# A Step by Step Guide to **Scaling Agile Across Project Teams and Departments**

**WHITEPAPER**

ZEPHYR
Real-Time Test Management

# Table of contents:

# **Success is the Sum of Small Efforts,** repeated day in and day out

Companies implement some flavor of "Agile" for many reasons, most boiling down to "we want to ship better software faster." Adoption of agile across an organization can be challenging for any group, but it's even harder for large organizations.

At its heart Agile is about faster communication and ability to react to change quicker. The idea is to speed releases of *value*, not just software, and do so in smaller, more frequent cycles. As shown below, many organizations ship only once or twice a year—some organizations don't even release that frequently!

Organizations adopting agile hope to make a transition to much more frequent releases. Some organizations might work towards two-week iterations. Other organizations might even take things farther and work to a Kanban-like process where individual units of value are released as soon as they're completed. These phases are shown below in Figure 1: Phases in transitioning to Agile.

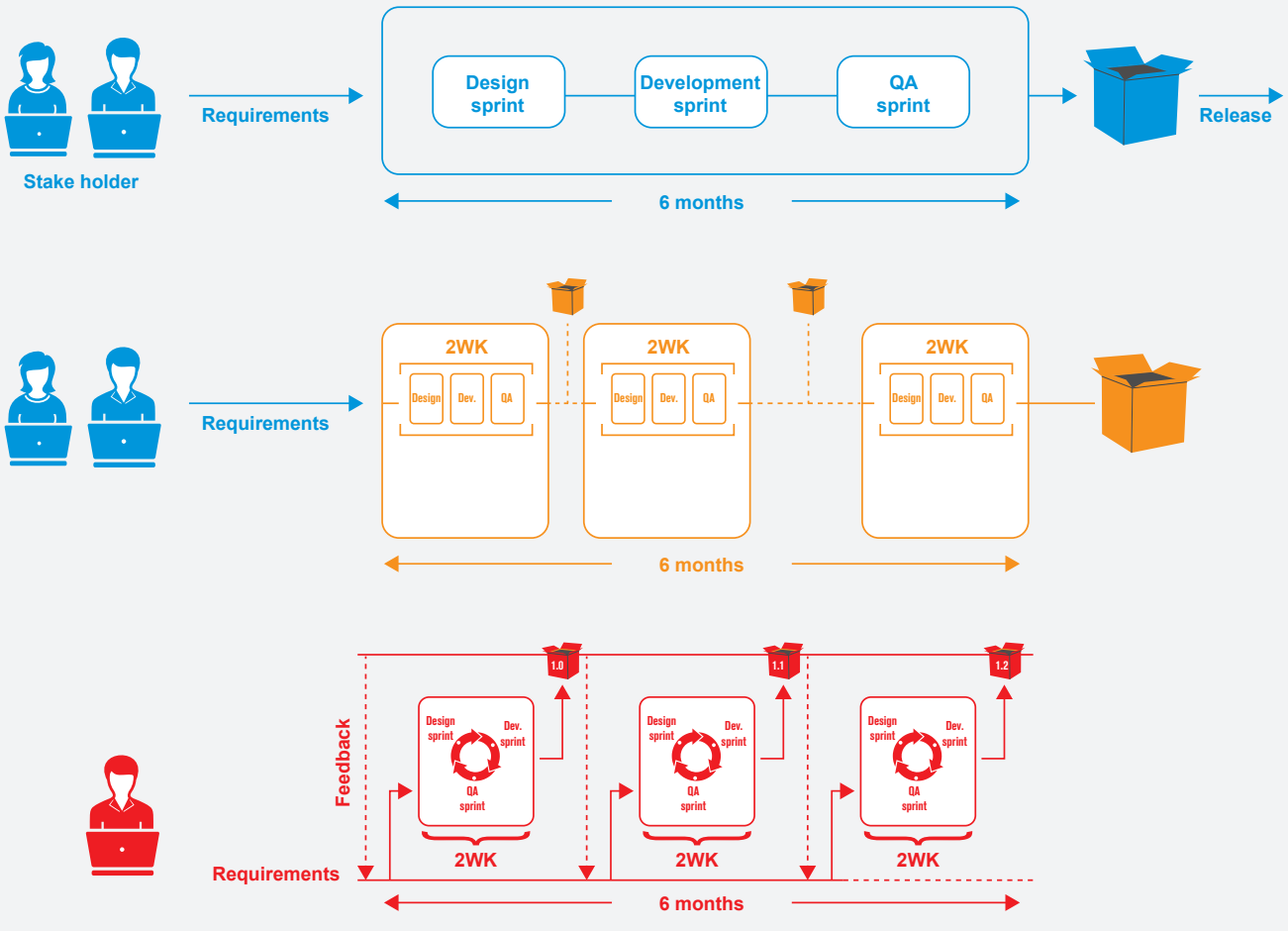


Figure 1: Phases in transitioning to Agile

The purpose of this white paper is to offer readers some concrete steps to bring Agile in to their organizations. The specific concepts and actionable steps in this paper include:

**Phase One**
(Shorter Release Cycles):

**Changing Culture:**
Moving to Shorter Release Cycles

**Phase Two**
(Two Week Releases):

**Engineering Steps:**
Creating Repeatable Short Release Cycles

**Phase Three**
(Continuous Delivery):

**Steering Your Success:**
Evolving to Continuous Delivery

