A Forrester Consulting Thought Leadership Paper Commissioned By Thoughtworks

Continuous Delivery: A Maturity Assessment Model

Building Competitive Advantage With Software Through A Continuous Delivery Process

March 2013

FORRESTER®

Headquarters | Forrester Research, Inc.
60 Acorn Park Drive, Cambridge, MA 02140 USA
Tel: +1 617.613.6000 | www.forrester.com

Table Of Contents

About Forrester Consulting
Forrester Consulting provides independent and objective research-based consulting to help leaders succeed in their organizations. Ranging in scope from a short strategy session to custom projects, Forrester's Consulting services connect you directly with research analysts who apply expert insight to your specific business challenges. For more information, visit www.forrester.com/consulting.

Executive Summary

"Software is eating the world." (Marc Andreessen)[1]

It seems like it was just a few years ago that the business world was divided into a small number of companies that lived and died on the quality of the software they shipped and a vast majority that could get by just fine with an "IT doesn't matter" mindset.[2] But that's changing as software-driven firms compete and displace traditional incumbents. We all know the stories: Netflix versus Blockbuster, Amazon.com versus Borders, or iTunes versus the entire recording industry. These cases are just the beginning of the larger shift, where companies that are able to innovate quickly with software will outcompete traditional market leaders. And this transformational process will move from industry sector to industry sector. It will also create market collisions, as Jeff Immelt, General Electric's (GE) chief executive officer (CEO), noted at the 2012 Minds and Machines conference.

"In an industrial company, avoid software at your own peril . . . a software company could disintermediate GE someday, and we're better off being paranoid about that." (Jeff Immelt, CEO, GE)[3]

The software you deploy, and especially the custom software you create, will increasingly be part of your competitive edge. But how do you make sure your company wins with its custom development efforts? One way is to increase your rate of software innovation. You can also reduce the cost of the software you develop and decrease the time it takes to get it right. But to take these steps, business leaders and IT leaders alike need to understand the connection between innovation and software delivery. In the fall of 2011, Thoughtworks commissioned Forrester Consulting to evaluate how business executives and IT leaders evaluated their challenges and whether their current software delivery processes were sufficient to meet the relentless demand for innovation. Our hypothesis was that companies that prioritize innovation and have organized their software development teams to deliver it quickly are able to deliver software faster and innovate quicker. This process of continuous delivery results in a closer business-IT relationship and generates more value by increasing the velocity of the feedback loop between customers and companies that serve them. As products get to market faster, the risk of investing heavily to build something customers don't want or won't use goes down. As organizations form hypotheses about customer behavior and test them, they gather actionable feedback and hard data that determine what new features and algorithms work best.

In conducting in-depth surveys with 325 business and IT professionals, Forrester found that these companies understand the growing importance of innovation, but most are not able to deliver new custom software solutions as fast as business leaders need them. A major reason for this is that most of the companies we surveyed have a low level of maturity when it comes to continuous delivery. As a result, they are not able to use their software development capacity to drive their business, and they are not able to release new applications to support their businesses as fast as they would like.

Key Findings

Forrester's study yielded four key findings:

- **Companies are looking to prioritize innovation through developing software services.** Most business and IT executives are currently focused on "keeping the lights on," but there's a strong desire to shift their future focus to more innovative services, delivered through custom software development.

- **Software development providers can't deliver new services at the rate business leaders want.** Most business leaders would like software development providers to deliver new services in fewer than six months, but that's not the performance they're currently getting from their software development providers.
- **Corporate culture and development process immaturity impede communication and slow service delivery.** A lack of collaboration leads to many IT shops being viewed as "order takers" by business execs, even though many IT leaders don't see it that way. And many IT shops are still working on daily execution of basic continuous delivery practices, which further slows service delivery.
- **Few IT organizations regularly perform advanced continuous delivery practices.** A small minority of IT leaders indicate that their software development teams are regularly executing mature continuous delivery practices like A/B testing, automated deployments, and test-driven development. This slows the rate at which these teams can deliver new releases of existing services and limits the rate at which developers and product owners can get customer feedback on the value of their work.

Innovation Is On The Software Development Agenda

What do business leaders want from their software development peers? Are they satisfied with the results that they are getting today? If the results from our survey are any guide, there's a lot of work to do and not much time to get it done.

When we asked business and IT leaders about the drivers behind their software development investments, we found that there is a mismatch between where most of them are and where they'd like to be:

- **Keeping things running occupies most today.** Both business executives and IT leader put keeping things running at the top of their software development priority list today. Forty-nine percent of business leaders view "keeping things running" as the most important priority or a top priority for their software development efforts (see Figure 1). That matches what IT leaders think — 38% rank keeping things running as their top priority.
- **Incremental innovation is what leaders want to spend time on.** Both business leaders and IT leaders push incremental innovation with software development to the top of their rankings when they look at a two-year planning horizon. It's a top driver or most important for 55% of business leaders and the top ranked priority for 38% of IT leaders (see Figure 2).
- **Business leaders focus on disruption, but IT leaders are slower to react.** As business leaders look forward, increasing numbers of them want to use their software development investments to disrupt the markets they compete in. Forty percent view market disruption as a top priority or most important priority two years from now. But IT leaders aren't quite on the same page here. Only 13% view market disruption as a top software development priority in two years.

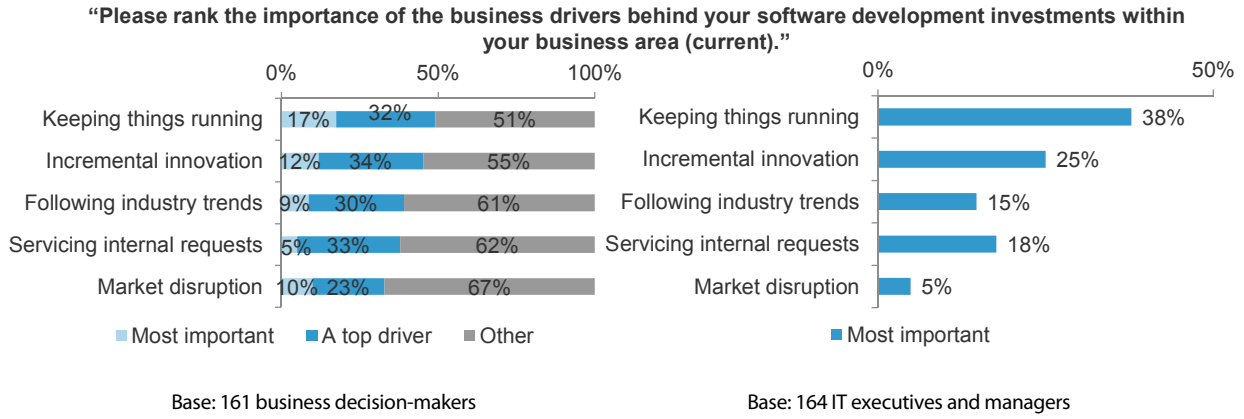
As the focus on software development shifts toward innovation, it's a fair question to ask: Are development teams ready to respond? Do they have enough capacity to innovate while responding to existing enhancement requests and keeping existing systems running? For most shops, the answer is in doubt — they aren't fast enough, and their processes aren't mature enough to support a rapid, sustained exercise in innovation. Our survey results tell the tale:

- **Business leaders want software delivered in six months.** When we asked business leaders how quickly they wanted strategic IT services or software products delivered, the majority responded that they wanted to go from concept to production in fewer than 12 months. In fact, just more than half (51%) want software development teams to introduce strategic capabilities in fewer than six months (see Figure 3).

