BCG

The Boston Consulting Group

GOING ALL IN WITH DEVOPS

By Andrew Agerbak, Kaj Burchardi, Steven Kok, Fabrice Lebegue, and Christian N. Schmid

For too many companies, moving to agile software development is like finding the perfect new strain of grass seed—after months of searching—and then planting the new seeds in your old backyard. Your lawn may ultimately look a little better, but it will take longer and the results won't be as great as they would be if you had first removed the hidden tree roots, put in new soil, and rethought the irrigation system.

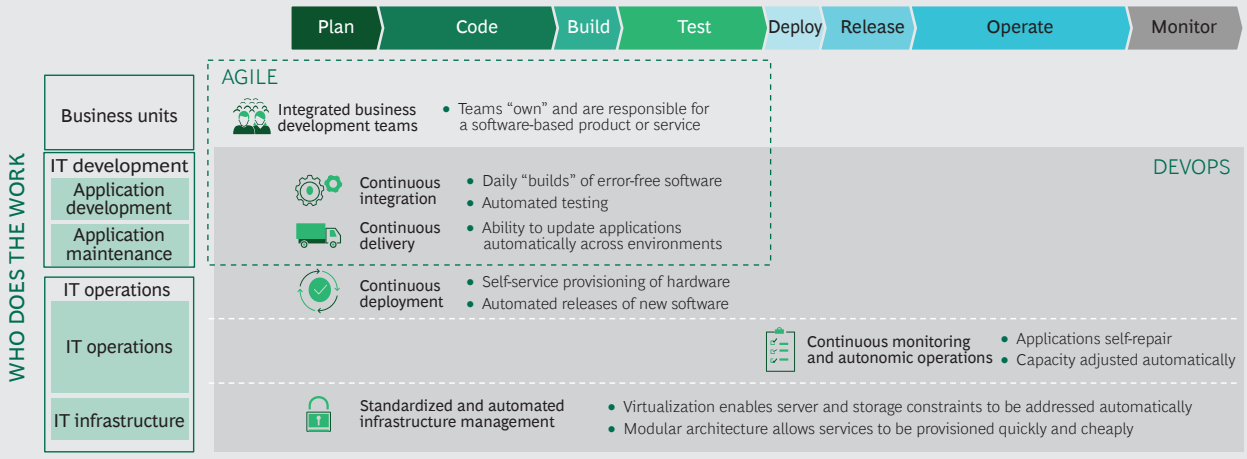
Many mainstream companies—in financial services, health care, manufacturing, consumer packaged goods, and other industries—have felt compelled to give agile software development a try. And they have waited expectantly for the benefits. In theory, agile's assignment of a business leader to development teams ensures that the most important software changes get done first, and its emphasis on short coding sprints ensures that the changes are implemented quickly. The fact that the quick part doesn't always happen has been discouraging. It has some agile newcomers wondering if there's something they're missing.

In many cases, there is. Agile does a good job of breaking down silos early in the software development process. But it can achieve only so much on its own. To turn themselves into digitally ready competitors, companies have to rethink the *entire* software development life cycle. They need agile, but they also need DevOps.

DevOps is an approach that integrates critical late-stage activities—like testing and deployment planning—into the code-writing part of software development. (See Exhibit 1.) With its emphasis on running multiple activities in parallel and on multifunctional teams, DevOps represents a break from the old "waterfall" model, in which planning, writing, testing, and deploying code were discrete steps managed by separate departments.

While many software companies use some form of the continuous software development and release that are the hallmark of DevOps, the approach (which continues to evolve and is starting to be referred to as DevOps 2.0 or BizDevOps by some of its

EXHIBIT 1 | Where Agile and DevOps Fit in the Software Development Life Cycle



Source: BCG analysis.

more advanced practitioners) is a lot newer outside the technology and internet services industries. Some traditional companies, notably in financial services, have some DevOps pieces in place, such as automated testing and mechanisms for provisioning hardware quickly. (See "Leaner, Faster, and Better with DevOps," BCG article, March 2017.) But the scope of these practices is limited. The companies haven't made the overarching changes that would allow them to capture DevOps' full range of benefits.

Making DevOps Work

To get the most out of DevOps, companies must make changes in controls and governance, IT organization roles, and operating models.

Rethink controls and governance. Most big companies' approach to developing and releasing software reflects controls they put in place years ago to maintain quality and avoid costly mistakes. The controls may have made sense at the outset. But as the pace of technological change has accelerated, the controls have lost their relevance. Now they're just obstacles.

For instance, a control about infrastructure provisioning—one of the hurdles that must be surmounted before a development team can begin its work—may have been implemented in the days before virtualization. Today, virtualization makes computing and storage capacity available with less operational complexity than before and at far lower cost. But we still see companies taking a month and 50-plus emails just to provision hardware and gather all the necessary permissions.