Pushing this far can be hard to follow when trying to implement agile in larger organizations. Most large organizations have evolved to value repeatable processes and stability as they've grown. Larger organizations hold the view it's easier to consistently execute at scale (tens, hundreds, or thousands of projects) when every project follows a standardized approach using approved tools and infrastructure processes. Larger organizations often build up checks, balances, and audit processes that value manual controls in order to align with various internal and external compliance requirements.

True agile programs, on the other hand, require the ability to quickly bring in new frameworks, stand up new servers, and adopt new tools at the drop of a hat. Agile programs can't afford to deal with the loss of hours required for simple tasks like moving between environments.

# **Step By Step Guide** for Your Delivery Organization

Transforming to agile impacts your entire organization; however, this paper focuses on the activities from your delivery team. In this paper's context "delivery" refers to the teams, roles, and activities primarily responsible for directly creating, building, and delivering the software. This includes your developers, testers, analysts, devops, and similar roles.

## **Changing Culture:** Moving to Shorter Release Cycles

Successful organizations foster cultures that support, encourage, and expect teams to grow into high-performing ones. Large organizations often release once a year or less, while Agile favors much smaller release cycles. Moving to a faster cadence requires changing the organization's culture to support this process change. The sections below lay out a number of aspects that contribute to such a culture.

