

# Crossing the Value Stream: Improving Development with VMware and Pivotal Platform

By Matthew Gunter

Table of contents

Introduction 3

Applying lean principles to software 4

Value-stream mapping creates consensus and actionable insight 5

Lead time, process time and other terms to know ..... 6

Use swimlanes to show overlap ..... 7

1. Provide a developer marketplace for self-service (and refocus operations) 10

Evaluating alternatives for improvement.....11

Seven principles of lean software development ..... 12

2. Move to smaller projects and smaller releases 13

Make the move to microservices..... 14

3. Reorganize into cross-functional, self-directed teams 15

4. Increasing defect visibility 17

Test-driven development and continuous integration..... 18

Make defects visible ..... 19

Developers own QA ..... 21

5. Reduce variability and uncertainty 22

Measuring development velocity..... 22

Conclusion 24

Recommended reading 26

Pivotal is now part of VMware, so some of these products and services are now part of VMware Tanzu™. [Learn more.](#)

## Introduction

Adopters of Pivotal Platform (now part of VMware Tanzu) have achieved remarkable gains in developer productivity.

[Allstate](#), [Citi](#), [CoreLogic](#), [Liberty Mutual](#) and many others tell their stories at industry events, detailing how they've achieved 3x and 4x developer productivity.

The [story behind Humana's Cue app](#) for the Apple Watch is representative. Four Humana developers delivered Cue, a health app, to Apple's App Store in five weeks. It was released on schedule, in time for the Apple Watch launch event. The app achieved 300 percent of the company's usage goal<sup>1</sup>, and was named a "Top 10 app" by MacLife.

It's common to hear Pivotal Platform users tell anecdotes such as "we delivered 3x the number of features this year compared to last year" and "our developers went from working on code only 20 percent of the time to 80 to 90 percent of the time."

Where does this productivity come from? This white paper will tell you.

Another goal of this paper is to help you understand—and adopt—the specific practices that lead to better software development processes.

We'll demonstrate the practices that companies use to transform developer productivity.

Each practice section includes three parts:

1. Reviewing the underlying principles that drive transformation
2. Explaining why these practices work
3. Proving examples where enterprises have used these practices to successfully transform at scale

Along the way, we'll also show how Pivotal Platform enables each practice.

Below are two quotations from Citi and Allstate who have detailed their experiences on how they have achieved developer productivity with Pivotal Platform.

---

"Over the last eight months, I'm most proud that we've released three times the amount of features."

BRAD MILLER, GLOBAL HEAD, DIGITAL & CLOUD, CITI

---

---

"Instead (of 12-person team and 14 months), the project ultimately required only a product manager, one user experience designer and six engineers who delivered the desired product in just six months."

RICHARD LEURIG, SENIOR VICE PRESIDENT, INNOVATION DEVELOPMENT CENTER, CORELOGIC

---

---

1. *Keynote: Consumerizing Healthcare through Lean + Agile Digital Product Development*. 2015.

## Applying lean principles to software

For decades, *lean principles have delivered remarkable results in manufacturing*. These proven practices can be applied to enterprise software. The pragmatic, grounded nature of lean is a welcome departure from the buzzwords and hype often associated with the next big thing.

This paper discusses Agile, DevOps, microservices and cloud native from the perspective of proven lean principles. This helps you understand how they fit together and contribute to the previously mentioned productivity gains.

We'll start by analyzing traditional software development with value-stream mapping. From there, we'll discuss five common practices of Pivotal Platform users:

- *Developer self-service*
- *Smaller projects and smaller releases*
- *Cross-functional, self-directed teams*
- *Increased visibility of defects*
- *Reduced variability and uncertainty*

These five steps combine to yield big results, transforming your waterfall development process into an optimized value stream.



FIGURE 1: Use lean thinking to understand digital transformation.

Let's get started.

## Value-stream mapping creates consensus and actionable insight

*Value-stream mapping (VSM)* is a lean process improvement approach for analyzing and optimizing how work items flow through a process. The VSM method has its roots in manufacturing physical products but applies to software development deliverables too.

VSM focuses on how companies create value for their customers. Companies have many value streams (e.g., sales, support, maintenance, recalls, compliance, etc.). Any deliverable valuable to a customer is the result of a value stream. Improving a value stream requires information sharing and collaboration across departments and at multiple management levels.

In the world of software, a value-stream map represents how code flows from concept to production (code can be an incremental enhancement, a feature or an entirely new application). When you analyze the end-to-end process—ideally with multiple stakeholders—you create a baseline view of an entire value stream.

The value-stream mapping exercise builds consensus across teams and leaders.

At a strategic level, companies adopt Pivotal Platform to achieve five outcomes:

1. Speed
2. Security
3. Stability
4. Scalability
5. Savings

We call this the Five S's, and different companies (and departments) tend to focus on just two to three of the possible outcomes. This focus is perfectly reasonable, but there are important linkages between the different outcomes. Speed requires stability and scalability. Security depends on speed and stability. Savings results from speed and scalability. You get the picture.

Many companies want high-velocity software development to create a sustained competitive advantage. It's hard to do that by focusing exclusively on speed. We have to also achieve a level of security, stability and scalability to be successful in achieving speed in a sustainable and safe way for the larger organization. For this white paper, we focus primarily on improving speed across the software lifecycle, but keep the other outcomes in mind, too, because they are interdependent.

What matters for improving speed?

Figure 2 shows an example of a value stream for a traditional application release cycle. It shows how a given release is funded, prioritized and executed.

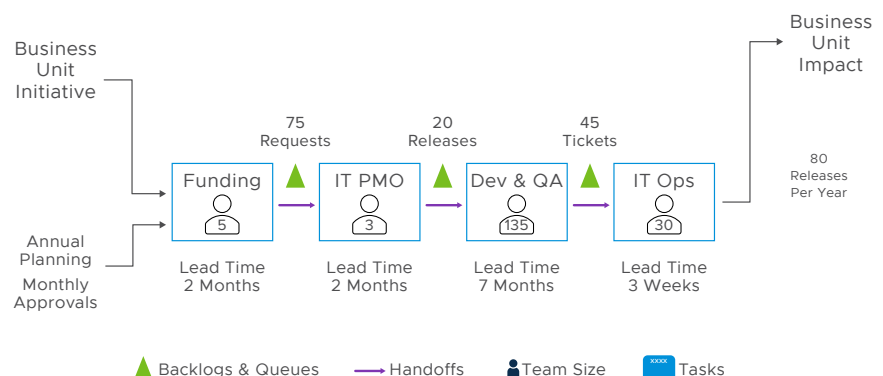


FIGURE 2: A value stream for a traditional application release.

