

CA Press

apress
open

DevOps for Digital Leaders

Reignite Business with a Modern DevOps-Enabled Software Factory

Aruna Ravichandran
Kieran Taylor
Peter Waterhouse

www.dbooks.org

DEVOPS FOR DIGITAL LEADERS

REIGNITE BUSINESS WITH A MODERN DEVOPS-ENABLED SOFTWARE FACTORY

Aruna Ravichandran
Kieran Taylor
Peter Waterhouse



CA Press
apress®

DevOps for Digital Leaders: Reignite Business with a Modern DevOps-Enabled Software Factory

Aruna Ravichandran
CA Technologies, Cupertino, California, USA

Kieran Taylor
Winchester, Massachusetts USA

Peter Waterhouse
Blackburn, Victoria Australia

ISBN-13 (pbk): 978-1-4842-1841-9
ISBN-13 (electronic): 978-1-4842-1842-6
DOI 10.1007/978-1-4842-1842-6

Library of Congress Control Number: 2016958432

Copyright © 2016 by CA. All rights reserved. All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

The statements and opinions expressed in this book are those of the author and are not necessarily those of CA, Inc. ("CA").

ApressOpen Rights: You have the right to copy, use and distribute this Work in its entirety, electronically without modification, for non-commercial purposes only.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Managing Director: Welmoed Spahr
Acquisitions Editor: Robert Hutchinson
Developmental Editor: Laura Berendson
Editorial Board: Steve Anglin, Pramila Balen, Laura Berendson, Aaron Black, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao, Gwenan Spearing
Coordinating Editor: Rita Fernando
Copy Editor: Kezia Endsley
Compositor: SPi Global
Indexer: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

The information in this book is distributed on an "as is" basis, without warranty. Although precautions have been taken in the preparation of this work, the author, Apress and CA shall have no liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

Printed on acid-free paper

www.dbooks.org

Advance Praise for *DevOps for Digital Leaders*

"Everyone on the DevOps journey wants to see how others solved similar problems in different business contexts. This book does a great job in chronicling how organizations have applied DevOps patterns and articulating the fantastic value they've created."

—Gene Kim, co-author of *The Phoenix Project* and *DevOps Handbook*

"DevOps for Digital Leaders addresses an important gap in the literature, presenting DevOps from a business and organizational perspective rather than the more common technology and process angles. Aruna, Kieren, and Peter have done a great job of assembling, presenting, and addressing the core challenges of a transition to DevOps. Not only do they look at the impact on the "path to production" of testing, deployment, and operational support, but they tackle the complex topic of API design and even present the ROI justification for DevOps, citing Dr. Nicole Forsgren's groundbreaking work with the State of DevOps reports. This book should be required reading for technology leaders wondering about the impact DevOps will have on their organization."

—Dan North, DevOps and Continuous Delivery pioneer,
Principal at Dan North & Associates Ltd.

"If *DevOps for Digital Leaders* had existed several years ago when we started on our DevOps journey, it would have made our transition significantly easier. This book is an excellent resource for both experienced and new DevOps practitioners—more than just a 'how-to', it provides valuable insights and real-world examples to which everyone can relate."

—Dana Edwards, CTO/EVP, MUFG Union Bank

"As DevOps has risen to the top of the mind with IT departments big and small, new and old, there has been much written about the subject. Unfortunately, too much of it has been about 'what is DevOps' or shrouded in almost a Zen-like philosophy. What has been missing is the practical 'why' and 'how to' for practitioners. What's missing is a guide that will grow dog-eared and highlighted, with real-world examples and advice that practitioners will carry with them and go back to again and again. *DevOps for Digital Leaders* is that book. If you are looking for real-world examples and guidance on turning your organization into a high-performing IT operation, this is a must-have book for your library."

—Alan Shimel, Editor-in-Chief, DevOps.com

"DevOps is no longer just a concept. This book by Aruna, Kieran, and Peter places DevOps squarely in the real world. It is a comprehensive guide to help you discover the origins of DevOps, realize the challenges, and gain a practical understanding of where DevOps fits into your development program, IT operations, and business. It's a must-have book for the business side to truly comprehend DevOps, but also a great read for the IT community to understand how their work ultimately impacts the business. And most importantly, the book is loaded with valuable, actionable tips to help you navigate your DevOps journey."