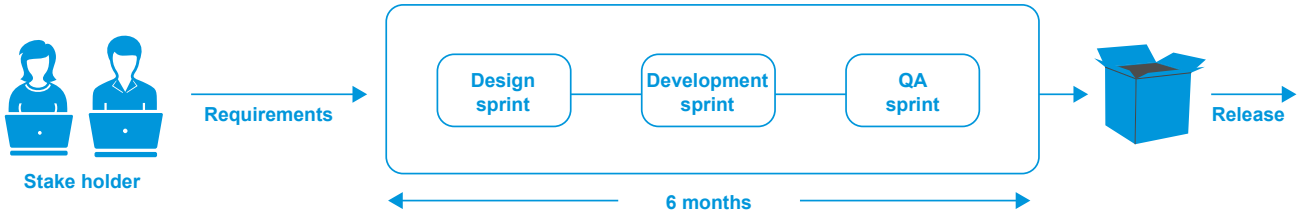


Figure 2: Shorter Release Cycles

### Break Down the Culture of Business Analyst vs. Developer vs. Tester

| | |
|---|---|
| WHAT | Analysts, developers, and testers need to be extremely close allies, not adversaries. Bring these three roles together as closely as possible. |
| WHY | These three roles are at the heart of your entire delivery process. Healthy relationships in this group directly contribute the business's goals: high-quality software bringing great value to your clients. Trust and healthy communication amongst these roles is central to that success. The collaboration of these roles is often referred to as Three Amigos and can reap huge benefits. |
| HOW | Set the expectation early that honest, open, respectful communication is critical for all teams. Take time each release train for team-building games that connect to a purpose and illustrate concepts around great communication. Use the Jenga game, the Dice, Rocks, Marbles game, or similar exercises. |
| RISKS | Building this relationship can take a long time, especially if the prior environment was "challenging." Work hard to reward success and ensure you're quickly squashing retrogrades (in private). Consider enlisting an outside facilitator. |

“I’ve worked on projects with and without close collaboration–it’s like night and day for me. Three Amigos empowers the tester: I can ask what I need to perform effective testing and the Dev team supports me. It helps me to identify requirements gaps immediately. Three Amigos conversations are the source of knowledge for my domain/business.

Three Amigos conversation enables parallel as I have acceptance criteria and implementation details as my outcomes of 3 Amigos. This means I can develop my test suite well before the code is ready. This enables faster testing and delivery. As a tester my inputs are respected and I feel better as I am contributing more to the project/company.”

Alex Michael, tester at a Fortune 10 global automobile manufacturing company

## Get Delivery Teams Involved With Decisions at the Envisioning/Ideation Phase

**WHAT**

Get developers, testers, and analysts involved early discussions when features are being determined.

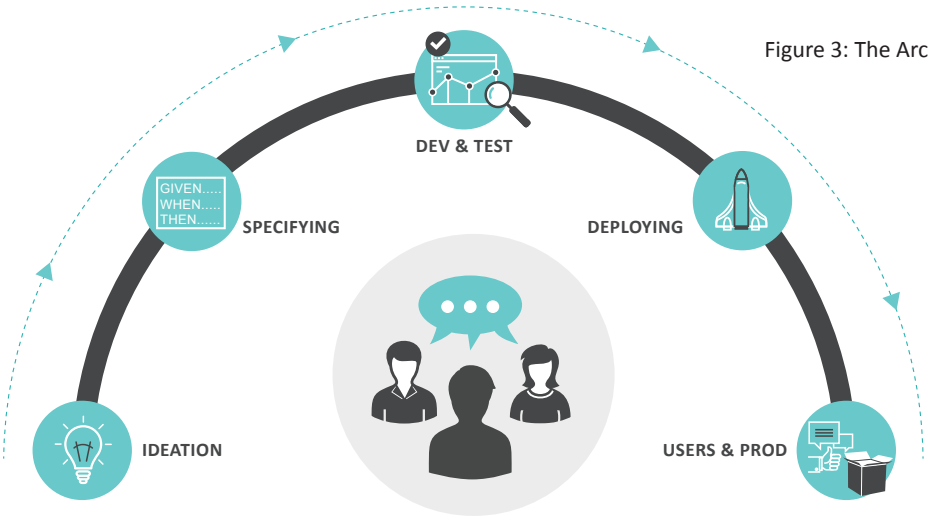


Figure 3: The Arc of a Feature

**WHY**

Too often organizations don't involve delivery teams in early feature planning. This results in a lack of knowledge which can skew value/risk decisions in a bad direction.

Consider a feature a business user would like:

- Pull data from an existing system
- Provide modest transforms to display data in a different business context
- Display data on a web page
- Offer same data on a mobile view

An initial estimate might be four weeks; it seems reasonable to the leaders and managers present because similar work's been done in the past. The business value of this particular feature is fairly high, so the decision is made to move forward with it.

Imagine how that conversation might have differed if representatives from developer, tester, and analyst activities were included: Integration with new data sources could add a week or two. Transforms might take two weeks longer than expected. Mobile testing adds another two weeks for testing.