This particular flow starts with the business unit objective and ends with the release having an impact (the apps can be customer-facing or internal). The work passes through several departments; handoffs and work backlogs are highlighted.

Differences of opinions and emotions tend to disappear when the facts and process details are clearly visible. The VSM helps everyone see how new techniques are linked to speed and other factors, so that we create synergistic changes rather than conflicting or isolated improvements. Understanding these linkages is key to driving business value from better software productivity.

### Lead time, process time and other terms to know

The power of VSM lies in analyzing the individual tasks of a larger process and finding ways to improve the whole value stream. You begin by examining the lead time for a given task (i.e., the total duration required to complete the task). Then, review the process time (the amount of time when work is actually happening).

The ratio of Process Time (PT) to Lead Time (LT) is the Activity Ratio (PT/LT). The difference between LT and PT is Idle Time (LT-PT). When a process is improved for the first time, it's common to discover an activity ratio of well below 10 percent (and sometimes below 1 percent). That's a lot of time when work is idle.

PT is further segmented into Value-Added effort and NonValue-Added effort. Figure 3 shows a single task and the basic metrics of LT and PT.

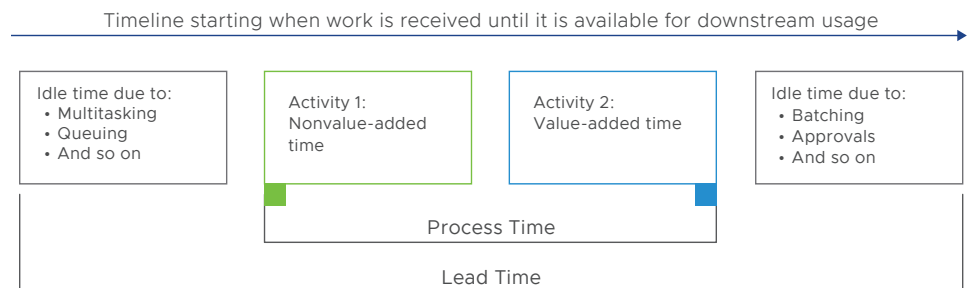


FIGURE 3: A sample analysis of a task in the software development stream.

Any work in software development can be decomposed into a map similar to Figure 3. For example, consider a developer who requests an environment for testing, or an operator that configures a new server. Both tasks can be analyzed with value-stream mapping. The level of granularity (i.e., the task breakdown level) to use in a map is somewhat arbitrary. It can be helpful to start mapping with larger, simplified tasks, and break them down as the analysis evolves.

Here are a few other valuable lean metrics you should know:

- Percent Complete and Accurate (%CA) measures the quality of a task's output. If a downstream task has to add, clarify or correct anything about the work item output, the initial output isn't considered complete and accurate.
- The number of staff helps calculate the potential labor savings that result from a given task's improvement.
- Barriers to flow are the reasons why a task is not closer to an ideal state (an ideal state is when work is done without delay and requires only value-added effort). It is insightful to realize that removing a single barrier to flow can have a profound impact on the way work is done within a task and through the entire value stream (later in the paper, we will show how Pivotal Platform removes several key barriers for developers).

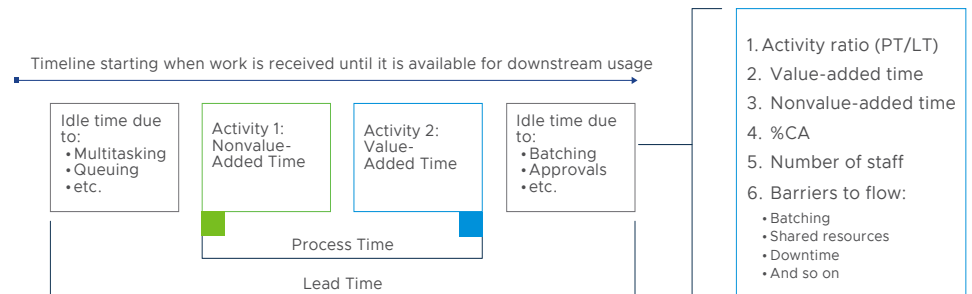


FIGURE 4: A sample analysis of a task in the software development stream and lean metrics.

### Use swimlanes to show overlap

We've reviewed the most important metrics in value-stream mapping. Now, let's apply them to a sample software release value stream.

Let's use a map that includes swimlanes. Swimlanes highlight how work is shared and handed off between teams. This clarifies the functional team responsible for a task. It also shows which tasks are sequential and which are concurrent. This distinction matters because:

- For sequential tasks, the entire value stream improves if task lead time and %CA are improved.
- For concurrent tasks, the tasks must be more carefully coordinated to minimize waiting or made completely independent so there is no waiting. In either case, it is important to reduce the costly back-and-forth interactions that lead to long or unpredictable delays and nonvalue-added communication and coordination efforts.

The software value stream is redrawn with swimlanes in Figure 5. The first problem we will highlight is the inefficient interactions between the development team and operations.

In Figure 5, operations work is nonvalue-added activity. Development interaction with operations is also nonvalue-added activity.

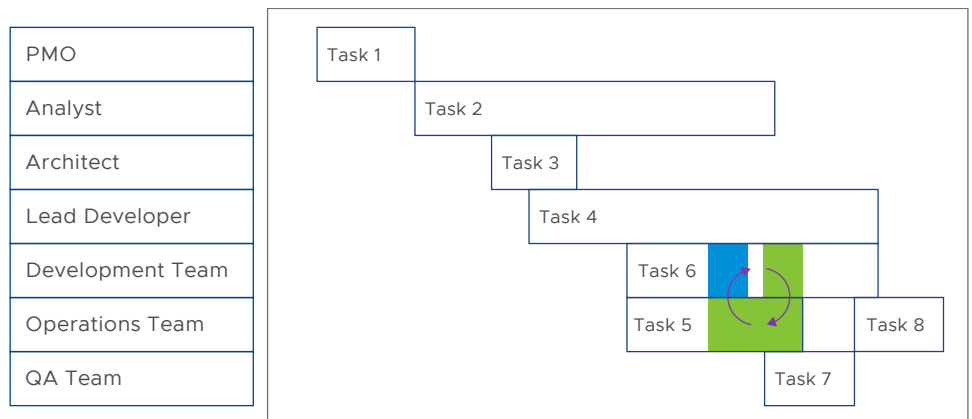


FIGURE 5: An organization that releases applications and updates infrequently.

Here, development and operations work overlap. Development starts while test, staging and production environments are provisioned. If operations is also responsible for dev and test infrastructure, significant ongoing interaction with development is required. (The teams have to jointly debug, configure and examine log files.) You often see a ticket backlog for operations, a sign that significant improvement is possible.

