Compliments of IBM

IBM Limited Edition

Application Release & Deployment

FOR DUMMIES®

A Wiley Brand

Learn to:

- Assess and improve your current software delivery methods
- Adopt deployment standards for rapid delivery of quality applications
- Successfully implement application release and deployment solutions

Eric Minick
Jeffrey Rezabek
Claudia Ring

Application Release & Deployment

FOR DUMMIES®

A Wiley Brand

IBM Limited Edition

by Eric Minick, Jeffrey Rezabek, and Claudia Ring

FOR DUMMIES®

A Wiley Brand

Application Release & Deployment For Dummies,® IBM Limited Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN: 978-1-118-84448-9 (pbk); ISBN: 978-1-118-84532-5 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Table of Contents

Publisher's Acknowledgments

We're proud of this book and of the people who worked on it. For details on how to create a custom *For Dummies* book for your business or organization, contact `info@dummies.biz` or visit `www.wiley.com/go/custompub`. For details on licensing the *For Dummies* brand for products or services, contact `BrandedRights&Licenses@Wiley.com`.

Some of the people who helped bring this book to market include the following:

Acquisitions, Editorial, and Vertical Websites

Project Editor: Carrie A. Burchfield

Acquisitions Editor: Connie Santisteban

Editorial Manager: Rev Mengle

Business Development Representative: Sue Blessing

Custom Publishing Project Specialist: Michael Sullivan

Publishing and Editorial for Technology Dummies

Richard Swadley, Vice President and Executive Group Publisher

Andy Cummings, Vice President and Publisher

Mary C. Corder, Editorial Director

Publishing and Editorial for Consumer Dummies

Kathleen Nebenhaus, Vice President and Executive Publisher

Composition Services

Debbie Stailey, Director of Composition Services

Business Development

Lisa Coleman, Director, New Market and Brand Development

Introduction

Software applications are large drivers of business revenue, and the timely release and deployment of those applications has become a critical part of the business life cycle. After all, what good is it to create an innovative application if you can't deploy it to test environments efficiently or deliver it to users on schedule?

Traditionally, *deployment* is defined as the promotion of components from one environment to the next. *Release* encompasses the deployment of a whole application or multiple integrated applications to production. Release and deployment differ but have the same objective: to deliver quality applications.

Because release and deployment have similar goals, the terms are often used interchangeably. For purposes of this book, we refer to application release and deployment as defined in the preceding paragraph and distinguish them as required.

Throughout the book, we give you insight into the differences between application release and deployment. We also show you how to leverage release and deployment solutions to help your organization speed time to market, drive down cost, and reduce risk.

About This Book

We wrote this book to serve as a relatively simple introduction to what can be a very complex topic. Use it as a reference, not as a manual. If you're interested in certain topics but not in others, feel free to read only certain chapters. Also feel free to skip around. You don't have to read the chapters in order.

Icons Used in This Book

You'll find the following icons in the margins of this book:

The Tip icon points out helpful information.

Anything that has a Remember icon is something that you'll want to keep in mind.

You don't have to read Technical Stuff material unless you want deeper understanding of a topic.

Be sure to read anything marked with a Warning icon, which alerts you to risk.

Beyond the Book

Throughout this book, we talk about the benefits of application deployment, automation, and release coordination solutions. We include a few industry success stories as well as best practices.

You can also find more information on application release and deployment by visiting the following web pages:

- **IBM DevOps Solutions for Application Release and Deployment:** `http://ibm.co/devopsRaD`
- **IBM UrbanCode Deploy product page:** `http://ibm.co/UCDeploy`
- **IBM UrbanCode Release product page:** `http://ibm.co/UCRelease`
- **7 Proven Practices to Strengthen Release Management:** `http://ibm.co/7ProvenPractices`
- **The ABCs of Continuous Release and Deploy in a DevOps Approach:** `http://ibm.co/ABCsRaD`

Finally, you can read these other IBM Limited Edition *For Dummies* books:

- ***DevOps For Dummies:*** `ibm.co/devopsfordummies`
- ***Agile For Dummies:*** `ibm.co/agilefordummies`
- ***Service Virtualization For Dummies:*** `ibm.co/servicevirtualization`

Chapter 1

What Drives Effective Release and Deployment?

In This Chapter

- Tracking the software development life cycle
- Understanding the business benefits of effective software delivery
- Understanding the technical practices that drive application development

An application takes a tremendous journey before being introduced to users in production. This journey is known as the software development life cycle (SDLC). In this chapter, we walk you through the elements of this cycle. Then we discuss the business and technical drivers of software release and deployment.

Following the Life Cycle of Software Development

The SDLC can incorporate every aspect of an application's life, from the initial planning to retirement. In this section, we discuss an important part of the SDLC that we call the *delivery pipeline,* which consists of multiple *environments* — deployment targets for a set of items that work together toward a common goal. Environments build on one another to increase the quality of an application before it reaches its intended user. Applications don't have to go through a set number of environments, but we've noticed four fairly typical environments:

- **Development (DEV):** In the *Development* environment, developers build and deploy code in a test lab, and the development team tests the application at the most basic level. When the application meets certain criteria for advancement, it moves to the next environment.
- **System Integration Testing (SIT):** In the *System Integration Testing* environment, the application is tested to ensure that it works with existing applications and systems. When the application meets the criteria of this environment, it's deployed to the next environment.
- **User Acceptance Testing (UAT):** In the *User Acceptance Testing* environment, the application is tested to ensure that it provides the required features for end users. This environment usually is production-like (see Chapter 3). When the application passes these requirements, it's promoted to the final environment.
- **Production (PROD):** In the *Production* environment, the application is made available to users. Feedback (see the nearby sidebar) is captured by monitoring the application's availability and functionality. Any updates or patches are introduced in the DEV environment and follow the same cycle.

