Release Management: Solutions for Today's Challenging Software Development Environment

Abstract

Release Management: Automation Solutions for Today's Software Development Environment

This white paper proposes automation as an excellent tool for Release Management (RM) to resolve the problems associated with the handoff from Development to Production (Operations) when creating and releasing new products using today's Software Development Life Cycle (SDLC). The different approaches and focus of these reduces the efficiency and effectiveness of the process, including delays in ship dates for product releases; time wasted tracking down and addressing code and script problems; friction between the Development and Production teams; audit failures due to the lack of critical information about the code and process; and additional down-time because of slower builds internally and customer deployments externally.

Automation helps Release Management identify problems early and allows them to take corrective action sooner; eliminating many of the problems caused by the handoff. The automation of testing, build deployments, and error-prone manual processes, also improves the speed of the entire SDLC. This allows RM to focus on issues unique to the current release.

A good automation solution includes several key components:

- Extensive information tracking is needed so that the Release Manager can answer all of the questions about the release to meet today's strict auditing and reporting requirements. For example: What has been added? What has been changed? What tools have been used? What problems have been fixed? How do you know that the "right stuff" has been handed off to Production?
- Test automation to remove error-prone, time-consuming manual and regression tests.
- Build automation to create reliable and repeatable deployments for all kinds of builds; and Deployment automation for scripting, a fragile, often manual, process that leads to time-consuming searches for bugs.

Introduction

The Release Manager has a difficult job. He is often the nexus in the tug-of-war between Development and Production. He needs to be able answer all of the questions about the release during all phases of the SLDC: What has been added? What has been changed? What tools have been used? What problems have been fixed? How do you know that the "right stuff" has been handed off to Production?

As a result, the handoff between Development and Production (Operations) becomes cumbersome and results in several problems:

- Delays that are the result of flawed manual processes
- Different work environments and expectations between Development, Test and Production
- Audit failures
- Excess downtime due to slower builds
- Time consuming script and code errors

Why is this so? Several reasons:

- Complex, interdependent software deployment cycles
- Human error
- Demand for faster, traceable, error-free releases

In a typical shop, the Release Manager is responsible for transitioning the application between Development and Production. She gets the job done. It just takes a long time, which is a problem when the company pressures both areas for a shorter release cycle and more frequent releases.

This paper identifies common problems in the Release Management environment and how automation solves these problems for today's Release Managers.

Market Drivers

The push for automation is driven by these factors:

- Complex software deployments
- Fast software development cycles
- Different environments in Development and Production
- Stricter requirements for auditing and reporting

Complex software deployments

Today's software projects are complex, interdependent software deployments that often impact other areas of development, other teams working within the company, and other employees and contractors working in remote offices and foreign countries.

Currently, most development efforts center on web applications. These applications hide a number of interdependent components. This led to Service Oriented Architectures (SOA). While an advantage of SOA can be the flexibility to update a single service, adding new features to an application often requires modifying numerous components. Releasing these components, even to a test environment, requires careful coordination from the Release Manager. Otherwise migrating the release to different environments, from Development to Test, and Test to Production, can create time-consuming delays to the product rollout.

