Continuous Delivery Sounds Great, but Will It Work Here?

**IT'S NOT MAGIC,
IT JUST REQUIRES
CONTINUOUS,
DAILY IMPROVEMENT
AT ALL LEVELS.**

JEZ HUMBLE

Continuous delivery is a set of principles, patterns, and practices designed to make deployments—whether of a large-scale distributed system, a complex production environment, an embedded system, or a mobile app—predictable, routine affairs that can be performed on demand at any time. This article introduces continuous delivery, presents both common objections and actual obstacles to implementing it, and describes how to overcome them using real-life examples.

WHAT IS CONTINUOUS DELIVERY?

The object of continuous delivery is to be able to get changes of all types—including new features, configuration changes, bug fixes, and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.

It is often assumed that deploying software more frequently means accepting lower levels of stability and reliability in systems. In fact, peer-reviewed research shows that this is not the case; high-performing teams consistently deliver services faster and more reliably than their low-performing competition. This is true even in highly regulated domains such as financial services and government.

This capability provides a competitive advantage for organizations that are willing to invest the effort to pursue it. It allows teams to deliver new features as they are ready, test working prototypes with real customers, and build and evolve more stable, resilient systems. Implementing continuous delivery has also been shown to reduce the ongoing costs of evolving products and services, improve their quality, and reduce team burnout.

While continuous *deployment*, the practice of continuously releasing every good build of your software, is mainly limited to cloud- or datacenter-hosted services, continuous *delivery*—the set of practices described here that enables continuous deployment—can be applied in any domain.

A number of principles and practices form the continuous delivery [canon](#).

COMMON OBJECTIONS TO CONTINUOUS DELIVERY

While people may know about continuous delivery, they often assume that “it won’t work here.” The most common objections cited are these:

- ➡ Continuous delivery is unsuitable when working in

highly regulated environments.

- ➔ Continuous delivery is only for websites.
- ➔ Continuous delivery practices can't be applied to legacy systems.
- ➔ Continuous delivery requires engineers with more experience and talent than are available here.

In this section these claims are examined and debunked, followed by a discussion of the *real* obstacles to implementing continuous delivery: inadequate architecture and a nongenerative culture.

Working in highly regulated environments

Objections to the use of continuous delivery in regulated environments are usually of two types: first, the unfounded perception that continuous delivery is somehow “riskier;” second, the fact that many regulations are written in a way that is not easy to harmonize with the practices of continuous delivery.

The idea that continuous delivery somehow increases risk is in direct contradiction to both the entire motivation of continuous delivery—to reduce the risk of releases—and the data. Four years of data show that high performers achieve high levels of both throughput and stability.² This is possible because the practices at the heart of continuous delivery—comprehensive configuration management, continuous testing, and continuous integration—allow the rapid discovery of defects in code, configuration problems in the environment, and issues with the deployment process.

In continuous delivery, automated deployments to production-like environments are performed frequently throughout the deployment pipeline, and comprehensive

automated tests are run against the builds thus deployed, resulting in a higher level of confidence that the software being built is both deployable and fit for purpose.

In contrast, many organizations employ risk-mitigation strategies that, in practice, amount to theater: endless spreadsheets, checklists, and meetings designed more to ensure that the process has been followed than to actually reduce the pain and risk of the deployment process. All this is not to say that more traditional risk-management processes can't work when done well. Rather, this shows that continuous delivery provides an alternative risk-management strategy that has been shown to be at least as effective, while also enabling more frequent releases.

The idea that continuous delivery is at odds with common regulatory regimes also deserves closer inspection. Much of the guidance concerning the implementation of controls designed to meet regulatory objectives assumes infrequent releases and a traditional phased software delivery life cycle complete with functional silos. It's typically also possible, however, to meet control objectives in a continuous paradigm. One example of this is Amazon, which in 2011 was releasing changes to production on average every 11.6 seconds, with up to 1,079 deployments in an hour (aggregated across Amazon's production environment).⁵ As a publicly traded company that handles a substantial number of credit card transactions, Amazon is subject to both the Sarbanes-Oxley Act regulating accounting practices and the PCI DSS (Payment Card Industry Data Security Standard).

