

February 7, 2011

Five Ways To Streamline Release Management

by Jeffrey S. Hammond

for Application Development & Delivery Professionals



February 7, 2011

Five Ways To Streamline Release Management

“Next Practices” That Will Improve Release Visibility And Speed

by **Jeffrey S. Hammond**

with John R. Rymer, Mike Gilpin, Adam Knoll, and Sander Rose

EXECUTIVE SUMMARY

Our first release management survey confirms that IT leaders are frustrated with slow software delivery, including dissatisfaction with the release management process. While Agile speeds software design and development, it doesn't do much to speed up release and deployment — creating a flash point where frequent releases collide with slower release practices. This is motivating development organizations to work with their peers in operations to streamline release management. They are achieving this by improving prebuild processes, expanding release management throughput, optimizing their release pipelines, designing software for rapid change, and creating common release portals. Any one of these practices will bring improvement, so pick one, start small, measure the results, and then expand to more in time.

TABLE OF CONTENTS

- 2 **IT Leaders Are Frustrated With Slow Software Delivery**
- 6 **Improve The Release Cycle One Step At A Time**
- 9 **Five Next Practices That Improve Release Flow**

RECOMMENDATIONS

- 19 **Better Release Management Is Not Beyond Your Reach**

WHAT IT MEANS

- 20 **Better Release Management Is A Logical Next Step For Agile Dev Shops**
- 20 **Supplemental Material**

NOTES & RESOURCES

Forrester interviewed more than 20 vendor and user companies and fielded a survey on current release management practices.

Related Research Documents

“The Forrester Wave™: Agile Development Management Tools, Q2 2010”

May 5, 2010

“Best Practices In Release Management”

December 13, 2007

“Case Study: Eclipse Convinces Its Projects To Board A Single Release Train”

December 13, 2007

IT LEADERS ARE FRUSTRATED WITH SLOW SOFTWARE DELIVERY

At Forrester, we often hear clients say, “Our business is pushing us to deliver new features and functions faster.” As software becomes more embedded in their business, firms doing custom development are finding that the velocity of business change is now limited by how quickly they can deploy software into production. If they can only release twice a year, then they can only change the business twice a year. But it’s actually worse than that, because this assumes developers get changes right the first time, every time — but few shops achieve such perfection.

A twice-a-year update may be fine in slow-moving industries, but it doesn’t work so well if your competition is introducing new customer Web self-service features or snazzy new iPad apps every time you turn around. Firms now have more communication channels to employees or customers than they’ve ever had before, and nimble companies are taking advantage of mobile devices, tablets, kiosks, push technology, and cloud computing to create engaging experiences that build customer loyalty. They don’t call it the rat race for nothing!

Agile Development Is A Good Start, But It’s Not Enough

Adopting Agile development practices is the logical first response to an unmet “need for speed.” Agile practices like continuous integration and daily stand-ups allow teams to build software faster and make midcourse corrections as needed. And Agile adoption is up: In this year’s Forrester/Dr. Dobb’s Global Developer Technographics® Survey, 37% of developers tell us that they use Agile development as their primary development process, up six points from last year’s survey (see Figure 1).

However, many Agile teams find that early success at the project level falters when teams try to scale their work. Multiple teams are building software in one- to two-week sprints, but they aren’t necessarily getting the right software released into production any faster. Things slow down because:

- **Agile techniques don’t address the entire application life cycle.** Agile techniques were originally created by developers for developers. It’s true that some techniques like test-driven development benefit other roles, but they do so within the context of the development sprint. Common Agile practices don’t really speak to the needs of infrastructure and operations professionals. These team members crave stability, and their primary goal is to meet the service-level agreements (SLAs) they’ve agreed to uphold. They need to deploy apps into production to support new releases while making sure that new software won’t break anything else. Agile techniques don’t offer them much guidance or help.
- **The definition of “working software” varies by team.** What does working software mean to your team? Does working mean that code compiles? That the app is unit tested or system tested? Alternatively, is it when they release an application into production? Once developers have working code, it’s tempting for them to move on to the next set of tasks atop the backlog. In the meantime, someone else needs to push delivered tasks for testing and deployment as

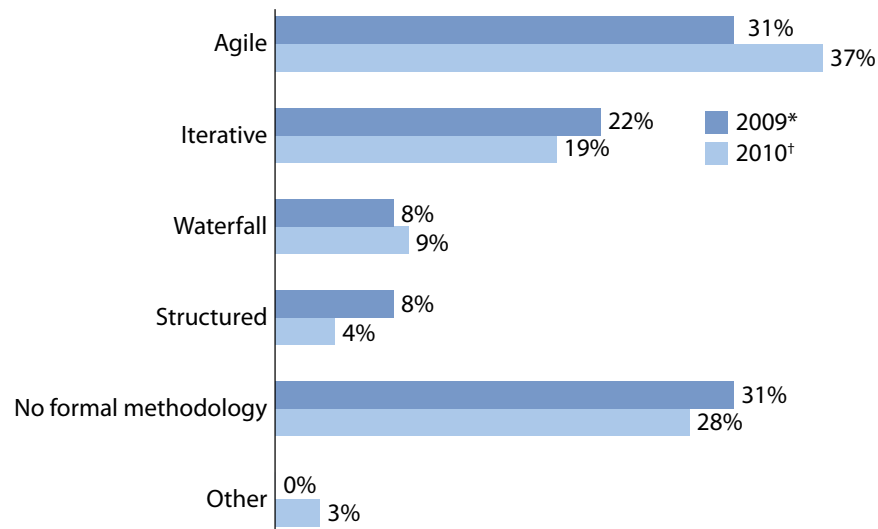
modifications to production code. And when release managers need to incorporate these changes into a larger release train, the rate of deployment into production can slow considerably.

- **Componentization drives complexity downstream.** As developers use more open source libraries and break apart monolithic applications into modular systems of systems, they are reusing more code and writing less. But this increase in reusable components comes with a price: Components evolve independently, increasing the challenge of managing system dependencies. As a result, the correct assembly of modules — or configuration — becomes a critical component to a successful deployment.

Developers on Agile teams may be happy because they are moving faster in the earlier phases of their projects, but this still leaves a big gap between building software that compiles and a system that’s ready to release into production. That’s where an updated set of release management “next practices” become a logical follow-on to your Agile process investments.