The interaction between Dev and Ops is depicted by the red arrows (see tasks 5 and 6 in Figure 5). This nonvalue-added communication and coordination is a major barrier to the value creation (i.e., coding and testing). It's common for analysis of the interaction to show delays, rework and other problems. To compensate for delays, developers often multitask by working on something else, thereby losing focus due to delays and context switching. Aside from the productivity impact, this complex interaction undermines the certainty and simplicity required for a smooth value stream.

The metrics behind this waste are captured in the value-stream analysis. That information can foster a shared understanding and build consensus for specific improvements. For example, the tables in Figure 6 compare a developer's week with and without Pivotal Platform improvements. Notice the reduced time spent with cross-team coordination and environment maintenance, along with over 2x increase in time available for coding.

CURRENT STATE			PIVOTAL PLATFORM ENABLED		
STEP	ACTOR	HOURS	STEP	ACTOR	HOURS
Coding	Developer	7.5	Coding	Developer	20
Unit tests	Developer	2	Unit tests	Developer	2
Sprint activities	Developer	4.5	Sprint activities	Developer	4.5
Cross-team coordination	Developer	16	Cross-team coordination	Developer	7.5
Code/design review	Developer	1	Code/design review	Developer	5
Environment maintenance	Developer	8	Environment maintenance	Developer	0
0 value meetings	Developer	1	0 value meetings	Developer	1
Sprint hours		40	Sprint hours		40
Productive development hours		9.5	Productive development hours		22
Activity ratio		23.75%	Activity ratio		55%

FIGURE 6: Example comparison of developer's week with and without Pivotal Platform.



Our experiences with value-stream mapping show that many interactions between Dev and Ops teams cause delay and nonvalue-added activity, including:

- Requesting, provisioning, and configuring infrastructure and software
- Clarifying (and correcting) infrastructure configuration
- Communicating about architecture, security and shared infrastructure
- Coordination of access to diagnose, trace, debug, access server logs and other restricted systems
- Resolving differences in configuration or behavior between environments
- Propagating changes (such as JVM memory settings) consistently across different environments
- Removing performance and resource constraints through scaling or tuning

The more you eliminate these interactions, the more independent and smooth development work becomes. Ideally, developers should be able to access just enough resources, just in time. Instead, most development teams deal with waiting, ticket-processes and over-provisioning just in case it will be needed. They are also often burdened with the ongoing maintenance of development servers and environments.

Pivotal Platform removes this problem. How? With a pull-based self-service marketplace that empowers developers to access, deploy and manage applications. Applications and environments are in full control of developers. They never have to wait on operations.

A pull system means you maintain resources, ready to be pulled, by the consuming step (developers in our case). Maintenance of the resources (i.e., compute, disk, memory, IP's, DNS names, etc.) is done by operations asynchronously.

This brings us to our first recommended improvement for the value stream: Provide a developer marketplace for self-service provisioning and deployment.

1. Provide a developer marketplace for self-service (and refocus operations)

Pivotal Platform enables self-service provisioning and deployment. This shift from push to pull dramatically reduces nonvalue-added activities, such as ticket processes and maintaining bespoke automation.

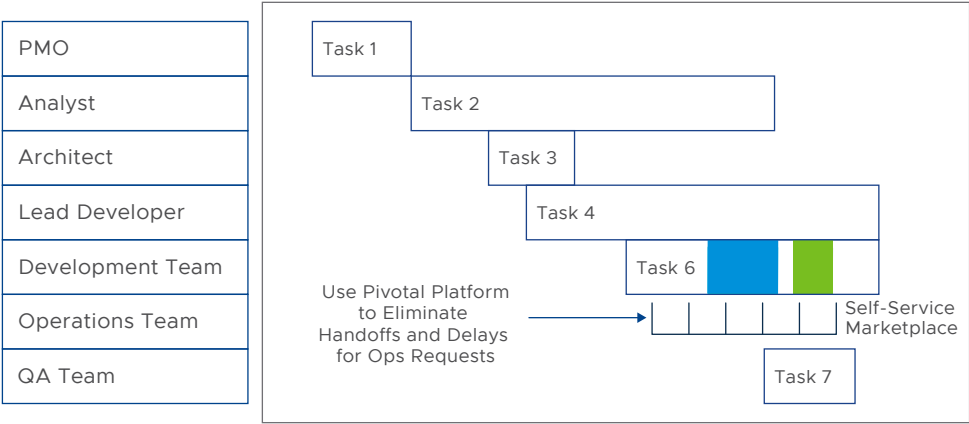


FIGURE 7: Value stream after eliminating task 5 for operations.

Pivotal Platform automates and coordinates operational tasks, such as creating environments, updating networking changes and adding capacity. This means developers have self-service access to environments in minutes. They can deploy apps as well, and they never have to wait on an operations team or take on the burden of managing their own infrastructure.

Self-service yields major reductions in nonvalue-added effort and uncertainty. As a result, there is simply more time for value-added activities, thereby reducing development lead time and variability.

“Wait-time for a new environment went from 90 days to 15 minutes.”<sup>2</sup>

ANDY ZITNEY, ALLSTATE

What about on the operations side? Pivotal Platform helps this group, as well. Instead of working on individual applications, operators can focus on running Pivotal Platform for all developers, projects and applications.

This one practice can double developer productivity. That means developers spend twice the amount of time on development. Want to learn more? Read the VMware white paper, [From Months to Hours: Accelerating Software Deployment with Pivotal Cloud Foundry](#).

2. Keynote: Developing the Freedom to Disrupt. 2015.

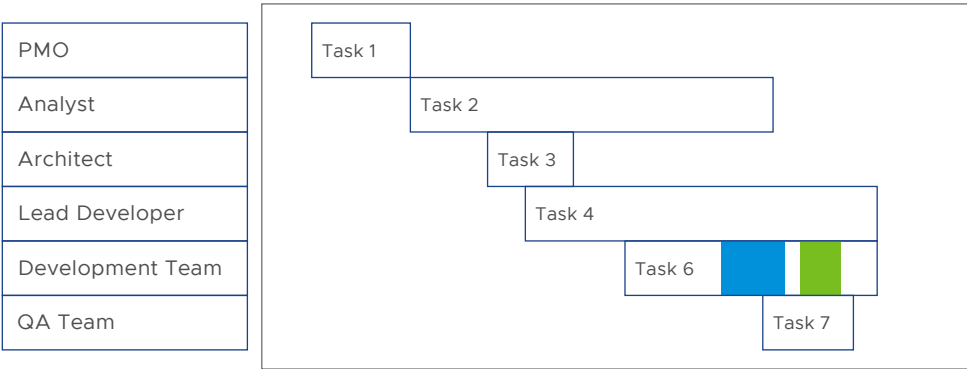


FIGURE 8: Value stream after eliminating the operations swimlane and task 5.