Likewise, a control requiring multiple go-live approvals for new software may have been justified when there were only a few software updates a year and each one involved a critical part of a monolithic system. But it shouldn't require two dozen people to approve a minor tweak—like changing the color of the screen users see.

With software now a key way of addressing fast-changing business and customer needs, the prolonged delays caused by controls that have become irrelevant put companies at a fundamental competitive disadvantage. The delays can pose a reputational risk and even a survival risk if a company is the target of a cyberattack. (See "Develop a Cybersecurity Strategy as If Your Organization's Existence Depends on It," October 2017.)

Governance is another area that requires adjustments in the move to DevOps. This includes adopting new approaches to funding. In agile, funding isn't allocated on a project basis: for a set period of time, against a defined set of deliverables. In-

stead, funding is allocated to critical "products"—like a mutual fund company's "my account" function or a retailer's order-and-ship system—that require the attention of teams for many months or even years. DevOps adds complexity by forcing companies to figure out how to allocate some application maintenance and infrastructure costs within the product funding paradigm.

Another area where DevOps should trigger a governance change is in the decision rights related to cloud solutions. In the past, these decision rights belonged to the IT organization, and no one questioned that. But that's changing as more business units create software directly using cloud-based tools.

We saw questions about such decision rights at a company where digital product development teams were pushing for direct access to an Amazon Web Services account, and the IT operations group, concerned about standardization and security, was resisting. In truth, there is no single right answer to the question of where the decision rights should lie, for this company or any other. But it is an issue that must be tackled in DevOps, which redraws the boundaries of software development along multiple dimensions.

DevOps should also prompt a change in how companies deal with buggy software. At companies that haven't fully adopted DevOps, there isn't a "you built it, you own it" governance philosophy. Instead, if an issue arises involving software that has been released, IT support teams report it through a ticket system (such as ServiceNow), and the issue becomes the responsibility of an application maintenance team. But the original developer of the code, having gotten wind of a problem, may go back in and try to fix it. The net result is that sometimes both the development and maintenance teams end up working on the same software, at the same time, resulting in inconsistencies, integration problems, and stability issues. By contrast, at companies that adopt DevOps practices, issues with released software automatically register on the backlog of the development teams, which are expected to make the fixes. There is no one farther down the line who would even think of fixing the code, and no possibility of different departments touching the same code simultaneously and working at cross-purposes.

Ultimately, the best test of governance practices is cycle time. If companies can substantially reduce the time between when they plan software and when they release it in a reliable, high-quality form, that is a sign that their governance processes are working and that they have the technical capabilities they need.

Redefine the role of the CIO and the IT organization. If DevOps is to succeed, there must be changes—some subtle, some more dramatic—in the role of the chief information officer and the information technology organization. In companies that adopt agile models, the specifications for new software—and the coding work itself—become the implicit responsibility of business units. If this relieves the CIO of responsibility for individual lines of code, in most cases he or she still shoulders the larger burden of quality. That is, the CIO must still recruit and train software developers. He or she must also put in place a better delivery model, one that includes an operating environment—standards, services, processes, tools, and infrastructure—that allows developers to maximize their productivity. The CIO must also front-load more activities in the software development life cycle.

The term of art for this sort of front-loading is the "shift left," referring to how one would diagram various activities on a software development life cycle chart. In DevOps, technical staff that would once have sat in the IT operations function—whose work kicks in later—are moved into the product development teams, where they have a say in how the code is built. There should also be input early on from those responsible for a company's data architecture and cybersecurity. The shift left of activity and expertise is one way all the code that's being created—often in many different business units—can get to market quickly and with the necessary level of security.

A particularly important CIO responsibility with DevOps is the implementation of an optimal infrastructure environment. The business-oriented development teams need infrastructure-independent platforms so that they don't have to worry about compatibility. In DevOps, managing this and providing the application development toolkit are significant parts of the IT organization's responsibility.

Netflix, the global streaming video service, provides an example of the kind of benefits that can come from embracing DevOps. Netflix captures these benefits through the efforts of a central engineering operations group (a sort of specialty IT team) whose mandate is to maximize the performance of newly released software and to make software development teams more efficient.

At Netflix, developers benefit from a common set of tools, services, and infrastructure management capabilities—a "paved road," as Netflix calls it—to traverse the normally bumpy path to new software creation. The paved road and the engineering operations group have been instrumental in helping Netflix release new code—secure, reliable code—to multiple geographic regions within minutes.