- **IT leaders can't deliver software as fast as the business wants it.** When we asked business leaders about their actual expected software delivery speeds, only 30% answered that software could be delivered as part of an innovative new idea within the six month time frame that they want (see Figure 4). What's worse, more than four in 10 said that it would take a year or more for their IT teams to develop new software to support an innovative new idea. The result? This speed gap keeps businesses from using their software development capability to disrupt markets by introducing new products or features first. Instead, these businesses spend their time following industry trends — reacting to the market instead of shaping the playing field.

Figure 1

Today Business And IT Leaders Are Focused On Keeping Things Running



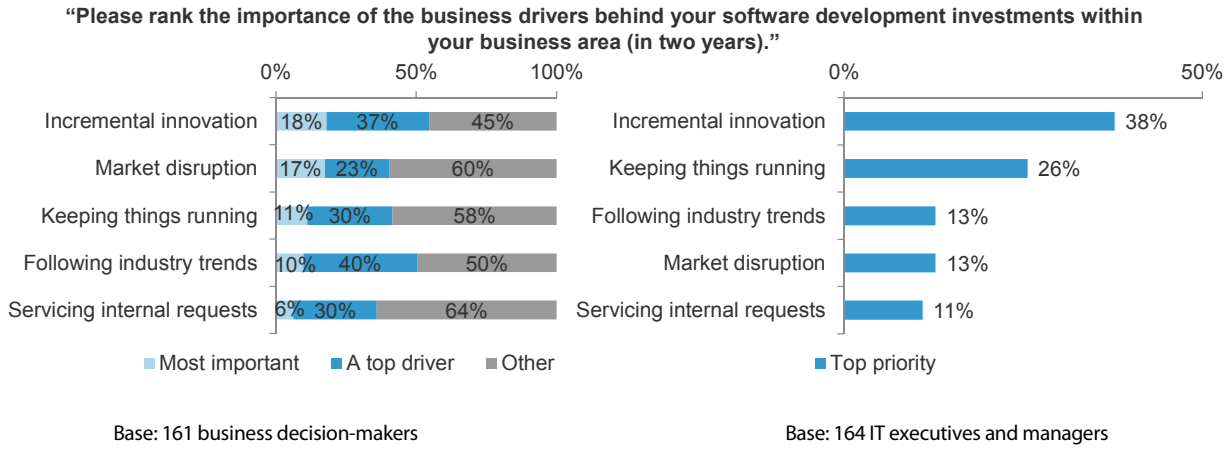
Base: 161 business decision-makers

Base: 164 IT executives and managers

Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

Figure 2

The Focus On Software Innovation Will Shift To Innovation And Market Disruption



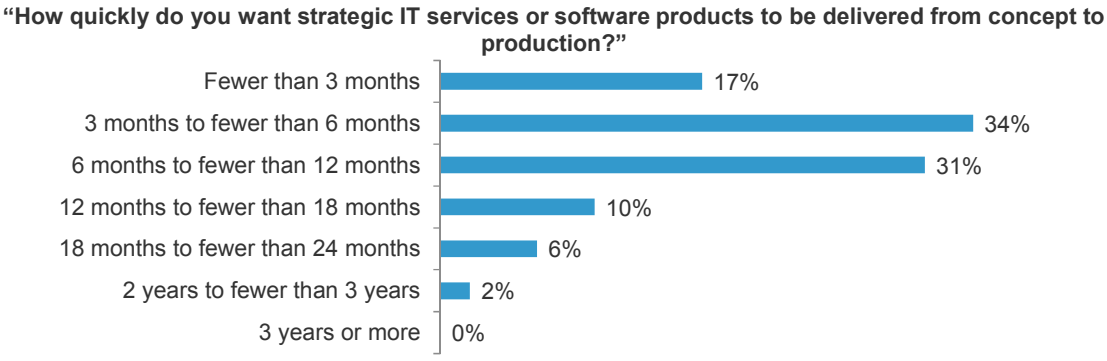
Base: 161 business decision-makers

Base: 164 IT executives and managers

Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

Figure 3

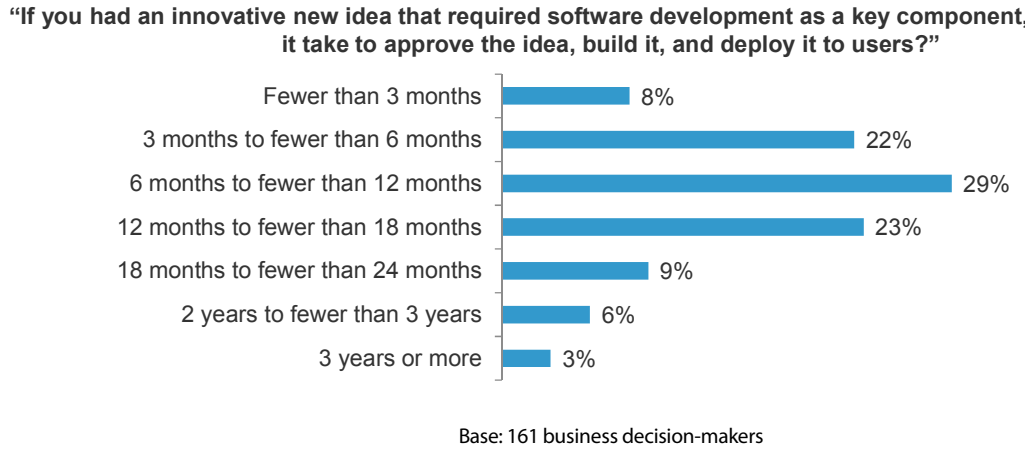
Business Leaders Expect New Services Within Six Months



Base: 161 business decision-makers

Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

Figure 4
Service Delivery Today Does Not Meet Expectations



Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

Culture And Process Impede Communication And Slow Service Delivery

Business leaders would like to shift their custom software investments to focus on innovation and market disruption. They would also like to deliver those software services faster than they currently do. This creates a challenge, because in many businesses, organizational culture and development process maturity make it difficult to speed up service delivery.

One organizational challenge that frequently gets in the way of faster service delivery is who controls the process. In our survey of business and IT leaders, we see distinct differences in their opinions about the relationship between those who develop software services and those who use them:

- **Many business leaders still view IT as an order-taking organization.** When we asked business leaders what level of influence their software development provider has when it comes to deciding which business services or products get delivered, 42% responded that they view their software development providers primarily as order takers — they use them as-a-service (see Figure 5). Slightly more business leaders (43%) viewed their software development provider as a partner, while 14% said that their IT or engineering group drives technology innovation today.

- **IT leaders see a different role, even in firms where software development is not strategic.** The IT and engineering leaders in our survey see their service delivery role differently. In firms that consider custom-developed software to be strategic to the business, only 7% of IT or engineering leaders see themselves as order takers, while 59% see themselves as partners (see Figure 6). Even in IT organizations where custom software development is not strategic to the business, only 25% of IT leaders see their organization acting as a service provider, while 57% see themselves as a partner.

