Compliments of

Continuous Testing

for dummies®
A Wiley Brand

IBM Limited Edition

Learn continuous testing practices

Test smarter with DevOps

Adopt continuous testing

Marianne Hollier
Allan Wagner

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Continuous Testing

IBM Limited Edition

by Marianne Hollier and Allan Wagner

for dummies®
A Wiley Brand

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Continuous Testing For Dummies®, IBM Limited Edition

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

Copyright © 2017 by John Wiley & Sons, Inc.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permissions.

Trademarks: Wiley, For Dummies, the Dummies Man logo, The Dummies Way, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries, and may not be used without written permission. IBM and the IBM logo are registered trademarks of International Business Machines Corporation. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, or how to create a custom *For Dummies* book for your business or organization, please contact our Business Development Department in the U.S. at 877-409-4177, contact info@dummies.biz, or visit www.wiley.com/go/custompub. For information about licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com.

ISBN: 978-1-119-36583-9 (pbk); ISBN: 978-1-119-36584-6 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

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

Project Editor: Carrie A. Burchfield

Editorial Manager: Rev Mengle

Acquisitions Editor: Steve Hayes

Business Development Representative: Sue Blessing

IBM Contributors: Bernie Coyne, Jennifer Moore, James Hunter, John Chewter

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Table of Contents

INTRODUCTION 1
About This Book 1
Icons Used in This Book 2
Beyond the Book 2

CHAPTER 1: **Defining Continuous Testing** 3
What Is DevOps? 4
Why Test Continuously? 6
Testing Is Costly 7
The Need for Quality and Speed 9
Finding the Right Set of Tests 10

CHAPTER 2: **Looking at the Key Elements of Continuous Testing** 13
Managing Defects 14
Managing Tests 15
Automating Tests 16
Analyzing Effort 18
Creating Test Environments 18
Virtualizing Dependent Services 19
Gathering Test Data 20

CHAPTER 3: **Testing Smarter** 21
Looking at Current Testing Challenges 21
Shift Left 23
Shift Right 24
Testing across Industries 25

CHAPTER 4: **Adopting Continuous Testing** 27
Finding the Path to Achieving Continuous Testing 27
Determining Where to Begin 29
Identify bottlenecks 29
Determine areas with high return on investment (ROI) 30
Processes in Continuous Testing 31
Making Continuous Testing Work: One Scenario 32
Technologies in Continuous Testing 35

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

CHAPTER 5: **Ten Continuous Testing Myths** 37
Continuous Testing Is Only Executing Test Scripts 37
Continuous Testing Is Just a Fad 38
Continuous Testing Is Only for Agile Teams 38
Continuous Testing Is Only for Testers 39
Continuous Testing Isn’t for Regulated Industries 40
Continuous Testing Isn’t for Large, Complex Systems 40
Continuous Testing Isn’t Part of DevOps 41
Continuous Testing Isn’t for Cloud or Hybrid Cloud Applications 41
Automating Tests Means We Need Fewer Testers 42
Quality Is the Test Team’s Responsibility 42

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Introduction

Welcome to *Continuous Testing For Dummies,* IBM Limited Edition. We hope you find this material helpful as you begin or continue your journey toward higher quality software and faster delivery. Successful companies are honing their software development approach to be as efficient as possible. This allows the software development, testing, and ops teams to spend their time on innovation instead of rework and manual, error-prone tasks.

Software testing can be a bottleneck or a competitive advantage, depending on your approach. Many IBM clients are adopting the right capabilities and best practices to achieve continuous testing that supports their DevOps transformation. And they're seeing solid improvements in their speed and quality.

The goal of taking an IBM DevOps approach is to accelerate software delivery while at the same time balancing speed, cost, quality, and risk all with a goal of improving the client experience — no small task, as you may expect.

And when the testing finds that the code isn't up to snuff, it can have a devastating effect. If an announced date is missed or the quality of the release is compromised, you can damage your brand's reputation or, even worse, lose customers. But when all your development, testing, and ops teams are focused on testing earlier and more often, they'll be able to consistently deliver quality software in a timely manner. And you can grow your market share while delighting your customers.

About This Book

In this book, we share how IBM software and best practices can help software development, testing, and ops teams adopt a continuous testing approach. Continuous testing means adopting the right set of automated tests along with service virtualization, which allows the team to simulate missing services and environments so they can start testing earlier and more frequently.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

By becoming more efficient and effective, teams can reduce their costs as well as decrease the time it takes to get high-quality, innovative software to users. You can't test everything, and you can't automate all your tests, so in this book we share how to find the right balance. You also discover the key concepts for enabling your software development team to test earlier, or *shift left*, so the team improves software quality and gathers feedback faster than ever.

Icons Used in This Book

You'll find several icons in the margins of this book. Here's what they mean.

The Tip icon points out helpful information on various aspects of continuous testing.

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

The Warning icon alerts you to critical information.

Technical Stuff material goes beyond the basics of continuous testing. It isn't essential reading, however.

Beyond the Book

You can find additional information about continuous testing and IBM's DevOps approach and services available by visiting the following web pages:

- **IBM Continuous Testing (solutions):** `http://developer.ibm.com/testing`
- **IBM DevOps (website):** `http://ibm.com/devops`

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

» Explaining DevOps

» Seeing the need to test continuously

» Understanding why testing is costly

» Delivering quality and speed

» Identifying the most important tests to run

Chapter 1
Defining Continuous Testing

You may have heard the term *continuous testing*, presented as one of the DevOps (short for development and operations) practice areas, but are struggling to understand what it is and how to achieve it. Like continuous integration and continuous deployment are adopted to remove bottlenecks in the delivery pipeline, project teams need to improve their ability to test each valid software build as it becomes available. Continuous testing relies on test automation integrated as part of a deployment process where software is validated in realistic test environments. Adding service virtualization to the mix allows teams to *shift left*, which means to begin checking software quality earlier in the life cycle, by simulating dependent yet unavailable software and systems.

Service virtualization simulates the behavior of select components within an application to enable end-to-end testing of the application as a whole. Test environments can use virtual services in lieu of actual services or systems to conduct integration testing earlier in the development process.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Combining test automation with service virtualization enables teams to test applications end to end providing immediate feedback on quality — so issues can be resolved earlier and at a lower cost. Encouraging and embracing this immediate feedback on the quality of the solution enables businesses to predict delivery with greater accuracy and deliver high-quality innovative solutions to the market quickly.

Continuous testing helps project teams execute tests when needed, not when possible.

What Is DevOps?

The practices that make up DevOps are a broad set of capabilities that span the software delivery life cycle, and continuous testing is one of these practices.

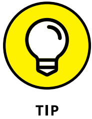
DevOps, like most new approaches, is only a buzzword for many people. In broad terms, DevOps is an approach based on lean and agile principles in which business owners and development, operations, and quality assurance departments collaborate to deliver software in a continuous manner that enables the business to more quickly respond to market opportunities and reduce the time to include customer feedback. However, opinions on what DevOps is and how to use DevOps differ greatly.

Some people say that DevOps is for practitioners only; others say that it revolves around the cloud. IBM takes a broad and holistic view and sees DevOps as a business-driven software delivery approach — an approach that takes a new or enhanced business capability from an idea all the way to production, providing value to customers in an efficient manner and capturing feedback as customers engage with the capability. To do this, you need participation from stakeholders beyond just the development and operations teams. A true DevOps approach includes lines of business, practitioners, executives, partners, suppliers, and so on.