Assisting and speeding up software deployments in this way require IT staff to develop skills they didn't need previously. For instance, enterprise architects must take a much stronger hand in defining IT architecture strategy, especially with respect to platform options. And IT organizations must adopt technical mechanisms—like containers and microservices—that allow coding teams to write reusable software and to do it faster. (See the sidebar "DevOps' Technical Underpinnings.")

IT organizations must also acquire some brand-new technical capabilities. For instance, they must hire or develop quality engineers. These engineers should be embedded in the software development team and should ensure that rigorous testing happens early in the process. The IT operations staff must likewise acquire or develop new expertise, such as reliability engineering and infrastructure service development. Without these capabilities, continuous delivery and continuous integration aren't possible, making agile's promised speed and reliability benefits hard to achieve.

Remake the operating model through automation. There is a huge benefit if a team, instead of going through a cumbersome approval process that might last weeks or months, can add a feature or plug a dangerous security hole with relatively little organizational oversight, and in the best case with just a few mouse clicks. Automation, one of the pillars of DevOps, makes that possible. But the decisions surrounding automation are complicated, and there is always the chance that a company will take a while to gain its footing. For this reason, the where and how of introducing automation is a key decision for any company moving to DevOps.

A good place to start is with test automation. In our experience, the benefits of covering more new code through automated testing can on its own justify a move to DevOps. Consider the ever-present risk of late-stage delays and the costs they create. With traditional waterfall and even sometimes with agile development, testing takes place once the code is complete. Significant problems may be discovered just as the code is supposed to go live. By contrast, in the DevOps paradigm, code is developed iteratively and tested regularly. This makes it less likely that coding issues will emerge at the last minute. (See Exhibit 2.)

Valuable as it is, automated testing must be rolled out in stages. Companies should start with the parts of their architecture where they have already started to transition to agile models. After they've had some success, they can use automation to cover more of their code.

Some of the companies that have set the pace in digital services, such as Google, have reliability targets well above 99%—meaning that they expect the software they release, with the help of automated testing, to work immediately and in pretty much

DEVOPS' TECHNICAL UNDERPINNINGS

IT staff must be familiar with various technical tools and approaches in order to implement DevOps. Here are seven of the most important.

Containers. A type of virtualization that keeps software running reliably when it is moved from one computing environment to another. By bundling new code with everything needed to run it, a container makes it possible for software development teams to ignore differences in operating systems and underlying infrastructure. Two open-source technologies that help with containerization are Docker and Kubernetes.

Microservices. A programming architecture that gives developers access to application functionality at a very granular level. Microservices make it easier to continually deliver and deploy large, complex applications.

Code Repository. A database containing the source code of an application. When centralized and actively managed, code repositories improve the consistency and stability of code, and help avoid version control issues. Among the open-source tools used for code repositories are Bitbucket and GitLab.

Continuous Integration. A development practice that promotes single-source code management and comprehensive automated testing, allowing developers to add code to a common repository as often as several times a day, in a highly automated way. This ensures that all development teams are using the latest version of an application. Open-source versions include GitLab CI and Jenkins.

Continuous Delivery. A discipline for building software that enables the software to be moved to a staging area at any time. Continuous delivery tools include Bamboo and Jenkins.

Continuous Deployment. A practice that allows tested software to be released, sometimes with not much more than the push of a button. Continuous deployment sharply reduces overhead and can be an invaluable tool for resolving issues quickly.

all instances. Google, of course, was built to enable rapid software releases and service improvements. Non–digital natives don't need to ensure reliability on the same scale, but when it comes to digital services they can learn from Google and other digitally advanced companies—and they must. After all, a traditional company with a mission-critical digital application—like a financial services company rolling out a smartphone payment feature—can no more afford to release bad software than Google can.

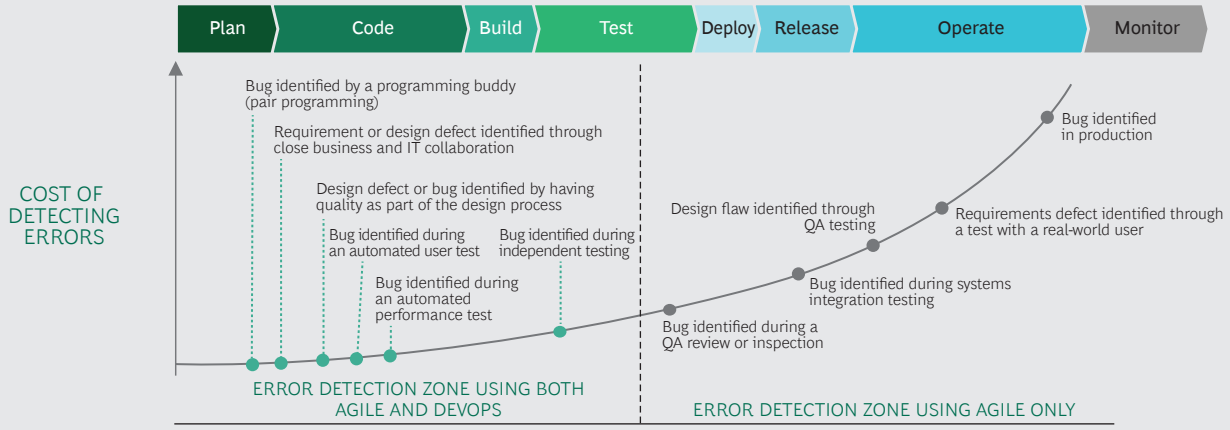
With traditional companies' legacy systems, such as payroll or enterprise resource planning, the dynamics are necessarily a little different. Companies can still do automated testing of their legacy systems, and in many cases they already do. But in order to support the faster release cycles agile development teams expect, the tests should be synchronized with batch processes, including prescheduled data transfers and transactions. Since batch processes are often designed to take place overnight, the IT organization may want to run the automated tests overnight, too.