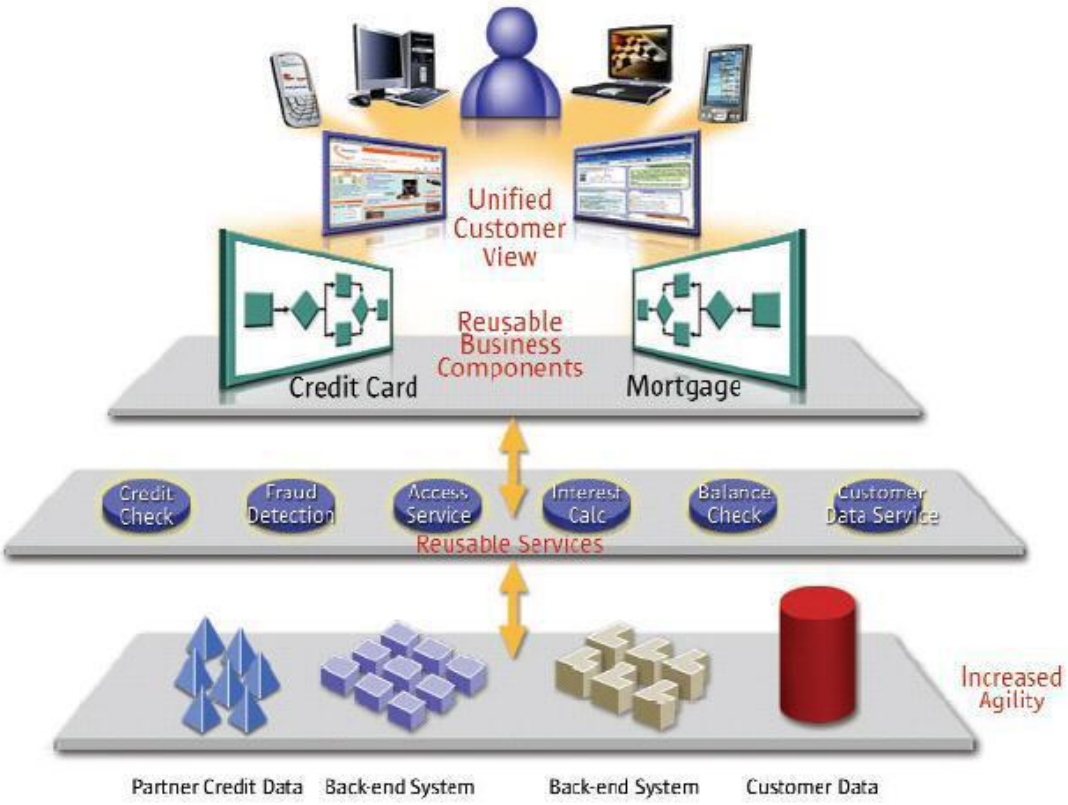

Figure 1 - Service Oriented Architecture (Image courtesy of ischool.tv)

Not only are the development pieces a problem, but a typical company has a number of these back-end services used by a wide range of departments, divisions, business applications, and products. Also, end users often interface with these services on branch kiosks, the web, or even their mobile phones.

As a result, Release Management then has the seemingly impossible challenge to make sure all elements are compatible, work together, and fulfill all essential requirements.

Faster software development cycle

The Agile methodology has changed the pace of software development, and established a new standard for a faster development and release life cycle. In many cases, Agile has cut the release life cycle time in half. Even companies that do not use Agile (or related methodologies) feel pressure from customers to release the product two to three times more often than before. The market now expects this kind of business speed, leaving IT departments scrambling to catch up. This faster pace often leads to production failures caused by not knowing all of the issues or applications involved with the deployment until the handoff from Development to Production.

Stricter requirements for auditing and reporting

IT organizations are increasingly under pressure to be compliant with laws and regulations. In the United States, the Sarbannes-Oxley Act of 2002 has created additional demand for traceability, separation of duties, and clear process. Separation of duties concerns has further broken up teams, ensuring that the developers and the deployment engineers are different people. As a result, there is greater need for a clear audit trail showing that the process was followed.

Problems

This section analyzes the challenges that today's Release Management team faces, including:

- Manual processes
- Different environments in Development and Production
- Audit failures
- Excess downtime due to slower builds
- Time consuming code and script errors
- Shipping delays

Manual processes

These days, even the most simple software development projects involve many handoffs between several different teams. When you combine multiple handoffs with the faster pace of development, it leads to errors.

These complex releases, which involve updating multiple components for a given application, have tasks that need to be done in a specific order, creating a long set of manual steps. Manual

deployments are generally executed by running through a checklist of steps defined in a document. They are fundamentally risky due to human error in following the checklist, or a chance that the checklist has not been updated, and so on. Running through this checklist can also be slow.