The DevOps movement has produced several principles that have evolved over time and are still evolving. Several solution providers, including IBM, have developed their own variants of the principles which are

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

- Develop and test against production-like systems
- Deploy with repeatable, reliable processes
- Monitor and validate operational quality
- Amplify feedback loops


TIP

To help you get started and then scale your solution, there are some public DevOps methodologies available to help. IBM's Bluemix Garage Method is one example that breaks down DevOps into teams and roles. It includes how-to guides on culture, best practices, tools, self-guided or hands-on training, and even sample code and architectures. You can find out more information at `http://ibm.com/devops/method`.

A successful organization is centered on a culture of innovation. The IBM Bluemix Garage Method shown in Figure 1-1 is divided into seven phases. Each phase includes a set of practices and tools to help you achieve your DevOps transformation goals. The phases are arranged in a repetitive cycle where the team continually gathers more intelligence on the application and its usage and iteratively improves the application . . . and the process itself.

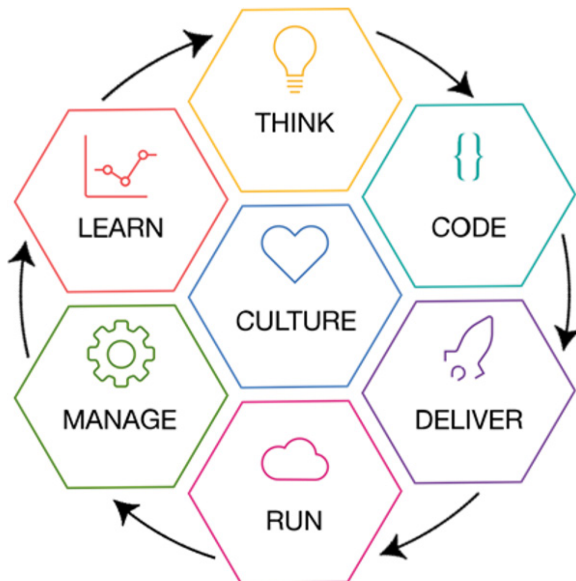


FIGURE 1-1: The IBM Bluemix Garage Method.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

EXPLORE CONTINUOUS TESTING SOLUTIONS

Find out more information about IBM continuous testing solutions by visiting `http://developer.ibm.com/testing`. This site is the IBM continuous testing community where you can explore plenty of information on all aspects of continuous testing, including IBM products, events, news, forums, blogs, videos, eBooks, how-to guides, product documentation, downloads, whitepapers, and the free starter editions of several IBM testing tools.

Wondering why you don't see Test explicitly listed as one of the phases? After all, this is a book about continuous testing, right? In an agile world, programmers and testers are both considered to be developers — full-fledged members of the delivery team. And as test automation requires the same disciplines as software development where developers build, manage, and maintain functioning software in the form of test automation assets, it is combined under the coding phase.

Why Test Continuously?

Today's customers demand change. They want functionality, which satisfies their needs; they want new releases delivered with speed; they want high-quality software, and they want their voices to be heard when planning for the next release. Today's end-user is the final tester and determines the financial success of the application. These demands require businesses to deliver software that meets the needs of today's consumer and deliver it with speed.

Having the ability to get feedback on software quality throughout the entire development life cycle helps business leaders predict with greater accuracy. Shift left and *shift right*" (potentially) means doing things faster and more efficiently, including getting software into the end-user's hands sooner than ever before.

Shift right is deploying software to the end-user that may not be fully tested or development may not be complete, in order to close the feedback loop earlier. Find out more on shift right in Chapter 3.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

In a nutshell, continuous testing means that organizations don't have to wait for all the pieces to be deployed before testing can start. Continuous testing is checking the software through a variety of techniques — including the execution of unit tests authored by programmers — with each and every build.

The sooner and more frequently an organization can get feedback on software quality the sooner it can react to architectural flaws, poor design decisions, erroneous application functionality, security vulnerabilities, and scalability issues. Continuous testing is the practice where this capability can be a reality.

Testing Is Costly

It should be no surprise that for many testing is a necessary step in their software development life cycle. Testing is also an area under constant scrutiny because testing software isn't cheap and is often a source of delay in trying to get software to market. So, why is testing so costly? Well, you don't have to look any further than the movement toward service based architectures, microservices, and container technology. New approaches to application development and adoption of cloud technologies challenge your delivery teams as they need to validate both the new innovative application front ends while ensuring the systems of record maintained in the legacy systems remain safe and secure.

Software development and delivery is evolving from complex, monolithic applications, whose many dependencies are resolved at build-time, toward a more distributed, service-centric architecture whose dependencies can be resolved at runtime. Most enterprise applications are a combination of existing applications originally designed for a pre-cloud environment (also called *systems of record*) and new systems of engagement applications developed for the cloud. Their architectures tend to be complex due to their many dependencies, and they use application program interfaces (APIs) to bridge between the new systems of engagement and the existing systems of record. They leverage API management and cloud integration technologies to enable integration while addressing the organization security requirements. Their workloads can run across multiple environments: on-premises, private cloud, public cloud — the combination of which is an architecture also referred to as *hybrid cloud.*

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

TESTING IMPROVEMENTS DELIVER SIGNIFICANT SAVINGS AND A COMPETITIVE EDGE

Because of potential delays or bottlenecks related to testing, it is an area ripe for increasing efficiency. The average spend on quality assurance, as a percentage of the total IT budget, is projected to rise to 40 percent by 2018. According to the World Quality Report, costs are aggravated by the fact that many organizations spend more than a third of their testing budget on test environments. So, efforts need to be focused on process improvement and test automation.

Why? Competitive advantage comes from being better, cheaper, faster — better quality, lower cost to produce, faster and more frequent deployments. Get it right and you can disrupt the competition. Offering a product that is superior in quality and developed at a lower cost to the business (and also offering frequent improvements) can be the difference between growing your business or going out of business.

REMEMBER

Hybrid cloud architecture is becoming the norm for both cloud-enabled and cloud-native applications. In fact, IBM released a white paper in 2016 that stated that hybrid cloud provides flexibility in deployment, enabling organizations to choose the right platform to run their workloads. And the International Data Corporation (IDC) predicts that 80 percent of enterprise IT companies will commit to hybrid cloud architectures by 2017.

For many organizations who are unable or unwilling to adopt a shift-right approach, testing simply costs what it costs as they can't afford to have buggy or non-compliant software escape into production. To decrease the cost of testing, organizations are working to adopt test automation and develop frameworks to increase testing efficiency. While test automation can certainly help in testing efficiency, one must not overlook the fact that creating test automation assets is development — an effort that requires management, resources, and funding and doesn't necessarily decrease testing costs. But the speed at which software functionality can be validated and delivered into the hands of the customer to increase market share, disrupt the competition, and establish leadership in a new market could offset the cost of any investment.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

AND THE CLOUDS COME ROLLING IN

Another piece of the testing puzzle comes with the new hybrid cloud applications — where part of the application is on-premise and part is hosted in either a private or public cloud, or both. From a testing perspective, as long as testers have access to those systems, the automation tools needed for continuous testing (test automation, service virtualization, deployment automation, and so on) can do their robotic magic as well. Understanding this requirement at the outset of a project will help prevent barriers to achieving continuous testing.

The Need for Quality and Speed

Business stakeholders demand project teams deliver new applications, integrations, migrations, and changes as quickly as possible. In turn, project teams ask testers to validate the following:

- Technical environments work as required
- Business processes and transactions run as expected
- Solutions scale to the expected usage
- Applications are secure and user data is protected

Regardless of the industry or technologies involved, a careful balance needs to exist between speed and quality. With the increased acceptance of cloud technologies, software teams have more tools than ever at their fingertips to make their work more efficient. However, while software and system programmers do the best they can to avoid making mistakes, human errors are inevitable — which is why you test in the first place. The risk created by *not* testing greatly outweighs the cost of performing even a small number of tests.

