CHAPTER

DEVOPS AND CONTINUOUS DELIVERY

10

I've been studying high-performing IT organizations since 1999. What I've noticed is that there was this downward spiral that happens in almost every IT organization. It led to more fragile applications in production, longer deployment times, building up of technical debt, and the business goes slower and slower. I think one of the reasons that I cared so much about this is that it led to preordained failure. Where people felt, especially downstream (operations, test, security), trapped in a system where we were powerless to change the outcomes. No matter what we did, failure has been preordained. I think the way out of the downward spiral is what organizations like Google, Amazon, Twitter, LinkedIn, and GitHub are doing. These organizations are doing tens, hundreds, or even thousands of deploys a day. And this in a world where most of us are stuck with one deploy every nine months with catastrophic outcomes.

Gene Kim, coauthor of The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win and the upcoming DevOps Handbook.

INTRODUCTION

Software drives business and today, businesses have a "need for speed." The speed of a proprietary trading system gives a financial services firm a split second cost advantage over competitors. Routing algorithms enable a package delivery firm to deliver faster and increase revenue by shaving fuel costs. A social media firm can continuously tweak its website to engage visitors longer and sell more add-on services—and it is always a race to translate new ideas into code faster than competitors.

In each of these examples, software correlates directly with revenue. And in each of these companies, **Development** and **Operations** teams are under the gun to deliver software faster, more efficiently, and at higher levels of quality. Today, IT is finally being perceived as a revenue producer versus simply a cost center, and virtually every IT organization has the opportunity to directly impact business success. Far from being "the guys in the back room who run the financial system," as they used to be viewed, IT professionals are now responsible for architecting, building, and delivering one of the most valuable assets of any company—software.

However, taking advantage of this opportunity requires the ability to accelerate the delivery of infrastructure and **applications**. People, processes, and tools all impact the delivery, and to some degree flexibility and risk tolerance are part of the picture as well.

Traditionally, IT Operations teams in particular had a vested interest in maintaining the status quo; change has long been the enemy because modifications to IT systems often caused production problems. IT organizations in general still walk a tightrope, balancing business demands for new services with the hard realities of day-to-day production support.

Application Performance Management (APM) in the Digital Enterprise. http://dx.doi.org/10.1016/B978-0-12-804018-8.00010-3

In psychology, an approach/avoidance conflict[1] arises when no choice is the perfect choice—in other words, every option generates both positive and negative results. "Can't win" situations result in high levels of inertia combined with high levels of stress. Yet modern IT organizations are confronted with the ultimate in dueling objectives: to keep production systems running flawlessly while absorbing constant and ever-increasing levels of potentially disruptive change.

Within this fast-moving and competitive business scenario, **DevOps** and **continuous delivery** are agents of change. When done in a disciplined manner, both can significantly reduce the risks associated with high rates of change while directly benefitting bottom line revenue. In an atmosphere of seemingly irreconcilable differences between the static and dynamic forces impacting every IT organization, both DevOps and continuous delivery are gaining traction.

Particularly when the two are viewed as linked versus disconnected processes, they have the potential not only to reconcile these opposing forces/outcomes, but also to fulfill the IT-driven business objectives so coveted in today's corporate environments.

AGILE DEVELOPMENT

To fully understand the emergence and value proposition of DevOps and continuous delivery, it is first necessary to understand the impact of **agile** practices on application lifecycle **management** and application delivery in general. Agile practices were designed, in part, to reduce the risks of the "catastrophic outcomes" Gene Kim describes in his quote at the beginning of this chapter. **Independent software vendors (ISVs)** of enterprise software such as **enterprise resource planning (ERP)** or **customer relationship management (CRM)** systems traditionally required between 9 and 18 months to develop a full software release. "Waterfall methodologies" delivered hundreds or thousands of features and packages in enormous bundles that could require weeks or months for customers to install. On the customer side, a tremendous amount of prep work went into the deployment of these packages and, once installed, the impact to production was often catastrophic.[2] In many cases, this meant that customers simply did not upgrade and remained on old software releases for years. This remains a persistent problem for virtually every ISV developing and delivering on-premise software.

Over time, agile development practices replaced waterfall methodologies as the new de facto standard for software development. The resulting software is more frequently useful and less frequently catastrophic than traditional software packages often were. There are multiple definitions of agile, but the key ideas listed in Table 10.1 seem to resonate across methodologies and practitioners.