Figure 1-1 shows a simple diagram of these four environments.

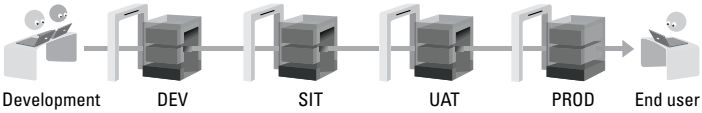

Figure 1-1: The four basic environments of SDLC.

A *deployment* is defined as the promotion of components from one environment to the next. *Release* encompasses the deployment of a whole application or multiple integrated applications to production. The image in Figure 1-2 helps illustrate the difference a little more.

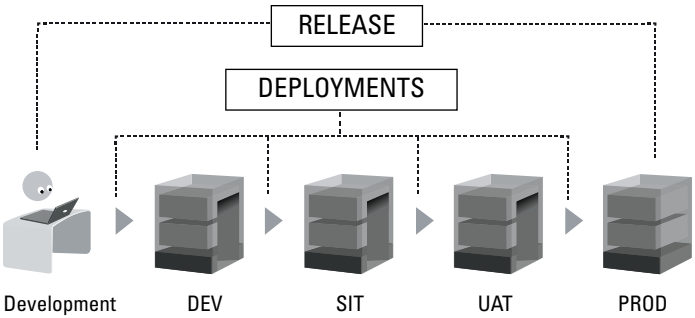

Figure 1-2: A visualization of the difference between a release and a deployment.

Running a feedback loop

Some software delivery practices involve monitoring the application in every environment and then returning feedback to the development team. Based on the new requirements, the development team deploys the modified application to the DEV environment, and the cycle starts again. After promotion to each succeeding environment, the application is monitored, and feedback is returned to the development team. This process is known as the *feedback loop.*

Reaping the Benefits of Effective Software Delivery

The need to provide software that adapts and responds to business and customer expectations drives the need for improved application release and deployment practices. The IBM Institute of Business Value reports that the number of organizations that *value* effective software delivery vastly outweighs the number of organizations that actually *provide* effective software delivery. Check out Figure 1-3 for more information.

In the following sections, we discuss some of the business benefits of efficient software delivery.

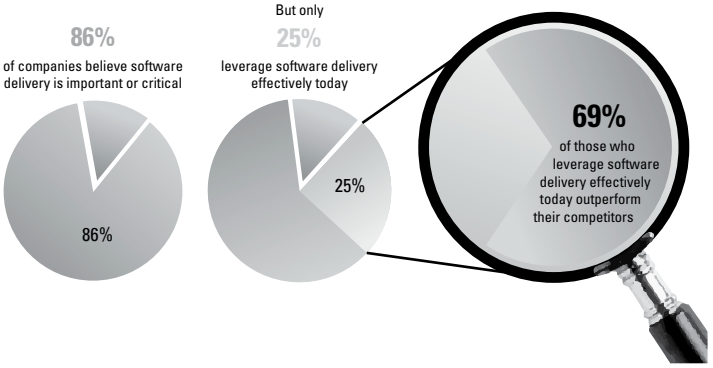

Figure 1-3: The value of effective software delivery.

Getting to market faster

The amount of time that customers are willing to wait for a desired new service keeps shrinking. Customers expect and crave instant gratification. Just think how often your mobile applications require updates driven by customers. If a service isn't available at the exact moment when a customer wants it, that customer is only a click or tap away from landing on a competitor's web page.

With the right application release and deployment methods, both upstart and established companies can quickly provide the services that customers demand. As effective software delivery is a key to business success, equipping your release and deployment teams with the right people, process, and tools can make the difference between long-term success and failure.

Decreasing expensive failures

The cost of a deployment failure depends on the environment in which the failure occurred. For the most part, failures in early environments, such as DEV and SIT (refer to Figure 1-1 earlier in this chapter), are far less expensive than failures in production. Finding an application error in these environments allows you to make a correction that may prevent a critical break in production — or worse, a critical breakdown for users. Forbes.com noted that in the summer of 2013, Amazon experienced a 30-minute outage that resulted in a projected loss of $66,240 per minute, or almost $2 million in total lost revenue.

Estimating the cost of failure

A failure not only can cause a loss of revenue but also can cost your organization more money to fix the failure. You can begin estimating the cost of failure of an application release by calculating an hour's pay for an engineer who is responsible for correcting the problem. A rough calculation of additional costs is 20 percent to 40 percent of salary. Here's the formula:

Estimated annual salary / 52 weeks per year / 40 hours per week * 1.3

If you have an engineer who makes $80,000, for example, a reasonable estimate of hourly total cost is

80,000 / 52 / 40 * 1.3 = $50

Now use this formula for every person on the recovery team, and multiply the result by the average number of hours spent taking corrective action. If a six-person cleanup team spends ten hours fixing a broken release, for example, the cost is $3,000 for that single release. That cost is merely productivity cost, however; it doesn't account for *opportunity cost:* the loss in revenue or the intangible potential loss of credibility due to the failure.

The cost of failure certainly drives the need for improved application release and deployment practices. In later chapters, we look at ways to prevent these failures from occurring.

Scaling complex releases and deployments

Organizations now tend to have more deployment targets than ever before on a variety of devices: local, cloud-based, and physical and virtual machines. Scaling a highly manual process up to an efficient, enterprise-wide system of interdependent application releases while maintaining security, traceability, and visibility is a difficult task.

The number of manual steps that may go into a single application's release may make up a laborious process, but the application inevitably ends up in production. With larger, more complex releases, risk grows with the number of interdependent applications being released. Manual processes combined with primitive release tracking solutions such as spreadsheets invite human error and, inevitably, costly release failures.