TECHNICAL STUFF

A recent survey conducted by Application Development Trends found that testing is by far the number one reason for delays when deploying to production. Using analytics and testing insight, IBM focuses on optimizing the test effort and the related deployment operations so more resources remain available for innovation.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

With the increasing adoption of Agile and DevOps practices, re-executing manual tests in every iteration is not a sustainable pattern. There is never enough time, and adding more personnel to execute manual regression tests results in diminishing returns. Even more importantly, the slow speed of feedback to the programmers decreases productivity significantly. Test effectiveness is a critical aspect to keeping up with the faster-paced development life cycles. It is optimized by running the fewest number of tests that find the largest number of problems. Even teams working in more traditional development life cycles, where all the testing is performed in a single phase, have found that they can't keep up with the regression testing each time they get a new build — with defect fixes, changes to existing features, and even new functionality all bundled into the new build.

Figure 1-2 shows that after just a few iterations, the quantity of new features, and therefore number of tests, increased significantly.

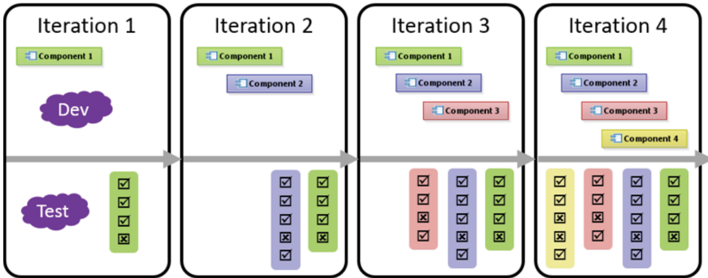


FIGURE 1-2: Test accumulation.

The only way to keep up with the needed regression testing is to automate the right set of tests to ensure the change hasn't impacted existing functionality. To accomplish this balance, mature DevOps teams use a combination of test automation and manual exploratory testing, both running in a continuous pattern.

Finding the Right Set of Tests

Attempting to automate all tests is impossible, and the cost of trying to do so eventually outweighs the benefit. For any reasonably complex application, you can't test every possible path

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

through the system because even if there is only a single loop in the application, the number of possible paths becomes infinite. If you then add in the test data permutations, you quickly realize that attempting to test everything is just not feasible.

REMEMBER

The key is to identify the most important subset of tests — the ones where you

- Typically find issues
- Have seen regressions
- Have customer complaints
- Know the occurrence of a failure would be significant or even catastrophic

Impact analysis of the new code changes is a critical aspect in identifying which regression tests to run. But without good change set input, the data and analysis of the code changes can be misleading. This analysis of what are the *right* tests to automate should involve the entire team, from business to development, to test, to operations, and support. Each role brings a different perspective on where things can and do go wrong, so it's important to include everyone.

Some examples of where test automation can be applied include the following:

- **Data-driven tests:** Tests where the data sets are complex and cover critical aspects
- **Business logic validation:** Scenarios that ensure the correct results are achieved
- **Integration with a third-party system:** Tests where programmers and testers might not have full access to that third-party system
- **Low-intensity performance testing:** Running small-scale stress, load, volume, or memory leak checks across builds to identify degradations early, well before conducting formal load testing
- **Verifying error scenarios are handled:** Ensuring that applications behave consistently and correctly when external dependencies are in error

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

- **Installation and upgrade of customer-installed software:** Testing across the many platforms and operating systems for commercially available software
- **Scanning for security vulnerabilities:** Important tests when financial or personal information could be at risk

Test automation implementation comes in all shapes and forms; some key examples are

- Unit testing
- Functional testing at the user interface (UI) layer
- Functional testing via APIs
- Performance testing, via the UI or API
- Security testing

In addition, there are tests that are extremely valuable when conducted manually. Continue these tests in parallel with test automation:

- **Exploratory testing:** Unscripted tests where the tester analyzes different aspects of the system without a pre-scribed end result. This helps find new scenarios that carry defects but are not yet covered by automated tests.
- **Usability testing:** End-users are asked to test specific aspects of the system and give verbal feedback as they progress. This allows the team to better understand what users are thinking when they use the system.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

- » Managing defects
- » Looking at test management
- » Understanding how to automate tests
- » Analyzing your efforts
- » Provisioning test environments
- » Increasing testing efficiency with service virtualization
- » Having the right sets of test data

Chapter 2
Looking at the Key Elements of Continuous Testing

Building a continuous testing culture requires people, practices, tools, and time. Finding the right balance of effort across all testing practices is critical in achieving continuous testing. But before you start deploying code for testing, one activity often overlooked is the code review. Everything that will be included in a build and used during deployment to an environment (testing or production) should be reviewed by a team of experts assembled for this purpose.

Code reviews need to be efficiently run and deemed to be effective; otherwise, the team may view them as a waste of time. The review process should confirm that the new and modified software is following the organization's coding standards and adheres to its best practices.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

With the code successfully passing the review step, Figure 2-1 shows the key elements that enable continuous testing, and teams need to spend effort on each of these practices — whether or not they want to.

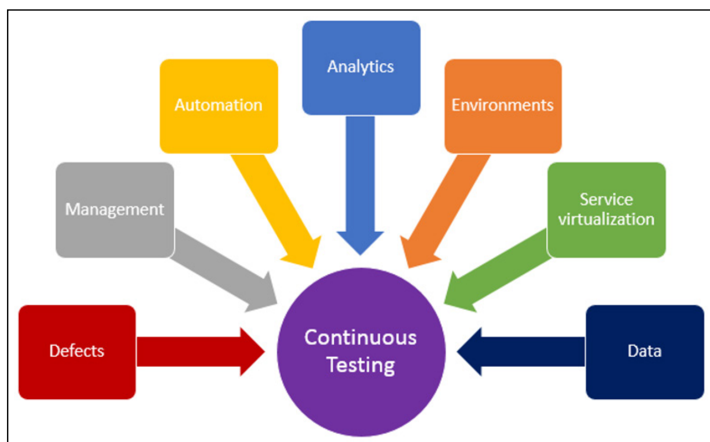


FIGURE 2-1: The important practices of continuous testing.

The sections in this chapter explain each element in Figure 2-1 and give you questions that may help your teams understand where they're spending time and perhaps where their time might be better spent.

Managing Defects

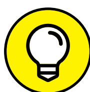
When applying a traditional testing approach, defects are the initial communication channel between testers and programmers. Testers think "I found a bug!", and programmers then agree or disagree that there is a problem with the code. Many times the defect isn't a problem with the code itself but with the requirements, design, or even the test. Sometimes, the problem lies outside the application itself — in the test environment, the test data, the test script itself, or some combination of these things.

Having a collaborative environment to log and track defects is essential in any software development life cycle, but managing defects is just the tip of the iceberg when it comes to adopting the right set of practices.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

When analyzing your level of effort that you spend managing defects, ask yourself the following questions:

- Are we spending too much time logging, triaging, or analyzing defects?
- What about time we spend on defects that aren't "real" defects, such as where there's a misunderstanding between the test and the code?
- What if we could prevent entire schools of defects from ever being created in the first place?


TIP

It is important that a definition or understanding of what's considered a defect and what's considered a request for enhancement (RFE, also known as a *change request, enhancement request,* and so on) is shared across the entire delivery team. Often personal opinions or preferences surface as defects, which should really be treated as future enhancements. If you're only submitting defects and not submitting enhancements, that can drive a wedge between programmers and testers where collaborating on the best way to deliver an RFE can lead to better teaming.