Figure 1 Agile Process Adoption Continues To Advance

“Please select the methodology that most closely reflects the development process you are currently using.”
(Choose one.)



*Base: 1,298 IT professionals who have knowledge of current development methodologies

†Base: 1,023 IT professionals who have knowledge of current development methodologies

*Source: Forrester/Dr. Dobb’s Global Developer Technographics® Survey, Q3 2009

†Source: Forrester/Dr. Dobb’s Global Developer Technographics Survey, Q3 2010

Release Management Creates Frustration And Slows Teams Down

As part of this research, we investigated whether the dissatisfaction we hear with the current status quo of release management is specific to the clients we talk to or indicative of a broader trend. In November, we surveyed just over 100 development shops about the current state of their release management practices. We found that development professionals who know about or are responsible for release management aren't that happy with their current environments (see Figure 2). Overall, on a scale of 1 to 10 (with 10 being completely satisfied), our survey respondents rated their release environments with an average score of 5.19. We found that:

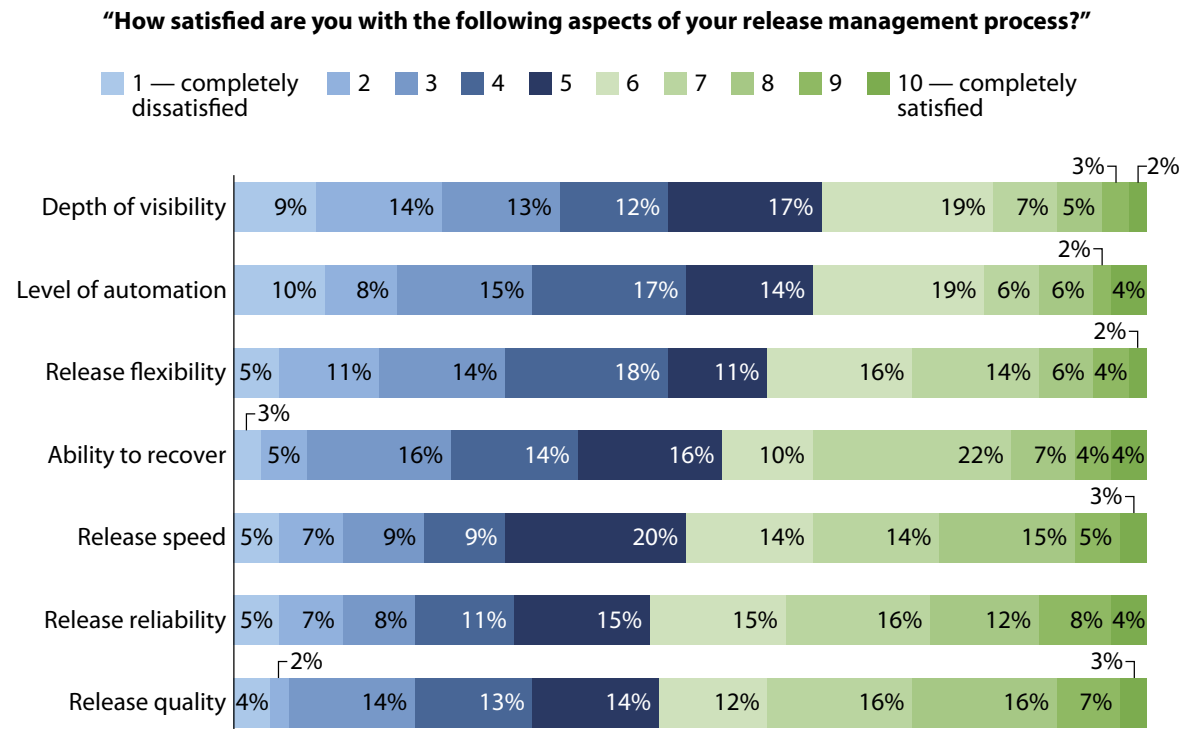
- **Depth of visibility into the process is the most frustrating problem.** Imagine a customer files a defect on your current project — let's call it defect 5479. They'd like to know which release will fix it, or when that will be available. Sounds like a simple question, right? In reality, most teams have trouble answering this question, because it's difficult for them to track the flow of changes through the release process. That's why it's no surprise that this is the most frustrating aspect of release processes for our survey respondents at an average satisfaction of 4.54 out of 10.
- **A lack of automation points to a desire to do things faster.** The next most frustrating element of the release process is the level of automation, coming in at an average score of 4.66 out of 10. Every time there is a manual task or a human involved in the release management process it's going to slow the overall process down. And let's face it, the tasks involved with creating releases aren't particularly fun or easy to do. Maybe that's why we commonly see development teams give the job to the newest/youngest member or make the person who breaks the build assume responsibility for fixing it until someone else gets put into the release management penalty box.
- **Satisfaction over "ilities" points to the desire for better release tuning.** The development professionals we surveyed were also dissatisfied with their ability to create different types of releases such as patch releases, virtual machine images, or complete releases (4.89 out of 10). They also showed frustration with their ability to quickly troubleshoot and recover from a bad release (5.38 out of 10). These "ilities" matter because they allow teams to make midcourse corrections in the release process the same way Agile techniques do in the development phase.
- **Speed and reliability aren't the worst problems, but they are still problems.** Release speed is the fifth most frustrating aspect of respondents' release management. It's tempting to conclude from its placement that speed is not a problem. That's not the case: It still rates only an average of 5.53 out of 10. In fact, two-thirds of survey respondents rate satisfaction with release speed at 6 or lower, so satisfaction is not that high — it's just higher than for other aspects of release management. Similarly, release reliability rates 5.66 out of 10.
- **Teams favor correctness over all other release management qualities.** It's no surprise that teams are most satisfied with the quality of their releases (although even that gets a relatively low average of 5.71 out of 10). What this indicates is that teams are most likely to sacrifice speed and

flexibility for a release that works — as they should. It also points out an important constant of the release process. Any change to tools, processes, or culture must either maintain or improve release quality, or teams will not adopt them.

This data confirms what we hear in client inquiries; there are many frustrating issues that keep teams from releasing software with visibility, predictability, and speed. At the same time, teams are not sure how to improve the current state of the art. Two years ago, we wrote about a number of release management best practices that firms are using to improve their release organization and their capacity to release software (see Figure 3). While Agile projects change how teams apply some of these practices, they are still valuable and bear repeating here.