Instead of the initial four-week estimate, leadership is now looking at ten weeks. The product owner might decide to take on completely different features!

**HOW**

Ensure you're bringing the right representatives from testing, development, and analysis in to early discussions when deciding which features to choose for the next release.

**RISKS**

Individuals may feel awkward or unsafe speaking up about risks, estimates, etc. when first starting out with this. Mitigate that by taking the time to build trust and ensure open discussion is honestly listened to and valued.

# Engineering Steps:
## Creating Repeatable Short Release Cycles

Once a team's culture has changed to support faster delivery the goal moves to making that delivery repeatable—as well as continuing to shorten it. Making a process repeatable means you'll need to invest in the actual engineering processes. You're shooting for improving the cultural changes you've started as well as investing in your tooling and methodology.

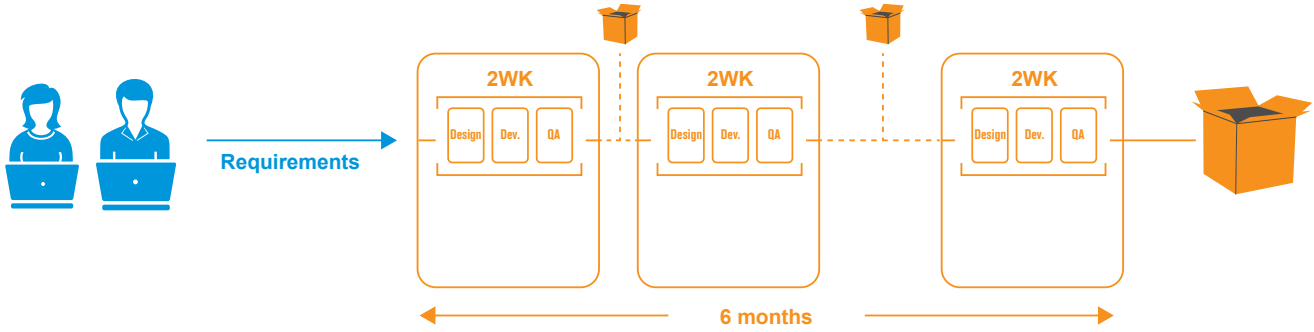


Figure 4: Two Week Release Cycles

## Create Automated Delivery Pipelines

**WHAT** Create an automated delivery pipeline as the first step in any software delivery project, no matter the size.

**WHY** Automated delivery pipelines take human error out of the problem space. Automated delivery pipelines ensure only secure, high-quality code makes it into appropriate environments. Automated delivery pipelines solve audit and compliance issues. ***An automated delivery pipeline is an organization's most important, highest-value feature, period.***