As applications scale to enterprise complexity, release teams often scale to dozens or hundreds of people, adding another layer of complexity. When IT teams scale to manage application complexity, job responsibilities and specialties become more siloed, and communication becomes less frequent as teams perform their respective processes. Silos occur due to job specialization as well as differences in time zones, cities, and countries. Distributed teams using different solutions and processes often discover that these differences double or triple their efforts and make them unable to deliver on time.

The coordination of each deployment, each process, and each associated set of manual steps is nearly impossible to execute without the use of a specialized tool. Using a specialized tool can help you coordinate each deployment, each process, and each set of deployment steps, and make them visible to each person involved in the release or deployment.

If you have ten steps to deploy an item to a server, and you go from one to ten servers, you go from having ten manual steps to having 100. Then, if you go from having one deployment item to ten items, you have another factor of ten, producing 1,000 manual steps. Manual processes won't scale as these three levels of complexity (number of items deployed, number of servers, and number of steps per deployment) come together to push exponential growth.

Organizations that can establish an efficient release and deployment process can keep up with the pace of development and can release smaller batches of applications, or even single changes or versions, at a time. Efficiency reduces risk and allows the organization to focus on timely satisfaction of business needs instead of process.

Identifying Software Delivery Practices

Several software delivery practices and methodologies enable organizations to speed time to market, reduce errors, and scale to enterprise level. Many organizations use a blend of approaches. Among the most popular practices are Agile,

Continuous Integration, Continuous Delivery, and ITIL, all of which we discuss in the following sections. Agile, Continuous Integration, and Continuous Delivery build on each other to help push applications to production.

Agile

As technology has advanced in leaps and bounds, customers have grown less patient and more demanding. The accelerated pace of change required to remain competitive in the market triggered the adoption of Agile practices.

Agile emphasizes focus on customer needs, small frequent releases, embrace of change, and collaboration among members of the development team. Unlike the traditional waterfall method of software development, Agile de-emphasizes extensive planning so that developers aren't locked into unchangeable plans. Agile focuses on smaller deliverables that allow developers to make changes as the need for change arises.

For more information, see *Agile For Dummies,* IBM Limited Edition, at `ibm.co/agilefordummies`.

Continuous Integration

Continuous Integration (CI) spun out from the Agile principle of regularly integrating code and testing builds to validate those integrated changes. Using CI, developers regularly commit to a common code line and integrate their changes with those of the other team members and with existing code. By sharing code quickly, they avoid the pain of waterfall's late integration phase and find problems faster. Developers should commit whenever a small block of code is complete — generally, at least once a day.

CI is facilitated by the use of a solution that automatically builds newly integrated changes when they're checked in. Often, automatic verification testing accompanies the automatic build so that defects can be found and addressed faster. This feature is especially powerful when combined with Continuous Delivery, discussed in the next section.

Combined with Agile development practices, CI is another step toward high-frequency, high-quality releases. It allows development teams to produce testable builds at a very high rate. It also puts more pressure on operations teams to deploy more frequently to later environments.

Continuous Delivery

Continuous Delivery (CD) is the next enhancement of CI and automatically deploys successful builds that have been qualified to test environments, triggering automated tests. Teams implementing CD seek to always have builds tested, enabling the business to choose to release features at any time with confidence.

CD takes its name from the Agile Manifesto and was created in response to the bottleneck that Agile and CI practices exposed for operations teams. As development teams began moving quickly and pushing out a high volume of builds, operations teams were pressured by a building backlog that needed to be promoted to production. Differences in language, platform, and testing further weighed down operations teams as they looked for ways to ensure a successful application release.

Just because an IT professional says that he's doing CD doesn't necessarily mean that he's deploying all the way to production. He may have implemented CD practices but could be hitting blockers — physical or technical obstacles that prevent a task from being completed — elsewhere in the SDLC. These blockers contribute to an exponential increase in risk, as more changes are accumulated in increasingly larger batches that wait to be deployed at once. The ideal form of CD uses automation to move an application to production, but you have major hurdles to overcome before you can achieve the benefits across the enterprise. We touch on these hurdles in Chapter 2.

ITIL

Information Technology Infrastructure Library (ITIL) is a set of best practices and processes that aid IT service management. ITIL consists of five major life cycles that advise practitioners on best practices to provide specific services for users.

The concepts in ITIL provide vendor-neutral, nonprescriptive practices that can be used in any industry. Most organizations that practice ITIL do so because they need governance, risk management, and control of their release and deployments.

Some ITIL shops are reluctant to automate because they think that going faster puts them at greater risk of making errors. In fact, automation can reduce risks and improve control, visibility, and auditability.

Taking a DevOps Approach to Software Delivery

DevOps is an enterprise capability for continuous software delivery that enables clients to seize market opportunities and reduce time to customer feedback. DevOps applies Agile and lean principles across the software supply chain to remove waste and bottlenecks. DevOps is made up of four major capabilities across the SDLC:

- **Plan and Measure** focuses on lines of business and their planning processes. It means understanding and increasing the effectiveness measures of these processes and the application portfolio that they cover.
- **Develop and Test** focuses on collaborative development, CI, and continuous testing. It focuses on streamlining development and testing teams' capabilities.
- **Release and Deploy** enables the creation of the Deployment Pipeline for Continuous Release and Deployment.
- **Monitor and Optimize** includes the practices of continuous monitoring, customer feedback, and optimization to monitor how applications are performing post-release, allowing businesses to adapt their requirements as needed.

For more information on DevOps, view *DevOps For Dummies* at `ibm.co/devopsfordummies`.

Implementing release coordination and deployment automation solutions are keys to solving a DevOps problem in your organization, as we discuss in Chapters 3 and 4.