As an example of an early application of agile practices, a well-known enterprise management ISV[3] began using agile techniques in 2006 to rearchitect and rewrite the products in its **business service management (BSM)** portfolio. The resulting products are simpler to deploy and manage, easier to

[1]Elements of stress introduced by social psychologist Kurt Lewin. Essentially, approach/avoidance conflicts are those choices in which the end result has both positive and negative characteristics.

[2]In fact, IT organizations still cite packaged applications as being more challenging to manage than even in-house developed custom software. *Automating for Digital Transformation: Tools-Driven DevOps and Continuous Software Delivery in the Enterprise.* Available at: www.emausa.com.

[3]BMC Software is headquartered in Houston, Texas.

Table 10.1 Characteristics of Agile Practices

Key Tenets of Agile Practices
Development of software in small increments via multiple iterative and incremental cycles
Flexibility to adapt and evolve requirements throughout the development cycle
Collaborative approach in which small teams (typically no more than 10 engineers) work closely together and meet frequently (typically for 15–20 minutes daily)
Continuous testing and ongoing integration of newly developed code
Stakeholder (such as customers or line of business project sponsors) involvement throughout the delivery process
Frequent software delivery, with stakeholder acceptance and signoff at project milestones

integrate to third-party vendor solutions, and easier to use than previous software versions. Delivery performance in terms of breadth of capability and time-to-market were also exceptional.

As a result of this initial success, the vendor was able to deliver new software to market two to three times faster than the industry norm, producing product releases in 4–5 month cycles, versus industry averages of 12–13 months for comparable releases.

Today, virtually every major ISV develops software in this way. For the vendor, not only is software delivered more quickly, but customers are more satisfied with the resulting product. It more often hits the mark because it is delivered within months of customer requests, versus the years required with traditional development methodologies. The collaborative, iterative nature of agile development practices means that customers have a far greater likelihood of getting features they actually want and need. In addition, delivery of smaller software packages makes the upgrade process less risky—customers can install new vendor releases in hours versus weeks or months.

Agile development has become the norm in enterprise IT as well, with more than 90% of companies leveraging agile techniques for at least some software projects. Going beyond software, however, it is also interesting to consider the applicability of agile practices to other areas of the business. From a big-picture perspective, the same techniques that can transform software development—historically a difficult and risky activity—can potentially empower other collaborative efforts within the business as well.

DEVOPS: "IT TAKES A VILLAGE"

INTRODUCTION

The adoption of agile techniques undoubtedly yields benefits, but it also has a dark side. As agile lifecycles accelerated, continuous delivery became the norm, increasing rates of production change and often adversely impacting production environments. Instead of deploying software once every 9 months, IT Operations teams were confronted with the need to deploy more frequently. Today, leading-edge companies are deploying small software packages hundreds or thousands of times per week; deploying monthly or more often is the industry norm.

To back up a bit, **change management** has always been an industrywide problem. For example, in 2007 most companies were still releasing software to production very slowly via waterfall development practices. Still, consulting teams reported extremely high rates of adverse impact from routine production changes, depending on the maturity of a given company and its change management processes. Highly

mature companies—those with structured, governed change control methodologies—reported that changes drove between 25% and 30% of production incidents.[4] Less mature companies—those with more of a Wild West approach to service management—reported that changes drove between 75% and 80% of production incidents.

Fast forward to 2016, an era in which some companies are making code changes thousands of times per week, and it is not surprising to see that the job of IT (delivering high-quality business applications) has become exponentially more difficult. Not only is software being delivered faster, software components are running on far more diverse infrastructure. A single transaction can span multiple languages and platforms, and the software elements themselves are becoming more granular. In this complex environment, the entire span of **application management** activities has become significantly more complex. At the same time, software applications must be designed, developed, deployed, architected, and monitored via an ongoing, never-ending lifecycle.

When issues occur, each issue must be traced to its root cause before it can be fixed. And while problem management and incident management were the traditional realm of IT service management practices, the reality is that modern applications are far too complicated to be supported by a single individual or silo team. In other words, "it takes a village" to support today's complex applications, and the name of that village is DevOps.

As agile became a mainstream standard practice for software delivery, DevOps became a standard term within the IT lexicon—and DevOps teams assumed a great deal of importance in terms of overall software quality. While there are as many definitions of the term as there are industry experts, a simple stripped-down definition is as follows: DevOps is a collaborative, team-based approach to software delivery leveraging specialists with cross-functional development and operations skills to address application-related issues.

Today, more than 80% of companies have such teams in place. Companies vary in terms of which stages of the lifecycle these teams support—in about 15% of companies they support the entire lifecycle, to a greater or lesser degree. In larger companies these are dedicated teams, while at smaller companies they are often composed of ad hoc groups of IT specialists skilled in some aspect of application support. The names of these teams also vary from company to company. They are known as DevOps teams in about 25% of companies, Application Management teams in about 30%, and Infrastructure Services in about 20%.