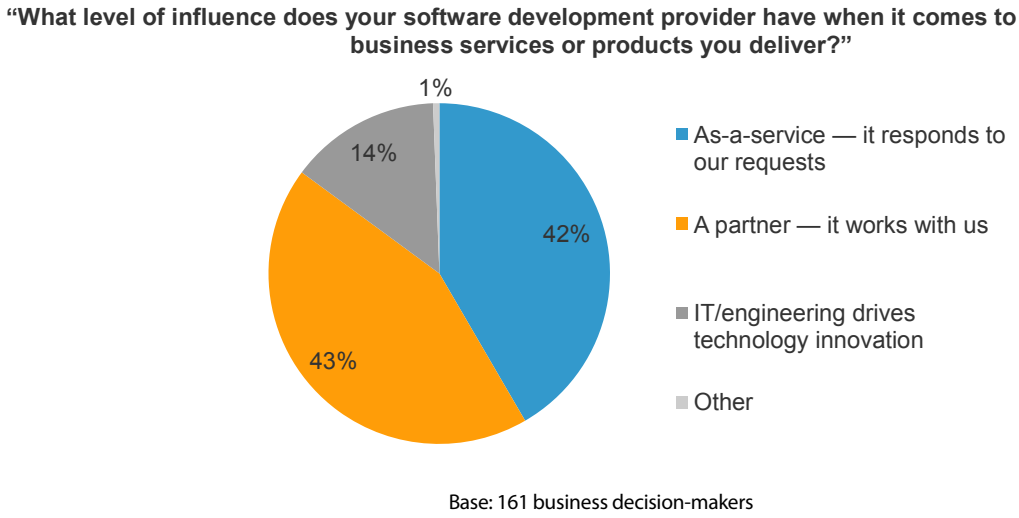
Closing this perception gap is an important step for IT leaders, because it creates a communication gap when it comes to setting expectations for how quickly software development teams should (and can) deliver innovative new software services or products (see Figure 7). When business leaders view their software delivery provider as a service, they tend to expect accelerated delivery — more than 50% expect delivery from concept to production in fewer than six months, and almost 90% expect delivery from concept to production within 12 months. When IT is in control of technology innovation, business expectations are bipolar. Some executives expect new service delivery in a similar time frame as when they are in control, but a small subset actually expects service delivery time to increase substantially, going out 18 months or more. We view this group as a sign that troubled software projects are still common enough that business leaders are conditioned to expect long Big Bang projects that are slow to deliver and that these leaders have set their expectations accordingly. When IT shops are set to function as an order-taking service provider, it's hard to create a constant stream of actionable customer feedback: Business sponsors or sales reps own the customer relationship and filter requests or soften the impact of poor software. Even worse, developers in IT may have goals based as much on cost control as on customer satisfaction and business growth.

In between the IT-as-a-service and IT-as-a-leader models is a business technology (BT) model, one where business leaders and IT leaders act in partnership. When we look at software delivery expectations in the partnership models, the vast majority of business leaders expect services to be delivered from conception to production in three months to fewer than 12 months. The distribution curve for the BT partner model closely resembles the desired distribution curve in Figure 1. Our conclusion: A BT model where the business and IT/engineering work in partnership to deliver new services from conception through production stands the best chance of matching the needs of business leaders with the reality of service delivery capacity as it exists today.

While a BT partnership model is best suited for matching service delivery capability to expectations, shops that engage in continuous delivery go beyond joint decision-making. They take an economic approach to managing their product portfolio of services — one that's based on data gathered from direct customer engagement. That's not the case for most organizations today: Almost half (47%) use a committee-based decision-making approach, 24% use financial modeling, and only 9% use a product portfolio approach. While committees are effective at getting buy-in to proceed from one phase of service definition to another, it's an unfortunate truth that they're also too often used to diffuse individual responsibility for making incorrect product decisions due to a lack of hard data on what customers want and how they react.