Chapter 2

Applying the Gold-Standard Deployment Process

In This Chapter

- Recognizing the three pillars of a gold-standard deployment process
- Seeing the effects of a gold-standard deployment process

In this chapter, we focus on addressing both business and technology drivers (see Chapter 1) by applying the gold-standard deployment process. At the end of the chapter, we show how applying a gold-standard deployment process can positively affect a release.

The Three Pillars of Gold-Standard Deployment

To ensure a successful application release, start by implementing the three pillars of the gold-standard deployment process:

- Use the same process.
- Automate.
- Perform incremental changes.

We discuss these pillars in detail in the following sections.

Use the same process

One of the main problems with scaling application deployments to more complex and higher volumes is that siloed teams and specialists create custom processes to fit their own responsibilities.

Deployment needs vary from team to team:

- Developers want to deploy and test quickly to implement more application changes. They often create shortcuts in the deployment process by writing scripts or by skipping burdensome steps that may be required for application performance in production.
- Testing teams strive to mimic the pace of development teams, but they understand the need to take their time in testing both the deployment process and the applications. These teams often create their own deployment processes to get to testing faster and avoid delays.
- Operations teams can tolerate a slower pace. Because they're responsible for keeping business-critical systems running, they can't endure the risks associated with rapid deployments. Their deployment process is designed to keep the production environment running smoothly.

When deployment processes differ between teams and target environments, the chance of errors increases. Undocumented steps, environmental differences, and lack of input from all teams increase the chances of deployment failure.

To reduce the risk that comes from using different deployment processes, use the same deployment process when promoting from environment to environment throughout the software development life cycle (SDLC) and through the continuous delivery pipeline to production.

When you use the same deployment process across the SDLC, you can test both the application and the deployment process.

The preferred method of standardizing the deployment process is to begin your design in production and work backward. Although this practice requires more thought, time, and effort up front, if you design the process for the move to production first, you can remove the unnecessary steps as you work your way back to earlier environments. We discuss this topic in more detail in Chapter 4.

Automate

Manual or half-scripted steps in a deployment process increase the risk of deployment failure. Even skilled operators make mistakes, and error is more probable when humans must run through a long list of manual steps. A deployment process (automated or manual) should include predefined gates or approvals that a team must perform or meet for the deployment to enter the next stage. Automating the deployment process, however, helps you scale, get to market faster, and reduce risk.

By automating what you can of the deployment process, you may add compliance and auditability through visibility into

- What components are included in the deployment
- Who deployed what application
- Where the application was deployed
- What version of the application was deployed
- When the application was deployed

Traceability and visibility are two keys to a streamlined release and deployment process, and you can attain them with proper automation, as we discuss in Chapter 4.

There's a difference between automating and writing a few scripts that a subject-matter expert still has to kick off. True automation removes risks of changing parts and allows authorized stakeholders to kick off a deployment at the click of a button. In addition, true automation provides self-service deployment capabilities, visibility, and traceability.

Deployment automation is such a major factor in successful application deployment that when we mention *application deployment* in this book, we're referring to application deployment through automation.

Perform incremental changes

The final pillar of a gold-standard deployment process is making incremental application changes. Ideally, your automation solution deploys only the components that need to be changed and leaves the remainder intact, as redeploying unchanged components carries risk. Deploying only what changed means being able to deploy more often and on demand. When you understand the application changes, you can see true versioning of what's deployed when, where, and by whom. You can deploy more often and on demand when highly repetitive tasks are automated, and agreed-upon approval and *quality gates* (criteria that an application must meet before advancing to subsequent environments) are established. If you're using the same process for every deployment, making small application changes is both possible and desirable. Smaller application changes should be easy to pass through testing environments and into production environments.

When you're performing incremental changes, you may not be using the complete deployment process. To make incremental changes, simply set up the gold-standard deployment process and use only the part of the process that's actually required to deploy incremental changes.

The Positive Effects of Gold-Standard Deployment

Employing the three pillars of a gold-standard deployment process (see the preceding section) can help you begin to align your organization's business needs with its technical needs, which can have positive effects on the application release. This section discusses some of those effects.

Avoiding the Monday Morning Effect

The Monday Morning Effect occurs the Monday following a major release weekend when the release team faces numerous problems after the release, such as a break in production or defects in the application. The release team is required to take heroic, immediate corrective action. Restarting the release, correcting the mistake or mistakes, and ruling out other possibilities without incurring a substantial outage window are nearly impossible. If the cleanup doesn't occur immediately or the problem isn't immediately obvious, the result is the Monday Morning Effect. To avoid the Monday Morning Effect, deployments must be automated, teams must use similar deployment processes, changes should be introduced incrementally, and release activity must be visible to everyone involved with the release.

Automating and using the same deployment process

When you use the same automated deployment process across multiple application life cycles, you greatly affect the overall release. The release is less error-prone and takes much less time to perform.

Performing incremental releases

Performing a large deployment is risky, but deploying a large release of applications is even more dangerous. Deploying small batches of change is less risky and easier to manage.

In addition, managing defects in a small release and correcting them effectively are far easier in smaller batches (see the next section).

To release and deploy on demand, you must have a streamlined process in place. For this reason, continuously improving your standard deployment process and automating error-prone tasks are the keys to successful incremental deployments and smooth releases.

Managing defects

Smaller releases are less risky than large releases because smaller releases decrease the number of potential defects that could cause an error or outage. There are three contributors to defects: code, configuration, and complexity. Complexity here refers to the number of interdependencies, or relationships, in the release package.

Whereas code and configuration defects increase risks linearly as batch sizes increase, complexity defects increase risks quadratically. As shown in Figure 2-1, if you have three features being released, you have three relationships to manage, but if you add just one more feature to the release, you have six relationships to manage. You can see that the addition of even one more application doubles the relationships and creates more complexity — and, therefore, risk.