Regardless, the teams are virtually always composed of senior IT specialists who are experienced in multiple software and infrastructure disciplines. At minimum, these teams typically include personnel with development and coding skills, as well as experts in network, systems, database, **integration**, and similar related areas. Their role is to span the technical silos, which, of necessity, exist within virtually every IT organization of size, and elevate the support function to an application versus infrastructure focus. In other words, they take a top-down approach to the support function—supporting applications from an end-to-end standpoint—versus a bottom-up, infrastructure-centric approach.

By collaborating and pooling their skills, these teams are empowered to support complex deployments, to deploy appropriate monitoring, and to troubleshoot the complex issues that arise in modern production environments. Collectively, they are far more efficient and powerful than would be the case if each worked separately at the silo level.

[4]All research numbers quoted in this chapter are from survey-based research conducted by Enterprise Management Associates.

IMPLEMENTATION

Although there are as many proposed approaches to DevOps as there are DevOps practitioners, Enterprise Management Associates (EMA) experts believe that the best approach is to apply cross-functional practices across the application lifecycle, if at all possible. It is also the case that automation is a key enabler for cross-functional support. Automating across the lifecycle while supporting automation with cross-functional skills and practices enables IT organizations to adopt a more nimble and iterative approach to application design, development, delivery, and support.

This view of DevOps also actively involves business stakeholders just as agile development practices do. **Line of business (LOB)** involvement helps ensure that the product delivered meets the needs of the business and that any requirements and modifications are incorporated into the software at the earliest possible stage in the lifecycle. Research demonstrates that software is far easier and cheaper to fix at earlier stages of the lifecycle than it is at later stages.

There are other factors driving the expansion of DevOps practices to a more lifecycle–centric approach. They include the following:

1. Iterative development impacting the entire lifecycle (not just deployment): The broad adoption of agile methodologies in the enterprise means that the deployment stage transitioned from being a point-in-time turnover of responsibilities to an ongoing process. Both Development and Operations are feeling the impact of this transition. Development must find ways to build and test code incrementally and iteratively, and Operations must find ways to engineer production environments capable of absorbing high rates of change. Neither group can do this without the skills of the other.
2. Abstraction of applications from hardware impedes manageability: **Cloud**, mobile, **sensor systems**, **virtualization**, and **containers** are freeing applications from the bonds of the physical data center. While moving applications and/or application components to the cloud and to consumers, retail facilities, and even train tracks can provide significant business benefits, the abstraction introduced by off-premise hosting of any sort makes applications more complex to manage, govern, and troubleshoot. When applications or their components are externally hosted, IT loses the visibility and control necessary to effectively monitor and manage the ecosystem. At the same time, IT still has responsibility for all aspects of application delivery including performance, availability, and troubleshooting. This means that **performance management** must often be done at the network level, since the network is the one common factor across these diverse components.
3. Application heterogeneity: From multibrand server farms to server clusters, load balancers, web servers, and heterogeneous databases, even applications hosted 100% on premise are so complex that supporting them requires a broad arsenal of skills. This is true across the entire six-stage application lifecycle, making it imperative that DevOps is approached as a lifecycle versus a point-in-time handoff.
4. Automated deployment: With the growth of highly virtualized environments deployed as both **private cloud** and public **infrastructure as a service**, software deployment is becoming increasingly **automated** and **metadata**-driven. However, as functionality previously supported by runbooks and manual job submissions is automated, the primary role of IT—to deliver quality applications to the business—remains the same. Many CIOs are seeking new and better ways to deliver on this mandate, and DevOps supported by automation is a proven approach for doing so.

5. Increased focus on LOB as part of the application lifecycle: A key element of agile methodologies is the involvement of LOB throughout the application lifecycle. From requirements to iterative checkpoints to acceptance testing, business stakeholders are more involved than ever before. Traditional development practices virtually ignore this group, which is becoming increasingly indispensable to customer satisfaction and project success. The DevOps lifecycle, shown in Fig. 10.1, includes LOB as a third stakeholder that assumes key leadership roles at given stages of the lifecycle.

DEVOPS ACROSS THE LIFECYCLE

The lifecycle model shown in Fig. 10.1 has two notable differences compared to DevOps models addressing a subset of the application lifecycle. Not only are DevOps concepts extended across the lifecycle, but Development, Operations, and business stakeholders are involved at every stage. This model of DevOps addresses the complexities associated with supporting today's dynamically changing and business-critical software ecosystems.