Figure 5

What Business Leaders Think About The Business-IT Relationship

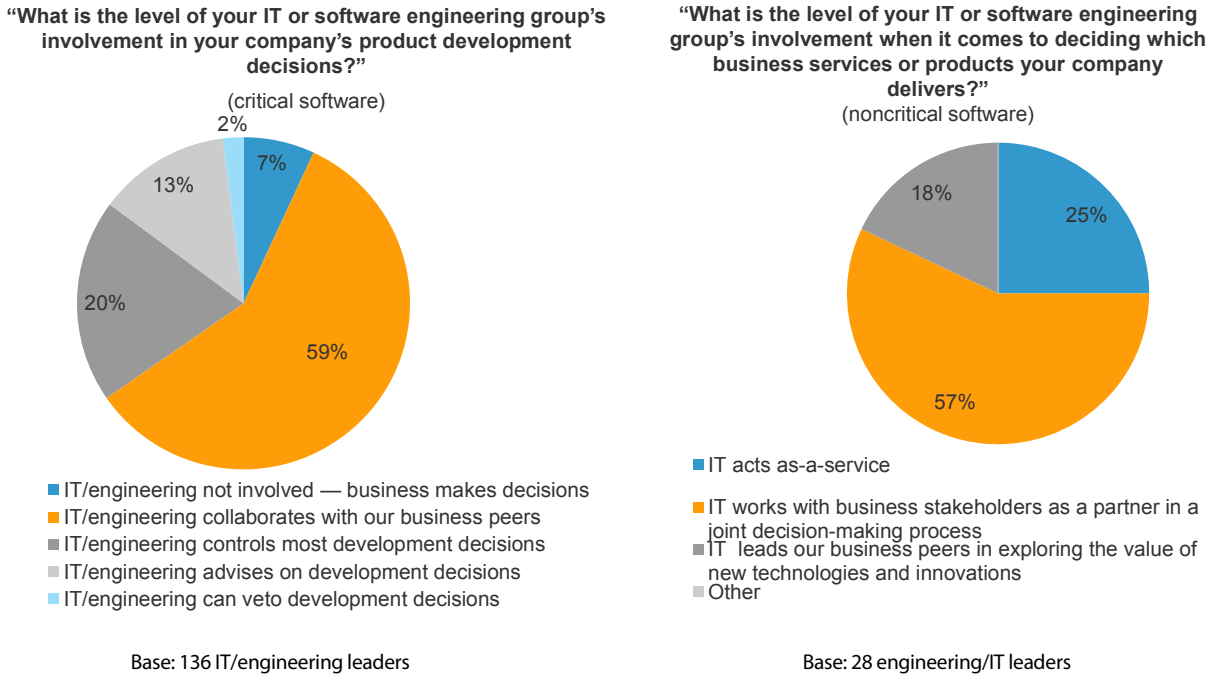


Base: 161 business decision-makers

Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

Figure 6

What IT And Engineering Leaders Think About The Business-IT Relationship

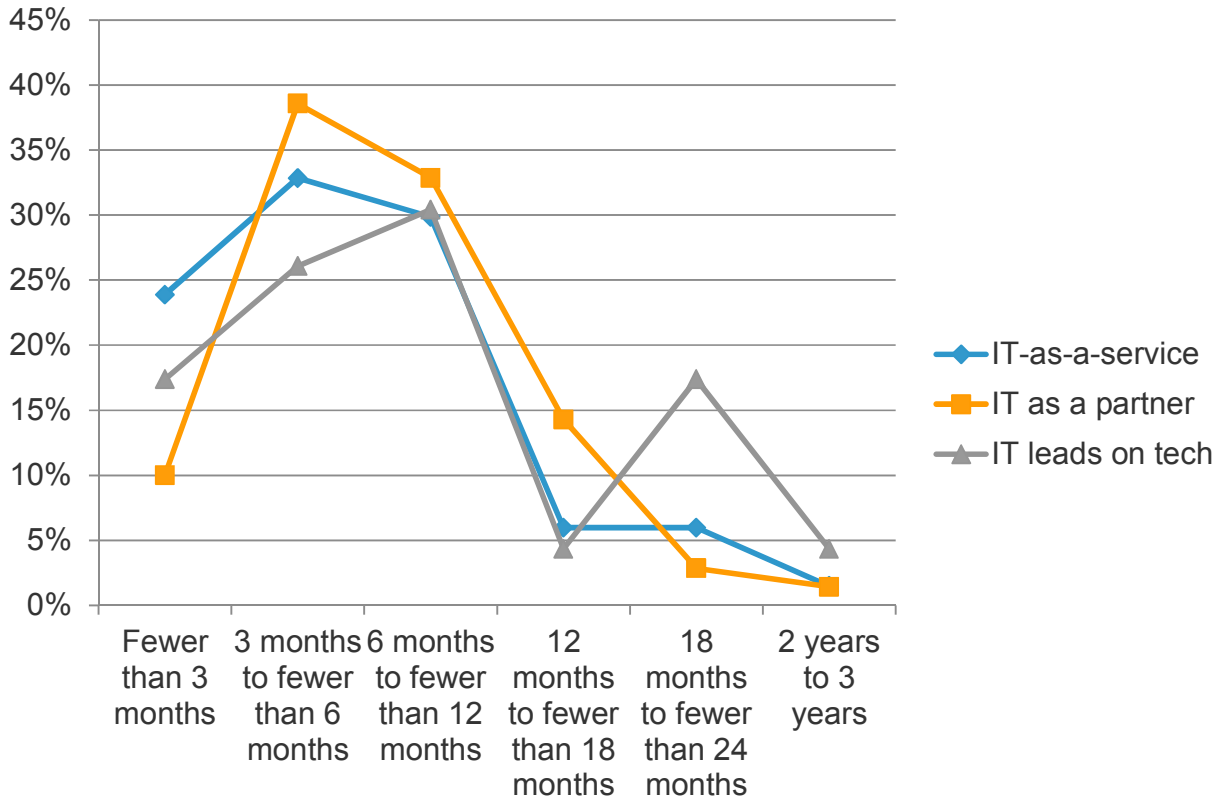


Base: 136 IT/engineering leaders

Base: 28 engineering/IT leaders

Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

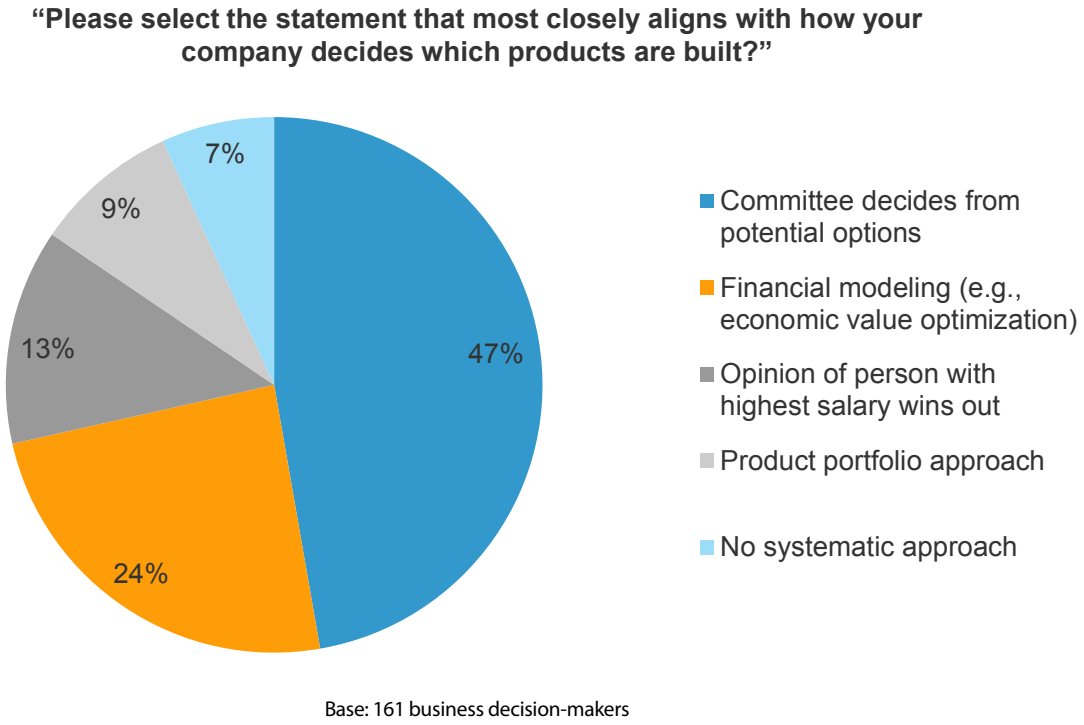
Figure 7
The Business View Of IT Affects Expectations Of Service Delivery Speed



Base: 161 business decision-makers

Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

Figure 8
Decision By Committee Is How Most Firms Develop Software Services



Base: 161 business decision-makers

Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

Development process immaturity also impedes the speed with which software development teams develop and deploy new business services. At Forrester, we've written at length about software development practices that speed up the rate at which software development teams work. Agile and lean development practices, continuous integration, and dev-ops techniques have an extensive record of reducing the amount of time it takes to release new software. These techniques span the software development life cycle — from idea conception to release to production. Over the past few years, we've seen many forward-looking development shops adopt these principles and those of the lean startup to enable a process of *continuous delivery*. Continuous delivery reduces the time it takes to release software and also enables early actionable customer feedback to minimize waste that many software development teams encounter because they spend time building the wrong features. And yet when we asked the IT leaders in our survey about their use of some of these practices, we see that there is still much adoption work to be done (see Figure 9):

- **A majority of development teams have mastered the basics.** We found that many software development leaders in our survey have mastered the basics of delivering software faster. As an example, we're happy to see that more than 60% are using source code management (SCM) tools to manage their software on a daily or weekly basis (although it's troubling to see that almost one in five teams still lacks SCM maturity). It's also good to see that six in 10 shops regularly have product owners assigned to software services. Product owners are a linchpin of business-IT collaboration because they facilitate the collaboration process between business leaders and IT leaders. It's also good to see that 57% practice continuous integration (CI) on a daily or weekly basis. While

purists might argue that CI is only really followed when developers are working off mainline and code is integrated on a daily basis, we've seen development teams find success even when performing integration builds every other day or several times a week — provided they actually fix the integration build when it is broken. Frequent builds are a core building block for more advanced continuous delivery practices.

- **A scarce majority of development teams are speeding up release management.** As continuous delivery practices move from day-to-day development to rapid deployments, we find that the maturity of software development teams in our survey begins to erode. Barely half of IT leaders have teams that employ automated testing to ensure release quality on a weekly (or more frequent basis). Without automated test cases, it really doesn't matter how often a team builds a software service; its release is gated by the team's capacity to execute a manual testing regimen. Likewise, we also see that a bare majority of teams are regularly staffed in a cross-functional manner — as a result, handoffs from development to test to production operations create stage gate barriers that a release must pass through as part of a deployment. Finally, a bare majority of teams regularly deploy their services to a virtual image or environment. This makes it difficult to quickly stage a full production deployment, because software is tied more tightly to bare-metal installations on dedicated hardware.[4] Lack of virtualization also makes it more painful for developers to perform system, performance, and security testing in integrated environments. Instead of simply moving images from one environment to another, developers or system administrators need to spend time and effort recreating releases — this is a place where differences in environments or deployment processes can cause needless waste.

- **A minority of software development teams employ extended continuous delivery practices.** Even when a software development team attacks the time spent deploying applications by automating testing, it merely shifts the deployment bottleneck downstream to the actual deployment itself. This is where the number of organizations ready to practice continuous deployment really starts to drop off. In our survey, only 44% of IT leaders said that they consistently automated the deployment of strategic software services. Even fewer (41%) give their development teams regular access to self-provisioned resources, like a public or private cloud. Many of the advanced development shops we've spoken with point out that an elastic self-provisioned deployment environment is a critical component of modern service architecture, as it enables advanced continuous deployment practices like blue-green deployment (or red-black deployment if you're Netflix) and testing in production.[5] Slightly more IT leaders (46%) say that their development teams regularly practice test-driven development (TDD) or require teams to pass a threshold of automated test cases before pushing new code into production. TDD further expands the focus of continuous deployment teams by recasting design and functional requirements specification as a measurable set of outcomes, which developers can proactively measure their code against.