While Amazon has chosen not to describe in detail how it was able to achieve compliance despite the dizzying

pace of changes, others have shared their experiences. For example, Etsy, an online handmade and vintage marketplace with more than \$1 billion in gross merchandise sales in 2013, described how it was able to meet the PCI DSS-mandated segregation of duties control while still practicing continuous deployment. Its “most important architectural decision was to decouple the cardholder data environment (CDE) from the rest of the system, limiting the scope of the PCI DSS regulations to one segregated area and preventing them from ‘leaking’ through to all their production systems. The systems that form the CDE are separated (and managed differently) from the rest of Etsy’s environments at the physical, network, source code, and logical infrastructure levels. Furthermore, the CDE is built and operated by a cross-functional team that is solely responsible for the CDE. Again, this limits the scope of the PCI DSS regulations to just this team.”⁴

It’s also important to note that segregation of duties “doesn’t prevent the cross-functional CDE team from working together in a single space. When members of the CDE team want to push a change, they create a ticket to be approved by the tech lead; otherwise, the code commit and deployment process is fully automated as with the main Etsy environment. There are no bottlenecks and delays, as the segregation of duties is kept local: a change is approved by a different person than the one doing it.”⁴

A well-designed platform-as-a-service (PaaS) can also provide significant benefits in a highly regulated environment. For example, in the U.S. federal government, the laws and policies related to launching and operating information systems run to more than 4,000 pages.

It typically takes months for an agency to prepare the documentation and perform the testing required to issue the ATO (Authorization to Operate) necessary for a new system to go live.

Much of this work is implementing, documenting, and testing the controls required by the federal government's risk-management framework (created and maintained by the National Institute of Standards and Technology). For a moderate-impact system, at least 325 controls must be implemented.

A team within the General Services Administration's 18F office, whose mission is to improve how the government serves the public through technology, had the idea of building a PaaS to enable many of these controls to be implemented at the platform and infrastructure layer. Cloud.gov is a PaaS built using mainly open-source components, including Cloud Foundry, on top of AWS (Amazon Web Services). Cloud.gov takes care of application deployment, service life cycle, traffic routing, logging, monitoring, and alerting, and it provides services such as databases and SSL (Secure Sockets Layer) endpoint termination. By deploying applications to cloud.gov, agencies can take care of 269 of the 325 controls required by a moderate-impact system, significantly reducing the compliance burden and the time it takes to receive an ATO.

The cloud.gov team practices continuous delivery, with all the relevant source code and configuration stored in git and changes deployed in a fully automated fashion through the concourse continuous integration tool.

Going beyond websites

Another objection to continuous delivery is that it can be applied only to websites. The principles and practices of continuous delivery, however, can be successfully applied to *any* domain in which a software system is expected to change substantially through its life cycle. Organizations have employed these principles building mobile apps and firmware.

CASE STUDY: CONTINUOUS DELIVERY WITH FIRMWARE AT HP

HP's LaserJet Firmware division builds the firmware that runs all its scanners, printers, and multifunction devices. The team consists of 400 people distributed across the U.S., Brazil, and India. In 2008, the division had a problem: it was moving too slowly. It had been on the critical path for all new product releases for years and was unable to deliver new features: "Marketing would come to us with a million ideas that would dazzle the customer, and we'd just tell them, 'Out of your list, pick the two things you'd like to get in the next 6–12 months.'" The division had tried spending, hiring, and outsourcing its way out of the problem but nothing had worked. It needed a fresh approach.

The target set by the HP LaserJet leadership was to improve developer productivity by a factor of 10 so as to get firmware off the critical path for product development and reduce costs. There were three high-level goals:

- ➡ create a single platform to support all devices.
- ➡ increase quality and reduce the amount of stabilization required prior to release.
- ➡ reduce the amount of time spent on planning.

A key element in achieving these goals was implementing continuous delivery, with a particular focus on:

- ➔ the practice of continuous integration.
- ➔ significant investment in test automation.
- ➔ creation of a hardware simulator so that tests could be run on a virtual platform.
- ➔ reproduction of test failures on developer workstations.

After three years of work, the HP LaserJet Firmware division changed the economics of the software delivery process by adopting continuous delivery, comprehensive test automation, an iterative and adaptive approach to program management, and a more agile planning process. The economic benefits were substantial:

- ➔ Overall development costs were reduced by approximately 40 percent.
- ➔ Programs under development increased by approximately 140 percent.
- ➔ Development costs per program went down 78 percent.
- ➔ Resources driving innovation increased eightfold.

The most important point to remember from this case study is that the enormous cost savings and improvements in productivity were possible only with a large and ongoing investment by the team in test automation and continuous integration. Even today, many people think that lean is a management-led activity and that it's about simply cutting costs. In reality, it requires investing to remove waste and reduce failure demand—it is a worker-led activity that can continuously drive down costs and improve quality and productivity. For more on this case study, see *Leading the Transformation: Applying Agile and DevOps Principles at Scale* by Gary Gruver and Tommy Mouser.