Evaluating alternatives for improvement

The Value-stream mapping also helps evaluate possible alternatives.

For example, one suggestion might be to give developers self-service access to servers, VMs or Kubernetes pods. But this approach does little to improve the value stream.

When developers have to install, configure and maintain infrastructure (or even Docker images), it merely shifts nonvalue-added activity to the development swimlane from the operations swimlane. The coordination of resources is still manual. That doesn't boost productivity, and the maintenance overhead can grow over time.

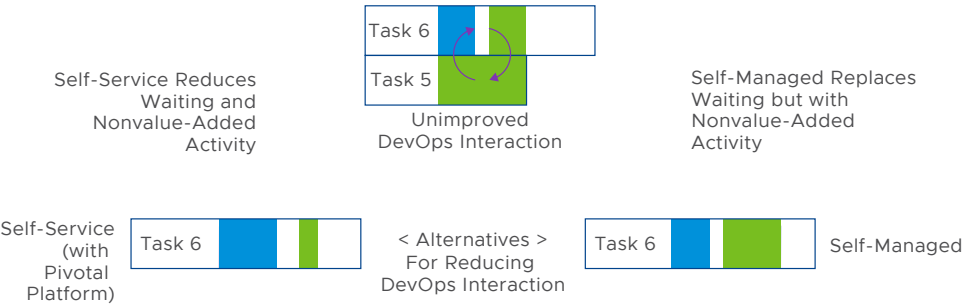


FIGURE 9: Comparison of developer self-service with developer self-managed.

### Seven principles of lean software development

Lean has been used to improve manufacturing efficiency for several decades. With any popular practice, a wealth of knowledge has been published. Techniques have been refined and tested through decades of real-world use. Best practices are learned, documented, shared and improved.

The *Institute of Industrial and Systems Engineers* maintains a formal “body of technical knowledge in the domain of lean and continuous improvement practices.” So, lean is an approach to process improvement that has been proven to work operationally, and there is broad understanding of why it works.

As we have seen already, we can apply lean with value-stream mapping to understand software development and the impact of Pivotal Platform, its benefits and how the platform enables agility and productivity for developers.

Applying lean to improving software development is not new. Making the software development process more productive through lean thinking was first seriously considered and explored in the book, *Lean Software Development: An Agile Toolkit*.<sup>3</sup> In it, the authors describe 22 techniques based on seven lean principles for improving software development:

1. Eliminate waste. “Doing the absolute minimum necessary to get the desired result.”<sup>4</sup>
2. Build in quality. “The quickest way to simultaneously increase your productivity and the quality of your software is by having a full suite of automated tests.”<sup>5</sup>
3. Create knowledge. “The best approach to improving a software development environment is to amplify learning.”<sup>3</sup>
4. Defer commitment. “Delaying decisions is valuable because better decisions can be made when they are based on fact, not speculation.”<sup>3</sup>
5. Deliver fast. “Compressing the value stream as much as possible is a fundamental lean strategy for eliminating waste.”<sup>3</sup>
6. Empower the team. “The people on the front line combine the knowledge of the minute details with the power of many minds.”<sup>3</sup>
7. Optimize the whole. “The idea is to seek out and remove the current constraint to growth, recognizing that the constraint will move to another place once the current constraint is addressed.”<sup>3</sup>

Because lean is a broadly understood and accepted methodology, it can help executives create clarity, build alignment and overcome confusion around organizational transformation efforts.

---

3. *Lean Software Development: An Agile Toolkit*. Mary Poppendieck and Tom Poppendieck. 2003.

4. *The Lean Book of Lean*. John Earley. 2016.

5. *The Art of Lean Software Development*. Curt Hibbs, Steve Jewett and Mike Sullivan. 2009.

## 2. Move to smaller projects and smaller releases

Self-service saves time. There is another benefit: Every step required to deploy to production has a much lower cost in time and complexity. Pushing code to prod—perhaps with a *blue-green approach*—is automated and repeatable. Ongoing management of many environments and deployments is also more efficient.

This efficiency means that application size, release size and project life span can be much smaller in scope and duration. This drastically improves time-to-market compared to the typical six- to nine-month release frequency.

Without self-service, projects and releases only made sense if the feature set was of a certain size and impact to justify it all. The operational overhead (and associated delays) for setup, maintenance and teardown of environments and infrastructure was too high.

Now, with self-service and lower operational costs, we can quickly develop, test and release smaller changes and new independent apps. Moving to smaller batch sizes is a classic lean technique for driving process efficiency and agility. For the same reasons, software development also benefits from building and releasing smaller feature sets.

Deploying to production with Pivotal Platform is as simple and low cost as it gets. Here are a few of the reasons why:

- *Orgs, spaces and roles* are key features for grouping users into teams with access to any number of isolated environments. Users with the org manager role can create and delete spaces (e.g., environments). Orgs and spaces usage is governed by quotas for memory and disk consumption. This permissions model makes it easy for users to have the right access for their role, while still adhering to common corporate governance policies.
- *Buildpacks* are a key technology in Pivotal Platform. Buildpacks preserve developer control over how applications are configured. They also allow operators to patch and upgrade the middleware components of the application and the underlying infrastructure.
- *Automatic containerization* is done when applications are deployed (or pushed) to Pivotal Platform. This minimizes developer concerns and enforces standards, security and efficiency.
- *Built-in networking* enables every deployed application to be immediately accessible via a configurable URL.
- Integrated application *logging* and *metrics* provide a baseline of observability and troubleshooting.
- The *services marketplace* provides self-service provisioning databases and other backing technologies.

When you deploy your application to Pivotal Platform, the platform automatically makes your containerized, running app accessible via a URL. This powerful capability allows new apps and features to be rapidly developed by individuals and very small teams. These teams can create additional Pivotal Platform space for testing or staging on demand and without significant cost.

Why do smaller projects and releases make so much sense? They are cheaper. They simplify decisions, and minimize requirements gathering. Small applications are easier to understand, debug and maintain. In short, they reduce complexity.

Your users get new features faster and cheaper—and with less risk. As shown in Figure 10, the developer activity ratio is higher with smaller releases, finished in weeks when compared to larger releases that take months or years.