REMEMBER

We differentiate between defects and enhancements as follows:

- **Defect:** An issue where the software under test doesn't match the application requirements
- **Enhancement:** An issue where the software under test doesn't function as expected and there's no application requirement that captures the expected behavior

Managing Tests

One of the next practices that test teams typically adopt is test management, which includes the following:

- Planning the testing effort
- Identifying needed tests
- Creating test cases and scripts
- Gathering existing tests

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

- Executing the tests
- Identifying redundant, duplicate, and overlapping tests
- Tracking and reporting on the progress of testing

This test management practice helps both teams and management understand at a glance if tests are passing, failing, or being blocked.

Questions to ask yourself in this area include

- Are we spending time manually crafting status reports and rolling up test execution results?
- Do we have a tool that provides test execution results in real-time and allows us to drill down as needed?
- How do we know if we are on schedule for our test effort (or behind, or even ahead!)?

Automating Tests

Test automation practices typically come after managing tests (see the preceding section), when teams discover that running all their tests manually is inefficient, ineffective, and, in many cases, downright impossible. Successfully creating a robust and maintainable test automation framework can be difficult if not approached as a software development project itself. There needs to be a shared vision, requirements, architecture, design, coding in some instances, and validation that the automation does what was intended. Without these aspects, test automation frameworks tend to be fragile, brittle, difficult to maintain, costly to refactor, and frequently abandoned.

IBM advocates automating testing at all layers of the application, including the component layer, service layer, and user interface layer. It is important to find the right balance of tests across those layers, and the industry is finding that the percentages shown in Figure 2-2 are a good starting place — depending on your application.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

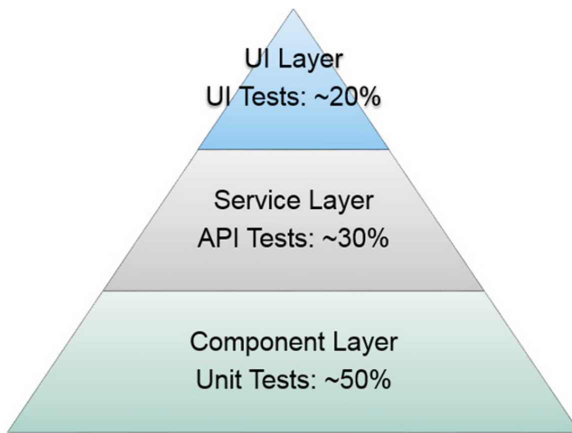


FIGURE 2-2: Recommended test automation percentages.

When determining or reviewing your test automation strategy, ask these questions:

- How efficient are we at re-executing existing tests?
- Do we run most or even all of the tests manually?
- If we automated tests, are we focused only on functional tests at the user interface layer, or are we running unit, functional API-layer tests, performance tests, and even security tests?
- Do we have a robust and maintainable test automation framework?
- Are we incorporating automated tests into the delivery pipeline?

TIP

As programmers write the code, doesn't it make sense to have them run the first round of unit tests? Executing unit tests to validate that programmer's code works properly in their development environment would go a long way to building trust across the delivery team. So, why is it that so many organizations aren't enforcing this as a policy? Ignoring best practices that could (and we argue, *should*) be delivered much earlier in the software development life cycle (SDLC) often results in unnecessary delays in later stages and increased costs. If you want to deliver software faster than ever before while decreasing risk to the business, every team member, including programmers, shares a responsibility in contributing to the quality of that software.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Analyzing Effort

Alongside test automation, test analytics and insights practices start to grow to help the teams know where to spend their precious testing effort. Code change impact analysis can be a difficult task, especially if the programmers aren't rigorous and consistent with their code change sets. Understanding what has changed from the last build is critical for selecting the right sets of tests to run. Defect density versus test execution coverage in specific areas of the application can provide insightful data to identify root causes of defects. However, analyzing that data can be time consuming. This is where analytical services can help. Without that analysis, over-testing — or retesting everything — is a tempting but costly answer.

When analyzing your testing efforts, think of the answers to these questions:

- How do we know what tests to run, and when, and even why we're running those tests at those times?
- How good is our test effectiveness? Are we running the fewest number of tests that find the largest number of problems?

Creating Test Environments

Another critical aspect of being able to test continuously is automating the provisioning of test environments. Standing up production-like test labs to meet the demands of modern software delivery teams requires four main components: dynamic infrastructure, deployment automation, test data management, and service virtualization. And if all the moving pieces can be brought together with orchestration making the process repeatable, reliable, and traceable, the benefit potential is huge! Automated creation and configuration of test environments decreases the time it takes to start testing a new build from hours, or even days or weeks, to minutes. This automation also reduces the number of false errors due to test environment issues, incorrectly installed dependent software, and other manual processes that introduce problems. In the IBM DevOps approach, test automation goes hand-in-hand with deployment automation.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Ask yourself these questions:

- Are we constantly waiting on our test environments to be provisioned and configured properly?
- Do we run tests and discover after the fact that our test environment wasn't "right," so we have to fix the environment and then rerun all the tests again?
- Do we hear from programmers "it works on my machine!", but it doesn't work in our test environment?

Virtualizing Dependent Services

In conjunction with the test execution and test environment automation, service virtualization, or *stubbing,* greatly increases testing efficiency. By creating virtual services from known and agreed-on interfaces, programmers and testers write code and test against the same interface, even when that dependent system isn't available. By deploying the application components that are available and virtualizing those that aren't, testing can begin much earlier and run much more frequently on every build. Service virtualization also allows testing of scenarios that might not be readily tested with a live system. For example

- Exceptions and errors
- Missing data
- Delayed response times
- Large volumes of data or users

Incorporating service virtualization into the overall test effort enables teams to build and effectively test the riskiest parts of the system earlier, instead of just building and testing the easy-to-test parts of the system. When you then automate tests for those risky parts of the system and execute them on each build, you get better coverage of those parts. This ensures regressions are identified as quickly as a new build is made available.

A typical scenario might be that mobile and web development teams work on cloud applications, and mainframe teams work on-premises. With service virtualization, the environment

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

dependencies in these hybrid cloud scenarios are decoupled. Teams then build and test at their own speeds and adopt the right DevOps approach that fits with their culture.

When virtualizing dependent services, ask these questions:

- Are we waiting for dependent systems to become available before we can "really" test?
- Are we using a "big bang" approach to conduct end-to-end system testing, where we throw all the systems together and hope they work and interact properly?
- Can we test exception and error scenarios before we get to production?
- Are we testing the easiest parts first just because they're available and delaying the high-risk areas for the end of our testing effort?

Gathering Test Data

Another aspect of testing that increases test effectiveness is having the right sets of test data. Using test data that is as production-like as possible provides better coverage with more scenarios. Extracting right-sized data from a production environment and automatically masking it for security purposes accelerates the availability of realistic test data. A good set of test data also helps the team create exception or error situations that might be difficult to test otherwise.

When gathering test data, think about the answers to these questions:

- Do we have the needed sets of production-like test data to ensure we are covering the right test scenarios?
- Are there exception and error scenarios that we can't execute because we don't have the right sets of test data?

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

» Recognizing the current challenges in testing

» Understanding the shift left concept

» Knowing when shift right is useful

» Approaching testing in all industries

Chapter 3 Testing Smarter

Have you ever been told, "Test smarter, not harder!"? Unfortunately, test and development managers send this message to their teams frequently. But, they don't provide guidance on how to accomplish this smarter testing. Teams are so busy trying to keep up with their existing test workload that they don't have time to figure it out for themselves.