—Pete Goldin, Editor and Publisher, *APMdigest* and *DEVOPSdigest*

www.dbooks.org

I would like to dedicate my first book to my mother, Sampurnam Thyagarajan, who has always been my inspiration, who stood by me every step of the way. I am who I am today because of her unconditional love and dedication. I love you Amma.

—Aruna

I dedicate this book to all the true IT believers and practitioners. Only from your collective effort can organizations build the generative DevOps culture needed to shape success. You know who you are!

—Peter

I dedicate this book to those who work to make the world a better place for future generations.

—Kieran

Contents

Foreword ix
About the Authors xi
Acknowledgments xv

Part I: DevOps: Conflict to Collaboration 1
Chapter 1: DevOps in the Ascendency 3
Chapter 2: IT Impasse 15
Chapter 3: DevOps Foundations 27

Part II: Essential DevOps Tooling 49
Chapter 4: Build 51
Chapter 5: Test 69
Chapter 6: Deploy 87
Chapter 7: Manage 105

Part III: Tuning and Continuous Improvement 123
Chapter 8: Practical DevOps 125
Chapter 9: DevOps and Real World ROI 139
Chapter 10: DevOps Finetuning 151

Index 171

www.dbooks.org

Foreword

As a software executive who's worn many hats across sales, services, development, support, operations, and even IT, I have a fairly unique perspective when I talk to business leaders about their technology strategies. Though the name DevOps didn't exist until fairly recently, the need for partnership between development and operations teams has always been around. Today DevOps has shifted from an emerging movement to a critical component of most enterprises' digital transformation strategy. In other words, DevOps is moving from the Kanban board to the board of directors.

While the idea behind this kind of collaboration is common sense, there are definitely challenges to driving mainstream adoption for DevOps. In my conversations with C-suite executives, they invariably ask for two things: documented best practices and actual case studies that they can take to their internal stakeholders.

That's why this book is so relevant right now. It's more than just theory or a summary of the ideal DevOps toolchain; it's a set of practical insights and better practices that came about through many, many interviews and interactions with frontline practitioners about what works and what doesn't work in the real world.

Aruna, Kieran, and Peter are career high tech marketers who shed light on the cultural and technological challenges to driving adoption for a DevOps philosophy across the software development lifecycle. With this book, they were able to bring together solid, actionable advice for anyone beginning their DevOps journey, including recommendations on how to measure success and return on investment from a business standpoint.

The days of organizational silos are coming to an end. Communication, collaboration, and automation are blurring the lines between development, QA, and IT Operations—and enterprises are embracing agile and DevOps practices to increase software release frequency while improving overall quality. I hope that in reading this, you find ways to optimize your technology strategy, investments, and business outcomes.

—Adam Elster, President of Global Field Operations, CA Technologies

About the Authors



Aruna Ravichandran is Vice President of DevOps Product Marketing and Solutions Marketing at CA Technologies. She has over 20 years of experience in building and marketing products in various markets, such as IT Operations Management (APM, Infrastructure Management, Service Management, Cloud Management, Analytics, Log Management, and Data Center Infrastructure Management), Continuous Delivery, Test Automation, Security, and SDN. In her current role, she leads the product and solutions marketing, strategy, market segmentation, and messaging, positioning, competitive, sales enablement across all DevOps products, which spans revenues over $1B. She is the key spokesperson for DevOps with analysts, press, customers, and major events.

Prior to CA, she has worked at Juniper Networks and Hewlett-Packard wherein she led executive leadership roles in marketing and engineering.

She frequently blogs for various publications such as SYS-CON Media, *Wired* Insights, Tech Target, Information Week, DevOpsDigest.com, DevOps.com, and Cloud Tweaks, to name a few. She frequently presents at various industry conferences and has presented at Gartner Symposium 2016, Gartner ITOM 2014, Gartner Data Center 2014, DevOps Summit 2014, Cloud Expo 2014, CA World 2014 and 2015, and HP Discover (2008-2012). She has authored several articles and publications as well.

Aruna earned her Master's in Computer Engineering and MBA from Santa Clara University.

In 2016, Aruna was named one of Top 100 Most Influential Women in Silicon Valley by the San Jose Business Journal and the 2016 Most Powerful and Influential Woman Award by the National Diversity Council.

www.dbooks.org



Kieran Taylor has 20 years of high-tech product marketing experience with a focus on application performance management, cloud computing, content delivery networking, and wide area network technologies.