These best practices will help teams get to a basic level of release capability, but it's the next practices we're interested in now. Read on to find out how forward-thinking development teams are putting these next practices into effect in an Agile project context.

Figure 2 Release Management Satisfaction Is Low



Base: 101 IT professionals involved in or aware of their company's release management processes (percentages may not total 100 because of rounding)

Source: Q4 2010 Global Release Management Online Survey

58422

Source: Forrester Research, Inc.

Figure 3 Release Management Best And Next Practices

Best practice	“How to”	Pitfalls
Build a release management team	<ul style="list-style-type: none"> • Create a formal release manager position to oversee work associated with each release • Group release management with related functions like build, SCM, and testing • Position release management as a centralized interface between apps and ops • Identify and invest in relationship managers who can represent business needs 	<ul style="list-style-type: none"> • Letting release managers become practitioners instead of governors • Allowing release managers to shift too far toward apps or ops • Hanging your release managers out to dry
Verify production readiness	<ul style="list-style-type: none"> • Verify that all parties have followed the release process • Confirm that the release meets all quality standards • Ensure that everyone in the business and IT is prepared for the release 	<ul style="list-style-type: none"> • Being satisfied with pro forma process conformance • Permitting system testing to slip into release testing • Slipping quality control activities to hit schedule or scope targets
Tune release frequency and type	<ul style="list-style-type: none"> • Reduce the number of releases by grouping changes by content or by date • Balance the costs of each release with the benefits • Maintain different classes of release with clear criteria for inclusion in each • Look for signs of pain that can be traced back to release frequency 	<ul style="list-style-type: none"> • Being too lenient in the name of customer service • Being too conservative in the name of risk management
Next practices	<ul style="list-style-type: none"> • Stay on top of your application and service dependencies • Automate the build-test-deploy process • Build a release portal 	

58422

Source: Forrester Research, Inc.

IMPROVE THE RELEASE CYCLE ONE STEP AT A TIME

The first step to improving the speed of releases is to attack the problem that development leaders find the most frustrating — the depth of visibility they have into their release processes. If you can’t measure the results of a process, it’s hard to know whether it’s getting better or worse, and why.

Get Started By Measuring The Release Process

On Agile teams, working software is a measure of progress. Working *released* software is also a good gauge of the release process; consistent, working releases are a sign that the process is healthy and that you don’t have to worry about dysfunctional behavior like individual project teams setting up their own private builds to get work done. If not all your releases go like clockwork today, then you’ll need to do a bit more digging to find out why that’s the case.

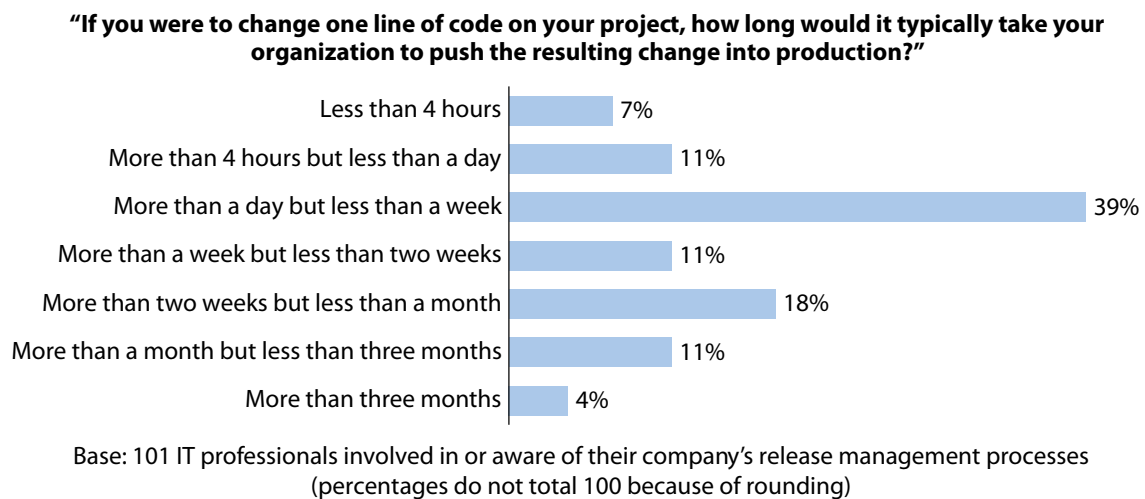
One way to get that effort started is by determining what your project’s *null release* cycle is. To do so, ask those who are involved in your release processes a simple question:

“If we changed one line of code in our application (or system), how long would it take us to deploy it into production using our regular release process?”

It seems like an easy question for your teams, but in most development shops the answer is anything but. Your team may ask for clarification of what you mean or dismiss the question entirely — but be persistent. What you’re after is the complete set of information to define all the technical and customary process steps needed to push your one-line code change through your release management process. Like sausage making, you may be horrified at the results.

We posed this question in our survey, and the results tell the tale (see Figure 4). Only 25% of the shops we surveyed have a null release cycle that is less than a day, and for 44% of respondents their null release cycle is a week or more. In practical terms, this creates a huge problem for Agile teams. Why should a five-minute task like changing a line of code kick off days of effort for release managers? This is where the fun begins in your quest.

Figure 4 How Long Does It Take To Release?



Source: Q4 2010 Global Release Management Online Survey

58422

Source: Forrester Research, Inc.

Kinks In The Release Pipeline Slow Flow

As you dig into the depths of your own null release cycle you’re likely to discover that there are all sorts of manual processes and checklists that create “kinks” in the flow of software changes in your development organization. Each shop is unique, but here are some common kinks to look for as you dissect your own null release cycle:

- **Manual handoffs between process steps.** Whenever there is a manual task like an explicit approval or a verbal conversation between release steps, it will create an indeterminate delay in the release process as humans execute these tasks. One or two may not be a big deal, but if you find a dozen or more manual handoffs, it's a good sign that your release process might need some re-engineering.
- **Manual testing limits release throughput.** If you only release two to three times a year investing in automated testing may not seem that important, but if you're seeking a faster release process you'll soon discover that manual testing is often a critical path task in your null release, especially for applications of any complexity.
- **Monolithic projects require monolithic release efforts.** We recently talked with a client whose null release cycle was several months long. The primary reason for this long cycle was that the application code was so tightly coupled that the client felt the need to completely test all the code for safety and regulatory reasons, even if making only one change. If a single change requires an entire battery of builds and test-passes then it may be time to consider refactoring to introduce more modularity into your software.
- **Bad builds lead to private builds.** Bad builds kill productivity, especially in large projects. Individual project teams need to test the changes they've submitted into the build stream, and if the integration build is always broken then individual project teams are likely to set up their own private builds for unit testing. While this creates short-term productivity gains, it also creates long-term integration nightmares as the baselines that different teams test against drift away from one another.
- **Rube Goldberg scripts make builds brittle.** Many release processes are composed of all sorts of odd Perl scripts, batch files, ant tasks, xUnit scripts, and/or make files. Each step in the process might be simple, but added together these layered scripts create a complex process that few understand and a situation where failures require someone with deep knowledge of the build process to troubleshoot. Hopefully the creator of the process is still on the project team and can remember the original motivation behind what he built.
- **Sparse reporting makes troubleshooting difficult.** When builds fail the first question that you need to answer is "Why?" Even if the build process itself might have completed correctly, subsequent testing might show that the wrong files were included or that developers introduced functional regressions or security flaws through the build process. Without logging and auditing of the bill of materials, troubleshooting failed builds can become a Byzantine process.

The goal of cataloging your own null release process is that it highlights the kinks that impede *your* release flow. For example, it may be great that you've created 1,000 automated test cases, but if you have to run them every time you change one line of code and it takes a day, maybe it's not so great.

Better to modularize the system so that you know which tests you need to run to get complete code coverage, enabling you to update a build configuration in only minutes. Documenting your null release also allows you to answer follow-up questions like what value each step in the process adds and whether you could automate or redesign individual steps to improve overall release flow.

FIVE NEXT PRACTICES THAT IMPROVE RELEASE FLOW

Over the past nine months, we interviewed a number of firms that are working diligently to improve their release management processes. As a result, many now have the capability to “release on demand.” That doesn’t necessarily mean they release every day or every week, but it does mean that they can release new functionality at the precise time their business partners demand it. These shops have improved release flow (and speed) by using a set of next practices, which are quickly becoming required practices for any shop that wants to streamline releases.

Next Practice No. 1: Improve Prebuild Processes

Any winning professional coach will tell you that the key to winning is a combination of preparation and perspiration. The same is true of your release management game. There’s a lot you can do to improve your results by properly preparing software for your release queue:

- **Outlaw testing against private builds maintained by individual teams.** Private builds are a development anti-pattern. They allow teams to focus on an interim deliverable, which customers never see. Teams don’t accept changes and new features from other teams into their own software change management (SCM) branches because they aren’t pushing their changes to a common release baseline in a timely manner. Even though it moves pain earlier in the process, committing early to a single release stream will keep technical debt from accumulating, which might cause a failed integration that could obstruct a major release milestone.
- **Make your continuous integration process change-aware.** One practice that improves release visibility is where teams link multiple related code changes to a single change task in change management tools like Atlassian JIRA or IBM Rational ClearQuest. The next step is to link these change tasks to the continuous integration process that Agile teams use to feed the release cycle. Linking delivery of code changes to logical tasks like “Fix defect 5479” or “Add the capability to override the ship date to the shipping screen” allows teams to answer the fundamental questions of what changes are in which releases and where they are in the release cycle.
- **Employ “preflight” builds before committing code.** One way to reduce the number of bad builds is to reduce the number of errors introduced into the build stream. Just because a set of code changes compiles on a developer’s local machine doesn’t mean the code is correct in a release context. Companies that can build on demand can use that capability to execute preflight builds. This practice runs a complete integration build on new code that a developer wants to

check in and only allows a check-in if the build completes successfully. While the principle is simple, the effect on the purity of the integration stream is profound. And the cleaner the integrated build stream is, the less likely that private builds will spring up to confuse the situation.

Implementing these three prerelease tactics will make the flow of changes into the release stream more predictable and keep the release pipeline free of garbage and pollution from code that's not up to snuff. Your developers may grumble at the start from the additional scrutiny on their work, but as they grasp how a single clean release stream aids them, they'll become believers in your pregame regimen.

Next Practice No. 2: Expand Release Management Throughput

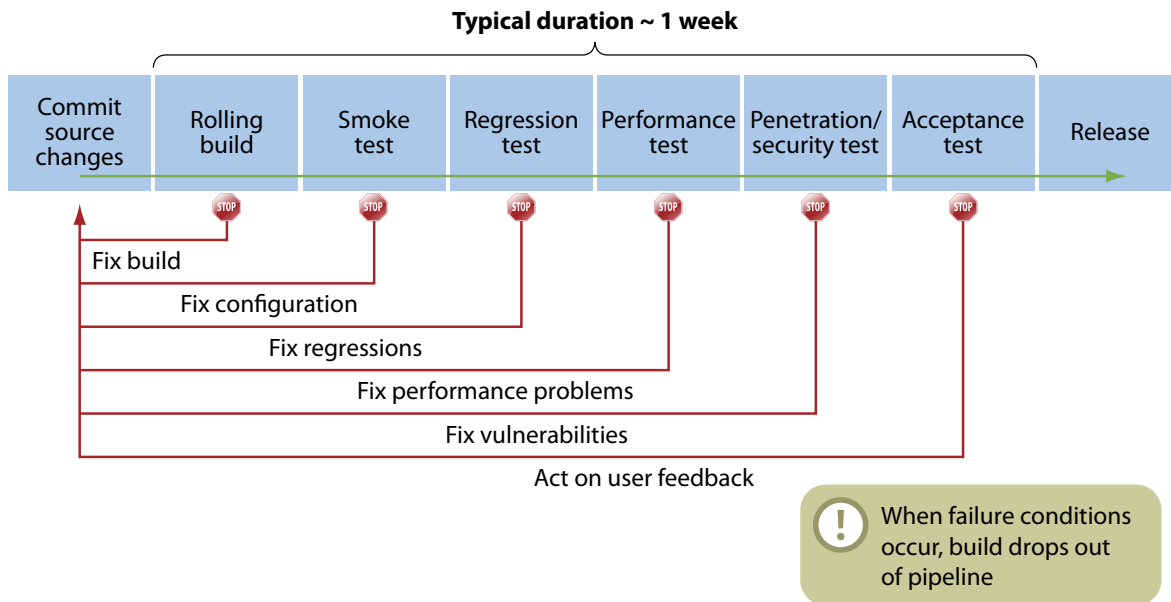
The downside of improving your prebuild processes is that you'll bring even more changes into the release management process. True, they'll be better organized and vetted, but there's no way to avoid needing to improve the overall throughput of your build and post-build processes, or all your other efforts will be for naught. To increase your release throughput:

- **Apply the concept of a release pipeline to your work.** Think of your release management process as a pipeline, and each build as the commodity that will flow through it.¹ Each activity in the release management process is a stage in your release pipeline (see Figure 5). One reason this metaphor works is that it gets your team thinking about overall flow instead of efficiency at individual stages. And like real pipelines, the flow capacity of a release pipeline is constrained by the capacity of its narrowest “choke point.” If that choke point is in the final stages and few builds actually reach it, that's not so bad. But if it's early on in the process — say, during the build process itself — it can back the flow of your changes up as surely as a stopped-up drain in the tub.
- **Use “smoke tests” to evaluate builds quickly.** Subjecting new builds to a basic battery of automated tests is a good way to ensure that basic functionality is enabled and that the build is suitable for further analysis. Design these “smoke tests” to catch basic build errors like those that result from linking in the wrong versions of build components or those that would prevent QA from installing the build and executing tests against it. If you find a recurring pattern of failure in your release management process, create a test to detect the condition and add it to your battery of smoke tests. In time, you'll find that the number of bad release candidates that make it into the later stages of your pipeline will go down, which will avoid wasting time and effort.
- **Decompose each process into the smallest possible logically independent steps.** Breaking up release tasks into individual processes like build, smoke test, package, and create deployment image lets teams apply filters and tests to the outputs of each stage before passing a release onto the next stage of the process. It's OK for builds to drop out because they've failed at a particular stage of the process — failed releases are a sign your QA processes are working — if most of the failures occur early in the release process. The later builds fall out, the more waste you created, and the more opportunity you have to improve productivity. Decomposition of a release process also gives you the opportunity to measure where errors are arising and to ask what could be done to detect recurring failures earlier, as in the initial smoke-testing phase.

- **Automate repetitive steps wherever possible.** Although people surpass computers in many things, computers are much better than people at repeatedly executing well-defined tasks. And when you decompose your release process into atomic tasks, you'll find many repetitive tasks that benefit from automation. The build process is a natural area for automation, but so are smoke testing, regression testing, static testing, and security testing. Start by automating the early steps in the process, speeding up the rate at which your systems can autonomously detect and remove bad builds from the pipeline. Teams that automate build, smoke tests, and regression testing find that they can significantly reduce their null release cycle — from days to hours.
- **Use parallelism to speed up builds.** As you break up and automate tasks, you can execute independent tasks in parallel to speed release management. One of the easiest places to do this is in the build process. Parallel builds aren't that important if you're building a departmental application with 10,000 lines of code, but it is if you are building multimillion-line systems like one large semi-conductor technology supplier we spoke with. In this firm's case, it was taking 10 hours to build one of its software systems. Bad builds and long build times were killing productivity. Going parallel with Electric Cloud's ElectricAccelerator cut the firm's build time to less than 3 hours and reduced the number of bad builds too. The net result is a significant decrease in its null release cycle.

As you focus on improving the throughput of your release pipeline, you'll find that cutting the time spent in the early stages of the process is eminently feasible with a combination of decomposition, automation, and parallelism.

Figure 5 A Simple Release Pipeline



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

58422

Source: Forrester Research, Inc.

Next Practice No. 3: Optimize The Release Management Pipeline

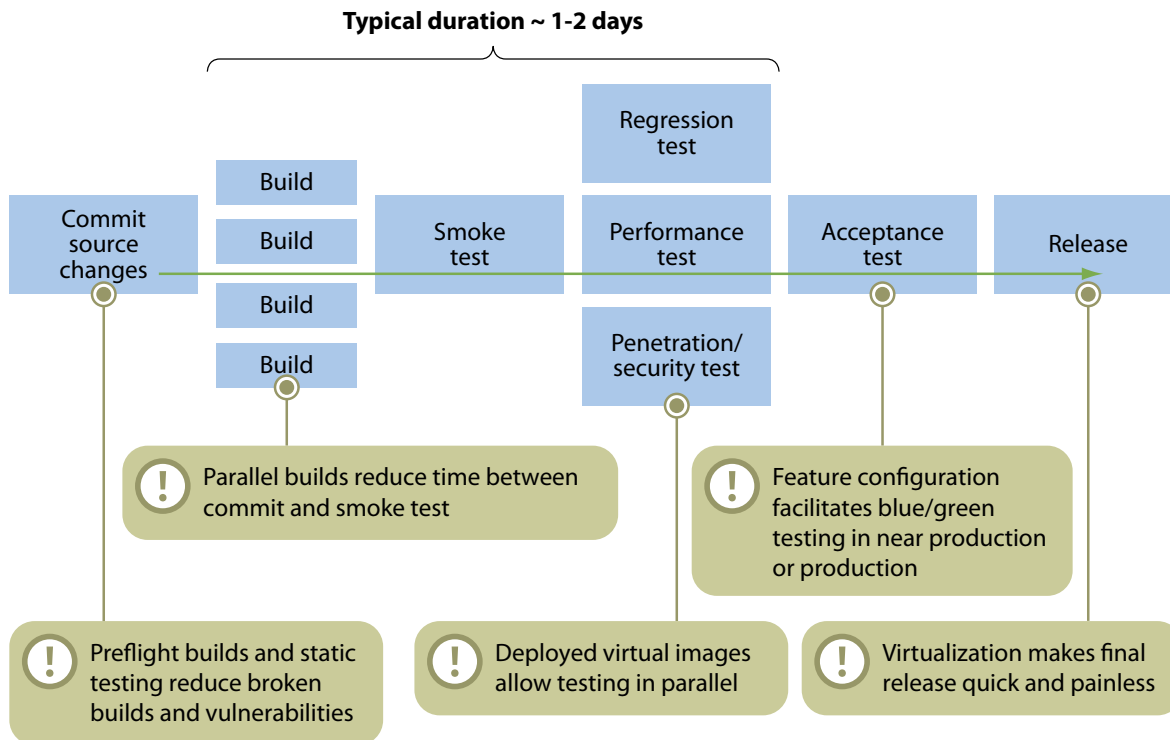
Even after you've expanded release throughput and automated a simple release pipeline there are still opportunities to reduce the total amount of time it takes to push a release through your process. Release speed freaks are adopting a number of additional tactics that further compress the time it takes to deploy a new release. To supercharge your release pipeline:

- **Use virtual images to enforce consistency throughout the process.** Teams that work in virtualized environments have discovered that they can use the images required by ops personnel to their advantage in the release process. They extend their build processes to package an image that they deploy to their VMware, Xen, or Amazon infrastructure. At Kroll Factual Data they've so highly virtualized their production and release management environment using Microsoft's System Center Suite and System Center Virtual Machine Manager technology that the final push to production is as simple as reassigning an image from one pool of hardware resources to another — in many cases the machine on which the image resides doesn't even change.
- **Use parallelism to compress the testing phase of the release pipeline.** Builds aren't the only task in the release management pipeline that teams may execute in parallel. It's also possible

to execute a battery of tests in parallel on a release that's passed smoke testing. Executing regression, performance, and security tests in parallel can significantly compress total time spent under test.

- **Use masked production data to improve testing fidelity.** One of the subtleties of the release process is that while applications move forward into production, the best test data tends to move backward from production databases. This can create a challenge when the data is sensitive — you don't want developers looking at real patient data or transaction histories. This is where a good data masking process can be a godsend — it lets developers get the range of values and breadth of data they need to test for odd errors while keeping you in compliance with regulatory guidelines. Masking processes can be scripts written by data professionals or automatically generated using tools like IBM's InfoSphere Optim Test Data Management Solution.²
- **Implement a developer self-service environment.** Ever wonder why empowered developers are drawn to public clouds like Amazon EC2 and Microsoft Azure? One of the biggest reasons is that they are able to self-provision their own instances for testing and even production. In the public cloud infrastructure, operations professionals are not gatekeepers, and few manual processes stand between making a code change and getting it into production. But there's no magic technology in the public cloud that prevents the same self-service capabilities from being implemented in a highly virtualized private cloud, except company culture and policy.

If you implement one or more of these pipeline optimizations, you'll find that the flow through your release pipeline will get even more compressed than the simplified pipeline described above and will start to look more like the optimized release pipeline pictured below (see Figure 6).

Figure 6 An Optimized Release Pipeline

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

58422

Source: Forrester Research, Inc.

Next Practice No. 4: Architect Software For Rapid Change

One of the biggest problems release managers face is that developers often don't think much about how they can build systems that are easier to deploy. But with a few tweaks, it's possible to make it much easier to deploy software when it changes. To enable rapid change, adopt these software architecture practices:

- Modularize your architecture.** Monolithic systems are the bane of the null release cycle, but with an injection of modularity and some sound architectural principles, you can reduce the level of coupling between different systems. For example, designing with application programming interfaces (APIs) and façades that hide system internals creates well-controlled access pathways that provide a convenient place to connect a test harness. And if you have six or seven completely separate subsystems and only one of them changes, then you can concentrate your testing against the API for the module that changed. A modular architecture also lets you

compare metrics like code churn or defect density between different subsystems. Based on this data you can determine how much remedial effort to expend on different parts of the system for each individual release based on objective measures.

A good example of modularization is the Eclipse Foundation's code base. We've written about its release train process before, but one key to its success is a layered dependency model that works because of well-established APIs and a commitment to an integrated build that's rarely broken.³ Individual teams rely on core level-0 projects that stabilize early and then level-1 projects that stabilize next before level-2 projects. The modular architecture also means that individual teams can opt out of a release if necessary when they are in the outer ring. They also know details of APIs well in advance of a release candidate or milestone build.

- **Manage deployment by configuration.** A modular application is a set of software artifacts that raise a set of dependencies on each other and on a set of hardware resources (virtualized or real). Release managers can express this combined set of artifacts as a configuration and can model the configuration the same way architects model software or enterprise architecture. An emerging set of open source and commercial tools like Chef, Cfengine, Puppet, rPath, Nolio ASAP, and Serena Application Release Management can capture and deploy software based on configuration models instead of Byzantine deployment scripts.

Implemented properly, configuration-based deployment allows organizations like Reliant Security to manage hundreds of unique customer configurations by treating each one as its own development artifact. First, configuration models are versioned, tested, and certified as ready to release. When customers want to move to a new release of Reliant Security's Redbox in-store system, the firm creates a new configuration from a known good release and certifies the entire image as ready for deployment into a PCI-certified environment. What Reliant Security has discovered is that many of the concepts it applies in its SCM repositories are equally useful to manage complex configuration trees, although the tools used are different (see Figure 7).

- **Design systems to support “blue-green” deployment.** One of the most important points in the release process is final deployment into production. Jez Humble and Dave Farley describe one trick that shops use to reduce their risk of a bad deployment in their book, *Continuous Delivery*. Shops alternate deployment to two different but highly similar production environments (one “blue,” the other “green”). If green is the current environment, then changes are made to blue, and when the cutover occurs, a load balancer or router reroutes traffic to blue. If something goes wrong, administrators can quickly cut back to green. If the deployment is successful then green becomes the new staging environment.