Getting Started

Companies can't just brush aside their current software development practices and make a wholesale move to DevOps; it involves too much change and training and would create too much disruption with existing systems and products. DevOps needs to be phased in.

The first step should be to find an application that has low levels of dependency

Exhibit 2 | With DevOps, Bugs Are Spotted Earlier and Fixed More Economically



Sources: Puppet, *2016 State of DevOps Report;* BCG analysis.

with other applications—perhaps a procurement portal for a manufacturing company or a savings platform for a bank—and run a pilot project to learn the DevOps model and fine-tune the practices.

In the pilot, a team of developers and IT engineers lays out a technical plan—establishing a central code repository and creating a testing framework so testing automation can start. Once this is in place, continuous integration and continuous delivery can begin. These processes make it possible for the development team to focus on writing code and not on manually checking for bugs and functionality problems.

At companies with complex legacy systems, continuous integration and continuous delivery are two separate phases. By contrast, at digital natives, both approaches are core to software development, and a digital native may already be thinking about other ways of enhancing the software release process. This explains why developers at the most digitally adept companies often see their code fixes go live within days, hours, or even minutes.

The DevOps pilot needn't go on indefinitely. Within six months, it should be possible to see benefits. These typically take the form of agility, which translates into more software releases per week; quality, which stems from increased testing coverage; and efficiency, in the form of lower costs of rework and an overall increase in the number of automated processes. After an introductory period like this, the company can create a road map to start applying DevOps practices to other software and infrastructure platforms and to other parts of its technology environment. The road map should include a decision about the suite of tools to be used and the sequence in which DevOps will be implemented in other parts of the company and for other platforms.

DevOps and the Customer

The example of a European travel company helps demonstrate why DevOps is turning into a must-have.

The company was unable to make pricing updates to its core booking system at the height of its main selling season. Previous updates had exposed the fragility of the system, and business managers had imposed a policy of no changes during peak periods.

There was nothing unusual about the company's monolithic software infrastructure or the policies to accommodate it. However, the deliberate approach to software development had left the company unable to respond, at the most important time of year, to new pricing or product propositions from competitors. If a seven-day trip to Belize was suddenly being discounted to

$1,800 on other travel websites, it would still be going for $2,100 on the company's site. Dynamic pricing updates required a software change, but the company's release process limited the speed at which such changes could be made.

Recognizing that its software development processes were hurting the business, the company adopted some DevOps practices, including continuous integration. As it did so, the quality of its software releases and the overall resilience of its system improved to such an extent that management lifted the change freeze. Thereafter, the company was able to be much more responsive to competitors' moves during the industry's peak selling season.

Sooner or later, most companies are going to find themselves in a similar position. That is, they are going to see that one of their competitors is doing something faster, with fewer security and quality issues, and at lower cost. And they are going to need to take action to narrow the gap.

DevOps is a way to do this. The implementation of DevOps involves organization and process changes that take place well out of sight of most customers. But customers will be expecting the benefits. For companies that don't deliver, there may not be a second chance.

About the Authors

Andrew Agerbak is a director in the London office of The Boston Consulting Group. You may contact him by email at agerbak.andrew@bcg.com.

Kaj Burchardi is a managing director of BCG Platinion in Amsterdam. You may contact him by email at burchardi.kaj@bcgplatinion.com.

Steven Kok is a project leader in BCG's London office. You may contact him by email at kok.steven@bcg.com.

Fabrice Lebegue is a managing director of BCG Platinion in Montreal. You may contact him by email at lebegue.fabrice@bcgplatinion.com.

Christian N. Schmid is a principal in the firm's Munich office. You may contact him by email at schmid.c@bcg.com.