While all three stakeholder roles work together to some degree at each stage, the lead role(s) (underlined in the diagram) change at each stage based on changing deliverables and objectives. Collaboration takes the form of cross-functional cooperative efforts, and leadership handoffs replace a full-fledged

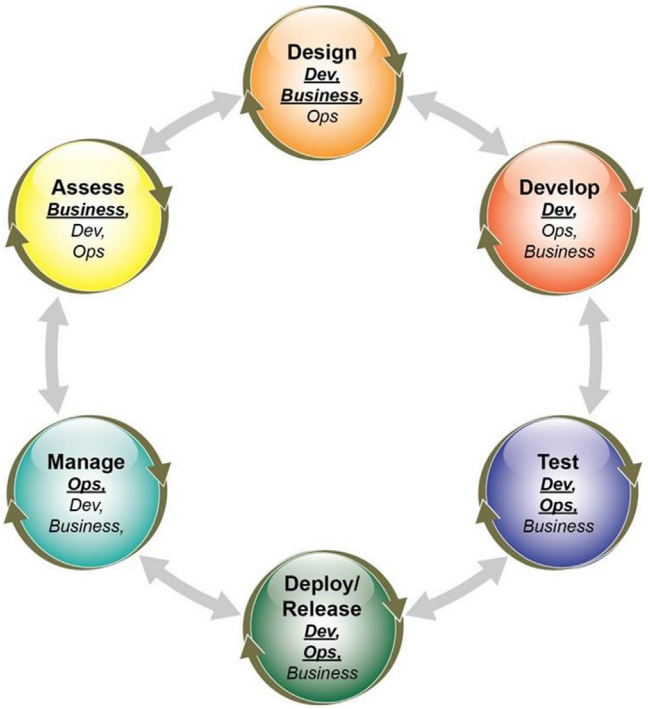

FIGURE 10.1

DevOps across the application lifecycle.

relinquishment of responsibility as software progresses through the stages. In addition, via automation, each stage of the lifecycle ideally shares relevant data with the previous and following stages, as depicted via the gray arrows between the stages.

In this model, the DevOps interactions at each stage are as follows:

1. Assess: Input from business stakeholders (for new services) and **service-level agreement** assessments (for existing services) is used to determine priorities for new business services and desired modifications to existing ones. Development and Operations stay in the loop because such improvements require new code, modified code, and changes to operational infrastructure.
2. Design: Development and Business take the lead in this stage, with Operations available to provide input as needed. Development has primary responsibility for incorporating requirements into a software design. Business has responsibility for educating Development about specific requirements, as well as for reviewing and signing off on the design. The role of Operations during this stage is to evaluate anticipated impact on production systems and assess any requirements for infrastructure acquisitions and/or enhancements.
3. Develop: Development is the lead during this stage and is ultimately responsible for building software that meets the needs of the business. Particularly in agile shops, business stakeholders perform functional reviews and sign off on application functionality on an iterative and ongoing basis. Operations is brought in as needed to support development teams in building development and testing environments and/or to be advised of infrastructure **configurations** and anticipated delivery timeframes.
4. Test: Development [including **quality assurance (QA)**] and operations are the leads during this stage, as final preparations are made for unit testing, integration testing, and release to production. While Development and QA teams perform unit and integration testing, Operations participates in integration and load testing to assess operational readiness. Acceptance testing becomes a critical role for business stakeholders, and all three groups must collaborate to agree on a final go-live plan.
5. Deploy/Release: This is the traditional DevOps handoff stage, but in this scenario, the handoff is a change in lead roles versus a turnover of responsibility. Development and Operations (or DevOps) teams lead this stage, while business stakeholders conduct final user acceptance processes.
6. Manage: During this stage, infrastructure, systems, and application management tools monitor production environments and applications. **Service-level management (SLM)**, performance and availability management, troubleshooting/root-cause analysis, and **capacity management** solutions all monitor and measure **application performance** as part of ongoing assessments. The resulting metrics can then be iteratively pumped back into the Assess stage (Stage 1) to ensure that the current state of an application is known and monitored in preparation for any modifications that may occur over time. In addition, DevOps teams are typically responsible for monitoring, managing, and troubleshooting the overall health of the application, and for working with Ops to see that production issues are fixed.

DEVOPS TOOLING: BRIDGING DIVERSE TASKS, GROUPS, AND SKILLS

EMA research has shown that, for applications written in-house, Development or DevOps groups remain in the loop in terms of application support for the life of the application. This is particularly true

for custom applications written in-house versus packaged applications purchased from an external vendor. Cross-functional collaboration becomes critical because this is where applications actually touch end users and impact the business.