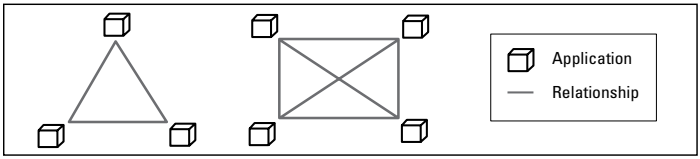

Figure 2-1: Understanding relationships and complexity.

For organizations that use manual deployment processes or slow releases to gain better control of quality, the risk of failure has grown exponentially. Larger release batches laden with interdependencies carry the risk of multiple failures. Using the same deployment process, automating manual tasks, and performing incremental changes, however, enable a streamlined release process.

For more information on managing release risks and defects, visit `http://ibm.co/UCVlog4`.

Chapter 3

Choosing Solutions for Application Release and Deployment

In This Chapter

- Preparing your organization for change
- Considering release and deployment solutions
- Finding a release coordination solution

Managing your application through the build and delivery pipeline with deployment automation while orchestrating and coordinating the overall release process requires specialized solutions. Application deployment automation solutions and release coordination solutions are designed to get high-quality applications to production as quickly as possible.

In this chapter, we discuss how to prepare your organization for the changes that application release and deployment solutions will introduce. Then we present basic criteria for evaluating these solutions.

Preparing for Changes

Before you begin to look for a solution, remember the goals of application deployment automation and release coordination tools:

- **Speeding time to market:** Increasing the frequency of software delivery with increased compliance through end-to-end transparency, auditability, and reduced time to feedback
- **Reducing cost:** Reducing the amount of manual labor, resource wait time, and rework by eliminating errors and providing self-service deployments
- **Reducing risk:** Delivering high-quality applications through automated, repeatable deployment processes across the development, testing, and production environments

Achieving most business goals requires acknowledging the changes required for individual team members, interactions, processes, and solutions, as we discuss in this section.

Acclimating your people to the proposed changes may be the longest and most difficult step in the implementation process, but it's required for streamlined, fully automated release and deployment practices.

Role changes

Some of the most important changes you're going to be making involve individual team member's tasks and the interactions among team members. Taking a note from the Agile Manifesto, your team members should be ready to adjust their methods of interaction as well as the frequency of those interactions. They must also be ready to accept automation solutions and the new roles that they will play within the organization.

Discuss changing team members' roles with management so you can assure employees that the automation solutions will help them perform their day-to-day job responsibilities. The solution isn't meant to replace human talent; instead, it's used to help facilitate the outcome the organization wants to achieve by reducing the amount of risky repetitive tasks and enabling team members to focus on the tasks that require more creative thinking.

Try to find an internal champion who sees the benefits associated with the change. He or she can provide peer-to-peer insight and perspective on the long-term effects of the

solution. A team-level champion who is on board with the new solution gives his or her peers a chance to hear the benefits from someone who faces similar challenges.

You should introduce this cultural shift as early as possible. Chapter 4 gives you a rough idea of how long various changes may take.

Process and solution changes

As individuals and interactions change to achieve business needs or goals, day-to-day business functions also change. When introducing the solution you want to use, be aware that you'll face some resistance. Keep your team motivated, and prepare for a troubleshooting period. Getting through the first major project with the solution will force everyone to confront the new solution and new process and to discover the most efficient way to interact.

If you just bring in the solutions and the process without preparing your team adequately, you're likely to miss the mark for successful implementation. It takes changes across processes, solutions, individuals, and interactions to achieve a business goal.

Evaluating Release and Deployment Solutions

Application deployment automation solutions manage application components and their versions and track which version is deployed to which environment. These solutions are essential for application release and deployment practices, as we explain later in this chapter.

When you're looking for a deployment automation solution, depending on your business goals, you should seek a solution that has the following capabilities:

- Reuses deployment processes across environments
- Coordinates application deployments across multiple tiers

- Integrates with existing technologies
- Provides role-based security
- Maintains logs of all commands executed in deployments
- Tracks who deployed which version of a certain deployable artifact to which target

Prioritize the features that your organization needs most to achieve the desired business goal. Your organization may want to ensure auditability and governance, for example, or to minimize the effort required to pass audits. In such a case, the solution's capability to provide role-based security and maintain deployment logs is more important than its other capabilities.

Successful deployment with IBM UrbanCode Deploy

A financial organization was developing a new trading platform to serve as the lifeblood of the organization. The development team used Agile development practices to produce results faster, but the process of deploying applications across hundreds of servers consisted of a mostly manual operation with customization required for each application. Introducing Agile development practices actually caused changes to build up for the operations team, whose deployment process wasn't equipped to handle frequent changes. This problem eventually brought development to a halt.

The financial institution still wanted to achieve the benefits of Agile development methodologies. After carefully evaluating numerous solutions, the organization replaced its manual deployment practices with IBM UrbanCode Deploy, which provided several benefits:

- Deployment times went from three days to two hours.
- The organization saved more than $2 million in the first year alone by eliminating the cost of manual deployments.
- The organization achieved compliance and gave teams a self-service option for deploying applications.

Evaluating Release Coordination Solutions

Planning, managing, and executing a major application release is typically done with the help of spreadsheets or release documents that dictate the tasks, their owners, and the sequence required for a successful release. The problem with executing a release in this manner is that there's no way of tracking who did what, what code was deployed where, and when the release happened.

Release coordination solutions are designed to help in the planning and execution of a release by providing collaborative release planning that encompasses application and infrastructure changes. Ideally, a solution provides full visibility into the release process as well as end-to-end planning and execution capabilities.