A key aspect of testing smarter is to test earlier and more often, or shift left, in the delivery life cycle. This way, the team can test the riskiest elements early, and those tests can then be continuously reused. This smarter approach provides early, iterative feedback on code quality directly to development teams to ensure that fewer problems are found late in the life cycle where they're more expensive to fix.

This chapter describes current testing challenges as well as approaches that may help, including shift left, shift right, and industry-specific concepts.

Looking at Current Testing Challenges

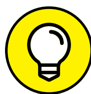
They say that money and time can solve most problems. However, those are exactly the two major challenges testers face (and also maybe figuring out who "they" is). There is never enough

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

time, and with increased pressure to accelerate delivery, testers need to find ways to increase efficiency in how they check application functionality, performance, and security. And how often, in recent times, have you heard of organizations increasing testing funds — something that's already viewed as being too expensive. So, we discuss each challenge individually and how to uncover a resolution to these challenges.

There never seems to be enough time. In a traditional waterfall approach to software development, requirements gathering, analysis, design, and development always seem to take more time than what was allotted. And with the release date holding strong, the expectation is that testing needs to get things done in whatever time is left. Unfortunately, things often slip, and steps are missed. In agile or iterative shops, testers are often testing a previous sprint's deliverables while the programmers are working on developing the next sprint — putting the delivery team out of sync. Then there's the biggest bottleneck: the exorbitant amount of time spent standing up test labs and creating test data, deploying the bits, or waiting for help from others. Test teams spend days and even weeks waiting for dependent systems, test environments, and new builds.

Money is also a challenge. Businesses are looking for ways to save while the cost of testing continues to skyrocket. Teams can free up budget by becoming more efficient so savings can be reinvested to add valuable features, increase productivity, and even decrease time to market.



TIP

So, how do you begin to address these time and money challenges? Here are some tips:

- Start by focusing on testing the high risk business transactions to ensure they are working properly as early as possible and to protect the business.
- Consider leveraging cloud technology to quickly stand up and tear down test environments on demand and embrace infrastructure as code to provision environments rapidly, while turning capital expenses (CAPEX) into operating expenses (OPEX).
- Look at deployment automation solutions that can
  - Continuously move application changes through the many test labs en route to production

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

- Instantiate virtual services to "mock" dependent yet unavailable software and systems
- Reset test databases
- Kick off the execution of automated tests orchestrating a repeatable, reliable, traceable process

These points are all proactive steps that teams can take in adopting a shift left approach to testing, which we discuss in the next section.

Shift Left

The term *shift left* refers to a practice in software development in which teams focus on building quality in, work on problem prevention instead of detection, and begin testing software earlier than ever before. The goal is to increase quality, shorten long test cycles, and reduce the possibility of unpleasant surprises at the end of the development cycle — or even worse and scarier, in production.

In many organizations, automated testing of today's composite applications is being executed via the user interface after the complete application has been developed and deployed to a test environment. Waiting for all the application components to become available before testing commences, however, often causes delays, adds risk to the project, or results in discovery of late-stage and architecturally significant defects — when they're more expensive to fix.

Shift left practices help avoid rework, delays, and churn that can occur when major defects are discovered late in the testing cycle — after all integrations and product components are finally brought together as a composite application and made available for the team to test. It aims to avoid these issues by performing integration tests as soon as the code is deployed in a realistic production-like test lab. If any of the dependent application components are not available to test, virtual services can mimic the real components' behavior until they're ready.

Figure 3-1 shows an example of a shift left testing solution by combining IBM Rational Test Virtualization Server and IBM UrbanCode Deploy to work in conjunction to provision

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

production-like test environments no matter where in the life cycle you are. For more detail on this shift left solution, visit `https://developer.ibm.com/urbancode/products/urbancode-deploy/features/shift-left`

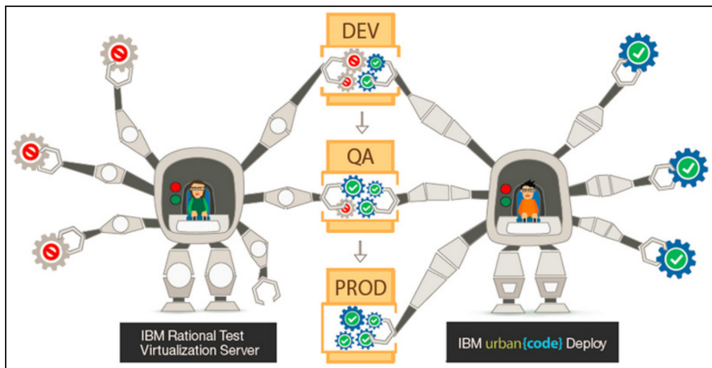


FIGURE 3-1: Shift left with virtual services and automated deployment.

Shift Right

Shift right describes an organization's willingness to carry a level of risk in production by deploying software to the end-user that may not be fully tested or development may not be complete. This is often done to accelerate capturing user feedback on software quality or to hear what users have to say about the new functionality being delivered. By leveraging deployment strategies like canary deployments, performing A/B testing, or inserting business toggles into the code base to manage the scope of functionality to the end-user community, businesses can reduce their risk of exposure. The value lies in software getting into the hands of the end-user as quickly as possible and obtaining feedback faster. Shift right practices allow organizations to leverage the feedback, perform sentiment analysis, and predict outcomes with greater accuracy.

TECHNICAL STUFF

Canary deployment is a technique to reduce the risk of introducing a new software version in production by gradually rolling out the change to a small subset of users before incrementally making it available to everybody.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

In the testing domain, shift right is often associated with testing in production. Can you think of any better audience in your world who will provide truthful feedback on new functionality during A/B testing than the "real" end-user? You want to manage risk, but in this new DevOps approach to speeding software delivery, the reward probably outweighs the risk.

Shift right testing also means that you're testing the software in an actual production environment. This provides companies with the ability to observe and measure the stability and scalability of the application as well as capture real usage patterns — how the end-user actually uses the application. In fact, some organizations will even inject performance testing robots or probes into their production environments or use an application performance monitoring solution to do synthetic analysis, monitoring end-user response times from locations around the world.

The shift right practice is a great way to accelerate delivery and close the feedback loop while minimizing risk to the business by controlling the size of the audience.

Testing across Industries

Different industries require different testing capabilities. For example, a banking or financial service project might need to test a scenario that uses a financial-based API to manage the flow of a payment. On a healthcare-industry project, there is more likely to be a need to test the flow of Health Level-7 (HL7) messages, which is a standard for transferring data between applications used throughout the healthcare industry.

HL7 refers to a set of international standards for transfer of clinical and administrative data between software applications used by various healthcare providers.

As well as domain-based industries such as banking and healthcare, there are also technology-based industries such as Mainframe, WebSphere, SAP, Blockchain, and others. In a similar manner, each of these also has its own specific technology interfaces, constraints, and behavior that need to be tested.

To help those tasked with testing activities on industry-focused projects, a common approach exists that can be applied to testing

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

across all industries — or as is more commonly found on projects, a combination of industries. The provision and consumption of multiple-industry specific APIs means testers follow these steps:

1. **Ensure test coverage of the points of contact (the APIs) throughout the technical ecosystem.**

   The nature of industry-based business use cases often means there is a dependency on multiple systems — some public, some private, and some purchased just for the execution of a particular business scenario. Ensuring test coverage of these points of contact means a team is able to test the technical ecosystem required to execute the business scenarios.