At the same time, production application quality depends on far more than manual support or efficient code. Approaching DevOps as a lifecycle has significant implications in terms of tool design, choices, and options. As DevOps progresses from being a point-in-time handoff to an ongoing process of creating, deploying, and supporting business software assets, tools become a primary factor unifying lifecycle stages, roles, and leadership changes.

The tooling implications are significant and center on several key areas:

- Integration: Approaching DevOps as a lifecycle requires information sharing across discrete lifecycle stages and toolsets. Tools **interoperability** is a unifying force across diverse teams, skills, technology languages, and methodologies. Modern applications don't exist as siloed entities. Instead, they exist as distributed ecosystems that execute across a host of technology elements, all of which must efficiently interoperate for an end-to-end application to become a reality. In the same way, tools must support the fact that technology silos and lifecycle stages don't live in a stand-alone world. They are a product of a multistage lifecycle continuum and an end-to-end execution environment. While each stage brings with it specialized toolsets to support its own internal lifecycle, these tools must also interoperate to support seamless collaboration across stages.
- Workflow: As applications themselves become increasingly complex, building, deploying, and managing them becomes more complex as well. Workflow management is essential to governance and traceability, particularly since most IT processes combine human and automated tasks. One of the biggest challenges facing today's technology teams is managing the orchestration of complex, multistep processes in context with manual tasks, automated steps, and reviews/approvals. This is particularly true for cases in which multiple projects, tasks, or deliverables have dependencies on others, and where multiple projects (and deployments) are being completed in parallel. In such environments, the sheer effort of keeping track of bottlenecks, work queues, point-in-time responsibilities, and similar factors can be gargantuan. This, combined with the touch points across stages that lifecycle DevOps entails, makes **application lifecycle management (ALM)**, **workflow automation**, **release automation**, and **business process automation** tools essential elements for large-scale automation of the application lifecycle.
- Cross–lifecycle Service Quality Supported by Tooling: Research studies have repeatedly found that the best way to reduce the cost of supporting applications is to find and fix flaws early in the lifecycle. A famous **National Institute of Standards and Technology (NIST)** study found that if it costs x to fix an application problem during the design stage, making the same fix during production will cost between 470x and 880x. If the fix requires an engineering change, the cost could be as high as 2900x.[5] The clear implication is that testing early, often, and iteratively is a far more cost-effective strategy than waiting to find problems in production. In terms of DevOps, this means moving application viability testing to a point far earlier in the lifecycle

[5]The Economic Impacts of Inadequate Infrastructure for Software Testing, www.nist.gov/director/planning/upload/report02-3.pdf.

(often referred to as "shifting left"). It also reinforces the value proposition of application performance management (APM) and **user experience management (UEM)** tools. Such tools can be used during preproduction (as well as during production) to enable IT specialists to better understand the projected impact of a software release once it hits production.

Service virtualization solutions are important to this process as well. These are a new class of tools supporting integration testing, which has traditionally been shortchanged. Traditionally, integration testing was done by either testing against stub programs, essentially dummy programs leading nowhere, or testing against expensive duplicates of production environments. While the latter was preferable, high costs limited access by developers and testers and therefore the frequency of test runs. For example, one CA Technologies (www.ca.com) customer was interviewed as part of a case study profiling the vendor's service virtualization solution. The client had already installed two integration testing environments at a cost of $4.5 million apiece. However, software releases were still being held up because testers and developers had to schedule testing time in advance, and available time slots were rapidly filled.

Faced with the prospect of building a third testing environment, the company sought alternatives and discovered that using service virtualization products, developers and testers could test as often as necessary against virtual models that realistically mimicked interactions with production systems. Software delivery times were accelerated and there was no longer a need for an additional $4.5 million testing environment.

Budget, Staffing, and Quality Efficiencies: Tools that enable data sharing across the lifecycle not only extend the value proposition of tools investments, they also empower technical personnel to move beyond the boundaries of their own skill sets. It is unreasonable to expect that developers will be operations experts, that operations personnel will be **security** experts, or security experts will understand how to install an **operating system** on a mainframe. As an alternative, integrated toolsets and workflow automation allow IT specialists to effectively collaborate with a common language and view of software ecosystems.

There are a number of tools-related capabilities that contribute to these positive outcomes. One is that IT specialists should be able to see the impact of their own silo on the quality of the application as a whole. Another is that IT specialists should be capable of collaborating in such a way that they can rapidly solve application-related problems in complex ecosystems. Finally, and potentially most important, is that tools should be unifying forces across silos instead of roadblocks to cross-functional collaboration.