Handling legacy systems

Many organizations hold mission-critical data in systems designed decades ago, often referred to as legacy systems. The principles and practices of continuous delivery, however, can be applied effectively in the context of mainframe systems. Scott Buckley and John Kordyback describe how Suncorp, Australia's biggest insurance company, did exactly this.

CASE STUDY: CONTINUOUS DELIVERY WITH MAINFRAMES AT SUNCORP

Australia's Suncorp Group had ambitious plans to decommission its legacy general insurance policy systems, improve its core banking platform, and start an operational excellence program. "By decommissioning duplicate or dated systems, Suncorp aims to reduce operating costs and reinvest those savings in new digital channels," said Matt Pancino, then-CEO of Suncorp Business Systems.

Lean practices and continuous improvement are necessary strategies to deliver the simplification program. Suncorp is investing successfully in automated testing frameworks to support developing, configuring, maintaining, and upgrading systems quickly. These techniques are familiar to people using new technology platforms, especially in the digital space, but Suncorp is successfully applying agile and lean approaches to the "big iron" world of mainframe systems.

In its insurance business, Suncorp is combining large and complex insurance policy mainframe systems into a system to support common business processes across the organization and drive more insurance sales through

direct channels. Some of the key pieces were in place from the “building blocks” program, which provided a functional testing framework for the core mainframe policy system, agile delivery practices, and a common approach to system integration based on web services.

During the first year of the simplification program, testing was extended to support integration of the mainframe policy system with the new digital channels and pricing systems. Automated acceptance criteria were developed while different systems were in development. This greatly reduced the testing time for integrating the newer pricing and risk-assessment system with multiple policy types. Automated testing also supported management and verification of customer policies through different channels, such as online or call center.

Nightly regression testing of core functionality kept pace with development and supported both functional testing and system-to-system integration. As defects were found in end-to-end business scenarios, responsive resolutions were managed in hours or days, not the weeks typical for larger enterprise systems.

In the process, Suncorp, which oversees several different brands, has reduced 15 complex personal and life insurance systems to 2 and decommissioned 12 legacy systems. Technical upgrades are done once and rolled out across all brands. The company has a single code base for customer-facing websites for all its different brands and products. This enables faster response to customer needs and makes separate teams, each responsible for one website, redundant.

From a business point of view, the simpler system has

allowed 580 business processes to be redesigned and streamlined. Teams can now provide new or improved services according to demand, instead of improving each Suncorp brand in isolation. It has reduced the time to roll out new products and services, such as health coverage for its Apia brand customers or roadside assistance for its AAMI customers.

The investment in simplification and management of Suncorp's core systems means the company can increase its investment in all its touch points with customers. In both technology and business practices, Suncorp increased its pace of simplification, with most brands now using common infrastructure, services, and processes.

Suncorp's 2014 [annual report](#) notes that "simplification has enabled the Group to operate a more variable cost base, with the ability to scale resources and services according to market and business demand." Simplification activity was predicted to achieve savings of \$225 million in 2015 and \$265 million in 2016.

Developing people

Continuous delivery is complex and requires substantial process and technology investment. Some managers wonder if their people are up to the task. Typically, however, it's not the skill level of individual employees that is the obstacle to implementation but, rather, failures at the management and leadership level. This is illustrated in an anecdote told by Adrian Cockcroft, previously cloud architect at Netflix, who was often asked by Fortune 500 companies to present on Netflix's move to the cloud. A common question they had for him was, "Where do you

get Netflix's amazing employees from?" to which he would reply, "I get them from you!"

Continuous delivery is fundamentally about continuous improvement. For continuous improvement to be effective, process improvement must become part of everybody's daily work, which means that teams must be given the capacity, tools, and authority to do so. It's not unusual to hear managers say, "We'd love to introduce test automation, but we don't have time," or "This is the way we've always done it, and there's no good reason to change." The one common factor in all high-performing organizations is that they always strive to get better, and obstacles are treated as challenges to overcome, not reasons to stop trying.

Where workers are treated as fungible "resources" whose roles are to execute the tasks they are given as efficiently as possible, it's no wonder that they become frustrated and check out. Continuous improvement cannot succeed in this type of environment. In the modern gig economy, workers are defined by the skill sets they possess, and many organizations make little effort to invest in helping their workers develop new skills as the organization evolves and the work changes. Instead, these companies fire people when their skills are no longer necessary and hire new people whose skills fit the new needs, and then wonder why there is a "talent shortage."