In addition, if a strict process is not established and enforced, the "informal" process of not properly checking in and out of changed code, or emailing files and solutions of active issues to other users may take root. Users may seek out solutions from others or outside the company for tools and techniques to help address the problems they encounter. For example: A developer performs a build on her desktop and hands it over to a tester to validate it. The tester then emails the files to Production for deployment. This is a Release Management nightmare. The build process is completely opaque and key steps can be missed. Worse, since the software is constructed on a developer's machine, a malicious developer could introduce arbitrary code that is not traceable to source control that performs any action.

Even the most diligent, well-intentioned user who is following a detailed deployment document or checklist can make mistakes and these mistakes can lead to delays. Some teams recognize that developer builds are too risky and designate engineers to perform builds on a controlled server using a set process. Although more controlled, this process can introduce additional bottlenecks and opportunities for miscommunication. Figure 2 shows how complex the development process can become.

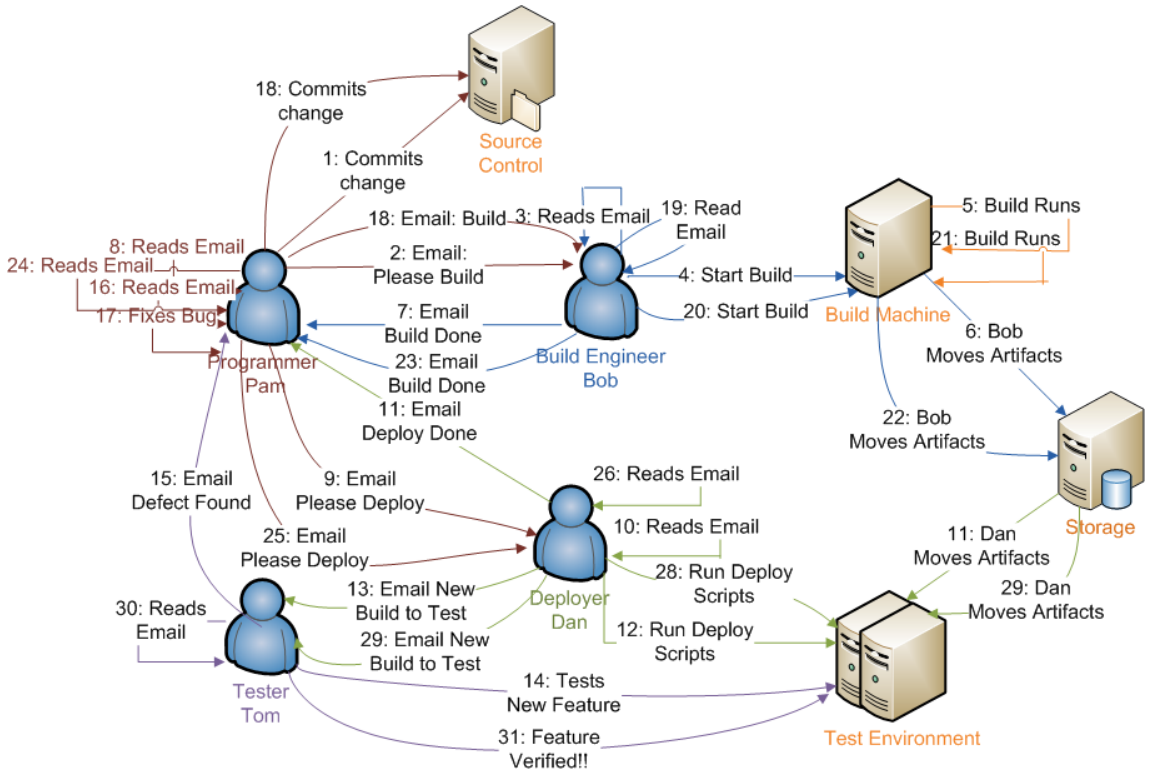

Figure 2 -- Manual Bug Fix & Retest Process Example

Different environments in Development and Production (Operations)

Complex software development has created a need for distinct, specialized teams to attack separate phases of the development process. As a result, the Development, Production and Test environments can all be different. Also, different teams may use different tools and configurations that best meet their specific needs.