2. **Make sure the end-to-end business scenarios work as expected across the different industry-specific technologies.**

   When industry-focused business scenarios are defined, they're typically driven by business processes, industry regulatory requirements, and end-user experience. After ensuring the dependent technical ecosystem is available (see Step 1), the project can begin to test the end-to-end business scenarios on a dependable test environment.

3. **Certify the scalability of both the technology ecosystem and the business processes.**

   The cost and impact of industry-led regulatory control and reporting can be significant so it's important that the business scenarios can scale. After the technical environment has been validated (see Step 1) and the business scenarios have been validated (see Step 2), it's important to ensure both can scale appropriately. The objective is to execute performance tests against both business scenarios and across the technical ecosystem.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

» Taking the right path to achieving continuous testing

» Knowing where to begin on your journey

» Looking at the processes in continuous testing

» Making continuous testing work: one scenario

» Explaining the technologies in continuous testing

Chapter 4
Adopting Continuous Testing

Adopting continuous testing practices may seem insurmountable, but the sections in this chapter should get you well along your way to achieving your goals. There is no "right" answer on how best to embrace continuous testing practices, so finding the right path to start your journey is the first step.

The sections in this chapter also explain the processes and technologies used in continuous testing, as well as an end-result scenario of how continuous testing can work seamlessly in practice.

Finding the Path to Achieving Continuous Testing

The path to achieving continuous testing is not one size fits all. You can get there through different avenues. To help get you started in setting up this process, we offer a basic strategy-planning exercise.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

1. **Understand the maturity level of your testing capabilities.**

   This step is your starting point. Is your team primarily manual testers, are your programmers and testers working in sync, do the testers have a development background or programming skills, are you leveraging test automation today? These basic questions help you get started, but the list of other things to consider is quite long. See Chapter 2 for more questions to help you understand your testing maturity level.

2. **Determine what your near-term and longer-term objectives are.**

   You want to begin with the end goal in mind. However, to simply state that "From this day forward, we will test continuously" without providing the cultural transformation, giving the opportunity to review and change process, and implementing tools to automate the process and support the team, you're likely to fail.

Starting with people and the cultural change is important because behavior defines your culture. You need to support, acknowledge, and reward certain behaviors to establish the culture you seek. Certain behaviors can be

- Open collaboration and transparency across the team
- Everyone on the team having an equal voice
- Everyone openly sharing their opinions without fear of being ridiculed or called out in front of others
- A world where everyone contributes to software quality
- A world where finger pointing no longer exists and the name of the game is experimentation

REMEMBER

Don't forget that you also need to provide training — perhaps not formal training but providing opportunities for testing communities of practice to share ideas across squads, learn from each other, and report on the result of past experiments. Lunch-and-learns or meet-ups are great for this. Training could also involve *short* videos that show how to do a task combined with just-in-time mentoring (don't use the dreaded 1-hour classroom video — nobody has time for those and you forget what you learned by the time you need it).

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Another hurdle many organizations face is that automation is perceived as head-count reduction, which builds fear. To get buy-in to test automation and continuous testing, testers will need to see the value and how they benefit from automation. Work to convince them that automation will free them up to work on more value-add activities that use their minds instead executing as robots. Remember that testing is so much more than automating the checking of applications to see if they're working properly. After testers are convinced that they will finally get to do the job they were hired to do, the move to automation will become less of an issue.

Determining Where to Begin

After you convince your testers that automation and continuous testing benefit them, the next step is to figure out where to get started on your continuous testing journey. We suggest taking the initial steps outlined in this section.

Identify bottlenecks

Your first step is identifying the processes and process steps that are slowing you down:

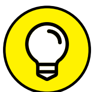
- **Test efficiency:** Test efficiency measures if you're able to run the needed tests and provide feedback quickly. Industry experts say that you should be able to run your daily regression tests in less than two hours.
- **Test effectiveness:** Check your test effectiveness, which means finding out if you're running the fewest number of tests that find the largest number of problems. If you are running regression tests that never find a defect, perhaps those tests aren't needed anymore.
- **Look upstream and downstream:** Bottlenecks may also be upstream or downstream from actual test execution effort. Don't wait on dependent systems or test environment provisioning.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Determine areas with high return on investment (ROI)

Spend time baselining your ability to test software. Then you can look at all aspects of your test effort to find the biggest pain points and focus on eliminating those bottlenecks first. Next, you need to start measuring your progress because those past and current measurements help you determine the best ROI. To identify waste during a value mapping stream exercise, think about the following questions:

- **How long are your delays in test environment provisioning, and are the environments accurate when they're provisioned?** If you have issues here, automate your test environment provisioning and configuration to eliminate manual errors and gain accuracy and reliability.
- **Are you waiting on dependent systems to start testing?** If so, use service virtualization to simulate those unavailable systems so you can start testing sooner.
- **Are you unable to run all the needed tests before the next build is available?** If this is the case, automate your high-risk regression tests to improve test efficiency.
- **Do you find performance problems late in the life cycle or even in production?** Outages can result in long-term damage to a business, so conduct load, performance, and stress testing earlier in the life cycle.
- **Do you find significant problems when all systems are integrated?** Identify and agree on system interface and data definitions and create automated API level tests for the high-risk integration points.
- **Are there misunderstood requirements, so code and tests don't align and a lot of defects are logged?** Work with the whole team early to make sure requirements are well understood by both programmers and testers. Business analysts and programmers tend to see the glass as half full, and testers tend to see it as half empty, so by working together they can flesh out a set of robust requirements.



Value stream mapping is a lean-management method for analyzing the current state and designing a future state for the series of events that take a product or service from its beginning through to the customer. As part of this exercise, organizations will begin

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

to understand how long each step takes and identify the constraints in their end-to-end process. The objective is to identify, analyze, and prioritize the bottlenecks with the intent of discovering opportunity for improvement. During this analysis of each process step, it is important to look at not only the lead time to completion (or when the clock starts) but also the process time (the time when the work actually starts until the work is completed). This process is shown in Figure 4-1. Increasing the efficiency of a process step while maintaining the thoroughness of that step can often be achieved by simply reducing or eliminating the wait time. To learn more, you may want to further explore the "Theory of Constraints" concept.

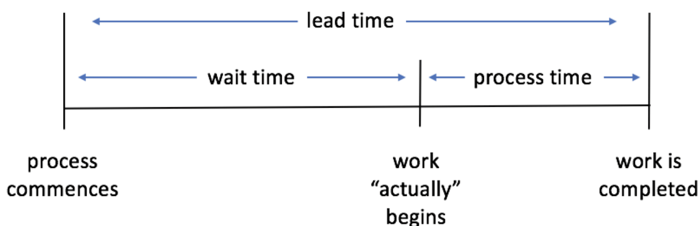


FIGURE 4-1: Value stream mapping.

Processes in Continuous Testing

In the context of continuous testing, test automation is a requirement for success. In order to have unattended automation from code commit to production, teams need to deliver several levels of automated tests to measure the quality of what's being delivered, as well as to quickly understand the state of the application.

TIP

An obvious benefit of automating testing, in contrast with manual testing, is that testing can happen quickly, repeatedly, and on demand. It becomes a simple matter to verify that the application continues to run as it has before. In addition, using the practices of test-driven development (TDD) and behavior-driven development (BDD), where test cases and scripts are created before a line of code is written, has shown to improve coding quality and design.

To learn more about the process of continuous testing as part of the IBM Bluemix Garage Method (we discuss in Chapter 1) visit `www.ibm.com/devops/method/content/code/tool_rtw`.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Making Continuous Testing Work: One Scenario

Imagine a situation where a live application has low quality and bad reviews. Here, we give you a scenario that describes how the project team can improve its ability to test when needed, not just when possible.