When evaluating a release coordination solution, you should search for a solution that can do the following:

- Streamline the release of multiple applications in a release
- Update the release plan on the fly
- Display the progress of the release in real time
- Track changes in applications and infrastructure
- Notify stakeholders of release escalation
- Sequence and coordinate automated activities and manual activities in the release workflow, all relationships among activities, and all related communications to people and automated systems
- Automatically promote applications that meet the entrance criteria of lower environments

Successful release with IBM UrbanCode Release

A not-for-profit organization was spending far too much time planning and coordinating releases. Team members were using spreadsheets to track release activities and meeting multiple times to review the plans and changes in plans. The organization wanted to achieve better visibility into release efforts and to cut down on meeting time during the release process. It eventually selected IBM UrbanCode Release, the first solution designed specifically for complex application releases. IBM UrbanCode Release enabled the organization to reduce release meetings by 50 percent, reduce the length of each meeting by 50 percent, and add visibility to the entire release. By the end of their second release with the solution, team members had a good release template in place. By the end of the third run, the team leader who implemented the solution felt so comfortable with the solution that he went on vacation during the release.

Although doing so isn't necessary, it's advisable to use a deployment automation solution with your release coordination solution. Just as application release and deployment solutions often work together toward the same goal, release coordination and deployment automation solutions often work together to help build an automated delivery pipeline. Release coordination solutions integrate with deployment automation solutions to kick off automated deployments of multiple applications during release time.

Chapter 4

Rolling Out the Solution

In This Chapter

- Putting an application deployment automation solution to work
- Introducing a release coordination solution

After your organization selects the best application release and deployment solutions for its needs (see Chapter 3), you face the task of implementing those solutions. This chapter gives you some pointers.

Implementing an Application Deployment Automation Solution

As we mention in Chapter 2, the goals of automation include traceability and visibility via a solution that tracks a build through the stages of the software development life cycle (SDLC). The main components of a sturdy build process in this context are a versioned artifact repository for completed builds and dependency management for dependent projects.

One way to maintain those two components is to have a Continuous Integration (CI) server; the other way is to integrate and track builds manually. With a manual integration method, you must version and contain working builds in a functional artifact repository. With a CI server, you must also ensure that those two steps are part of your solution or process. When you have visibility into what each build contains, and when each artifact is set to redeploy if something goes wrong in a higher environment, you've established the minimum requirements for implementing your application deployment automation solution.

To implement a deployment automation solution successfully, you must have a consistent, reliable build process in place. If your builds are inconsistent or unreliable, deployment automation may find the flaws of that process sooner, but the intrinsic quality issue remains.

Choose the ideal time for deployment

When you're moving forward with the implementation, your two major challenges are overcoming resistance from your team and finding the project or application that has the qualities you need for automation. These two challenges are intertwined. Choosing the wrong application for automation creates resistance, and automation that's poorly planned or poorly documented is likely to fail.

The best practice, therefore, is to choose the right time to deploy the application or project in question.

One ideal time is during a *greenfield project,* when a new application has a deployment plan that requires automation. Another potential ideal time is during a *brownfield project,* which is already under way but is also ready for automation. Choosing the right greenfield or brownfield project requires careful consideration.

You have to take a deep look to tell whether your project or application is ready for automation. Applications that call for automation have a well-documented deployment process and (ideally) have a series of repetitive tasks. If you're looking at a greenfield project, it may use an application that's similar to a previously released application, or its existing deployment plan can simply be moved to the automation solution. The same rules apply to a brownfield project. If you have a manual deployment process in place and are feeling pressure to deliver faster, you can move the process to an automation solution.

To choose the proper greenfield or brownfield project, you need to know how your teams typically work and where to step in. If you usually do development-driven deployments, for example, you should move the deployment power and design to the operations team. If this process usually ends

up creating a wall between development and operations, the team deploying to production should be in charge of that process and the automation. Knowing where the process usually breaks down can be an asset in solution implementation, and timing must come together with the right people, processes, and solutions (see Chapter 3).

Figure 4-1 shows the tasks that you should be thinking about to get an idea of how to plan and roll out your solutions.

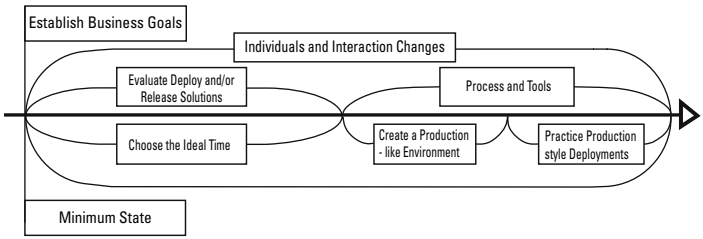

Figure 4-1: An example of an implementation and rollout timeline.

Create a production-like environment

As soon as you have your project in mind, you should design a production-like environment as a starting point for development. This environment will likely be smaller but should use the same operating systems, middleware, and configurations as the production environment. As we discussed in Chapter 1, UAT is one of the four typical environments of the SDLC and provides an opportunity to test in a production-like environment. Production resources that are unavailable to test environments should be simulated through service virtualization, if possible.

For more information on service virtualization, see *Service Virtualization For Dummies,* IBM Limited Edition, at `ibm.co/servicevirtualization`.

A production-like environment improves the accuracy of your testing for both the application and deployment processes. You can simplify your environments as you work your way back to earlier environments and remove unnecessary components.

Involve your operations teams as early as possible instead of saving the critical final deployment to production for last and hoping for the best. Allowing these teams to step in at the beginning of the SDLC allows the development teams to develop and test their builds and leaves the operations teams free to test and deploy the applications. It also ensures a self-designed environment to keep a working application in working order. Shifting some of the business responsibilities to development improves coordination between development and operations and creates a beneficial process for all teams.

This shift in thought and process is the essential goal of DevOps. It forces development to take Operations concerns into consideration throughout the SDLC instead of just at deployment time. In turn, Development has access to production-like environments that enable them to develop and test against a more realistic system, like the one that users will use. Operations teams also benefit by getting a preview of how their environment will react to the application and where support enhancements can be made.

