Agile Release Management

Towards Frequent, Low-Risk Releases

By Jez Humble | Build and Release Principal | 14th July 2010

Everybody in IT has the same goal: rapid, reliable, low-risk delivery of high-quality, valuable functionality to users. This white paper discusses the application of Agile and lean techniques to release management in order to achieve this goal. The approach set out here is informed by the work of the DevOps movement, which aims to apply Agile principles and practices to release management and operations.

Agile Release Management

IT organizations face two different and conflicting pressures. On the one hand, organizations need to respond rapidly to a changing business environment. That means delivering high-quality software and services fast—and continuing to push new features into production through the life cycle of each project. Thus, the business, and hence development teams, require the ability to make changes to the live environment frequently. On the other hand, operations teams face many constraints that often limit their ability to support change, such as the need to support complex and interdependent systems, including legacy applications, on a variety of heterogeneous platforms. Their aim is to protect the stability of the live environment, which is most easily achieved by ~~limiting change~~.

The Tension at the Heart

It is a truism that we live in a world where the pace of change is accelerating around us. As a result, businesses need to respond even faster to new opportunities. They also need to evolve rapidly in order to maintain a competitive advantage. For many organizations, their ability to do so is reliant on the software that supports their business. IT organizations need to be able to deliver new software to support new programs as well as add new functionality to support existing programs. Since businesses need to keep their cost base as low as possible, they often also look to IT for savings, which means IT must do more with less and continually be on the look out for ways to consolidate and produce efficiency savings. A key capability of IT organizations is performance—the ability to deliver more with less.

On the other hand, businesses have to be increasingly careful about managing risk. This also exerts a direct pressure on IT organizations, where operations teams are measured on their ability to help businesses meet service level agreements around capacity, availability, service continuity, and security. They also have to meet regulatory requirements, which means each change must have its risk assessed (for example, its impact on the business continuity plan) and be auditable. And, of course, operations teams also have to manage business as usual, which is a regular source of fires to be fought. Thus, the other key capability of IT organizations is conformance—the ability to adequately manage risks to the business.

The apparent conflict between these two requirements—performance and conformance—is most keenly felt during releases. These are often painful, risky affairs that take hours or even days, resulting in late nights and frayed tempers. At worst, they can result in service interruptions and end in a roll back. At best, people lose their evenings, mornings, or weekends. Hence an essential capability in organizations is release management. ITIL® defines release management as "the process responsible for planning, scheduling and controlling the movement of releases to test and live environments. The primary objective of release management is to ~~ensure that the integrity of the live environment is protected and that the correct components are released~~."

Why Delivering Software is Painful

The goal of Agile release management is to enable organizations to achieve both performance and conformance by mitigating the risks of releasing such that it becomes a ~~routine, push-button event~~. The key to achieving this goal is a holistic approach that tackles the whole process of software delivery from project inception through to operation. In order to understand how to create an effective approach to software release, we must first analyze the problems most organizations face.

Releases are Infrequent, High-Risk Events

Many organizations have a release process that takes hours or days. Why is this so? The most common reason is that releasing new versions of software is ~~predominantly a manual process~~ that involves changes to many separate systems. Operating systems, infrastructure, and middleware must be configured, and their state verified. Binaries must be deployed, and the application configured. Data migrations have to be performed. Integration with other systems must be checked. The system needs to be smoke-tested to prove it is working. When these processes are manual, they are error prone. Because they are so expensive to perform, they are rarely tested, at least until near the release time. As a result, ~~serious problems are not found until near the release date~~.

Furthermore, when releases are painful, the usual response is to do them less frequently, and surround them with ceremony such as bureaucratic change management processes. But when releases are performed less frequently, the delta between each release becomes large, which only increases the risk of each release. Long cycle times and infrequent releases also increase the time it takes for businesses to get feedback on each release. If it takes you several months to release a new feature or service and you find out that users aren't interested in it, you have wasted a great deal of money. If you release regularly and frequently, you can benefit from user feedback to improve your software incrementally and start making money early on.