He is presently Senior Director of Product and Solutions Marketing at CA Technologies and is responsible for thought leadership and sales enablement for Application Performance Management as well as CA solutions that help enterprises implement DevOps. In previous roles at Compuware and Adobe, Kieran's responsibilities included corporate awareness, product marketing, field enablement, and partner marketing.

For over a decade, Kieran was Senior Director Global Marketing at Akamai with oversight of the go to market strategy, channel marketing, and demand generation for Akamai's entire solutions portfolio. While at Akamai, Kieran launched several CDN market innovations, including J2EE-based EdgeComputing and the open-standard Edge Side Includes (ESI), a markup language for dynamic content assembly and delivery at the edge, which is used today by many leading enterprises.

Kieran led product marketing and management roles at DataPower Technology (now IBM), Nortel Networks specializing in XML/XSLT, VPN, and remote access technologies. Kieran has also worked as a broadband consultant to major service providers while at TeleChoice Inc. and was Wide Area Networks editor for publications at McGraw Hill.



Peter Waterhouse is Senior Director, DevOps solutions at CA Technologies. With more years in tech than he cares to mention, Peter's experience is broad and varied—ranging from implementing "big iron" monolithic ERP software applications, to writing crude Python code on his Raspberry Pi to automate his son's Lego creations.

At CA, Peter is honored to be part of a fantastic team of marketing professionals and solution strategists; folks who each and every day work tirelessly to ensure their customers get the most value from their technology investments.

Passionate about how disruptive technologies and DevOps in particular can transform business and society for the better, Peter writes and blogs on a variety of topics, including organizational culture, transformational business models, lean thinking, and metrics. His articles and whitepapers address the

purposeful application of DevOps, Cloud computing, Mobile, and the Internet of Things; appearing in publications ranging from *CIO Review Magazine* and *InformationWeek* to *App Developer* magazine and *Network World*.

Living in Melbourne, Australia, but raised in the north of England explains why Peter supports Manchester City, and much to the annoyance of his loving family, still listens to Joy Division and The Stone Roses.

Peter has a Bachelor's degree in Commerce and Economics.

www.dbooks.org

Acknowledgments

In the spirit of true DevOps collaboration and sharing, we would like to acknowledge the contributions of the following CA Technologies product marketing colleagues.

Alan Baptista; Scott Edwards; Amy Feldman; Matt Hines; Jeffrey Hughes; Tim Mueting; Tyson Whitten; Brendan Hayes

Their insights, thought leadership, and expertise have been invaluable in the development of this book.

PART

I

DevOps: Conflict to Collaboration

www.dbooks.org

CHAPTER

1

DevOps in the Ascendency

"Be ahead of the times through endless creativity, inquisitiveness, and pursuit of improvement."

—The Toyota Precepts[1]

Accelerating Agile Practices in Today's Software Factory

In 2016, Formula 1 (FI) racecars, the ultimate in four-wheeled technology, are awash in wireless sensors and transmitters.

Running your eyes across the graceful, sculpted bodywork of these 200+ mph engineering wonders, the only discernible interruption of their low-slung carbon fiber noses are small clusters of wireless antennas jutting up directly into the drivers' sightlines.

[1] "Toyota Quotes," *The New York Times,* Feb. 10, 2008: http://www.nytimes.com/2008/02/10/business/worldbusiness/10iht-10facts.9900222.html?_r=0

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_1

As subtle as these transmitters may be, why would designers, who spend endless time and resources attempting to improve the aerodynamics and drivability of the machines, accept such a stark concession? The answer is simple—to arm their teams with priceless real-time data used to continuously improve performance.

Along the pit row wall, in the trackside garages, and back in the very design centers where F1 engineers continue to refine their handiwork, telemetric information provided by those tiny antennae is immediately translated into change.

While team principals and technical directors sitting directly adjacent to the track communicate directly with the drivers, advising them how to adjust settings on the fly, their colleagues in the garages prepare to adjust everything from tires to aerodynamics when the machines pull into the pits.

Meanwhile, back at the factory, live data is consumed by all manner of engineering teams who continually produce and modify new components to be utilized in subsequent competitions—beginning the process of further advancement even before the current race is complete. The push for innovation in F1 is a perpetual activity; in this sense, perhaps more so than in any other sport, the only constant is technological change.

That Toyota Motor Corp., the global automotive giant best known for advancing such data-driven, process-centric "just in time" manufacturing practices—concepts that revolutionized mass production of passenger cars—had such a short-lived and disappointing record in F1 can only be classified as ironic. However, the Kanban and Kata methodologies leveraged by Toyota since its inception, paralleled on a smaller, more specialized scale by today's F1 teams, stand as the most widely emulated management frameworks in history.