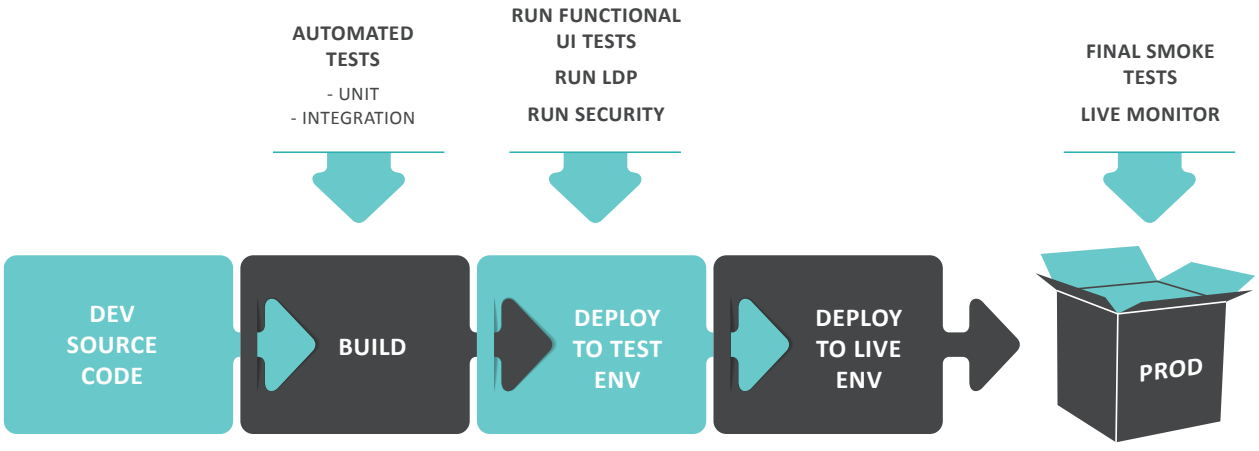


Figure 5: A Delivery Pipeline

HOW

Experiment with widely available tools like Jenkins, TeamCity, or Microsoft's Team Foundation Server. Choose a tool that has wide adoption across the software community, not an "Enterprise-level" tool with narrow adoption.

Every team in an organization should have some form of an automated delivery process that, at a minimum, handles the following steps:

- Pulls the latest version of the software from source control for all teams contributing
- Builds that software
- Runs automated tests as appropriate
- Runs static code analysis and other code-quality tools
- Deploys software (and data!) to an environment
- Runs post-deployment automated tests as appropriate (integration, security, performance, functional, etc.)
- Stores the deployable artifacts (binaries such as jars, database schema, etc.) plus related quality information (test results) into an appropriate repository

This pipeline is in addition to regular automation and quality checks that might run on a nightly basis to check other forms of quality such as load and performance.

RISKS

Pipeline tools take some time to learn. Set expectations properly. Pipeline tools also need someone to administer and monitor. They also require infrastructure to run on. Ensure you're setting up people and resources to properly support these tools.

## Automate Quality Checks

WHAT

Automate quality checks via testing frameworks, load and performance tooling, and static code analysis tools.

WHY

Automating quality checks provides an extraordinarily fast feedback loop on your software. Additionally, you'll have a reliable safety net that makes it much easier and safer to modify, extend, or otherwise alter your codebase.

HOW

First, if your team isn't already doing some form of automated testing, start planning for it immediately. There are many types of automated testing, and it's critical to have proper understanding of how to effectively implement automation. Types of test automation to consider include:

- Unit Testing
- Integration/Internal Service Testing
- External System Testing
- Functional/UI Testing
- Security Testing
- Load and Performance Testing

Starting test automation is extremely difficult. Good test automation requires practical experience, knowledge of specific patterns of software design, and the ability to plan out a workable implementation of automation that will focus on high-value aspects while meeting goals of long-term maintainability.

Consider bringing in outside skills to help get your team(s) started if you don't have practical experience in test automation. Bring in outside craftsmanship coaches, draw on skilled practitioners in other departments, or look to other places in your organization to get help from.

RISKS

Test automation done badly can actually be much worse than no testing at all. Bad tests fail for no good reason, they pass for the wrong reason (and that's scary!), they suck up a huge amount of maintenance time, and they crush trust in the teams. Mitigate this by ensuring your teams have members or coaches with *practical experience.* This can't be over-stated enough!

## Archive the Output of Your Pipelines

WHAT

Archive the artifacts your pipelines produce. This means the deployable assets (jar files, DLLs, database schemas, etc.) as *well* as the quality report artifacts.

WHY

Auditing and compliance are a major part of large organizations. Imagine you've discovered a critical fault in a production system. Archiving means you can pull down that exact set of binaries that were pushed to Production. You also have the entire set of quality reports generated from your test automation tools. Automating this archival process ensures you'll meet compliance requirements in your organization. Automating the process also builds confidence among doubters by proving to them required documentation and quality gates can be automatically generated.

HOW

The mechanics of implementing this will differ between organizations, but the concept is the same:

- First, automate your delivery pipeline
- Ensure each step of the pipeline is a quality gate that stops progression upon failure
- At the end of the pipeline assemble all quality artifacts (reports from L&P, unit and integration tests, functional checks, static analysis, etc.)
- Zip or otherwise package those artifacts with the deployed assets
- Push those to a repository such as Nexus, SharePoint, etc.