Recommendation: A good release management process is one that results in more frequent, less risky deployments.

People Involved in Delivering Software Do Not Collaborate Effectively

Another major contributory factor to painful releases is poor collaboration between the people involved in delivering software. Developers, testers, DBAs, operations staff, and managers all need to work together from the beginning of every project to ensure its success. Instead, it is typical to see developers throwing work over the wall to testers, testers entering bugs into defect systems rather than working with developers to fix them, and the operations team trying to deploy software that is not production ready, and in many cases, hasn't even been tested in a production-like environment.

ITIL® focuses on two key characteristics of services, which also apply to application software. It should be ~~*fit for purpose*~~—in other words, it should deliver the expected value to users in terms of functionality–and it should be ~~*fit for use*~~ , which means it should be production ready in terms of meeting capacity, availability, and security characteristics. Poor collaboration between developers, testers, and users results in software that is not fit for purpose. Poor collaboration between developers, testers, and operations leads to software that is not fit for use. Bureaucratic change management processes that are put in place to reduce the risk of releases often create silos that further inhibit collaboration.

Delivering software is an inherently collaborative exercise, so it should come as no surprise that effective release management of software that is both valuable and production ready requires organizational change to enable collaboration between everybody involved in delivery.

Recommendation: Create cross-functional teams for delivering software if possible, and when it is not, encourage frequent collaboration throughout the project lifecycle through regular stand-up meetings, showcases, and retrospectives that are attended by all stakeholders.

Automation and Collaboration: The Keys to Effective Release Management

An effective release management process protects the integrity of production and ensures that software is fit for use while ensuring businesses get valuable feedback quickly and the fastest return on their investment. The key to achieving these goals is to release as frequently as possible. Releasing frequently might seem like a counterintuitive way to protect the integrity of the live environment, especially to teams used to spending weekends performing releases. However, it is possible to ~~make software release a push-button event that is cheap and low risk~~. This requires *automation* of the delivery process, including the provisioning and management of test and live environments and data migration and better *collaboration* between the people involved in delivery.

Automation is essential because it allows activities such as building, testing, and deploying software to be done fast, repeatably, and without errors. Automation requires an up-front investment, but the return on that investment is enormous. Not everything can be automated—showcases, exploratory testing, and usability testing are skilled activities that need to be performed by humans—but much of the rest of the delivery process should be fully automatic. An automated build, test, and deployment process means that software can be continually tested both for its fitness for use and its purpose.

