able to view the evolution of the Azure DevOps product from versions 2013 to 2018 and included a number of customizations at the request of the client. Consultants were also embedded directly into teams to help ensure best practices in Agile and the use of the tool while also contributing directly to delivery.

The client also utilized Atlassian's Confluence software as their Wiki solution, allowing another unique window into the differences in philosophy between the two products.

6.1.1. Core Findings

- With the release of the Build vNext framework in TFS 2017, teams at this organization who had been utilizing Team City and Octopus Deploy to great effect abandoned these tools in favor of a more integrated approach. While both sets of tools were extremely capable, the ability to use a single tool for building and deploying software across several environments was critical in reducing DevOps maintenance.

- Teams took various approaches to using Confluence – some using it as a developer repository for How-Tos, and others using it as a staging for requirements. The latter approach was poor practice overall, as it split crucial information across 2 platforms. Even with non-optimal usage, Confluence's strength over TFS' wiki component was clear.
- After helping enforce strict agile discipline and correct usage of fields in TFS, consultants were able to increase overall velocity for teams by a significant amount. More importantly, sprints became extremely predictable. The use of heuristic analysis through a mix of customized reports and queries allowed Scrum masters and project managers to identify risky areas of their applications and almost completely eliminate crunches.
- Manual QA improvements were vast, with increased reporting and bug detail enabled by TFS' manual testing tools. Automated testing coverage increased across some teams but varied widely according to their legacy status.

6.2. Online Trading

- Team size: 10-15 developers
- Development Process: Agile (Scrum)
- Maturity Level: Skilled developers new to tooling
- Languages/Tooling: C#, JavaScript, AngularJS, Node.js, Atlassian JIRA, JetBrains TeamCity, Octopus Deploy

This engagement involved building a replacement front-end for an existing trading platform written in Java and PHP. It was difficult to add new features to the existing platform, and time-to-delivery for the new platform was critical. New management introduced agile thinking, and specifically the Scrum process, to help increase velocity and ensure early and often feedback into the development process.

As part of the Scrum process, the team selected JIRA for lightweight work tracking for ongoing sprints. The product owners entered all work into the JIRA backlog, and the development team would assign points to each work item for relative effort. Based on previous sprint success for a given number of points, the product owner would select the top items for us to work on that could be completed in a two-week sprint. The QA team was embedded into the development team for the sprints, so their work efforts would also be captured in the points. A work item was considered "done" when QA completed their testing.

While Atlassian JIRA can be directly integrated into other Atlassian products (like Bamboo for CI/CD, Bitbucket for source control, and Confluence for wikis), in this engagement it was exclusively used for work tracking, sprint sizing, and reporting. The team used other products, such as GitHub, TeamCity, Octopus Deploy, and Slack for source control, CI/CD, and team communications. The tool choices were primarily based on cost in a startup-like environment.

5.2.1 Core Findings

- The development team, the scrum master, and the product owners were all able to use JIRA to accomplish their tasks, without spending an undue amount of time "managing the tool". The scrum master was non-technical, but able to set up and administer the JIRA system without much effort. The product owners were able to enter items and re-arrange them without difficulty. Most importantly, developers spent minimal time in the work tracking system, while still providing useful information to management regarding sprint health and overall progress.

- By separating the work tracking from the technical efforts, no time had to spent on integration with other more technical tools in JIRA. The scrum master, for example, did not need to understand source control or CI/CD. JIRA was intentionally used in a simple manner to allow easier separation of responsibilities for the full ALM lifecycle.
- JIRA's flexibility in working with non-Atlassian tools helped minimize the impact to DevOps when adopting a work tracking tool. The team was able to completely utilize existing build and release definitions with their preferred tools and a proven track record.
- The development team had a high level of skill across multiple languages and platforms. Without this skillset, it would have been more difficult to integrate the distinct ALM products. Also, if the work was more focused on using one programming language or platform, it would have made more sense to use one suite of tooling from start to finish (e.g., Azure DevOps or VSTS if purely .NET). At the time of the engagement (2013-2015), it made more sense to use these discrete ALM products for cross-platform, multi-language development.
- Overall, JIRA played a crucial role for work tracking. It was full-featured enough to support ongoing development and management efforts across multiple teams, and it was lightweight enough to enable agile delivery of high quality software products.

6.3. Healthcare Provider

- Team Size: 20-30
- Development Process: Waterfall
- Maturity Level: Low
- Language/Tooling: Microsoft

This was a process consulting engagement where Hylaine consultants were tasked with increasing the use of Azure DevOps features. Multiple tools were being used to track work across the Application Lifecycle, with shadow-it and poor connectedness ruling the day.

The development team was split among various products at the brokerage, allowing different approaches to products that varied from entrenched legacy software to cutting-edge applications using the latest in front and middle-tier tooling.

7. Whitepaper V1.1 Updates

After initial publication of this whitepaper, both suites of products underwent significant changes. Some of these were improvements, and others that could be considered lateral changes.

7.1. Azure DevOps

A significant change to Microsoft's ALM product was the name change from "Visual Studio Team Services" to "Azure DevOps". This name has a couple issues. It was essentially a fourth revision to the product's name, making previous references to the platform confusing. It also contains Azure in the name, when it technically supports any cloud platform and any on-prem infrastructure. Even more confusingly, the on-prem product now is called "Azure DevOps Server" which has nothing to do with Azure at all.

ADO also underwent a significant user-interface redesign, shifting focus from a traditional top-mounted navigation bar to a left-mounted navigation menu. Ironically, this user interface looks more like Jira and Confluence, though because ADO has so many more features there is liberal use of color and collapsible menus. Most teams experiencing this transition took roughly a month to re-acclimate to the new UI. Even with this and some significant new features, others have been lost.

The wiki tool in ADO has improved, but still lacks many features in confluence Confluence in order to support its key new feature, connection via code repositories.

7.2. Atlassian Suite

The Atlassian suite has continued to improve since the initial authoring of this whitepaper. There has been a much more minor UI redesign. Most importantly, the entire suite has had significant improvements in integration between each product. The integration of Bitbucket Pipelines has added a true in-house deployment tool and can be used with or without Bitbucket. It currently lags its more mature competitors.

8. Conclusion

Regardless of what path an organization chooses, the benefits of covering the entire application lifecycle with tools focused on automation and simplicity cannot be overstated. Development teams that have issues with software quality, requirements tracking, or multi-phased work cycles should assess their current toolchain and the benefits of vendor consolidation. Both Atlassian and Microsoft have made it easier than ever to build software and database solutions with their enterprise-level products. As a technology-agnostic consultancy, Hylaine can help organizations choose and implement ALM tools and processes to achieve their business goals.