Development and Production also have competing goals. Development's role is to create new, innovative products quickly, which requires a risk tolerant mind-set. Production's role is to make sure that new products do not break the customers' systems which lead to downtime. This requires a risk-averse mindset. These opposing roles lead to conflict between the two groups and often management unwittingly encourages further conflict by implementing bonus structures that causes a clash between these teams.

Audit failures

Although the presence of many handoffs, manual processes, and different development environments contribute to audit failures, the main cause of them is the lack of needed information to meet the new stricter requirements for regulatory compliance.

Release Managers need greater awareness of these production environment changes to meet these requirements. They also must have a clear audit trail so that they know where everything is, who did it, and what specific changes have been made.

Excess downtime due to slower builds

The demands of Agile development insist on fast internal code changes and builds; however, checks, processes, and testing (especially regression testing), must occur to prevent problems and to assure successful builds for Development.

While a slower pace can be tolerated for the rare event of a production deployment, it is painful in earlier testing environments, where waiting for deployments significantly delay the testing of new features and the identification of defects. If an organization wants more frequent, high-quality releases, then the organization needs to focus on improving the efficiency of testing.

Downtime is also a nightmare for the Production team. To reduce downtime, external builds need to be easy to roll back and be repeatable so the team can address urgent bugs and customer outages quickly.

Time consuming code and script errors

Some IT environments develop deployment scripts independently, to automate part of this checklist, but these scripts are often fragile and lack the ability to track what has been done and on which machine. As the applications evolve, the configuration settings change and these scripts become out of date and must be maintained. The overhead of maintaining these scripts causes the team to lose some of the productivity that was gained by creating the scripts in the first place.

When there are mistakes, they can be difficult to diagnose. The script debugging process has become expensive and time-consuming.

Scripts help the automation of effort, but they also don't provide the control over the console that identifies what actually went into production. This impacts the ability to do easy remediation and reinstallation. For example, if a situation where the configuration has drifted among various machines, the scripts don't help identify what change is causing the problem and resolve it.

Shipping delays

When an organization has delays in Development, Testing and Production, the result is often a missed delivery date.

Automation is the Answer

Implementing automation along the build, test, and production chain is a key part of the solution. It improves the speed and control of your development process. Some people see this as a tradeoff: If you gain speed, you lose control, or vice versa, but this is not the case. Automating manual processes and procedures improves both speed and control.

Automation helps meet the demands of shorter development and delivery cycles that companies need for a competitive advantage. Successful automation requires Development, Test and Production to work together to manage risk, limit end-to-end issues and minimize downtime.

Here are some of the ways that automation can be implemented in each phase of the SDLC.

Automation in Development

- Track code changes by identifying what has been added, what has been changed, who made the updates, and what tools were used.
- Generate rapid builds which can be rolled-back or migrated as necessary.
- Establish a release infrastructure to manage hardware, software, network connections, and so on for the entire SDLC.
- Use automation to establish quality standards that need to be met before turning the build over to Test.

Automation in Test

- Track code changes by identifying what has been added, what has been changed, who made the updates, and what tools were used.
- Use automation to establish and enforce standards that Test must meet before sending the release over to Production.
- Perform regression tests.
- Manage the test process to substantiate that the process has been followed.
- Manage the test environment so that it closely matches both the Development and Production environments.
- Generate fast builds to quickly implement bug fixes and updates.

Release Management in Production (Operations)

- Establish and enforce the final requirements checklist needed before the final release.