Practice production-style deployments

If the ideal state that you want to achieve is a streamlined release consisting of automated deployments, you need a consistent, repeatable process. To achieve this process, start by practicing production-style deployments that can be simplified for earlier environments. The environments themselves should also be as similar to production as possible. In a new implementation, finding a project that can adapt to this criterion is key.

Design for production first

Because production is the most complex environment, deployments to earlier environments should be designed as simpler versions of production deployments. Building up in deployment complexity from development to production won't work, just as increasing the size of a square won't create a cube. The dimensions and complexity in production don't have an adequate framework in development, so if you build a deployment process based on development, you end up having to rework it greatly in production.

Designing the process for production first provides several benefits:

- It allows teams to practice before the critical final deployment (see the preceding section) and provides the template for remaining processes toward the beginning of the SDLC.
- It allows teams to refine and further streamline the process. Automation furthers this goal by stabilizing previously manual steps in a complex process and reduces unnecessary effort in future deployments.
- It eliminates the disconnect in deployment process complexity between earlier and later environments. Starting the design process in development actually prevents alignment with the goals of the operations teams, because a self-designed, self-tested process can't reach the level of complexity that operations teams require.

Developers focus on testing their individual contributions and the representative piece of the entire application that applies to their function in the project. For this reason, developers often don't take into consideration or don't know what's needed to keep production environments stable and functional. Due to the magnitude and scale of what needs to be tested and functional in production, the deployment process shouldn't be designed by teams that aren't familiar with the production environment. This is inevitably why allowing developers to design a Continuous Delivery process as an extension of Continuous Integration stops before Production and hits the wall of Operations.

To fully align your teams and facilitate collaboration from the start of a project, you must recognize the importance of starting with the most complicated processes and removing steps to simplify them. It's easier to remove steps than to add during a crisis or when a deployment has failed in production.

Implementing a Release Coordination Solution

Release coordination is ideal for organizations that deploy multiple applications at the same time or that desire more

visibility into a complex release process. As we mention in Chapter 3, you can use a release coordination solution with or without deployment automation solutions.

If your release process is a long list of manual tasks contained in a spreadsheet, you should evaluate solutions that aid in that process. If you want to implement release coordination with deployment automation, however, additional best practices for deployment automation should be in place.

Identify a realistic release model

To achieve success with a release coordination solution, you need to be familiar with your current process, the applications being released, and the complexity of those applications. You should start with a small release and work your way up to the release of multiple applications. Even in a small or sample release, you should still know the application(s) and feature(s) being released and who is involved in the typical process.

The key to this concept is scale. You want to begin with the smallest logical model of a typical release process. You should use a release that, though small, will be useful in scaling up to future releases by representing your ideal process from the beginning, which means including the people who will be part of the most complex releases.

Don't bog yourself down with "what-if" scenarios, because you won't be able to predict everything until you actually begin using the solution and discover areas for improvement. You and/or your team should find the way to implement the release coordination solution that provides the greatest value.

Choose an implementation path

After you choose a release to serve as a baseline for future releases, you should identify a logical path toward implementation. Three main paths provide a smooth transition to the use of a release coordination solution:

- **Use the new solution in parallel with the existing process.** The first path is to use the solution in parallel with an existing release process. This path enables the release

team to map an existing process to the process in the release solution while they're both running. In practice, this means conducting a release in the legacy fashion while running the release in the new solution to see how tasks differ using the tool. This practice not only helps the team acclimate to the idea of the new process, but also provides a no-risk dry run for first-time use, thereby reducing anxiety about the next release, demonstrating the parallel tasks in each method of release, and providing a before-and-after picture of the process.

- **Conduct a post-release run.** The second path is to conduct a post-release run by using the solution to model a recently completed release — that is, releasing an application by using the current-standard process and then using the same process to release an application with a release coordination solution. Much like running the solution parallel to a release in progress, this practice gives the team a before-and-after picture of the process and a safe way to use the solution for the first time.

The first two paths are exercises that provide realistic use cases for the solution and give team members a chance to realign their expectations and interactions.

- **Dive into the deep end.** The third path, which is less conservative than the first two paths, is to dive into the deep end and simply use the new solution for a live release. You should set up your teams for success by choosing a small release that you know they can execute with the solution. Take the selected release down to its minimum representative state of complexity and give it to your team as a low-risk live first use of the solution.

Chapter 5

Ten Myths about Application Release and Deployment

In This Chapter

- Seeing why automation can help, not hinder
- Understanding what a specific solution can and can't do

Sometimes, the best way to understand what's true about a concept is to understand what's false about it. In that spirit, here are ten myths about application release and development.

Automating Deployment Means Writing Scripts

Although kicking off individual deployment scripts may work for small deployments to a few servers, that technique won't hold up under the scale and complexity of deploying interdependent enterprise applications.

Relying on a subject-matter expert (SME) to kick off deployment scripts puts a strain on the whole deployment team and especially on the person who has that responsibility. If the SME is unavailable to kick off the script (or is no longer with the organization when a change is ready to advance), you can miss deadlines or miss the step entirely. In addition, if the SME is responsible for running the scripts for multiple deployments, the chance of errors increases as the likelihood of

bugs increases with batch size (see Chapter 2). Also, human constraints such as fatigue, anxiety, and nerves affect performance when a person is performing a long series of repetitive and high-pressure tasks.

The best solution is to let an application deployment automation solution execute deployments, ideally using integrations. If your chosen solution can effectively integrate with your existing tools, you should be able to create a deployment process using a simple process designer that does not rely on scripting. Effective integrations and a drag-and-drop process designer can remove most or all scripting from your deployment process.