These problems are related. An effective organization invests in developing people's skills to help solve new problems, not the problems that existed at the time they were hired. One way to help achieve this is to problem-solve to remove obstacles to improved performance,

learning new skills along the way: exactly what is required to effectively implement continuous delivery.

The barrier to achieving this is organizational culture, particularly the way leaders and managers behave.

OVERCOMING OBSTACLES TO CONTINUOUS DELIVERY

The principles and practices of continuous delivery can be implemented in all kinds of environments, from mainframes to firmware to those that are highly regulated, but it's certainly not easy. For example, Amazon took four years to re-architect its core platform to a service-oriented architecture that enabled continuous delivery.³ Typically, the biggest obstacles to this transformation are organizational culture and architecture.

Culture

What is culture? Edgar Schein, author of *The Corporate Culture Survival Guide*, defines it as “a pattern of shared tacit assumptions that was learned by a group as it solved its problems of external adaptation and internal integration, that has worked well enough to be considered valid and, therefore, to be taught to new members as the correct way to perceive, think, and feel in relation to those problems.”⁶

There are many models of culture, but one created by Ron Westrum,⁷ illustrated in figure 1, has been used to research the impact of culture on digital systems. Westrum's research emphasizes the importance of creating a culture where new ideas are welcomed, people from across the organization collaborate in the pursuit of common goals, people are trained to bring bad news so it can be acted upon, and failures and accidents are treated

FIGURE 1: WESTRUM'S THREE CULTURES MODEL

PATHOLOGICAL (POWER-ORIENTED)	BUREAUCRATIC (RULE-ORIENTED)	GENERATIVE (PERFORMANCE-ORIENTED)
low cooperation	modest cooperation	high cooperation
messengers shot	messengers neglected	messengers trained
responsibilities shirked	narrow responsibilities	risks are shared
bridging discouraged	bridging tolerated	bridging encouraged
failure leads to scapegoating	failure leads to justice	failure leads to enquiry
novelty crushed	novelty leads to problems	novelty implemented

as opportunities to learn how to improve rather than as witch-hunts.

The DevOps movement has always emphasized the primary importance of culture, with a particular focus on effective collaboration between development teams and IT operations teams. Research shows that a win-win relationship between development and ops is a significant predictor of IT performance. Practitioners in the DevOps movement have also used a number of tools to help organizations process information more effectively, such as [ChatOps](#), [blameless postmortems](#), and [comprehensive configuration management](#).

Indeed, the highest-performing companies don't wait for bad things to happen in order to learn how to improve; they create [controlled] accidents on a regular basis so as to learn more quickly than the competition. Netflix took this to a new level with the Simian Army, which is constantly breaking the Netflix infrastructure in order to continuously test the resilience of its systems.

Architecture

In the context of enterprise architecture, there are typically multiple attributes to be concerned about—for example, availability, security, performance, usability, and so forth. Continuous delivery introduces two new architectural attributes: testability and deployability.

In a *testable architecture*, software is designed such that developers can (in principle, at least) discover most defects by running automated tests on their workstations. They shouldn't have to depend on complex, integrated environments to do most acceptance and regression testing.

In a *deployable architecture*, deployments of a particular product or service can be performed independently and in a fully automated fashion, without the need for significant levels of orchestration. Deployable systems can typically be upgraded or reconfigured with zero or minimal downtime.

Where testability and deployability are not prioritized, much testing requires the use of complex, integrated environments, and deployments are “big bang” events that require many services be released at the same time because of complex interdependencies. These big bang deployments require many teams to work together in a carefully orchestrated fashion with many hand-offs and dependencies among hundreds or thousands of tasks. Such deployments typically take many hours or even days, and require scheduling significant downtime.

Designing for testability and deployability starts with ensuring that products and services are composed

of loosely coupled, well-encapsulated components or modules.

A well-designed modular architecture can be defined as one in which it is possible to test or deploy a single component or service on its own, with any dependencies replaced by a suitable test double, which could be in the form of a virtual machine, stub, or mock. Each component or service should be deployable in a fully automated fashion on developer workstations, in test environments, or in production. In a well-designed architecture, it is possible to achieve a high level of confidence that the component is operating properly when deployed in this fashion.

To aid the independent deployment of components, creating versioned APIs that have backwards compatibility is worth the investment. This adds complexity to systems, but the flexibility gained in terms of ease of deployment will pay for it many times over.

Any true service-oriented architecture should have these properties—but, unfortunately, many do not. The microservices movement, however, has made explicit priorities of these architectural properties.