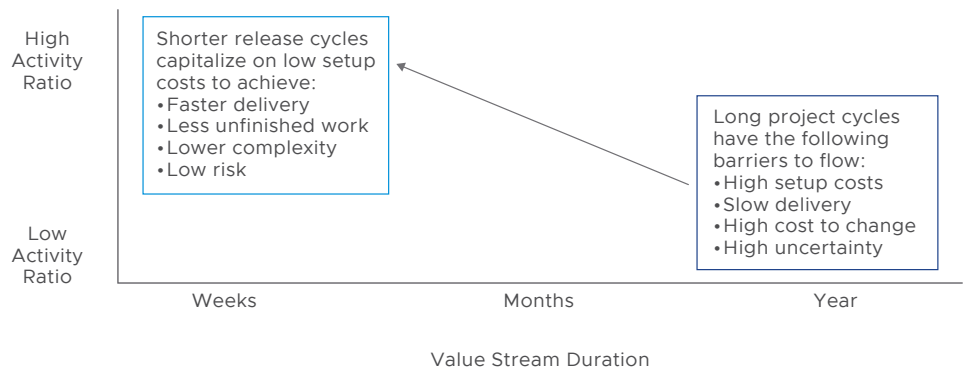


FIGURE 10: Smaller releases drive up the activity ratio for a number of reasons.

Smaller batches are a proven way to improve activity ratio and resource utilization. This approach can drastically reduce nonvalue-added activity. (As a fun aside, take a look at the [penny game](#). It helps visualize the power of breaking sequential work down into smaller pieces.)

Another benefit of smaller projects: It's far easier to add developers. Small teams that work on separate applications have less communication and complexity to overcome. So adding a new team and a new app is simple. This technique helps with scaling development even if the different applications interact as part of a larger system.

### Make the move to microservices

Many Pivotal Platform shops take this idea a step further with a microservices architecture that supports small releases. Greg Otto at Comcast [cites](#) that their most strategic applications are being developed for Pivotal Platform. This work is part of a broader strategy to systematically decompose monolithic legacy apps into a more flexible microservices model.

How does Pivotal Platform and the Spring Cloud framework help with the challenges of building and managing microservices? Find out more in the white paper, [Running Microservices on Cloud Foundry](#).

Corporate organizations have traditionally built enterprise systems to support their business. We have learned from a particular client that these systems were built on classical shared system models which are not conducive to continuous deployments. With a core focus on this client's experience and needs, we've been on a journey to deliver solutions faster and more efficiently. This endeavor has encompassed building agile practices and cloud ready applications. Our transformation leveraged the 12-factor methodology, thereby allowing us to build cloud ready applications with modern patterns. We have shared these lessons that we have learned, as well as provided guidance on building scalable software-as-a-service applications.<sup>6</sup>

6. *Tech Modernization: A Cloud Migration* - Henri van den Bulk & John Berry, Schwab, 2017.

3. Reorganize into cross-functional, self-directed teams

Let’s revisit the value-stream map. Our process still includes tasks spanning several different functional swimlanes. PMO, business analysts, architects and developer teams have a role in the delivery of a new application.

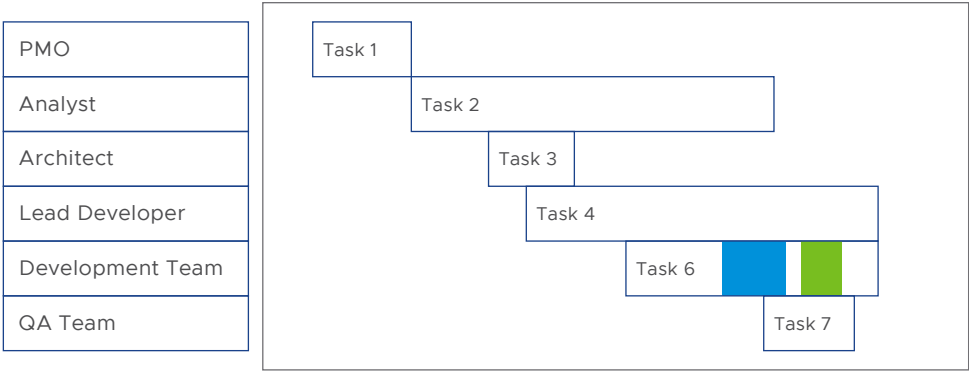


FIGURE 11: The value stream.

We saw earlier how Pivotal Platform simplifies the value stream by removing the operations swimlane with Pivotal Platform and self-service. Operations staff no longer need to touch or even be aware of applications moving through the value stream. This is a big win. But how can we reduce the number of swimlanes further?

With smaller releases that deliver working code into production quickly, it is realistic to combine the other functional expertise (such as architects, DBA skills and analysis skills) within the development team. This eliminates the handoffs and interaction that used to be required. It also fosters autonomy and agility.

Here’s our new value-stream map with the combined team.

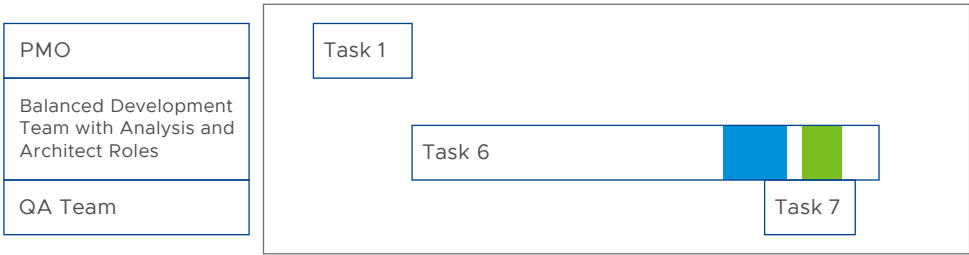


FIGURE 12: Value stream with a cross-functional development team.

A combined team is the most efficient. But it can take some getting used to. VMware helps companies adopt this model through *VMware Pivotal Labs* engagements. Engineers from an enterprise work alongside VMware staff within a small, balanced team. *Pair programming* and *overlapping pair rotation* are used to cross-train team members.

Organizing teams in this way cultivates shared knowledge across the entire value stream and project. *Studies show* that rotations and knowledge sharing maintain high productivity, even as team members come and go over time.

Another consequence of learning by pairing is that junior staff quickly become productive. As you can imagine, this boosts productivity also.

---

“A cross-functional product team now has all the skills it needs, and with a self-service platform it can do that work when it needs to do it, without delays.”<sup>7</sup>

DANIEL JONES, ENGINEERBETTER

---

Productivity and quality are proven to be improved by organizing around cross-functional teams that pair and cross-train. However, the value stream must be optimized for efficient development first. For example, it's counter-productive to pair program if developers are frequently delayed by ticket processes, interrupted by other duties or doing a lot of context switching.

Starting with the self-service provided by Pivotal Platform, you are able to make the move to small releases. This, in turn, enables agile, autonomous, cross-functional dev teams that deliver rapid time to market.

---

7. *The Anthropic Sympathy of Platforms*. 2017.