- **Adoption of advanced continuous development practices is still a work in process.** The IT leaders we surveyed were even less likely to employ advanced continuous deployment practices like infrastructure as code and automated deployments that included database provisioning and migration. As a result, there are still substantial parts of the services they build that are deployed manually, slowing the rate at which services can be updated after an initial version is deployed. We also note that few organizations (fewer than 25%) automate the feedback gathering process using analytics or introduce new features to watch how (and whether) users adopt them (see Figure 11).

And it's not just immature development processes that slow down the rate of service delivery. When we asked development leaders about specific items that slowed their software releases down, a number of surprising items filtered up to the top of the list (see Figure 10).

- **A lack of slack time for continuous improvement.** Almost half of the IT leaders we surveyed pointed out that even if they wanted to improve their software service deployment processes, they didn't have enough slack time in their schedules to implement a continuous improvement process. We've recently seen some development teams build this type of slack into their Agile sprints, by assigning one sprint in four or five to reducing the level of technical debt in their software, including the systems used to automate its delivery.[6] Other teams will designate cooling-off periods at the end of a series of functional improvements, where systems are upgraded before a renewed set of sprints (i.e., like a factory where the assembly line get retooled between production years).

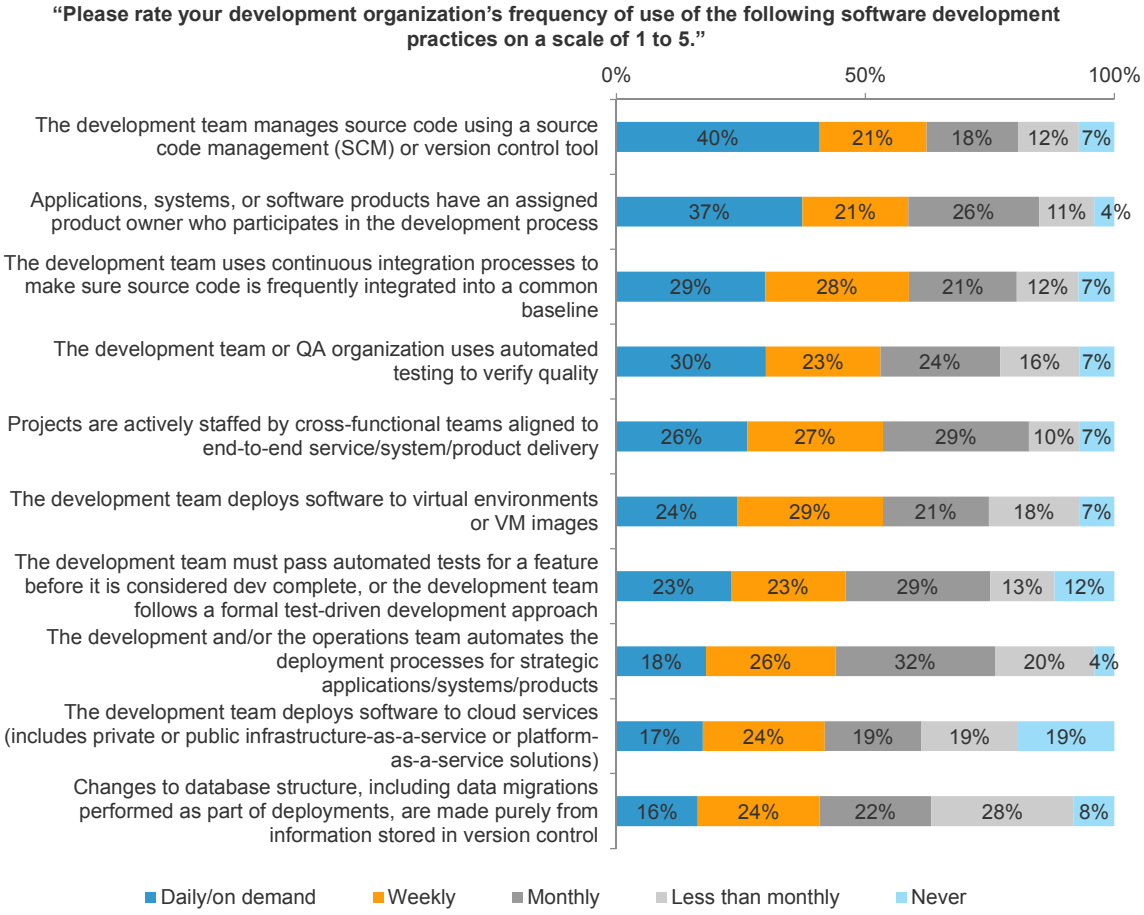
- **Restricted release windows.** It's probably not a real surprise that teams struggle when a system must be taken offline in order to upgrade it or release new capability. And it's a real barrier for 44% of the IT leaders we surveyed. It's also increasingly unnecessary where modern application architectures are concerned. As developers evolve systems to run on scale-out hardware and move toward Internet-centric application interfaces, it's easier than ever to use a load balancer in front of multiple services and then redirect traffic to different machines with different versions of a service. In a sense, teams practicing continuous delivery don't just test in production; they also update in production.

- **Unforeseen issues that cause rollbacks.** Deploying software is akin to playing sports or a musical instrument. The more you practice, the better the actual performance. Deployment systems act like muscles in the sense that they develop memory through repetition. When there's a lack of repetition, unforeseen issues tend to crop up, and that's the case for 37% of the IT leaders we surveyed, who cite unforeseen rollback as a significant block to faster releases. Regular releases also tend to be smaller, and regular releases also tend to be more highly automated, as the development works to eliminate repetitive manual tasks from their day-to-day jobs. Repetition and automation (practice if you will) put continuous deployment teams on the road to perfection.

Businesses that want to improve the speed of innovation through software development would be well advised to carefully examine the state of their own corporate and development culture and perform a self-assessment of their readiness to move toward a continuous deployment model. In particular, it's important to assess the maturity of the IT or engineering organization that's responsible for developing software services and what can be done to improve that maturity by introducing new processes, skills, and systems to improve service deployment capability.

Figure 9

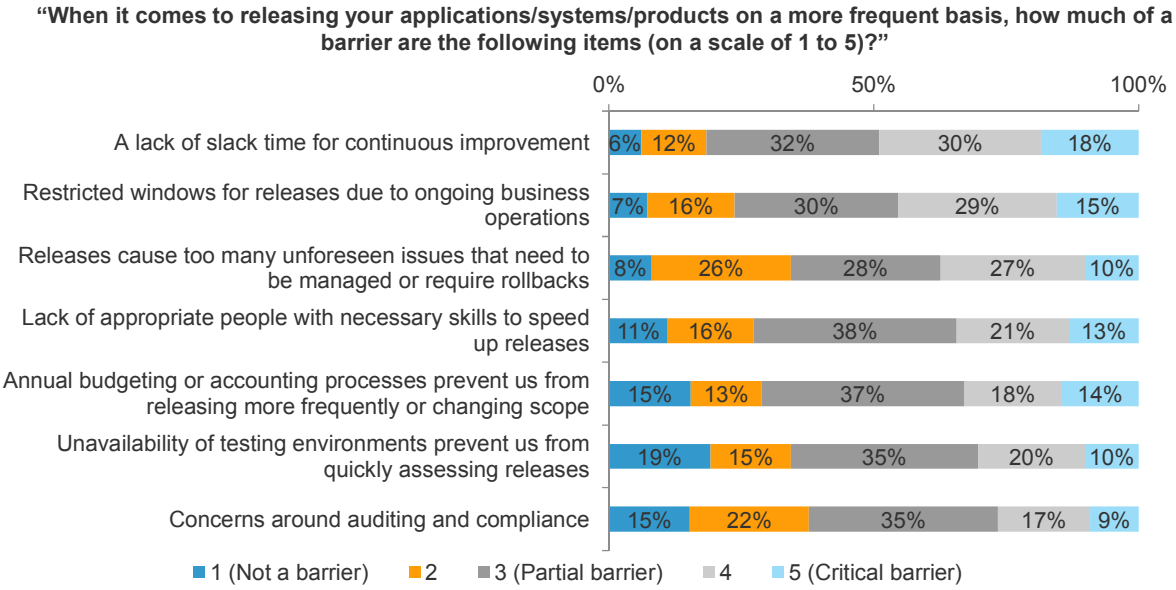
Many Teams Are Still Adopting Continuous Delivery Practices