- Verify that the release infrastructure is still being followed and meets the requirements for Production.
- Control production scripts that used to be maintained manually.
- Generate builds and rollback builds quickly to minimize customer downtime.

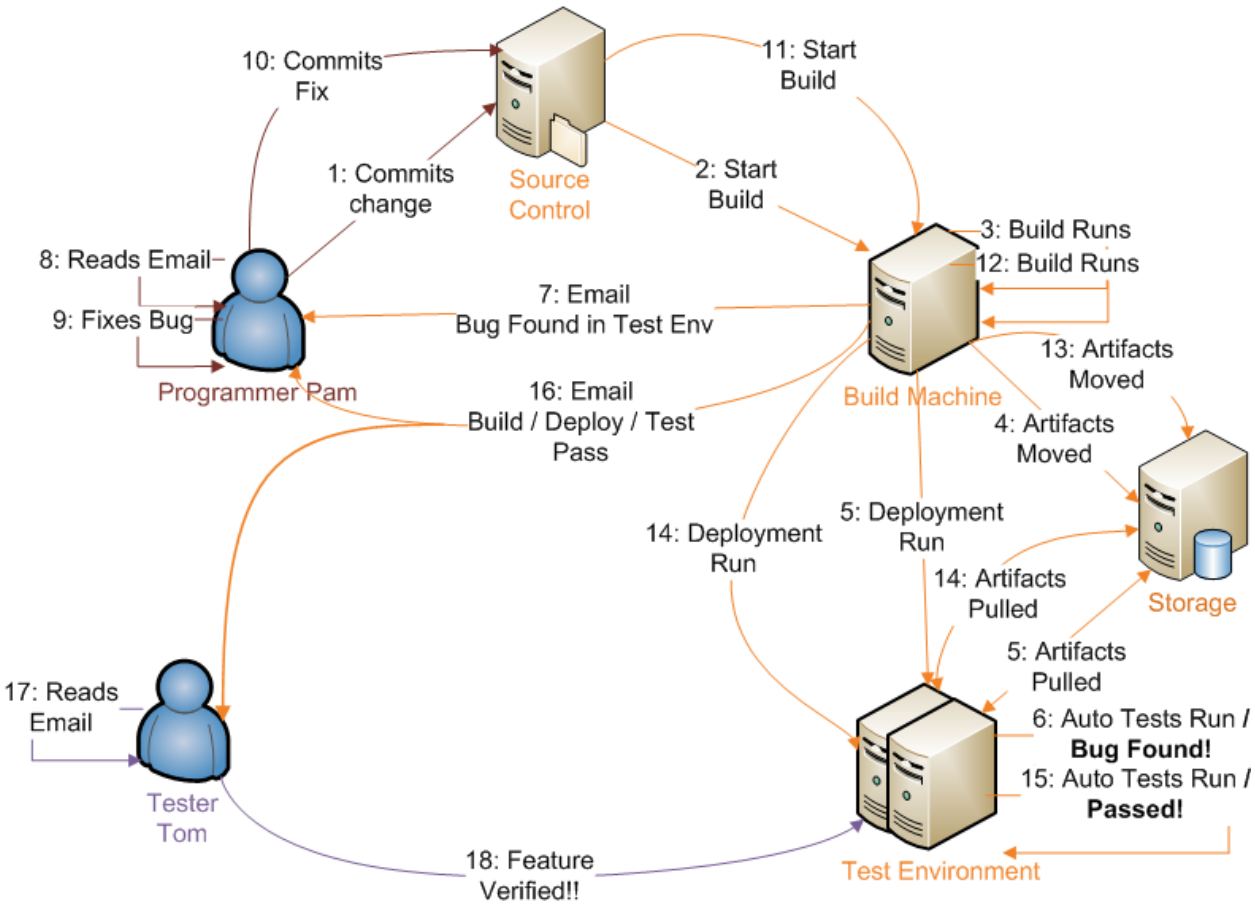

Figure 3 Bug fix and retest process with automation

What to Look for in an Automation Solution

The following sections provide detailed information about the criteria of a good automation solution.