#### 4. Increasing defect visibility

The software value stream can also be improved by boosting the %CA quality metric.

In our analysis so far, we made a simplifying assumption: Quality is perfect (i.e., zero defects). We did not consider the effect of the %CA measurement for each task (and the subsequent rework and risk that it causes).

Let's once again consider our development process and focus on the rework and handoffs associated with testing. Each task in Figure 13 has a %CA of less than 100 percent. Therefore, each handoff from coding to testing could end up having a defect reported back to the coding task via the bug database. Likewise, each handoff to final stakeholder review may require additional corrections or enhancements before it is production ready. How does this situation impact the value stream?

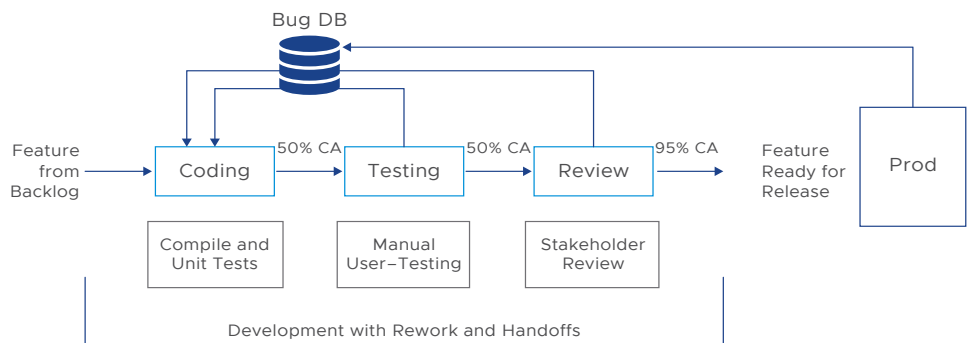


FIGURE 13: Detailed value-stream map showing defects, rework and handoff overhead.

Clearly, defects must be kept very low for the value stream approach to work. If the defect level is high or uncertain, then management is forced to add validation and testing steps before production deployment. This necessity pushes back to more of a waterfall process and undermines lean improvements made so far.

For example, the cost to fix defects goes up after every task handoff. Plus, any work done since the defective code was written can increase the cost to fix. So, ideally, we need to detect defects quickly, before handoffs and before new code is written.

In most enterprises, testing remains a separate swimlane. Not all defects will be found by the developers because they rely on a separate testing team for quality assurance. Let's do the math in our example from Figure 13. The %CA for the coding task and the testing tasks are shown at 50 percent. This means that a code change has to pass at least 2–3 times through the testing task on its way to the review task, and each review step requires multiple testing steps (another 2–3 times). So, that is approximately 4–6 different handoff delays and the communication overhead of tracking via the bug database.

With these factors in mind, it's clearly preferable to relocate defect detection upstream to the developer. The more real-time feedback developers have about quality and accuracy, the sooner the defects can be efficiently resolved. Our goal is to minimize the defect life span and reduce the rework effort. This also means eliminating handoffs and using bug-tracking to only manage the few defects found after code is released.

Defects have five negative effects on productivity:

1. Rework consumes time that could be used for value-added work.
2. Writing more code when there is an unresolved defect increases rework.
3. Rework itself can create more defects.
4. Rework creates unpredictability, such as a surge in defects after certain releases.
5. One serious defect can hold up the entire release.

Figure 14 depicts the handoff delays caused by the Dev-QA interaction in the overlapping swimlanes, and the purple arrows that represent communication and other nonvalue-added activities.

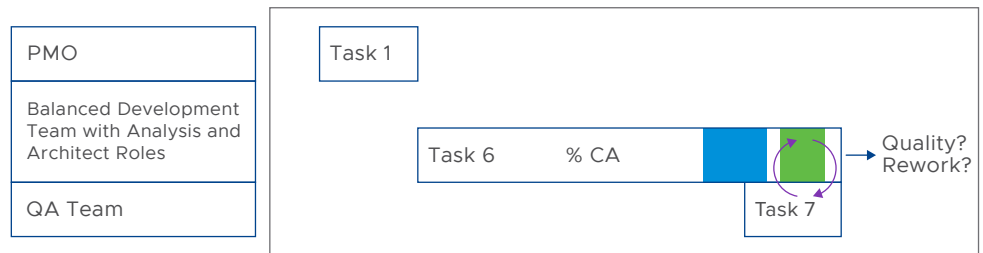


FIGURE 14: Development and QA interactions are slow and nonvalue-adding.

The value-stream analysis with %CA metrics confirm that defects, rework and handoffs severely limit the productivity of the development process. What can your team do to overcome this particular barrier to flow?

Three countermeasures can be combined to make Defect Visibility and speedy Resolution a top priority:

1. *Test-driven development (TDD) and continuous integration (CI) testing*
2. *Make defects visible* (and stop writing new code until the defect is fixed)
3. *Developers own QA*

Each of these are proven and are described further in this section.

### Test-driven development and continuous integration

The first technique paves the way for the others: Always be testing and validating the correctness of code and design assumptions.

TDD and continuous integration and continuous delivery (CI/CD) are used by high-velocity teams to provide fast, reliable feedback on the current quality of a piece of code. TDD is a popular development methodology, whereby tests are written before writing the associated code. With this approach, a new line of code can be tested immediately.



FIGURE 15: Common CI/CD pipeline from code repository check-in to production.

These practices lead to very high test coverage (i.e., greater than 96 percent) and very low defect rates. Defects that are not caught via CI testing indicate that a new test is needed so that these false-negative results are prevented from recurring. Add this test to the test suite, and the defect will be identified automatically next time.

The VMware white paper [\*Speed Thrills: How to Harness the Power of CI/CD for Your Development Team\*](#), provides details on how Pivotal Platform and the open-source CI tool Concourse help improve %CA, reduce rework and improve productivity.

It's important to point out that continual testing and maintaining of a pipeline, as previously described, is much easier with Pivotal Platform and Concourse. Pivotal Platform provides self-service environments, environment consistency, scalability and scriptability. Missing any single capability makes this practice much more difficult. Concourse is also essential. It's designed to enable versioning of pipelines and repeatability.

Together, these products simplify concurrent development and testing on multiple versions of your app. Even if you don't normally have multiple active versions of an app, it is good practice to be able to keep the source code, tests and pipeline definition always in sync.