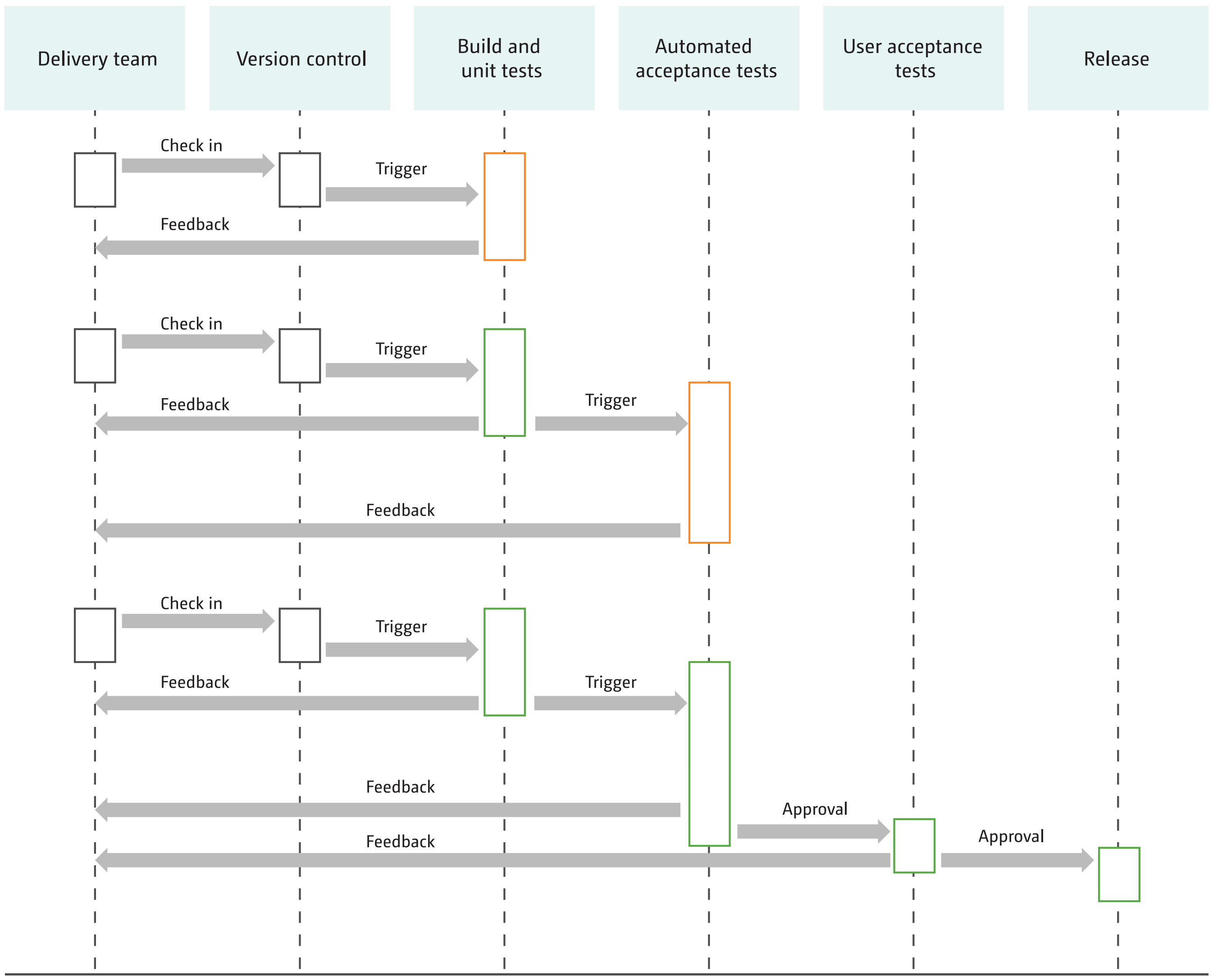
Much of the pain of delivering software comes toward the release date. The solution is to ~~bring the pain forward~~ by creating an ~~automated deployment process~~ and running ~~automated tests in production-like environments~~ from the beginning of the project, so that problems can be discovered early when they are cheaper to fix. If your architecture is not going to deliver the required capacity and availability, the best time to discover that is a couple of weeks into a project, not a couple of weeks before a big-bang release.

Creating an automated build, test, and deployment process requires close collaboration between developers, testers, and operations. Operations teams need to be involved in creating an automated deployment process that will be used to perform deployments to the testing, staging, and production environments the system will exist in. Database administrators need to work with developers and operations to ~~automate the data migration processes~~. **However, automation can and should be done incrementally:** use the theory of constraints to identify the bottlenecks in your delivery process, and use automation as part of your strategy for relieving them.

Recommendation: Release managers need to bring key stakeholders together to ensure that teams are performing automated deployments in production-like environments and automated testing to verify the production readiness of their software at the beginning of projects, not the end.