Base: 164 IT/engineering leaders

Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

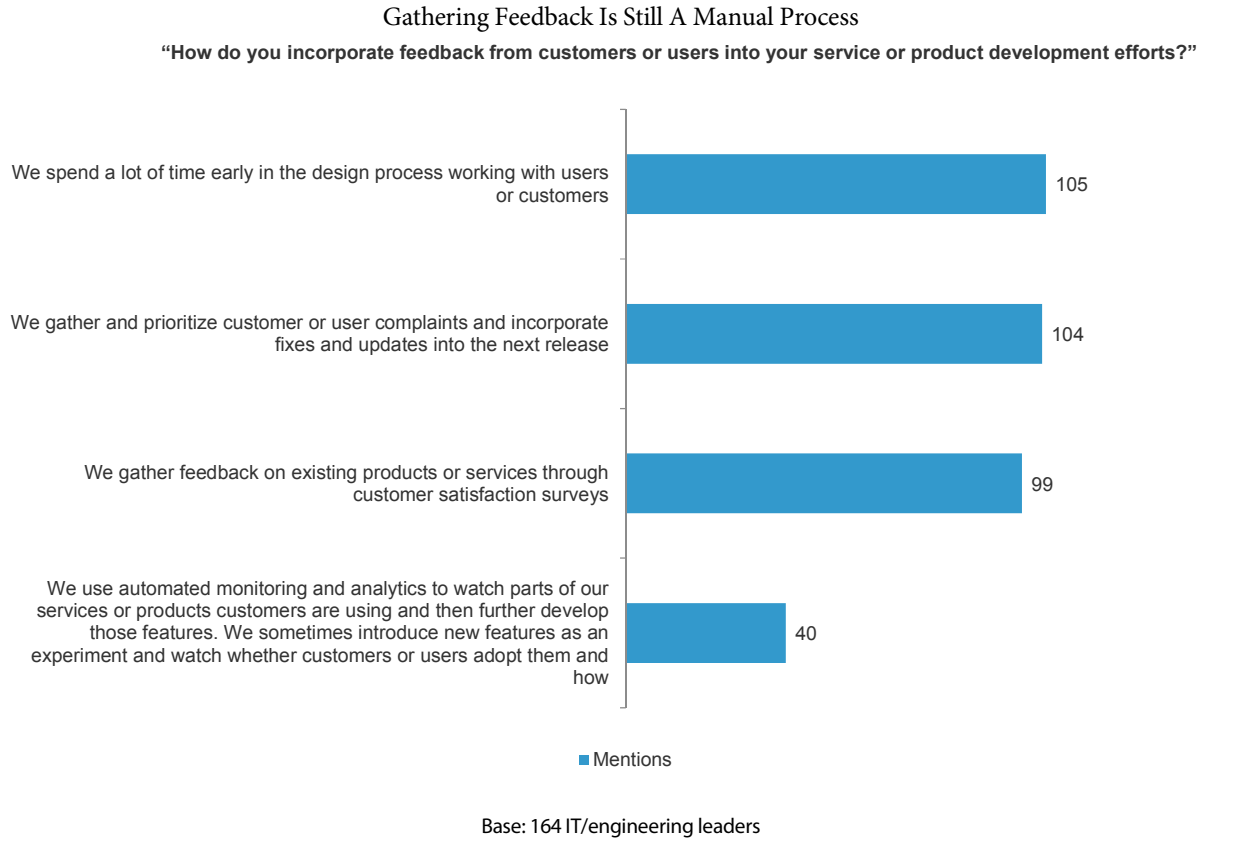
Figure 10
Barriers To Faster Service Deployment



Base: 164 IT/engineering leaders

Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

Figure 11



Source: A commissioned study conducted by Forrester Consulting on behalf of Thoughtworks, September 2012

Assessing Continuous Delivery Capability: A Maturity Model

So what's the best way to assess your software development shop's continuous delivery capability? We've put together a maturity model to help you with your efforts. The goal of this model is to help development leaders understand the existing capabilities their development shop has and determine current gaps in maturity so that it's clear which development capabilities to focus on improving to support faster development and delivery of strategic business services. We'd recommend using this matrix by focusing on introducing specific practices, one or two at a time, with the goal of achieving measurable improvements within an individual area over a few months. By slowly introducing the capabilities suggested in each stage of maturity, you'll be able to create sustained results. You should also focus your effort on individual teams and work with them to advance on an individual basis: Don't expect to take the entire development organization from level 2 to level 5 in a matter of months. Remember — you're dealing with culture change as well as changing processes and tools.

Table 1
Continuous Delivery Maturity Model

Level	Delivery focus	Characteristics	Result
5: optimizing	Hypothesis-driven delivery	**Teams focus on optimizing cycle time to learn from customers.** • All new requirements describe how the value of the feature will be measured. • Product teams are responsible for implementing metrics to gather this data through techniques such as A/B testing. • Systems are architected with continuous deployment in mind, supporting patterns such as dark launching to decouple deployment from release. • Database changes are decoupled from application deployments.	Continuous deployment capability ***enables*** business innovation/ experimentation
4: quantitatively managed	Release on demand	**Delivery teams prioritize keeping code trunk deployable over doing new work**. • Deployment pipeline automatically rejects bad changes from version control. • Cross-functional end-to-end product-centric teams manage products throughout life cycle. • Comprehensive automated test suites are created through TDD/ATDD and maintained by developers and testers working together. • Teams monitor and manage work in process and deliver work in small batches.	Release on demand: Software is always in a releasable state. Release time box is well defined and equal to, or less than, business need.

3: defined	Regular releases over a defined period with interim milestone builds	**Teams build quality into release process.** • Teams practice trunk-based development with continuous integration of all changes. • There are enough automated tests that critical defects are detected and prevented fast and automatically. • Provisioning of integrated testing environments is fast and mostly automated. • No work is considered done until it has passing automated unit and acceptance tests associated with it. • Testers are not primarily focused on regression testing. Database changes are versioned and scripted.	Regular release cadence: Release time box is well defined, but duration from idea inception to production release is greater than business need.
2: managed	Time-boxed releases (the team sets a release date and manages to it)	**There is an adaptive delivery process.** • Clear product ownership and chain of responsibility are in place. • Change management controls are implemented, including a process to detect unauthorized changes with consequences defined. • Business participates fully and regularly in development activities and decisions related to delivery. • There is some automated acceptance testing. • Production-like testing environments are available for projects early on. • There is some scripting to reliably and repeatedly configure environments and build packages from version control. • Teams work in iterations of one month or less and showcase integrated	Planned release: Release time box is well defined, but duration from idea inception to production release is greater than business need.

		working software at the end of each iteration.	
1: initial	A few smart people performing heroics	**There is an ad hoc release delivery process** • Teams rely mainly on manual testing after development is complete to find defects. • System integration is painful and happens after development on a module is completed. • Provisioning production-like integrated testing environments is expensive and manual. • Deployment process is manual. • Developers, testers, operations, and management have goals that bring them into conflict. • Change management is ad hoc or heavyweight and often circumvented or ignored.	Ad hoc deployments

Source: Forrester Research, Inc.