#### Make defects visible

Your development pipeline should show the state of the build. That's another reason VMware recommends Concourse. The tool includes a very simple way to visualize quality. Development pipelines are color-coded to reflect status:

- Green means good
- Red indicates a problem
- Yellow indicates a step is actively executing

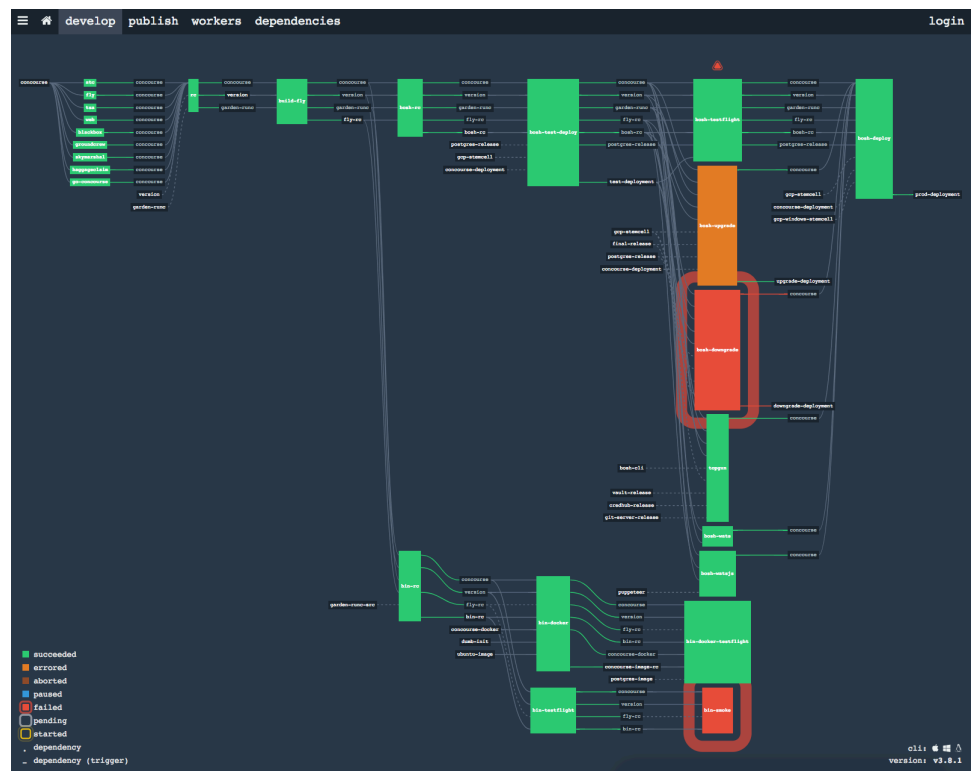


FIGURE 16: The Concourse dashboard makes failed tests highly visible.

Figure 16 shows a Concourse pipeline dashboard. This is usually displayed prominently on flat-screen monitors in the team's development area. When there is a problem, it's immediately known and defects are resolved quickly.

## Developers own QA

Now that we're doing test-driven development (TDD) and continuous integration (CI) to increase the visibility of defects, we have the development process shown in Figure 17.

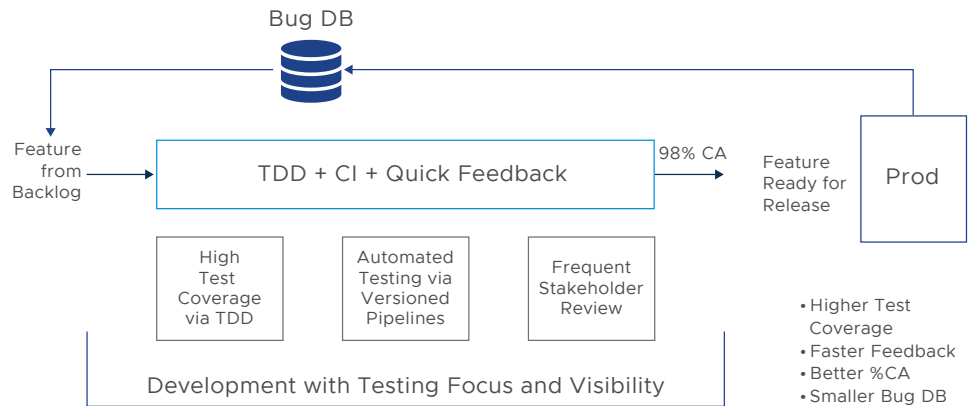


FIGURE 17: Continuous testing and quick feedback raises defect visibility.

When this approach is taken, testing and coding are inseparable. Organizationally, we then can merge QA with development, and we have this final swimlane view of our VSM.

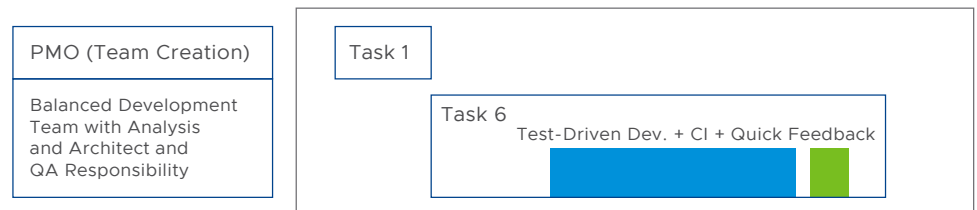


FIGURE 18: Highlight-optimized value stream.

With QA now part of the balanced, cross-functional development team, we can shift our value stream focus toward improving velocity, consistency and predictability.

It can take 6–18 months to get the practices shown in Figure 18 in place. At that point, the development process will be up to 3–4 times more productive than the pre-existing process.

“[Pivotal Platform] improved our ability to deliver quickly... In the old way of doing things, we configured everything through Chef. We built servers. We had this dodgy build system. It really could take weeks to get the feature into production. Now with this tool set that we have along with this CI/CD capability, [Pivotal Platform] has really accelerated that. We could push something in 5 minutes... It's been a dramatic improvement for us.”<sup>8</sup>

PHILIP GLEBOW, DIRECTOR IT ARCHITECTURE, PRODUCT ARCHITECT, GAP INC./GAPTECH

8. *Replatforming: Moving Custom Apps to the Cloud*. 2020.

## 5. Reduce variability and uncertainty

Finally, we turn to the topic of reducing variability. Variation is a synonym for risk, increased cost, missed dates and loss of information. Variability also makes it hard to predict how much effort and time tasks will take, contributing further to uncertainty. The waste and inefficiency due to variation and uncertainty are normally only visible after removing the larger sources of delay and nonvalue-added work from the value stream.

How can we start to see this type of waste?

### Measuring development using velocity

Software productivity is notoriously difficult to measure. You can use lines of code, function points or something else—it's still hard. By starting with the value-stream mapping approach, we don't try to measure productivity early on. Instead, we measure and eliminate waste in the form of delays, handoffs, rework, defects and batch sizes. This approach is more concrete and actionable. However, to continue to improve the development process, we now need a unit of work and a unit of work per week type of metric.