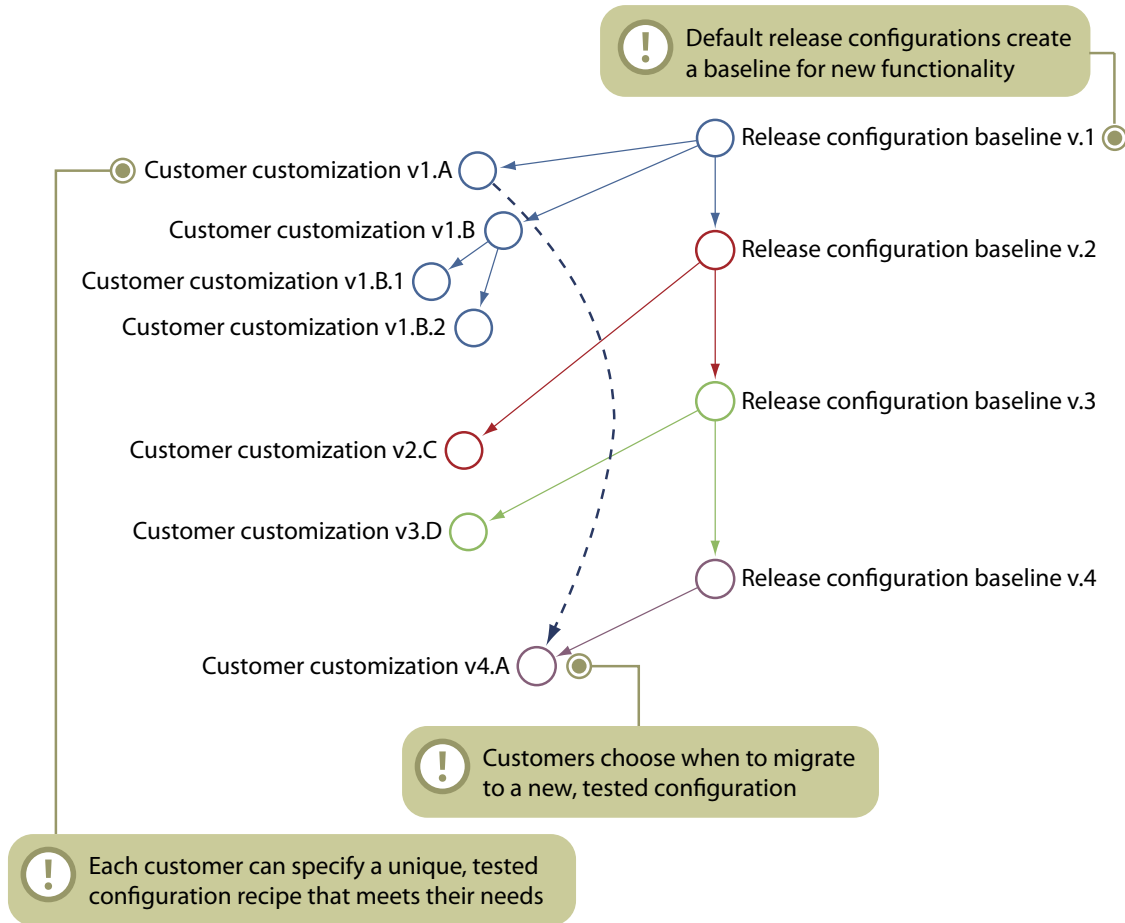
You might think that “blue-green” deployment is unworkable because it would double your hardware costs, but that's not necessarily the case. First, it doesn't have to be a monolithic cutover, it can be a more gradual spin-up and spin-down of individual servers or even

application tiers or customer segments. You may also already have excess capacity on standby that you can retarget (like that robust system testing instance or disaster recovery environment you use once a year). Blue-green deployment works especially well in scale-out or elastic environments like the cloud.

- **Use runtime configurations of features to avoid rollbacks.** If you've ever wondered what it's like to deploy multiple times in a single day then you should study the mechanisms that Flickr.com uses to encourage cooperation between developers and operations.⁴ In addition to many of the practices mentioned above, Flickr uses the concept of "feature flags" that allow individual versions of features to be turned on (or off) via the release configuration file. The nice thing about this runtime configuration capability is that release managers can turn a feature on (or off) without requiring a new build and a subsequent release. More importantly, it eliminates the need to "roll back" a failed release. Operations can simply edit the configuration file to revert to the previous feature version, in effect containing the failure by "rolling forward."

These architectural changes are not quick hits for an existing code base — they take a good deal of time and cultural change to get right. We recommend that you concentrate on some of the next practices described prior to this one first before you tackle the significant code refactoring and process changes required to adapt an existing software architecture for rapid change, unless you have already implemented those practices and are ready to step up to the next level. However, if you're lucky enough to be starting a new project from scratch, think hard about adopting some of these architectural practices right away — your business sponsors will thank you for it!

Figure 7 Reliant Security Uses A “Leaf-Node” Configuration Model



58422

Source: Forrester Research, Inc.

Next Practice No. 5: Create A Common Release Portal

The information that provides visibility into the status of any given release is scattered among an overwhelming number of tools. Centralizing this information is the final, but perhaps most important, next practice that release management teams should implement. Centralize this information in a portal that unifies sensor data from portfolio management, build, SCM, testing, and deployment tools. To achieve high release transparency, your release portal should include:

- **The status of your release pipeline.** Your release portal should make it clear what builds are in flight, who initiated them, and what stage of the release process they have reached. Indicate builds that have failed and make it clear that they have, and why. Red and green lights and colored text are traditional, but icons are even better as they also work for color-blind practitioners. A well-designed portal will allow stakeholders to get a quick view of how things are going at a glance. Consider making this top-level view into a wall-board information radiator that's easily viewable by anyone (and provide an online equivalent for stakeholders who are remote).⁵
- **Drilldown metrics on individual releases.** For each release, project members should be able to navigate to detailed metrics about the release. Be sure to include test coverage (it's a good predictor of release quality), tasks delivered (which answers questions like "What did we do?" or "When will this defect be fixed?"), and time spent in each phase of the release (which helps identify problem areas in the release process).
- **Consistent locations and links for "latest releases."** Many team members won't really care much about the health of the release process — they just want the latest, greatest release for testing or to give to customers for a private beta. The release portal should make it clear what the best releases are, and users should be able to bookmark the location once to return to whenever needed. Some teams go even further and provide consistent links to multiple type of releases with labels like "latest stable release," "latest milestone release," or "latest nightly build."
- **Links to all configuration recipes and instances.** Version and store each successive release configuration so that you can replicate it at any point in the future, especially when a team is supporting multiple releases. Organizations that work in highly regulated environments would also be wise to store logs and outputs associated with release instances linked to the configurations used to create them. This allows developers and regulators to build a particular release at a later date and compare the outputs to ensure consistent execution and outputs.