First, all stakeholders (business, development, test, operations, and so on) work together to understand the root causes of production defects. By using defect data and analysis, the team discovers that the most significant defects arise from two areas:

- Integration with other systems
- User response delays

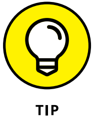
TIP

When you are working to understand root causes of production defects, this activity should not be executed as a "witch hunt" but rather a no blame postmortem. Remember to focus on driving out the "What happened?" and avoiding pointing fingers by spending time trying to find out "Who caused it?" If people feel that they are at risk, activities which contributed to the issue may be covered up. You need to know all the facts, and people need to be willing to openly share what happened without fear. Root cause analysis should be considered to be a learning exercise with the goal of process improvement trying to discover ways to make sure the issue doesn't repeat itself.

The team decides that it needs to test the high-risk integrations earlier in the development process and not wait until just before deployment to production. It is agreed that the solution is to increase collaboration and capture the agreed-on interface and data definitions as part of its system architecture.

The team also decides to conduct some low-intensity performance testing — checking the stability of the application with very few users — as soon as a build is available, so critical issues can be identified and fixed much earlier. Performance tests across each build are tracked so the team can immediately see if there's any degradation in performance.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.



In performance testing, the user load is gradually increased from low intensity to high intensity. Testing the performance of the application with very few users is referred to as *low-intensity performance testing.* Typically, you require a separate automated testing solution to do performance testing. However, with IBM Rational Test Workbench, you can do functional testing and low-intensity performance testing with the same functional test assets.

Based on the interfaces, the programmers and testers work together to create virtual services, or stubs, for each high-risk interface. The interface stubs mimic the other systems and allow the programmers and testers to test the high-risk areas earlier in the development process — as soon as the code is written and built.

The team then automates many of the functional integration tests and key response time tests — needed after root cause analysis and source of the significant production defects. Now whenever a new build is made, the successful build process triggers the automated activities shown in Figure 4-2.

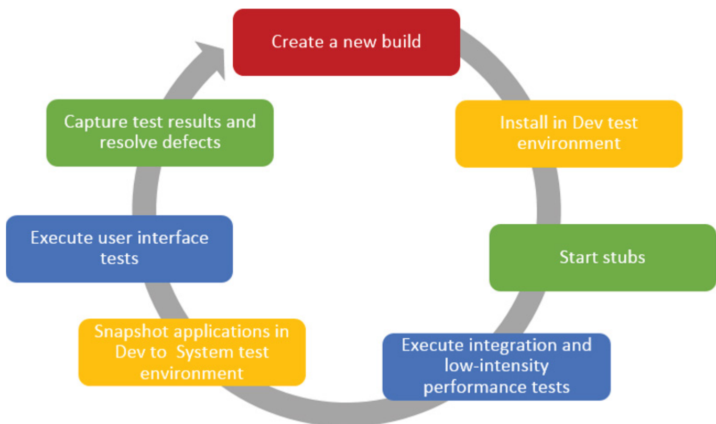


FIGURE 4-2: A typical continuous testing scenario.

The activities from Figure 4-2 are described as follows:

1. **Create a new build.**

   The new application build is automated and unit tests are run.

2. **Install in Dev test environment.**

   The new build is deployed into the automatically provisioned development hybrid cloud test environment.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

3. **Start stubs.**

   The stubs for any missing dependent services or test scenarios are started.

4. **Execute integration and low-intensity performance tests.**

   An automated integration test suite is triggered as part of the automated deployment process for execution, followed by the low-intensity performance tests. The test execution results are then made available to the entire team. The team analyzes the results for any regressions, especially around application performance.

5. **Snapshot application to System test environment.**

   Provided the integration and performance tests pass or conditions of the quality gate are met, a snapshot of the specific application component versions and the deployment assets is created. This snapshot is then used to consistently move the software to the next test environment — the System test environment.

6. **Execute user interface tests.**

   If the deployment to the next environment is successful, the automated user-interface-based test suites are triggered and executed in the system test environment. Testing the software at a different layer is the next step in the process.

7. **Capture test results and resolve defects.**

   Test results are captured, feedback is provided, defects are resolved, and a new build is created.

After the dependent systems are available, the team runs those same automated integration and performance tests again, against the real systems, to confirm their application still behaves as expected without the use of stubs. Re-using the automated tests across test environments not only saves time but also highlights any inconsistencies found in the applications being validated.

This scenario can be applied to any pattern of defects. It enables whole teams to work together to not only start testing earlier in the life cycle but also to automate the most important tests first, so those tests are executed repeatedly. This ensures good test coverage of the riskiest parts of the application.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Teams that improve their test efficiency and effectiveness significantly reduce their costs and the time it takes to deliver high-quality, innovative solutions to end-users. You can't test everything, and we aren't suggesting that you need to do *more* testing, and you can't automate all your tests, so it is critical to find the right subset of tests in the riskiest areas. Remember that service virtualization is key and allows testing to begin as soon as a build is made by mimicking missing dependencies. By combining test automation and service virtualization, teams can test earlier, or shift left, so they gather feedback faster than ever.

Technologies in Continuous Testing

Technology orchestrates the continuous testing process, making it repeatable, reliable, and traceable. Maybe you already have a build automation solution in place, but what additional capabilities might you benefit from? Deployment automation with the ability to treat infrastructure as code and create blueprints in a graphical interface to accelerate the provisioning of test quickly comes to mind. Those same capabilities are delivered by IBM UrbanCode Deploy.

IBM UrbanCode Deploy is a tool for automating application deployments through your environments. It is designed to facilitate rapid feedback and continuous delivery in agile development while providing the audit trails, versioning, and approvals needed in production. For more information visit `developer.ibm.com/urbancode`.

Technology can also help in eliminating testing delays. The API testing ability of IBM Rational Test Workbench coupled with IBM Rational Test Virtualization Server would meet your needs, enabling the delivery team to create virtual components and deploy those stubs to the enterprise where they're shared across the entire organization. And when deployment automation and service virtualization are integrated, the team is now able to stand up complete applications in dev-and-test labs much earlier than ever before. This is a significant savings because teams may spend 30 to 50 percent of their time standing up realistic test environments or delaying testing until all the application components are available.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Finally, you need to automate the other tests that you probably want to run — functional, performance, and security. Along with API testing, IBM Rational Test Workbench can also be used to author, maintain, and execute functional and performance tests. And just to make sure we don't forget about application security, IBM Security AppScan scans your application for vulnerabilities that hackers are trying to exploit.

While you may want to investigate other solutions, the IBM software solution will help you take a giant step along the path of achieving your continuous testing goal.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

IN THIS CHAPTER

- » Breaking the myth of the fad
- » Knowing who continuous testing is for

Chapter 5
Ten Continuous Testing Myths

The time has come to attempt the potentially impossible — busting the myths around continuous testing. This task means changing some people's understanding of the test profession, test automation, and continuous testing after years of misconception or lack of understanding. Wish us well.

Continuous Testing Is Only Executing Test Scripts

If we're dispelling myths in this chapter, we should remove the misconceptions in this statement immediately. The execution of test scripts — manual or automated — is simply checking application functionality, performance, usability, reliability, and so on to verify and measure if it meets the acceptance criteria and quality standards of your organization.

Testing and checking application functionality are *not* equal. Testing is collaborating, planning, managing, thinking, exploring, analyzing, automating, checking, reporting, validating, reviewing, and so on — the list is long. To suggest that testing is limited to the execution of test scripts is actually insulting to the test professional.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Continuous Testing Is Just a Fad

Continuous testing *isn't* just a fad or a buzzword you hear in your work. In this era of DevOps, continuous testing is an essential part of a successful transition to the continuous delivery of software. It is an automated approach that accelerates getting new software with higher quality into the hands of customers. Those goals aren't going to change anytime soon and neither is the continuing automation of the life cycle. So we are confident that the term and the process will stick around for the foreseeable future.