One good solution: Use the concept of a [story point](#). It's not perfect. But story points do make a relative gauge of development output possible. When your team estimates the size of their work with story points, you can calculate team velocity—a good measure for productivity. We see Pivotal Platform adopters use the total number of story points a team completes in one week as the measure of velocity.

As the team gets accustomed to story points, the velocity of a team can be measured and improved. A consistent approach to sizing features and delivering at a steady velocity are hallmarks of a mature development process. This should be your long-term goal.

Low variability means velocity is predictable over time. When variability is low, teams can:

1. Estimate feature delivery timelines with very high accuracy.
2. Understand if changing a practice or a step in their process has had a beneficial impact on velocity.
3. Understand the impact of churn on the development team.
4. Prevent unhealthy, unsustainable practices (such as developers working nights and weekends).
5. Have a consistent work-life balance.

At Pivotal Labs, we have learned to reduce variability through several workplace practices:

1. Uniform start time (9:00 AM)
2. Breakfast in the office (8:30–9:05 AM)
3. Daily standup (9:06 AM)
4. Overlapping pair rotations
5. Team retrospectives
6. Eliminate interruptions

When these day-in-the-life practices become routine, there's that much less that can change in a given sprint. This consistency is key.

Another key to reduced variability: Pivotal Platform. The platform makes infrastructure and operational changes non-events that do not disrupt development. Teams that use Pivotal Platform aren't impacted by upgrades, patches, failovers or platform scaling.

Note: Optimized value streams do not operate at maximum capacity. This makes sense when you think about it. For software development to be predictable and responsive, there must always be some spare capacity to tap into. When variability is low, it can be absorbed by a minimal amount of spare capacity.

Repeatability becomes the ultimate objective rather than maximizing productivity or throughput. Repeatability indicates that the process is mature, and that additional improvements can be easily validated. Only with such consistency can you deliver a predictable level of output over time.

Perhaps best of all, focusing on consistency and repeatability reduces your reliance on superhuman heroics and other short-term compromises that are counterproductive to continual improvement.

## Conclusion

Let's take a final look at the improvements we have made and see how much lower the overall cost and schedule risk are for our optimized value stream.

PROCESS IMPROVEMENT PRACTICE	COST AND RISK IMPACT
Developer self-service	Lower operational costs and no ticket delays
Small releases	Faster completion and feedback, easy to pivot
Combined teams	Simpler staffing and estimation, less finger-pointing and fewer handoffs
Defect visibility	Fewer defects and less rework
Reduce variability	Predictability builds confidence and enables ongoing improvements

FIGURE 19: Cost and risk impacts to the optimized value stream.

Developer productivity improvements are a frequent justification for adopting Pivotal Platform. Combine this highly automated platform with lean principles, and you'll find there's easily 2x (and likely 3x–4x) improvement potential depending on your current value stream.

Pivotal Platform—along with community tools, practices and culture—makes it possible for enterprises to improve software development quickly following a proven path. Figure 20 shows how productivity and time-to-market improves for each practice that is adopted.

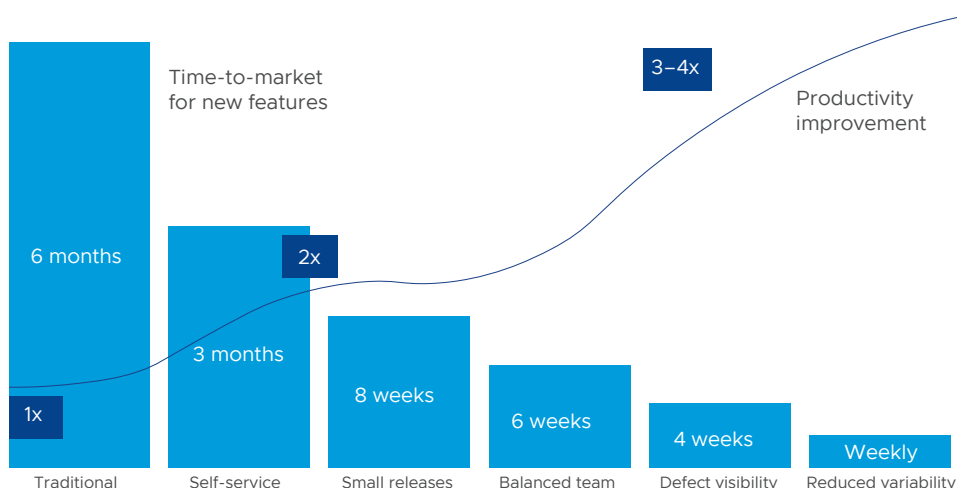


FIGURE 20: Pivotal Platform along with community tools, practices and culture.



Companies choose Pivotal Platform because they desire the strategic benefit of a predictable, high-velocity output of quality software.

Enterprises that use Pivotal Platform often have large backlogs of project work. The demand for new features and applications continues to grow. As the development value stream accelerates by three to four times, not only is more work being done, but developers benefit from less burnout and a more rewarding workplace.

Sure, you will save money as a result of using Pivotal Platform. But these cost savings are a secondary benefit compared to becoming better at software.

### Recommended reading

- Learn from your peers: [\*How Allstate boosted developer productivity by 350% with the cloud\*](#)
- Learn from your peers: [\*Citi: Accelerating Transformation\*](#)
- Learn from your peers: [\*CoreLogic: Transformation to Cloud Native Enterprise\*](#)
- Learn from your peers: [\*Liberty Mutual: Keynote at SpringOne Platform 2017\*](#)
- Learn from your peers: [\*Humana describes their digital transformation and the Cue app story\*](#)
- Learn from your peers: [\*Comcast Embraces DevOps for Faster, Smarter App Development\*](#)
- Best practices: [\*Six Myths of Product Development\*](#)
- White paper: [\*From Months to Hours: Accelerating Software Deployment with Cloud Foundry\*](#)
- White paper: [\*Running Microservices on Pivotal Cloud Foundry\*](#)
- Best practices: [\*Sustainable Software Development through Overlapping Pair Rotation\*](#)
- White paper: [\*Speed Thrills: How to Harness the Power of CI/CD for Your Development Team\*](#)
- White paper: [\*Multi-Site Pivotal Cloud Foundry: Deployment Topology Patterns and Practices\*](#)



VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 [vmware.com](http://vmware.com) Copyright © 2020 VMware, Inc.  
All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at [vmware.com/go/patents](http://vmware.com/go/patents). VMware is a registered trademark or trademark of VMware, Inc. and its subsidiaries in the United States and other jurisdictions.  
All other marks and names mentioned herein may be trademarks of their respective companies. Item No: Crossing the Value Stream Improving Development with VMware and Pivotal Platform 5/20