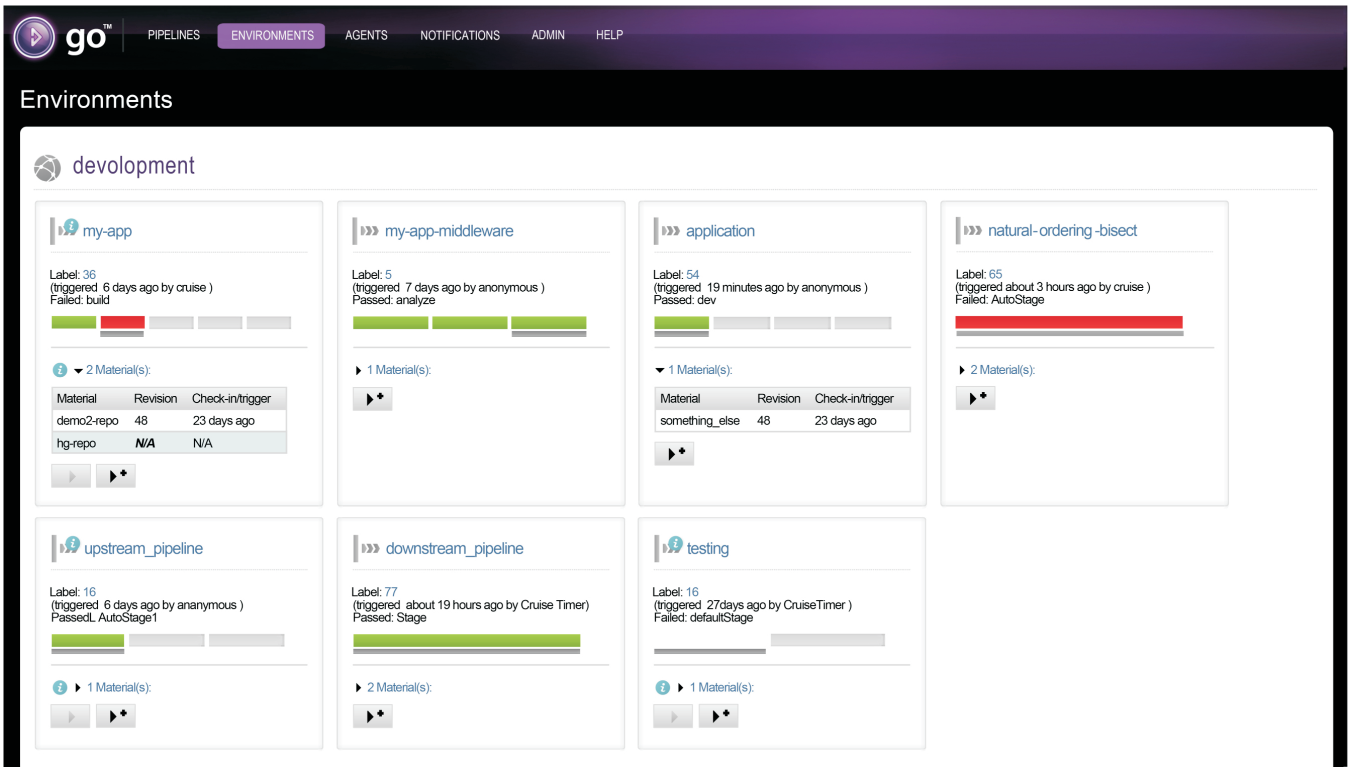
The Deployment Pipeline

The deployment pipeline is a pattern that has been applied in many projects both within ThoughtWorks and in other organizations. The aim of the deployment pipeline is twofold. First it creates *visibility* in the delivery process, enabling collaboration by providing ~~instant feedback on the production readiness~~ of your organization's software portfolio. Second, it provides ~~*control* over the delivery process~~, enabling developers, testers, and operations teams to perform push-button releases of any chosen version of an application into the environments they manage. The deployment pipeline also provides complete traceability from check-in to release, enabling straightforward auditing of the delivery process.



The deployment pipeline is a workflow system that embodies your process for delivering software from check-in to release. The first stage, known as the *commit stage,* is triggered every time somebody checks a change into your organization's version control system. The commit stage runs unit tests, analyzes the code to generate useful quality metrics such as code coverage and cyclomatic complexity, and builds the application, creating packages that can be deployed into every environment the application needs to run in.