Continuous Testing Is Only for Agile Teams

You may think that achieving continuous testing only works for teams following agile practices, but *any* team can adopt some or all of the continuous testing methods (even if that team is following a more traditional waterfall approach). For example, when all the testing occurs at the end of the project, teams can be proactively working to determine how they will ultimately create and execute the tests.

You must first understand the different systems that your team's application needs to interact with and then determine if those systems will be available when needed. If not, create virtual services for those systems based on known interfaces.

After that, even if the user interface (UI) isn't final, create non-UI automated API-level tests to ensure the data and messages exchanged both within and outside the application are accurate. As soon as a first build is made available for testing, those automated integration tests can be run quickly to provide insight on software quality.

Having production-like test environments and realistic test data are critical for any team. Getting the test labs provisioned and configured on demand is another proactive step that can happen well before actual test execution needs to take place.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Continuous Testing Is Only for Testers

Absolutely not! The entire team, from requirements analysts, architects, designers, programmers, testers, and operations engineers need to embrace the spirit of continuous testing and support the effort. Here's a rundown of how everyone plays a role:

- **Requirements analysts** trigger the testing process when they capture requirements, so ensuring the team has designable and testable requirements is key.
- The **architects** are critical in the continuous testing effort because they provide the full architectural picture of the application — and any dependent systems that the application needs — as well as test data needs. Architects also provide valuable insight into impact analysis on code changes, especially those changes that could impact performance, usability, or reliability. They should ensure that the ability to test is provided at all layers of the application in order to support the initiative.
- **Designers** have significant input into what data is needed, and when. They can greatly improve the overall test effectiveness by collaborating with the testers to help them understand where there are potential fragile parts of the application.
- **Programmers,** of course, can provide detailed information on the upcoming changes as they create new features and fix defects, so testers know which sets of regression tests to execute.
- **Operations engineers** are the people that actually keep the application running smoothly, and they know where things typically go wrong in production. They provide critical input to which tests need to be created and executed and why.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Continuous Testing Isn't for Regulated Industries

While continuous testing is beneficial for all industries, it can actually provide additional benefits for compliance-mandated and regulated industries. Continuous testing solutions deliver automation, and anything automated typically means audit logs are part of the process. For any component of any release in production, you can definitively trace its full history of versions and deployments.

What this means is that the audit files contain definitive "proof" of exactly what components have been tested and are currently running in production and more importantly what was running in production at any previous point in time. When continuous testing and continuous deployment solutions are implemented as part of a DevOps continuous delivery pipeline, organizations have the ability to prove compliance as a by-product of their automated delivery process.

Continuous Testing Isn't for Large, Complex Systems

Continuous testing can greatly improve testing productivity for large or complex systems. In the past, teams have assumed (incorrectly) that if each subsystem does its own testing, then when all the subsystems are integrated together, everything will magically work between those subsystems. We know this magic rarely occurs, and the root cause of the integration failures is due to misunderstood interfaces between the subsystems. Creating virtual services and API-level tests as part of the continuous testing practice can significantly reduce the time and effort it takes to fully test all the integrations between subsystems.

And even outsourced teams can design, develop, and maintain automated testing suites as well as virtual services, thereby relieving, or in some instances, paralleling the development team's work. This can help reduce costs, eliminate bottlenecks, and accelerate the delivery of software changes.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Don't be alarmed if your organization has invested in open-source or other tools. There is no need to rip and replace all of them in order to add continuous testing capabilities. There are a number of integrations that exist for many tools. Meaning, your team can keep the tools it's familiar with, and you can simply add new test automation and service virtualization capabilities on top of them.

Continuous Testing Isn't Part of DevOps

Perhaps this belief comes from the fact that the word *test* is not visible between *Dev* and *Ops* — DevTestOps. Another thought is that some people believe that you don't need testers at all. But thought leaders in the testing domain would argue that point to the bitter end. Perhaps it's best to finally accept that while the testers may not write application software, they do create test automation code — both they and the programmer fall equally into the Dev of DevOps and are full-fledged and equal members of the delivery team.

Continuous Testing Isn't for Cloud or Hybrid Cloud Applications

The cloud has been around for a lot longer than you might think — we just didn't call it that back when we were using a single CPU that had multiple virtual machines running to execute our tests. Who knew you could have your own little private cloud right there under your desk? And, testing applications that are in a dedicated on-premises data center, or hosted on a private cloud, or hosted on a public cloud, or any combination of those are not barriers to adopting continuous testing practices. All you need is access to the systems (which you would need anyway to test them manually), and voila! — you can now implement your continuous testing automation robots to check software deployed to cloud or hybrid cloud environments.

If anything, users of new cloud-based software demand faster delivery of high quality features, which increases the need to adopt continuous testing.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Automating Tests Means We Need Fewer Testers

This myth is an oldie but a goodie. Automating tests should never be used as an excuse to reduce head count. Automating testing should be implemented with the purpose of freeing up a tester's time to do more important things.

Let the automation robots repeatedly perform the same test steps over and over — they are good at it. They don't get bored, they can run the same test repeatedly with a high level of accuracy, and they do it without thought.

In contrast, the testers have analytical minds, they're pessimists by nature, and they like to think and figure out how things work and potentially discover issues. They do this by thinking outside the box, at times pushing the limits of what the software was intended to deliver and then reporting back their findings.

So, embrace automation as a means to free testers from their robotic existence, empowering them to spend more time learning and experimenting. And most importantly, make sure the testers in your organization understand the move to automation is because they're highly valued resources who shouldn't be wasting their time taking the place of a machine. The move to test automation should be embraced because management would prefer testers spent their time doing the things that a machine can't do.

Quality Is the Test Team's Responsibility

When the entire DevOps team focuses on quality, you get much better outcomes. Quality is a mindset right from the start of ideation through to the customer experience. Of course, blatant design flaws or bugs must be eliminated, but even more subtle aspects of components, such as UI design and application performance, must all be part of a quality mindset for an application to be perceived as high quality. That means taking a DevOps whole-team approach where quality is one of the parameters of every decision and role in the DevOps life cycle.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Notes

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Notes

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

These materials are © 2017 John Wiley & Sons, Inc. Any dissemination, distribution, or unauthorized use is strictly prohibited.

Embrace continuous testing on your DevOps journey

IBM software and best practices can help software development, testing, and ops teams implement a continuous testing approach. Continuous testing means adopting the right set of automated tests along with service virtualization, which allows the team to simulate missing services and environments so they can start testing earlier and more frequently. By becoming more efficient and effective, teams can reduce their costs while speeding delivery of high-quality, innovative software to users in today's DevOps world.

Inside...

- Current challenges in testing
- The shift left concept
- Achieving continuous testing
- How to begin your testing journey
- Technologies in continuous testing
- Breaking continuous testing myths

Marianne Hollier has 20+ years of experience in life cycle application development and deployment and is especially passionate about software testing. **Allan Wagner** is a Technical Evangelist with IBM, driving thought leadership, strategic initiatives, and tangible solutions with a specific focus on continuous deployment and testing within the larger DevOps topic.

Go to Dummies.com®
for videos, step-by-step photos, how-to articles, or to shop!

for dummies® A Wiley Brand

ISBN: 978-1-119-36583-9
Part #: KUM12367USEN-00
Not for resale

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.