CONTINUOUS DELIVERY

INTRODUCTION

Both agile development practices and DevOps have become essential elements contributing to the rise of continuous delivery. The value to the business is that good ideas can be translated into software much faster than was possible in the past with traditional delivery models. The term itself can be misleading, however, as "continuous delivery" also means different things in different contexts. At its most generic, the term encompasses an iterative and ongoing cycle of development, testing,

and delivery of software to a targeted destination. The destination could be a production environment, a staging environment, or a software product package. Regardless of the target, the goal is to accelerate the delivery of software functions and capabilities to support the business need for agile decision-making.

Although continuous delivery is often considered to be primarily an IT-related initiative, the primary drivers for continuous delivery are business- and customer-related. Businesses need new products and **services** to remain competitive, and customers expect to be able to interact with a company in efficient, seamless ways. In other words, from the business perspective, continuous delivery is no longer simply nice to have. It is a must-have for **digital transformation** as companies bank their futures on accelerating the speed at which new products are delivered to the marketplace.

IMPLEMENTATION

From the lifecycle perspective, DevOps and continuous delivery are intimately intertwined (see Fig. 10.2). Software components are developed via iterative stages, with each stage of the lifecycle having a specific purpose and its own inputs and outputs. Continuous delivery requires that each stage of the lifecycle be accelerated and optimized since a primary goal is to deliver software to the business as rapidly as possible.

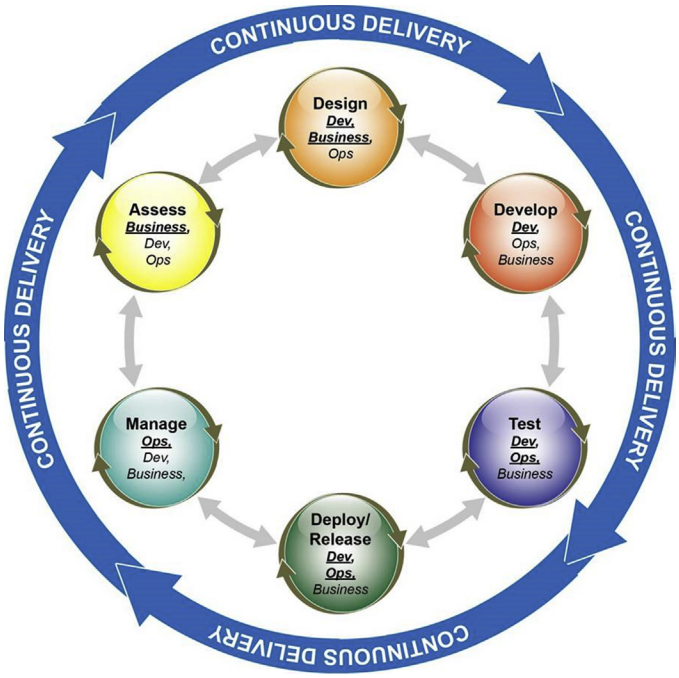

FIGURE 10.2

DevOps as a framework for continuous delivery.

Since each stage (and the artifacts generated at that stage) supports the next, DevOps practices can be viewed as a framework and foundation for accelerating continuous delivery. In effect, applying DevOps principles across the lifecycle paves the way for efficient delivery of application code. DevOps practices support staffing efficiencies necessary to free up both Development and Operations groups to actually work on new products. As the lifecycle is automated, software delivery naturally accelerates. Finally, automated processes produce more predictable results than manual ones, an outcome that also frees up Development and Operations staff from spending excessive amounts of time (and budget) on production support.

There is an old adage in the software development world that delivering high-quality software requires three key ingredients at optimal levels: time, budget, and skilled personnel. Reductions in budget or personnel extend timeframes, while reductions in timeframes require additional budget and personnel. That model can be reversed with the addition of two new ingredients: automation and the "grease" of DevOps practices. DevOps reduces the friction associated with the people- and process-related aspects of software delivery.

Governable, repeatable processes across the lifecycle support a factory approach much like Henry Ford's assembly line, which accelerated the process of building automobiles. Ford changed the way people worked, with each person performing a very limited number of tasks very efficiently, and therefore very quickly. He changed industrial production processes forever, and many other industries followed.

In the real world, IT organizations are, in fact, delivering software functionality faster than ever before. Almost 20% are delivering new code daily or even more frequently. However, it is not the case that delivery timeframes somehow magically become shorter. Accelerating software delivery requires acceleration of each stage of the underlying lifecycle. This can be done, in part, by reducing the volume of code contained in each release and by optimizing processes underlying the lifecycle.