If this stage passes, further stages can be triggered automatically, such as stages that run automated functional acceptance tests or capacity tests. Following this, a deployment pipeline provides the ability to deploy a selected build into testing or live environments. You don't have to automate every stage in your pipeline initially—you can start with placeholders for manual steps and automate incrementally. However, your pipeline must extend all the way to release—even if the final step is a manual one.



Environments

devolopment

my-app

Label: 36
(triggered 6 days ago by cruise)
Failed: build

2 Material(s):

Material	Revision	Check-in/trigger
demo2-repo	48	23 days ago
hg-repo	***N/A***	N/A

my-app-middleware

Label: 5
(triggered 7 days ago by anonymous)
Passed: analyze

1 Material(s):

application

Label: 54
(triggered 19 minutes ago by anonymous)
Passed: dev

1 Material(s):

Material	Revision	Check-in/trigger
something_else	48	23 days ago

natural-ordering -bisect

Label: 65
(triggered about 3 hours ago by cruise)
Failed: AutoStage

2 Material(s):

upstream_pipeline

Label: 16
(triggered 6 days ago by ananymous)
PassedL AutoStage1

1 Material(s):

downstream_pipeline

Label: 77
(triggered about 19 hours ago by Cruise Timer)
Passed: Stage

2 Material(s):

testing

Label: 16
(triggered 27days ago by CruiseTimer)
Failed: defaultStage

1 Material(s):

In this screenshot from Go, ThoughtWorks' Agile Release Management tool, the environment dashboard is shown. This shows you exactly which versions of each application are deployed into each environment. Using Go, you can trace back to the exact version in revision control it came from, which parts of your delivery process that build has been through, and what the results were. You can lock down who can do what so as to ensure you can meet your organization's control objectives.

Recommendations: Use a tool like Go from ThoughtWorks to create a deployment pipeline and work to incrementally automate your software delivery process.

Does Agile Release Management Work In Real Life?

ThoughtWorks consultants have been implementing deployment pipelines for a number of years on projects across a wide range of industries, including large, distributed projects. We have typically found that using the approach described in this white paper, organizations can move from performing releases once every few months to releasing several times a week, while improving the quality of the software delivered and increasing the stability of the production environment.

When we started on a project to replace a broadband provisioning system at a world-leading ISP, releases took a whole weekend, during which the service was unavailable for several hours, costing the company tens of thousands of dollars in revenue. We worked with the operations team and the development team to create an automated deployment process that could be used to deploy to testing and production environments. By the end of the project, deploying the application to any environment was a push-button process that could be performed in minutes, and down time when releasing—or when rolling back—was reduced to a fraction of a second using the ~~blue-green deployment~~ technique.

The Release Management Ecosystem

The deployment pipeline relies upon good configuration management and comprehensive automated tests. Good configuration management entails having everything required to create and test your application—source code, build and deployment scripts, automated tests, data migration scripts and application configuration options—in version control. It also means managing software dependencies, such as libraries and components. Finally, it means managing changes to your infrastructure and environments in a fully automated way.

Infrastructure and environments should be provisioned and managed using a fully automated process. New boxes should be provisioned using PXE or Windows Deployment Services from baseline images. Infrastructure and environment configuration should be kept in version control and rolled out using data center management tools such as Puppet, Chef, CfEngine, Microsoft SCCM, and BMC BladeLogic. Even networking infrastructure such as firewalls, switches, and load balancers can have their configuration managed centrally and pushed out via automated processes. Virtualization or cloud computing are also options for managing environments, again using a set of baseline virtual machine images. Virtual and cloud solutions allow organizations to decouple their hardware procurement process from their environment provisioning process to some extent.

Databases should also be managed in an automated fashion. ~~Databases can be versioned and migrated such that changes can be made to them incrementally~~, using tools such as DbDeploy, DbDeploy.NET, and ActiveRecord migrations. This allows DBAs and development teams to work collaboratively and iteratively on application development.