Of course, many organizations are living in a world where services are distinctly hard to test and deploy. Rather than re-architecting everything, we recommend an iterative approach to improving the design of an enterprise system, sometimes known as evolutionary architecture.¹ In the evolutionary architecture paradigm, we accept that successful products and services will require re-architecting during their life cycles because of the changing requirements placed on them.

One pattern that is particularly valuable in this context is the strangler application, shown in figure 2. In this pattern, a monolithic architecture is iteratively replaced with a more componentized one by ensuring that new work is done following the principles of a service-oriented architecture, while accepting that the new architecture may well delegate tasks to the system it is replacing. Over time, more and more functionality will be performed in the new architecture, and the old system being replaced is “strangled.”

FIGURE 2: THE STRANGLER APPLICATION

Related articles

➡ The Hidden Dividends of Microservices
Microservices aren't for every company,
and the journey isn't easy.

Tom Killalea

<https://queue.acm.org/detail.cfm?id=2956643>

➡ A Conversation with Tim Marsland
Taking software delivery to a new level
<https://queue.acm.org/detail.cfm?id=1066063>

➡ The Responsive Enterprise: Embracing
the Hacker Way

Soon every company will be a software
company.

Erik Meijer and Vikram Kapoor

<https://queue.acm.org/detail.cfm?id=2685692>

CONCLUSION

Continuous delivery is about reducing the risk and transaction cost of taking changes from version control to production. Achieving this goal means implementing a series of patterns and practices that enable developers to create fast feedback loops and work in small batches. This, in turn, increases the quality of products, allows developers to react more rapidly to incidents and changing requirements and, in turn, build more stable and higher-quality products and services at lower costs.

If this sounds too good to be true, bear in mind: continuous delivery is not magic. It's about continuous, daily improvement at all levels of the organization—the constant discipline of pursuing higher performance. As presented in this article, however, these ideas can be implemented in any domain; this requires thoroughgoing, disciplined, and ongoing work at all levels of the organization. Particularly hard, though essential, are the cultural and architectural changes required.

Nevertheless, as organizations of all types and sizes from fintech startups to the U.S. government implement these ideas, they have transitioned from being exceptional

to standard. If you haven't yet started on this path, don't worry—it can be achieved, and the time to begin is now.

References

1. Ford, N., Parsons, R., Kua, P. 2017. *Building Evolutionary Architectures: Support Constant Change*. O'Reilly Media; (<http://evolutionaryarchitecture.com>).
2. Forsgren, N., et al. 2014-2017. *State of DevOps Report*. Puppet and DevOps Research and Assessment LLC; (<https://devops-research.com/research.html>).
3. Gray, J. 2006. A conversation with Werner Vogels. *acmqueue* 4(4); <http://queue.acm.org/detail.cfm?id=1142065>.
4. Humble, J., O'Reilly, B., Molesky, J. 2014. *Lean Enterprise: How High Performance Organizations Innovate at Scale*. O'Reilly Media. 242-243.
5. Jenkins, J. 2011. Velocity culture (the unmet challenge in ops). O'Reilly Velocity Conference; <http://assets.en.oreilly.com/1/event/60/Velocity%20Culture%20Presentation.pdf>.
6. Schein, E. 1999. *The Corporate Culture Survival Guide*. Jossey-Bass.
7. Westrum, R. 2004. A typology of organizational structures. *BMJ Quality and Safety* 13(2); http://qualitysafety.bmj.com/content/13/suppl_2/ii22.

Jez Humble is coauthor of *The DevOps Handbook* (<https://www.indiebound.org/book/9781942788003>) (<http://www.amazon.com/dp/1942788002?tag=contindelve-20>), *Lean Enterprise*, (<https://www.indiebound.org/book/9781449368425>) (<http://www.amazon.com/dp/1449368425?tag=contindelve-20>),

and the Jolt Award-winning Continuous Delivery (<https://www.indiebound.org/book/9780321601919>) (<http://www.amazon.com/dp/0321601912?tag=contindelve-20>). He has spent his career tinkering with code, infrastructure, and product development in companies of varying sizes across three continents, most recently working for the U.S. government in the 18f office (<https://18f.gsa.gov/>). He is currently researching how to build high-performing teams at his startup, DevOps Research and Assessment LLC (<https://devops-research.com/>), and teaching at UC Berkeley (<http://www.ischool.berkeley.edu/people/faculty/jezhumble>).

Copyright © 2017 held by owner/author. Publication rights licensed to ACM.