CONTINUOUS DELIVERY TOOLING: ACCELERATION VIA AUTOMATION

As is the case with DevOps, automation is also a facilitator for continuous delivery. It is, in fact, the essential core for delivering on the continuous delivery vision, as it supports iteration within each stage of the lifecycle as well as the handshakes and artifact handoffs that must occur between the stages. In this scenario, each stage is appropriately instrumented and automated, and the artifacts generated at each stage are shared with the next. Requirements generated during the design stage, for example, will need to be available in subsequent stages to support development, testing, and service-level measurements. This underlines the importance of solutions that act as integration hubs supporting cross-tool, cross-stage data sharing.

As an example, Fig. 10.3, courtesy of CloudBees (www.cloudbees.com), shows the Jenkins ecosystem, which includes plugins to more than 1000 third-party solutions. These plugins allow Jenkins to act as an integration hub, supporting tasks at multiple lifecycle stages. Given the fact that today's software lifecycles heavily rely on cross-tool data sharing, these plugins are one reason for the success of Jenkins as a "continuous integration" platform: it provides a way for multiple tools to share information with one another across the lifecycle.

Another key value-add of high-quality enterprise management products is innate product intelligence. Imbued with industry-leading expertise in one or more key areas, modern toolsets are more than capable of executing skilled but repetitive tasks typically performed manually. From this perspective,

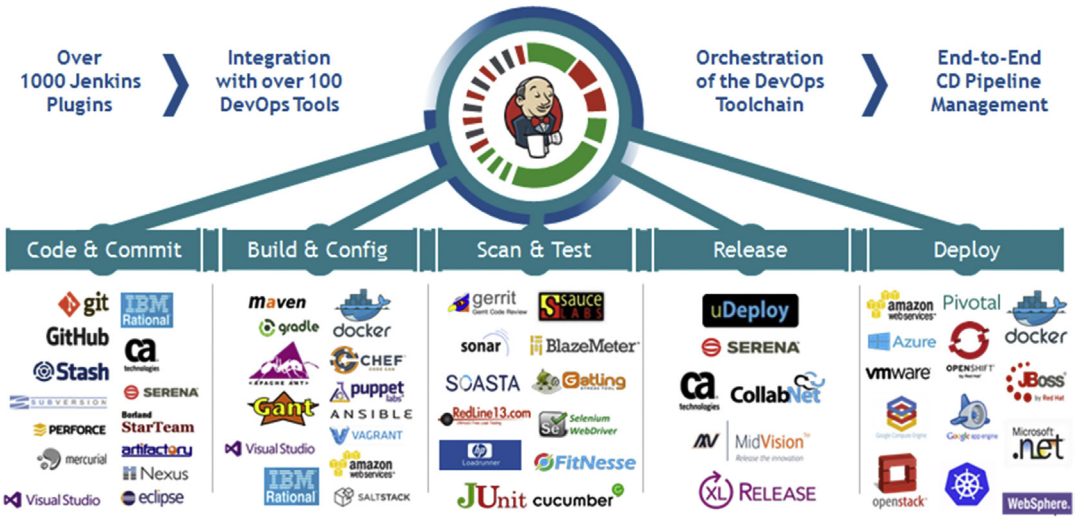

FIGURE 10.3

Jenkins as a hub for the DevOps/continuous delivery ecosystem.

tools investments extend IT budgets by enabling IT specialists to work smarter. They are also one of the best ways for IT organizations to accelerate continuous delivery by institutionalizing repeatable, governable processes without increasing headcount.

Since we are essentially talking about tools and practices that span the entire application lifecycle, there are other product types that should be part of the discussion as well.

- Collaboration tools are essential accelerators for requirements assessment, software design, and virtually every task executed by DevOps teams. Most IT practitioners indicate they currently lack a way to collaborate with peers and believe that such tools would make them more effective in terms of delivering software services to the business.
- Service virtualization and automated environment provisioning (release automation) are accelerators for QA testing, software deployment, and release management.
- Configuration, change, performance, and availability management tools reduce the manpower required to maintain the status quo in production application environments, freeing up personnel to more effectively address the needs of the business.
- Release automation tools govern, orchestrate, and apply software releases. They automate workflows as well as the provisioning and configuration tasks underlying software deployment at scale.

While they can definitely reduce the personnel needs associated with software deployments and delivery, their greater value lies in the fact that they make outcomes more predictable. The repetitive manual tasks underlying deployment of software releases are error prone and lack an audit trail. Every action performed by release automation tools is logged and the tools themselves ensure that deployments are always performed in the correct order and on the correct artifacts. Deployment becomes more predictable, and adverse impact to production is minimized.