RISKS

Archived artifacts may not meet all compliance/audit requirements. Ensure you meet with compliance/audit staff and walk them through the process. You'll need their oversight and buy-in.

## Adopt and Enforce an Appropriate Form of Test-First Development (TDD, BDD, ATDD)

WHAT

Test-first development takes many forms. The gist, regardless of which form you use, is that tests are written before system code. Then the team writes just enough code to make the test pass. Test-Driven Development generally leans on many small unit tests and is a very engineering-focused. Behavioral Driven Development works on having conversations to flesh out how the system should function. Those conversations generally make their way in to specifications in test code. Acceptance Test Driven Development focuses on the high-level business value to drive out tests which in turn drive out system behavior.

WHY

Test-first development, regardless of the specific form, offers a number of concrete, measurable benefits:

- Nearly complete automated test coverage (Teams may decide to leave off data objects which have no logic and only hold state.)
- A full set of automated regression tests that build tremendous trust in the ability to work fast while evolving and updating the system
- A much cleaner system design. Many who've used some form of TDD will often refer to it as much as a *design* effort as a *development* effort.

HOW

Becoming proficient at test-first methodologies can be extraordinarily difficult if a team is trying to learn by themselves. Successfully adopting a test-first approach requires extreme discipline and accountability across all team members. It also requires a significant amount of experience, and constant feedback in to learning loops. Here are proven ways of bringing teams' skills up to the necessary level:

**Novice Team (No skill in agile or test automation):**

- Hire outside coach to run training workshops and embed with team
- Bring in coach who is an experienced practitioner from other teams inside the organization
- Combine coach plus e-Learning routes such as Pluralsight courses

**Moderately Skilled Team (Some experience in agile and test automation):**

- Supplement team with experienced practitioner, either from other teams inside organization or as a new hire
- Bring in a coach to assist team adopting practice
- E-Learning

**Advanced Team (Experienced in agile and test automation):**

- Empower team by getting proper tooling in place
- Set expectation that test-first is required
- Provide additional support via coaches, outside practitioners, etc.
- Ensure team is given bandwidth for learning and constant improvement

RISKS

See the risks outlined in Automate Quality Checks. Additionally, maintaining the discipline necessary for test-first isn't always easy. Time pressure and human nature can cause back-sliding. Backing up the delivery teams and ensuring a non-negotiable, clear set of processes mandate the proper engineering practices will go a long way.

## Compliment Test Automation With Skilled Testers

WHAT

Skilled, experienced testers are a necessary compliment to good developers and automated tests.

WHY

Skilled testers bring different views and skills to the table from developers—even test-minded developers. Great testers help the team focus on the most critical business value parts of software delivery. They create realistic test scenarios, understand how to avoid overlapping coverage, and ask hard questions otherwise overlooked.

> Test-driven development is often billed as a panacea for software failure: ‘practice it correctly and you'll have clean designs and zero-defect code!’ When it inevitably fails to yield those results, the implication is you must not have been TDD'ing hard enough. There are other impacts of practicing TDD, however: focus on the problem at hand, obvious starting points, and a continual sensation of progress. Plus, it slows developers down to think about the purpose and usability of their designs. All of these benefits would be game changers for most teams, despite the fact that they're mere side effects of practicing TDD rigorously.

Justin Searls, co-founder of Test Double. Consultant, developer, trainer.

HOW

Finding great testers is every bit as hard as finding great developers. Look for testers who have the mindset and skills discussed in the "Why" section above. Invest in growing younger, less skilled testers, or those with some level of aptitude and interest. Your testers will need the same sorts of bandwidth you provide your developers for skills training and continuous learning.