RECOMMENDATIONS

BETTER RELEASE MANAGEMENT IS NOT BEYOND YOUR REACH

If you look at all the potential work you have facing you in the release management area, it's tempting to turn away and go back to more enjoyable strategic initiatives like building mobile apps or increasing the number of certified Scrum masters on your payroll. But don't look at the release process like the development equivalent of cleaning out the Augean stables. Instead, think of it as a target-rich environment that is creating waste and sapping productivity, and as a long-term process improvement project. Take small steps, and give yourself time to measure your progress so you don't get overwhelmed. You should:

- **Start at the beginning of your release pipeline for greatest effect.** If you've already invested in continuous integration then it makes sense to extend those efforts downstream into release management. By integrating continuous integration into software change and configuration management (SCCM), you provide a sound basis to gather metrics around the flow of change, and tracking this over time will let you know which tactics work for you and which provide no measurable benefits.
- **Find like-minded operations professionals and get them onboard.** Not all operations professionals like saying "no." In fact, a growing number of them are inspired by the Agile movement and are incorporating many of the tactics and principles discussed here into a body of work collectively called "dev-ops." Pay attention to this movement and see if any of your operations peers are already thinking along these lines. Maybe you should invite them to attend a dev-ops conference on your dime, or at least send them links to many of the excellent online presentations that describe how firms have implemented the next practices described here.
- **Admit to your own technical debt.** It's time to come clean — developers haven't always thought about how the decisions we make affect those who are further downstream in the release process. Instead of putting off testing to the end or setting up private builds, we can improve our own prerelease practices to make sure we put only clean, quality code into the release process. Start planning sprints that improve architecture and reduce technical debt that results in bad builds and bad releases. If you're already deep in debt you may need to allocate one sprint in four to reducing the debt — if not, it could be one sprint in six or eight. Implement a flexible, modular architecture, preflight builds and code scans, and feature flags, and watch your relationship with ops improve.

WHAT IT MEANS

BETTER RELEASE MANAGEMENT IS A LOGICAL NEXT STEP FOR AGILE DEV SHOPS

We've had repeated conversations with many of our clients about how they scale the success and increased productivity they have seen from implementing Agile processes on projects. One common next step is to invest in better demand management to make sure that Agile teams are building the right things. Better release management processes are another logical step, because a more transparent, flexible release environment allows development teams to release changes when the business needs them. In this sense, better release management is a precursor to transforming IT into BT. The best and brightest thinkers in release management are drawing inspiration from the Agile movement and applying it to infrastructure and operations. We expect to see significant growth in this dev-ops movement as the cloud, new tools, and new thinking create a perfect storm that shakes the traditional "us versus them" relationship that defines most IT shops today. In the end, the change will be as significant as the process transition from traditional development processes to Agile — get ahead of the curve now!

SUPPLEMENTAL MATERIAL

Methodology

Forrester fielded its Forrester/Dr. Dobb's Global Developer Technographics® Survey, Q3 2010 to 1,036 development professionals; however, only a portion of survey results are illustrated in this document. The respondents consist of volunteers who join on the basis of interest and familiarity with specific topics. For quality assurance, panelists are required to provide contact information and answer basic questions about their firms' revenue and budgets. Forrester fielded the survey from September to October 2010. Exact sample sizes are provided in this report on a question-by-question basis. Panels are not guaranteed to be representative of the population. Unless otherwise noted, statistical data is intended to be used for descriptive and not inferential purposes.

Forrester fielded its Forrester/Dr. Dobb's Global Developer Technographics Survey, Q3 2009 to 1,298 development professionals; however, only a portion of survey results are illustrated in this document. The respondents consist of volunteers who join on the basis of interest and familiarity with specific topics. For quality assurance, panelists are required to provide contact information and answer basic questions about their firms' revenue and budgets. Forrester fielded the survey in July 2009. Exact sample sizes are provided in this report on a question-by-question basis. Panels are not guaranteed to be representative of the population. Unless otherwise noted, statistical data is intended to be used for descriptive and not inferential purposes.

Forrester fielded its Q4 2010 Global Release Management Online Survey to 101 development professionals; however, only a portion of survey results are illustrated in this document. The respondents consist of volunteers who join on the basis of interest and familiarity with specific

topics. For quality assurance, panelists are required to provide contact information and answer basic questions about their firms' revenue and budgets. Forrester fielded the survey from October to November 2010. Exact sample sizes are provided in this report on a question-by-question basis. Panels are not guaranteed to be representative of the population. Unless otherwise noted, statistical data is intended to be used for descriptive and not inferential purposes.

ENDNOTES

- ¹ Jez Humble and David Farley lay out the concept of release pipelines in detail in their excellent (and extensive) book on improving release management. Source: Jez Humble and David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley, 2010.
- ² The presentation discusses data security trends and drivers; current data security challenges; why data masking is critical; data masking approaches; selection criteria for data masking implementation; and recommendations. See the July 9, 2009, "Why Data Masking Should Be Part Of Your Enterprise Data Security Practice" report.
- ³ For a more in-depth look at how the Eclipse release process works, see the December 13, 2007, "Case Study: Eclipse Convinces Its Projects To Board A Single Release Train" report.
- ⁴ You can catch an excellent version of the culture at Flickr.com from the Velocity 2009 presentation. Source: John Allspaw and Paul Hammond, "10+ Deploys Per Day: Dev and Ops Cooperation at Flickr," O'Reilly Velocity Conference, 2009 (<http://velocityconference.blip.tv/file/2284377/>).
- ⁵ An information radiator is a large display of critical team information that is continuously updated and located in a spot where the team can see it constantly. For some creative examples to inspire your work, check out the Atlassian-sponsored Ultimate Wallboard site at <http://ultimatewallboard.com>.

FORRESTER®

Making Leaders Successful Every Day

Headquarters

Forrester Research, Inc.
400 Technology Square
Cambridge, MA 02139 USA
Tel: +1 617.613.6000
Fax: +1 617.613.5000
Email: forrester@forrester.com
Nasdaq symbol: FORR
www.forrester.com

Research and Sales Offices

Forrester has research centers and sales offices in more than 27 cities internationally, including Amsterdam; Cambridge, Mass.; Dallas; Dubai; Foster City, Calif.; Frankfurt; London; Madrid; Sydney; Tel Aviv; and Toronto.

For a complete list of worldwide locations visit www.forrester.com/about.

For information on hard-copy or electronic reprints, please contact Client Support at +1 866.367.7378, +1 617.613.5730, or clientsupport@forrester.com.

We offer quantity discounts and special pricing for academic and nonprofit institutions.

Forrester Research, Inc. (Nasdaq: FORR) is an independent research company that provides pragmatic and forward-thinking advice to global leaders in business and technology. Forrester works with professionals in 19 key roles at major companies providing proprietary research, customer insight, consulting, events, and peer-to-peer executive programs. For more than 27 years, Forrester has been making IT, marketing, and technology industry leaders successful every day. For more information, visit www.forrester.com.