DEVOPS AND CONTINUOUS DELIVERY

Today, many companies are utilizing both DevOps and continuous delivery and finding that both approaches are revenue drivers. Together, they can be a powerful combination. Industry research has found strong correlations between the frequency of code delivery and revenue growth. Even stronger correlations were made between the quality of a company's interactions between Development and Operations and revenue growth. Research findings also reveal the revenue-related negative impacts of failure to evolve the software delivery process to meet the changing needs of a business.

However, there are multiple implications to accelerated software delivery, both positive and negative. While it can have a profoundly positive business impact, it also significantly compounds the difficulties associated with day-to-day management of production environments. For example, the research also indicates that the number one bottleneck hampering efforts to accelerate the continuous delivery pipeline relates to the adverse impact of constant change on production environments.

It is also interesting to note that while the primary benefits of continuous delivery are business-related, the negative impacts are felt primarily by IT. More than 50% of companies engaged in continuous delivery indicate that Operations is spending more time managing production environments and nearly 50% say that Development is spending more time supporting production. Forty-five percent say service levels have degraded, and more than 35% cite an increase in the number of performance and availability problems.[6]

So, while the positive side of continuous delivery lies in the potential for business growth, the negative side can mean a decrease in service quality or increased costs related to managing production. In short, increased frequency of code delivery too often equates to production environments that are far more dynamic and far less stable due to constant and relentless changes to production.

Strong DevOps practices and teams supporting continuous delivery can mitigate adverse production impact while maximizing the value proposition to the business bottom line. The research also shows that automation supporting continuous delivery in particular can reduce and essentially eliminate adverse production impact over time, as the process becomes increasingly automated and "cookie cutter."

SUMMARY

There appears to be a dichotomy at present between the organizational imperative of accelerating software delivery and the potential costs of doing so. What continuous delivery has done, in effect, is massively increase the rate of change to production environments, often with adverse results. However, both production issues and costs can be mitigated by well-chosen tools, and this is another argument for application-related automation. ALM, testing tools, release automation, change management, and **APM** platforms/suites all become valuable assets that reinforce an organization's commitment to accelerating the lifecycle.

Automation has always been one of the best ways to ensure application quality, and this is particularly true today. Automating personnel-intensive tasks (such as manipulating test data, building test

[6]All statements and figures in this section are from survey-based research conducted by Enterprise Management Associates.

environments, executing QA tests, deploying software packages, and troubleshooting production issues) minimizes the possibility for the human error inherent in multistep processes. It also supports "build once, run many" scenarios in which automated processes can be controlled by standardized tools with repositories, policies, **templates**, and similar organizational assets designed to promote and enforce QA.

We as an industry are nearing a point where the absence of enterprise management automation could well mean the demise of a business. We have already seen this in the security arena, and the rise of continuous delivery brings the message home to application support teams as well. It's very difficult to continuously deliver software when an organization lacks adequate release management tooling or when Development and Operations teams are consumed with production support issues.

It is also the case that manual support processes have now run out of runway; automated management toolsets have become a necessity for those companies seeking to deploy and support applications at speed and scale.

KEY TAKEAWAYS

- DevOps and continuous delivery are separate but related IT practices that are neither institutionalized by standards nor uniformly defined and understood.
- When approached as a collaborative initiative spanning the lifecycle, strong DevOps practices act as a hub supporting accelerated continuous delivery.
- While often considered to be an IT-related initiative, continuous delivery is being driven primarily by business and customer demand.
- Both the quality of a company's DevOps interactions and the speed of continuous delivery correlate strongly with double-digit year-over-year revenue growth on the business side.
- While continuous delivery can yield compelling business benefits, it increases the load on IT organizations tasked with managing the impact of ongoing change on production execution environments.
- Tooling supporting DevOps and continuous delivery should be both lifecycle focused and siloed. In other words, information created in silo tools should be integrated for reporting purposes and shared across stages to facilitate a common knowledge platform and collaborative practices. Investment objectives should focus on enabling collaboration, facilitating complex workflows combining manual and automated tasks, dashboard building, and integrating diverse tools within and across lifecycle stages.

  Examples of vendors with products in this space:

  Appvance
  BMC
  CA Technologies
  Chef
  CollabNet
  CloudBees
  Compuware
  Dell

HP
IBM
New Relic
NRG Global
OutSystems
Klocwork
Parasoft
Perforce
PuppetLabs
Riverbed
SaltStack
Serena
Tasktop