An Initial Level Of Continuous Deployment Capability Constrains Innovation With Software Services

At the initial level of maturity, organizations are capable of doing ad hoc deployments of services when they are ready, but they have a hard time predicting when those services will be production quality and are dependent on the actions of talented (or long-serving) individuals. If you've ever been on a project where the build engineer leaves and everyone else is afraid to touch the build scripts for fear of breaking something, then you know what it feels like to be in this initial state. In the initial state, manual testing slows the release process, and there's little organization to service testing. The manual testing that is performed tends to be concentrated between unit testing and exploratory testing.

At the initial level of maturity, there are many manual steps involved in deploying a new version of software service. As a result, a full release cycle (inclusive of testing) can take days or even weeks to complete. Final production deployment consists of taking existing systems offline and deploying new functionality, so an offline release window is required while the system functionality is updated (which often means that the teams get to work over a night or weekend). Because releasing a new version of a service is risky, it's a high-ceremony process, with approvals required by IT and business leadership.

The results of maintaining an initial capability are that it's difficult for business leaders to predict when new software services will be delivered, and the teams that construct these services will usually deliver them later than needed.

Customers will also frequently encounter regressions in functionality or system errors caused by mistakes in manual processes. As a result, firms with an initial level of continuous deployment capability will find their capacity to innovate through custom software severely constrained.

A Managed Level Of Continuous Deployment Capability Introduces An Adaptive Delivery Process

At the managed level of maturity, software development teams are able to work with business leaders to set release time boxes (i.e., a firm date on when they can release a software service). Software development teams then vary the scope of the work performed and the effort applied to meet the agreed release date. At the managed level, delivery time boxes are usually still outside the bounds of what the business may need. As a result, a managed state of continuous deployment is characterized by frequent negotiations over the priority of requirements, defects, and system capabilities versus budget and the release window. Because this negotiation process continues throughout the release, it's critical to have an identified product owner who can examine functionality and quality tradeoffs and maintain customer support for the service while it's being developed. At the managed level, business sponsors regularly participate in scope/resource/date tradeoffs and are well informed about the progress of service development.

A critical hallmark of the managed state is that automated processes and tools manage changes to requirements, code, test cases, and deployment scripts. This includes the capability to audit changes and restrict changes as a service nears production deployment. Another important step in creating a managed environment is significant automation of the build, test, and deployment process. This includes automation of regression testing, smoke testing, and performance testing activities as well as automation of the build, packaging, and deployment processes. As a result, the time it takes to ready a distinct release (inclusive of testing) drops to days.

At a managed level of continuous deployment maturity, development teams should be capable of setting a release date for a service and then managing their development schedule to meet that date (more or less). Business sponsors may choose to slip a release date if the scope of the project changes or unforeseen risks occur that need to be dealt with before deploying the service. At the managed level, the development team's basic release capability gives business sponsors some visibility into service development and provides limited opportunities for course corrections and reprioritization of requirements for subsequent development. While the speed of service releases increases under a managed state of continuous deployment, there is still enough latency to prevent release on demand. Projects that implement Agile development practices like regular integration and backlog management should find that they have the necessary process discipline to achieve a managed state, where even if they can't release new capability as fast as business sponsors need it, they can demonstrate regular process and confirm that an envisioned service will function as planned while meeting desired business objectives and performance measures.

A Defined Level Of Continuous Deployment Capability Builds Quality Into The Release Process

At the defined level of continuous deployment maturity, the release process becomes a regular key indicator of project health. Most development teams that operate at this level of maturity create a build at least once a day from trunk, and developers will ensure they don't have more than a day's work sitting on a branch in version control. Accordingly, branches to the main release trunk are short-lived, and the result is a significant reduction in system integration issues — which tend to fester when individual developers or subteams maintain private builds or source code branches.

Another process that is commonly seen with teams at the define level is automated testing of builds (also known as commit tests). Commit tests are a basic set of tests that are run against each code commit to the mainline source trunk.

They assess the health of each build and evaluate its suitability for further regression testing, performance testing, and exploratory testing. If commit tests fail, it's a big deal, and the development team is immediately alerted.

Regular builds are critical in large software projects, because they allow individual teams to make progress, and where dependencies between teams exist, it allows those teams to get fast feedback when their work doesn't integrate. Teams may work against the most recent CI build, or they may choose a defined milestone build, which is known to be good, while they work to stabilize their own service components. A regular pattern of builds with milestones provides flexibility at scale and is used to reliably ship software services that contain many millions of lines of code on an exceedingly predictable schedule.

At the defined level of maturity, deployment of new service versions is further accelerated by automating the process of provisioning integrated environments (i.e., setting systems up for testing and tearing them down when finished). In most cases, development teams at this level focus on automating deployment into a system test environment (as opposed to production or a blue-green setup). Changes to database components of the system (DDL, DML) are also versioned and controlled, database creation is automated, and migration is automated as part of the release management process.

When a team reaches the defined state of maturity, the result is a regular release cadence. The release time box is well defined, and it's easy to identify early warning sign that a project is in trouble. This makes it possible to exert early corrective actions like scope reduction, resource augmentation, and a schedule replan. While service delivery is much more predictable than the initial or managed stage of maturity, it may still not be as fast as business leaders desire.

A Quantitatively Managed Continuous Deployment Capability Enables Release On Demand

As development teams progress to the quantitatively managed level of maturity, they reach a critical plateau — one where they can deploy a new release whenever one is needed. From a business perspective, the speed of service development meets or exceeds that capacity of the business to assimilate new services. At this level of capability, software development capability no longer hinders business innovation — it enables it.

As development teams reach this level of capability, we also see a shift in organization structure. Instead of vertically organized centers of excellence (e.g., business analysts, development, quality assurance [QA], infrastructure and operations), we see an organizational shift to cross-functional product teams. Individual roles become less important than the tasks that need to be completed to release new capability, and teams are likelier to self-organize their works. As part of this transition, the entire team assumes responsibility for service-level agreements associated with their product.[7] And it's not just service testing and deployment that folds into the cross-functional organization. User experience should also be integrated into development teams at this level — that is, teams should be iteratively discovering and refining the design of the products and services they build.

As teams reach the quantitatively managed level, they find that they have eliminated all remaining bottlenecks in downstream build, test, and deployment phases. Deployments of individual changes in source code can now be performed in minutes, and there's no reason a team can't do multiple deployments to production in a single day. As software development teams move to this stage of maturity, delivery teams prioritize keeping the main code trunk deployable over doing new work. Development teams don't let the integration build stay broken for more than a few minutes — and anybody is empowered to revert breaking changes from version control. Some organizations go even further and have the build system revert breaking changes automatically.

The focus of testing services also shifts up the development process. Test-driven development and acceptance-test-driven development become core processes for development teams. These tests are layered on top of preflight builds to provide semantic validation of system function in addition to technical validation.

As code deployments become more continuous, traditional barriers between developers, testers, and systems administrators recede. Developers write tests, and system admins provide infrastructure- or platform-as-a-service for developers to use.[8] System deployment planning begins at service inception, with deployment modeling, and early simulations of deployment. New code that breaks the deployment loop results in dev and ops working together to understand why. As new capabilities and defect features are unbatched, teams start to focus on flow through the system and instrument the deployment systems to measure their progress and detect issues as early in the process as possible. From a developer's point of view, "done" means tested and working in production — not that the code has been checked into the SCM system or is dev complete.

An Optimizing Level Of Continuous Deployment Allows Software Developers To Drive Business Value

At the optimizing level of continuous deployment maturity, software development teams drive a continuous stream of incremental innovation. They form hypotheses about customer needs and how they can serve them, run experiments to test these hypotheses with customers, and then use feedback from their experiments to make design and service implementation decisions based on the best course of action. For example, if a new algorithm reduces server load by 5% without affecting response time, it can be implemented servicewide immediately. If a new web page generates 10% more click-throughs, it can be rolled out to more web servers for more extensive testing. In this hypothesis-driven development model, software development teams focus on optimizing cycle time in order to learn from customers in production, by gathering and measuring system feedback.