Replacement of manual and time consuming processes

This is the heart of automation. Automation eliminates the need to rely upon error-prone manual procedures. These processes can be stored in a matrix, repeated and updated to meet the current needs of the application environments. The added benefit is that these processes can also be easily rolled back, migrated to new machines for load balancing, and duplicated to new machines.

Automation also enables you to generate rapid builds. Since the build processes have been defined already, the application blazes through the steps in the checklist the same way each and every time, which is helpful with time-critical emergency builds.

Automation is also perfect for testing, especially regression testing. Faster testing means faster implementation of new features and bug fixes. Engineers spend more time finding and resolving bugs instead of performing tedious and repetitive tests.

Extensive information tracking and reporting

The manual effort used to validate that the release requirements have been met limits the speed of a release. By automating data collection in the release management process, organizations can identify problems earlier, resolve them sooner (while they are less expensive to deal with), and help accelerate throughput within the SDLC.

The ideal automation solution captures all information about all components of all applications that are going into a given release. This allows organizations to forecast what the relationships are and which pieces need to move together. This helps with forward planning.

Detailed information about the software development process – This solution must pull all of the information together, integrating it to show how the data relates to this release. This information helps cut down on the overhead of compiling and coordinating that information manually (such as through a tracking spreadsheet) at the time of release.

This solution provides data for the following objectives:

- To prove that the quality goals have been met for each phase of the release
- To know which files have been changed
- To verify that the code being delivered matches expectations, and matches the files that have been planned

- To verify that only those components that were intended to be touched have been affected
- To know who did what and when
- To know that the established process was followed
- To display all test results for the release

Ideally, the solution can trace artifacts deployed to any environment (including production) all the way back from the exact source code. This solution also provides detailed compliance and audit reporting to meet the needs of today's exacting audit requirements.

Real-time visibility – It should also provide ongoing visibility for all items in the release, as the components migrate from one environment to another. This identifies problems, such as areas with a great deal of code churn, so that they can be analyzed and addressed earlier in the process.

Customized milestone and tracking alerts – These help keep the team coordinated and allow everyone to see approximately when the team will be entering different phases of development and provide checklists of what requirements need to be achieved before reaching that development milestone. Tracking alerts generate notifications when a component is not meeting its dates so that these issues can be addressed, as opposed tracking it through manual processes.

Deliverable tracking for multiple teams – For situations where several teams are working on components that need to be released together, this component information must be tracked to make sure that:

- If one of the teams' schedule changes, does it impact the other teams?
- Do any of the components being worked on by one team impact any of the components of the other?

Bridging the DevOps Gap Deployment Automation

When it is time to deploy the release to Production, automation really saves time. Where before there was a lot of overhead to identify and fix the errors that occurred during the roll out to Production, the ideal automation solution reduces the time it takes to deploy and the number of errors found.

By automating the checklist so that it is run the same way every time in each environment, the release team eliminates much of the risk inherent in the manual process. This frees engineers to perform less rote tasks, and can be provided as a push-button service to testers and developers who want to perform frequent deployments to their test environments. In addition,

since the same process is used in all environments, deployments to the live environment have not only been tested extensively, but the deployment process has been tested as well.

Summary

Automation saves your organization time and money. It outsources tedious, error-prone tasks and frees up engineers to do what they do best: develop new features and resolve difficult problems. Automation provides additional visibility and accountability to the auditing process and identifies possible problems before they occur, when they are less expensive to deal with. Automation can reduce the tension between teams and help them work together as a cohesive unit.

With the pressure to cut costs in the current economic climate, can anyone afford to not look into automation as a possible solution?

Contact

For more information about how urban{code} can improve your release management process and implement our suite of automation solutions, contact Greg Wunderle at gaw@urbancode.com or give him a call at *(216) 858-9000*.