Development Teams Create the Best Deployment Processes

Developers know how an application should work but may not know how the application topology in a production environment works, because they don't need to. Assigning the development team the job of designing a release process is likely to result in a plan that neglects key operational concerns or production-specific configurations such as clustering, load balancers, and integrations with outside systems.

As we mention in Chapter 4, starting to plan the deployment process in production allows the development team to test the application against a production-like environment. It also allows the development and operations teams to test the deployment process early and often. The best deployment processes are the result of collaboration among development, operations, and release engineering.

Complex Releases Can Be Easily Managed without Specialized Solutions

Sure, you can manage simple application releases with various methods, such as e-mails, run-book documents, spreadsheets,

and deployment automation solutions. The first three methods, however, don't provide traceability or collaborative release planning, and none of these methods can be scaled to accommodate the complexity of an interdependent enterprise release.

Although it's possible to use a deployment automation solution to manage a release, it's better to use a release coordination solution. Release coordination solutions are designed to assist in planning and executing a release by providing collaborative release planning that encompasses application and infrastructure changes.

Continuous Delivery Means Constant Production Releases

Continuous Delivery (CD) isn't continuous release. Instead, it focuses on speeding a new version through the delivery pipeline as fast as possible and then waiting for the business to decide when to release. The same technology is used for the production deployments, but the decision is a human one.

For most organizations, their automated tests are insufficient for validating that a new version is ready for production. For them, Continuous Delivery minimizes the friction caused by moving versions through environments and maximizes the productivity of testing teams.

Automation Reduces Quality and Control

Automation boosts quality because it leverages computers' capabilities to run repetitive tasks that can be botched when performed manually.

Controls are also improved. The deployment button is behind role-based security, and approval and quality-gate rules can be enforced automatically. When you have a full audit trail showing who configured a process and who ran it, you know exactly what happened at all times.

A Spreadsheet Is a Good Release-Management Tool

Many release teams manage to function by using spreadsheets, but this method of coordination almost always results in human error. Spreadsheets do enable management of the release process at the highest level, but they require highly repetitive tasks to be performed manually, which increases the chance of error, and they also require constant maintenance to ensure that all members of the release team are on track. Finally, as team locations vary, teams grow, and applications become more complex and interreliant, spreadsheets become more error-prone and less useful because they don't scale with release complexity.

Unlike spreadsheets, release coordination solutions capture interdependencies and determine the most effective deployment strategy. They also alert members of the release time when certain milestones in the process have been met or when certain team members' skills are required.

Release coordination solutions are highly advised for organizations that have a complex release process or distributed teams, or that have tried and failed to manage their release process with spreadsheets.

Release coordination solutions can be used without deployment automation solutions, but it's advisable to use both to reduce the risks associated with repetitive manual tasks and to speed time to market.

One Large Release Is Less Risky Than Several Small Ones

In fact, large releases carry increasingly higher risk with the number of interdependencies included than small, frequent releases do. If you streamline the release and deployment process by using automation solutions, you free your people to troubleshoot the process itself instead of requiring them to perform highly repetitive manual tasks.

Smaller releases have fewer pieces and fewer independencies. Errors caused by misunderstanding or poorly accounting for interdependency are dramatically reduced in a smaller release.

Incremental changes through true CD are the ultimate goals of any organization that wants to deliver value by delivering new features at a faster rate than its competitors do. Small, frequent changes delivered through automation solutions are the first step in reducing the cost and risk associated with traditional releases.

Automation Is Separate from the Build Process

Unfortunately, until you reach the minimum requirements of a working artifact repository, dependency management, and high-quality output from your build process, you're not ready to look into deployment automation. Deployment items must be versioned or ready to be versioned to allow for tracking of what's being deployed and where the deployment item is deployed. To complete successful releases that provide visibility across the entire SDLC, you need to make sure that dependencies are manageable, visible, and evident as part of your build process.

You usually can achieve the required minimum state with the help of a Continuous Integration (CI) server that also provides versioning and self-service capabilities for deploying to testing environments. Manual forms of CI can give you a leg up on deployment automation but lack the full visibility that we recommend achieving before you move to deployment automation solutions. The primary goal of a streamlined build process is to create working builds that are versioned and ready to be deployed through testing and to production at a high level of quality.

Some application deployment automation solutions provide a versioning capability with a proprietary artifact repository. This capability allows you to skip versioning your own builds and use the application deployment automation solution to incorporate that step. One such solution is IBM UrbanCode Deploy.

A Backlog of Deployables Doesn't Indicate a DevOps Problem

If you claim to be practicing CD, but your operations team is facing a huge backlog from development, you have a DevOps problem, and you aren't really practicing CD. DevOps practices and solutions help you seize market opportunities and reduce time to customer feedback by enabling true CD. When you decide to implement release and deployment solutions, you should prepare to work in a DevOps culture, which involves changes for individuals, interactions, processes, and solutions (see Chapter 1 for more on DevOps).

When you've minimized differences between development and operations environments, standardized your release and deployment process, and automated most or all of your manual tasks, you have a DevOps solution in place. This DevOps solution accelerates software delivery; reduces time to customer feedback; improves governance across the SDLC; and balances quality, cost, and speed.

Release Coordination Solutions Fix All Problems

The move from a completely or mostly manual release and deployment process to an automated one is difficult, and it requires several considerations. One of the most important long-term changes is in the culture of your organization. Your people and interactions must be aligned with the new method of work and potentially shifting responsibilities. You should prepare your teams for the shift, expect resistance, and remain aware of what works and what doesn't work for your organization.

When you've selected the right project to implement application deployment automation and release coordination solutions, your teams will have no choice but to collaborate. Preparing them as much as possible for the change, however, will make the transition easier and will help you reach your business goals. Just as addressing build and CI issues reduce bottlenecks in deployment and CD, you may begin to see other opportunities for improvement in your SDLC after you introduce deployment and release solutions.

Notes

Notes