Frequent releases rely on excellent automated test coverage to ensure that the application is always working. This means a comprehensive set of automated unit, component, and end-to-end acceptance tests. Unit tests should be created using test-driven development and cover around 80% of the code base. They should run in a few minutes.

Automated acceptance tests are end-to-end tests that run against the application while running in a production-like mode. They ensure the application delivers the expected business value for users and protects it when teams need to make structural changes to the system. ~~Acceptance tests can run either directly against the UI or against an API that sits just under a thin UI layer~~. Acceptance tests should be layered to avoid coupling them to the UI and should not be created using record-and-playback tools that create brittle tests.

Testers and developers should work with customers from the beginning of the project to ensure quality is built in to the product and that it is always releasable and production ready. This is the key to protecting the live environment when performing regular releases and ensuring that releases are low-risk events.

Recommendation: Ensure your process for provisioning and making changes to testing and production environments and to databases is fully automated. Ensure that ~~everything required to create your application and the environments it runs in is kept in version control~~. Ensure your application's functionality is protected with a ~~comprehensive set of unit, component, and acceptance tests~~.

Effective Release Management Practices

Here are some practices that enable delivery teams to perform reliable, low-risk releases while protecting the live environment.

Develop Functionality Incrementally

An essential part of keeping your application production ready is ~~ensuring you can ship even when you are in the middle of adding a complex new feature or making architectural changes~~. Branching in version control is a poor way to achieve this goal since true continuous integration, and by extension the deployment pipeline, is impossible unless all developers are working off mainline. Instead of branching, ~~hide features~~ that are not complete either through configuration options or by ensuring the new functionality is not accessible through the UI—a technique known as "~~dark launching.~~" Perform re-structurings of applications using the "~~branch by abstraction~~" pattern. Make all changes to your application incrementally such that each change keeps the application working on mainline. Use a ~~modular, loosely coupled architecture~~ to manage collaboration on large or distributed teams.

Continuously Improve Your Change Management Process

It is essential to lock down your staging and production environments to ensure that unauthorized changes cannot be made to them—uncontrolled changes are a leading cause of downtime in live environments. However, change management processes need not be bureaucratic. As is pointed out in *The Visible Ops Handbook,* many organizations that perform best in terms of the MTBF (mean time between failures) and MTTR (mean time to repair) “were doing 1000–1500 changes per week, with a change success rate of over 99%.” Work to ensure your change management process is effective at preventing uncontrolled changes, while being efficient at enabling low-risk changes. It is essential that the tools you use for change management provide visibility into exactly which changes were made to which environments, when, and by whom, and to be able to trace them back to a change set in version control. In this way, when something goes wrong, it is relatively straightforward to determine the cause of the failure and roll back to a known-good baseline if necessary. Prefer transparency and automated remediation to decreasing the frequency of changes.

ThoughtWorks® STUDIOS

Deploy The Same Way To Every Environment

It is essential to use the same deployment mechanism to deploy to every environment, whether your continuous integration environment is testing, or live. This way you ensure your deployment to production is thoroughly tested. This involves developers collaborating with operations to ensure that the technology used to deploy the application is acceptable to the operations team. In most cases, it is better to use a combination of OS-specific packaging systems such as RPMs or MSIs, along with middleware-specific deployment technologies where necessary, rather than generic build tools.

Create Your Packages Once

You should only create your deployable artifacts once—~~in the commit stage~~ of your deployment pipeline. These should be the ~~same packages you deploy to production~~. This ensures that the thing you deploy to production is the same thing you tested all the way through your release process and removes the possibility of introducing errors through re-compilation or re-packaging. This means that you have to ~~separate out environment-specific configuration from your packages.~~ It is also important to associate packages with the revision that was used to create them in version control. You can either take a hash of the package and store it in a database along with the revision number used to create the package, or, if the packaging tool you use has a way to associate metadata with a package (such as Java Jar files or .NET assemblies), you can use this facility. This way it is easy to verify and audit which version of the application is in every environment.