A hypothesis-driven development model more closely resembles classic scientific inquiry than traditional application development. A long list of requirements are replaced by testable hypotheses. When new features are proposed, those requesting them must include criteria that specify how their value will be measured. Software development teams are responsible for implementing metrics to gather this data through techniques such as A/B testing. As a result, development teams working at this level may stand up multiple production environments so that they can observe the effect of large-scale changes or gradually stage changes from a beta customer population to that mainstream user population. Parallel production environments also allow teams to quickly pull the plug on bad experiments that exhibit unintended consequences.

Working at an optimizing level of continuous deployment also requires changes to software service architectures. Developers write code with continuous deployment in mind, supporting patterns such as dark launching and feature toggles to decouple deployment from release.[9] As deployment regularity increases, the releases of specific features or capability shifts from being a technical decision (i.e., when can we deploy it?) to a business decision (i.e., when do we want to turn it on and for what customers?).[10]

Teams that work at an optimizing level also tend to favor scale-out systems that can be slowly ramped and decommissioned as testing telemetry roles in. Asynchronous services allow load balancing work distribution among different variants of the same service. Application-programming-interface-based system communication allows operational systems to inject probes into real-time production operations to monitor application load, deploying more resources as necessary. Developers also test and evaluate the system as it functions in production — they may even

initiate failures to make sure that the system takes proper corrective action. At the optimizing level, database changes are decoupled from application deployments, and individual service endpoints are decoupled from each other. Large-scale services can be upgraded a component at a time. At the optimizing level of continuous deployment, software development teams can initiate business change by running experiments and proving business value. As the cost of each service deployment trends toward zero, the cost of running discrete experiments also drops significantly and becomes largely a factor of development labor costs.

KEY RECOMMENDATIONS

Forrester's in-depth survey with business and IT executives yielded several important observations. Innovation is on the agenda, but innovating through delivery of custom-developed software services is not as easy task. To solve this challenge:

- **Development teams must speed up their rate of service delivery.** The survey results are clear — there's a gap in when business leaders want custom-developed software services and when they get them. To avoid the perception that IT is slowing down business innovation, software development teams must adopt development practices that speed up the rate of service delivery and work to simplify the architecture of existing systems when they create a drag on IT's time and resources.
- **Business leaders and IT leaders need to improve their collaboration.** It's interesting that software-centric firms tend to be organized in a very different way than business with centralized IT groups. In order to compete with software-centric firms, traditional businesses need to shift their organization models to view software development providers less as order takers and more as partners in delivering BT. IT shops need to redesign their organizations to focus on end-to-end service delivery instead of organizing around functional centers of excellence. These changes are the cultural precursors to establishing a rapid pattern of continuous delivery. To put it another way, individual service teams need to act like their own self-contained startups and act collectively with a focus on the end customer's needs and feedback.
- **Agile practices are a good start, but they are not sufficient.** Many businesses are attempting to increase service delivery speeds by introducing Agile practices into their development teams. It's a good start, but to really increase the rate of innovation, it's important to also invest in improving downstream release management in order to increase overall delivery capability. Otherwise, all of the changes teams implement back up behind manual integration and testing processes and unreliable deployment efforts, and teams spend too much time delivering low-value software, because they are focused on emergency patches or system rollbacks due to deployment errors.
- **As teams get better at continuous delivery, development gets easier.** If your teams are not at level 4 on the continuous delivery maturity model, you should plan on how you'll get there as soon as possible. Why? While getting to the quantitatively managed level is a significant challenge, as your teams become experienced and automation increases, it becomes much easier to move small changes into production. Moving away from Big Bang deployment significantly reduces risk and unlocks a regular value stream. It also means less overtime, fewer weekend release windows, and improved team morale. If your organization is currently at level 1 or 2, then push your best-skilled teams toward level 4 first and then use their experience to seed other teams throughout your organization.

Appendix A: Methodology

In this study, Forrester conducted an online survey of 325 decision-makers in the US (N = 169), UK (N = 93), and Australia (N = 63) to evaluate their software development practices. Survey participants included line-of-business decision-makers (business decision-makers) and IT decision-makers in both manager and executive roles. Questions provided to the participants asked about software development practices as well as software development investments. Respondents were not told who commissioned the study and were each offered a small incentive as a thank you for time spent on the survey. The study began in August 2012 and was completed in September 2012.

Appendix B: Supplemental Material

Related Forrester Research

"The Future Of Mobile Application Development," Forrester Research, Inc., January 17, 2013

"Case Study: City Index Becomes Agile With Release Automation," Forrester Research, Inc., May 29, 2012

"Five Ways To Streamline Release Management," Forrester Research, Inc., February 7, 2011

Online Resources

ThoughtWorks Website (http://www.thoughtworks.com)

Continuous delivery site (http://continuousdelivery.com/).

Martin Fowler's Continuous Delivery page (http://martinfowler.com/delivery.html).

Print Resources

Jez Humble and David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley Professional, 2010.

Appendix C: Endnotes

[1] Marc Andreessen wrote a compelling analysis of why "Software Is Eating The World." Source: Marc Andreessen, "Why Software Is Eating The World," *The Wall Street Journal*, August 20, 2011 (http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html).

[2] In 2003, Nick Carr wrote a highly influential article entitled "IT Doesn't Matter" for the *Harvard Business Review*. Please go to the *Harvard Business Review* website for more detail. Source: Nicholas G. Carr, "IT Doesn't Matter," *Harvard Business Review*, May 1, 2003 (http://hbr.org/product/it-doesn-t-matter/an/R0305B-PDF-ENG).

[3] Source: (http://articles.timesofindia.indiatimes.com/2012-11-30/strategy/35484411_1_jeffrey-immelt-software-andreessen-horowitz).

[4] When companies deploy applications to virtual machine (VM) images, it makes it easier to move the entire image from one environment to another (e.g., test to production or blue to green). This allows companies to make the final release cutover in a matter of minutes or even seconds. Source: "Five Ways To Streamline Release Management," Forrester Research, Inc., February 7, 2011.

[5] Google allows developers to devote up to 20% of their time on whatever projects they want to work on to benefit its business. 3M, Hewlett-Packard, and Yahoo do something similar. Source: Jeff Atwood, "Make Your Job Feel Less Like Work with '20% Time,'" Lifehacker, August 7, 2012 (http://lifehacker.com/5932586/make-work-feel-less-like-work-with-the-8020-rule).

[6] Source: (http://msdn.microsoft.com/en-us/library/hh765983.aspx)

[7] Amazon is a good example of this transition (as per Werner Vogels' "you build it, you run it" [i.e., teams are operating and supporting the products/services they build]). Source: "A Conversation With Verner Vogels," Association for Computing Machinery interview, May 2006 (queue.acm.org/detail.cfm?id=1142065).

[8] For more information on this type of practice, read how Netflix does it. Source: Adrian Cockcroft, "Ops, DevOps and PaaS (NoOps) at Netflix," *Adrian Cockcroft's Blog,* March 19, 2012 (perfcap.blogspot.com/2012/03/ops-devops-and-noops-at-netflix.html).

[9] Dark launching is a technique of wrapping the code of new software features in a way that lets you turn them on or off. Source: Tom Cook, "Hammering Usernames," Facebook, July 1, 2009. (http://www.facebook.com/note.php?note_id=96390263919 for more information about how Facebook uses Dark Launching).

[10] For more information on decoupling deployment from release, read the related article. Source: Jez Humble, "Four Principles of Low-Risk Software Releases," *Inform IT*, February 16, 2012 (http://www.informit.com/articles/article.aspx?p=1833567).