RISKS

Getting a bad tester on a team is every bit as crushing as getting a bad developer. Bad testers focus on low-value edge cases that aren't realistic. They take pride in finding bugs instead of helping the team prevent bugs.

## Frequent Cadence Of Pushing To Production

WHAT

Pushing to production in a mature, high-performing team is a non-event. Those teams have solidified their processes, toolsets, and methodologies in order to deliver value as frequently as possible. This is the culmination of all the items in this paper.

WHY

Shipping frequently to production creates several advantageous situations.

Frequent pushes identify roadblocks that have to be resolved to push smoothly. Addressing these roadblocks eliminates constraints one by one. No build and deploy server, set up Jenkins or TFS, deployment scripts aren't working. Address them and get them stable. Automated testing fragile and low value? Address that next. Approach those problems step-by-step starting with the worst offenders. Soon you'll have a smooth pipeline.

Second, frequent pushes encourage your team to think in small, high-quality, high-value chunks of work. That's a wonderful habit to build because you'll be able to react in an agile fashion to your stakeholders' and customers' needs.

Third, your entire organization will get a tremendous boost in their trust of the ability to quickly fix bugs, ship new features, and react to changing markets.

Many other benefits result from frequent pushes. These three are just a few.

HOW

Approach this from a constraints-based view: what's the biggest constraint in front of our teams right now? Solve that problem first, then move on to the next issue. As Mark Watney said numerous times in *The Martian* (both the book and the movie!), solve one problem, then move on to the next.

RISKS

Teams may become pre-occupied with fast pushes to production at the cost of stability and reproducibility. As with all things in software, focus on creating a smooth process, then work on speeding it up. Frequent pushes to production may also overwhelm your sales, support, and business departments. It may even be challenging for your customers to keep up. Mitigate this by keeping all those groups tightly integrated into your release process. Over communicate and ensure you're keeping documentation, training, and related support activities as part of your done criteria.

> The real value a skilled tester brings to the team is the creativity and structured design techniques that provide insight into scenarios, unique combinations, hidden business rules and unexpected system behaviours that aren't immediately visible on or near the happy path to meeting customer expectations. The world is full of accidents and malicious players that skilled testers are trained to efficiently emulate in accelerated time frames.

Paul Carvalho, Agile and Testing Coach

# Steering Your Success: Evolving to Continuous Delivery

Constant feedback in to a learning loop is a fundamental aspect of agile. Measuring your success as you progress through your agile transformation is crucial. Feeding measurements back in to your tooling, culture, and process means you're likely able to close the loop on continuous delivery—a constant flow of value to your stakeholders.