Zero-Downtime Deployments

You can use techniques such as blue-green deployments and canary releasing to reduce the risk of deployments. Both of these techniques involve having both the current and the new version of the application being deployed, available in production and switching users to the appropriate version using routers. This makes it possible to route users back to the previous version of the application in case a roll back is required and provides a way to perform A/B testing to validate the utility of new functionality. If a new release of your software requires new resources or frameworks to be available in the live environment, put these dependencies into the live environment—having first tested in staging that they do not break existing functionality—before releasing the new version of your application. Resources and dependencies should be versioned so that multiple versions can be available in production at any time. It is also essential to consider whether and how data will be migrated in deployments and roll backs, but this topic is beyond the scope of this white paper.

Continuously Measure and Improve Your Delivery Process

It is important to gather metrics as part of the release management process, such as frequency of deployments, cycle time of releases (from check-in to release to production), time taken to perform deployments, time to detect an incident, time to repair an incident, what proportion of changes are successful, and so forth. Using these metrics, you can measure the success of any changes you make. Analyze successful and unsuccessful changes to look for patterns, determine the root causes of unsuccessful changes, and categorize them. Use the Deming cycle to continually improve your release management process: plan, do, check, act.

Conclusion

When organizations adopt Agile methodologies, the agility often stops when an application reaches the development complete stage. However, software doesn't deliver any value until it is live in production. The ~~last mile—getting software from dev complete to released—is often still a waterfall process~~, which is to say that it is risky, time consuming, expensive, and unpredictable. Organizations cannot truly claim to be agile until they apply Agile principles and practices to the release management process and to operations. The approach set out in this white paper is designed to make release management an integral part of the delivery process, ensuring that software is production ready from the beginning of the project, so that releases become frequent, reliable, and low-risk affairs.

ThoughtWorks® STUDIOS

About ThoughtWorks Studios

ThoughtWorks Studios is a global leader in Agile software development tools, and its products can be found in development organizations seeking sustainable Agile adoption. The company's Adaptive Application Lifecycle Management (ALM) solution provides a platform for managing all aspects of software development, from requirements definition and project management to test automation, quality assurance, and release management. Adaptive ALM comprises the integration of three products: Mingle (Agile project management), Twist (Agile testing), and Go (Agile release management). Each tool is available as part of a complete lifecycle solution or as a standalone product. Backed by more than 17 years of experience in Agile delivery, ThoughtWorks Studios is the product division of ThoughtWorks Inc., a pioneer in Agile development. ThoughtWorks Studios has over 400 customers in more than 20 countries, including 3M, Honeywell, BBC, eBay, Barclays, Vodafone, McGraw-Hill, and Rackspace. The company headquarters is located in San Francisco and Bangalore, with offices in London and select cities in Europe, Asia, and Australia. For more information, visit **www.thoughtworks-studios.com.**

mingle®

Mingle, an Agile management and collaboration tool, provides a common workspace for all team members and an automated system of record for all projects. Mingle can adapt any existing workflow process and easily manages daily development activities. Offering true-to-life visibility in the entire development process for all stakeholders, Mingle helps development teams become more open and collaborative.

go™

Go is a solution for Agile release management, which enables businesses to release software on demand. Go improves collaboration between developers, testers, and operations and provides fast feedback on the production readiness of your software. Using Go, teams can model the delivery process, perform push-button deployments, and trace from deployments back to version control.

twist®

Twist, an automated Agile testing solution, provides English-like constructs, making the testing process more productive for all team members. As applications grow in complexity, Twist helps to more easily maintain complex test suites. These suites keep pace with application development and are held as long-living assets.

ThoughtWorks Studios

CALL: 512-467-4956 (sales)
415-238-6497 (main) North America

EMAIL: studios@thoughtworks.com
WEB: www.thoughtworks-studios.com

+91 80-4064-9703 | Rest of the world

SAN FRANCISCO | CHICAGO | LONDON | BANGALORE | BEIJING | MELBOURNE