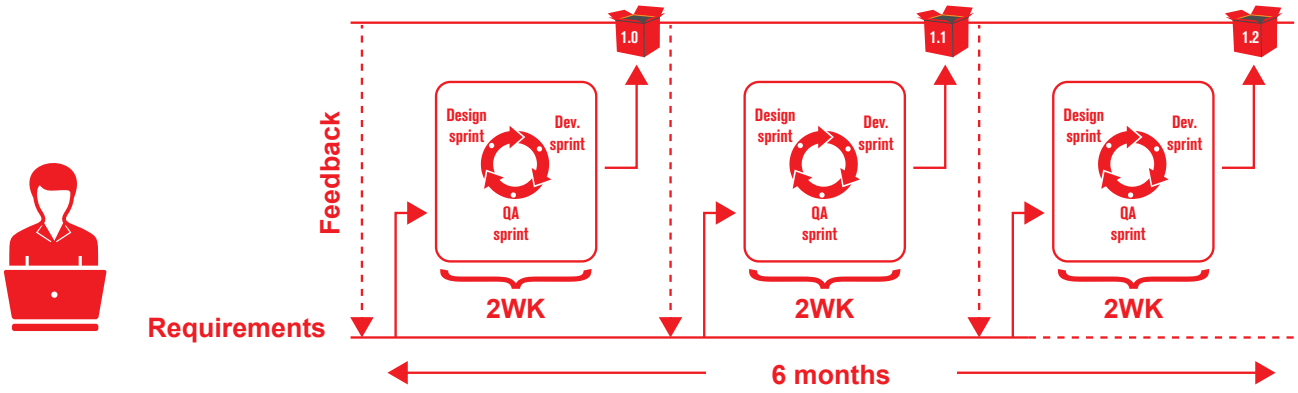


Figure 6: Continuous Flow Delivery

## Monitor Baseline Metrics

Metrics are both the boon and bane of software delivery organizations. Far too many metrics are misused (counts of test cases and bugs filed, anyone?), but a careful selection of metrics can give you reliable, actionable information. Mature organizations also understand that metrics are best used for trend warning, not as discrete measurements.

Below are a few metrics that are sensible to monitor as a team evolves its maturity in agile processes. These are not best practices (there isn't such a thing!), but are guidelines for organizations to consider.

| Work Item Size | Throughput | Cycle Time |
|---|---|---|
| Immature teams usually create work items that are far too large—weeks, or even months. Mature teams break work items into much smaller chunks: less than a week, and preferably one or two days. | How many stories / work items teams get from start to finish in an iteration. Improving throughput is an indicator many other processes are working well—work items are sized appropriately, little or no time is wasted in handoffs, the development/test/deploy cycle is working seamlessly, etc. | How long it takes to get a work item from start to finish. This is a direct factor in Throughput and indicates places where your process may not be smooth or resources/people are constrained. |

| Velocity | Commitment / Completion Ratios |
| --- | --- |
| Number of story points accomplished per iteration. Remember: velocity must never be used to compare teams' performance! Velocity likely differs between teams—it's a measurement unique to each team. | This should be measured both for points and stories. It's an indicator if teams are biting off more than they can chew. If so, have teams reduce their commitment. The point is not to make teams feel unsuccessful, it's to have them be open and honest about what they can accomplish. |

Many great articles exist on

- Agile Metrics: Agile Health Metrics for Predictability
- Five Agile Metrics You Won't Hate

### Run Regular Facilitated, Disciplined Retrospectives

Retrospectives are one of the best things you can do for your agile transformation (Three Amigos and effective daily standups are numbers one and two). Constant feedback is critical to making the agile transformation as smooth as possible; it's also crucial to your ongoing agile operations once you deem your transformation complete.

For guidance on running retrospectives look to reading material like Esther Derby's and Diana Larsen's *Agile Retrospectives: Making Good Teams Great*, or James Shore's and Shane Warden's *The Art of Agile Development.*

# Succeeding in Agile at Scale: **Continuous Improvement is Key**

As mentioned at the beginning of this paper, large organizations have unique challenges to implementing agile: slow communications, tedious standards and processes, and cultural inertia that can wear down the best intentions. Dealing with those challenges requires monitoring, feedback, and adjusting along the way.

Regardless of which practices from this paper organizations choose to implement, one overarching, foundational theme is critical to a successful agile transformation: Disciplined continuous improvement. Organizations striving to implement agile have to be honest about learning from the mistakes that will inevitably happen.

Every organization's agile transformation journey is different, and ignoring lessons learned along the way will only make it harder. Organizations that begin with clear expectations around a supportive, encouraging learning environment will have a faster, smoother transformation. Those organizations will be much better at breaking through the communication and inertia challenges found in any large organization's attempts to change their culture and business processes.

# About Zephyr

Zephyr is a leading provider of quality management solutions, powering intelligent DevTestOps for more than 11,000 global customers across 100 countries. Project teams and enterprises of all sizes use Zephyr's products to enable continuous testing throughout their entire software delivery pipeline to release higher quality software, faster. Zephyr is headquartered in San Jose, CA with offices in King of Prussia, PA, Europe and India. For more information, please visit www.getzephyr.com.

ZEPHYR
Real-Time Test Management

**Contact Zephyr Today!**
sales@getzephyr.com
www.getzephyr.com
+1-510-400-8656