With an emphasis on the adoption and evolution of processes designed to optimize resources while increasing production speed and quality—and most importantly promoting constant innovation—the "Toyota Way"[2] represents arguably the leading model for continuous improvement.

In the words of W. Edwards Deming—the legendary American engineer and management consultant who helped spearhead the reinvention of Japanese industry after WW2—Toyota's approach embodies the notion that, "It is not enough to do your best; you must know what to do, and then do your best."[3]

Grounded in constant observation and measurement of efficiency—from supply chain management through final production, with the goal of constantly improving output, including and in particular worker productivity—this meth-

[2]Toyota Motor Corporation Annual Report, 2003, page 19.
[3]"Made in Japan," MadeinWyoming.com, Rena Delbridge, copyright 1995: http://madeinwyoming.net/profiles/deming.php

www.dbooks.org

odology flies in the face of traditional "waterfall" workflows. American automaker Henry Ford may be credited with revolutionizing the assembly line, but Toyota is widely recognized for decoupling and perfecting it.

Rather than creating linear dependencies, where each successive activity is wholly dependent on completion of the task preceding it, this revolutionary approach emphasizes techniques wherein production is dynamically adapted on the fly to result in optimal efficiency and maximum quality.

Further illustrating the staunch devotion to continuous improvement represented both in the Toyota Way and in his personal doctrine, Deming famously said, "It is not necessary to change. Survival is not mandatory."[4]

This observation is neither subtle, nor, given industrial history, seemingly misplaced.

Embracing DevOps in the Application Economy

In today's rapidly evolving Application Economy, it is widely recognized that, driven by the evolution of digital channels—across both the business-to-business and consumer segments—many organizations are reinventing or recasting themselves as providers of software and digital services.

For example, the global population of mobile banking users is forecast to double to 1.8 billion by 2020, representing over 25 percent of the world's population, according to KPMG.[5] As a result, banks are increasingly focused on advancement of web-based and mobile applications, versus expansion of brick-and-mortar operations.

Furthermore, as business delivery mechanisms shift to the digital landscape, end users' tolerance of latency of such applications has grown increasingly narrow. According to a study published by *Wired* in June 2014, roughly 50 percent of consumers expect a web page to load in two seconds or less—or they move to abandon it.[6]

Given this transformation, as traditional business services are replaced by largely web-based and mobile-friendly applications, organizations are being forced to completely reexamine their software development and IT management practices. Technology is no longer viewed as a supporting dependency, but rather as a primary element of conducting business.

---

[4]Edward Deming, *Out of the Crisis,* (Cambridge, MA, MIT Press, 1982), p. 227.

[5]"Global Mobile Banking Report," KPMG LLP, copyright 2015: http://www.kpmg.com/channelislands/en/IssuesAndInsights/ArticlesPublications/Documents/Digital%20offerings%20in%20mobile%20banking%20-%20May%202015.pdf

[6]"Great Expectations: 47% of Consumers Want a Web Page to Load in Two Seconds or Less," by Nilesh Patel, *Wired*, copyright 2014: http://insights.wired.com/profiles/blogs/47-of-consumers-expect-a-web-page-to-load-in-2-seconds-or-less#ixzz40vkvFwVl

Within that context, most of today's organizations are moving aggressively to adopt more agile, efficient software delivery and IT management practices to meet customers' evolving expectations. Among the fastest growing and most widely adopted strategies, aimed at bringing Toyota-like efficiency to the world of the applications lifecycle, is the methodology known as *DevOps*.

Whether or not Patrick Debois understood that he was creating, or at the very least putting a name to, the movement that would completely redefine the manner in which organizations build and support software is unclear. What is known is that since Debois, a systems administrator, and a handful of like-minded software development and IT operations experts first coined the DevOps moniker in 2009, the concept has become a global phenomenon.

The underlying concepts encompassed by DevOps are, at first glance, straightforward, but represent seismic reformulation within the context of software production and support. Rather than maintaining discreet applications engineering ("Dev") and IT management ("Ops") competencies and organizations, DevOps dictates use of smaller teams with cross-functional expertise to improve software functionality and the processes used to deliver it.

As highlighted in the seminal DevOps novel, *The Phoenix Project*, this mindset eliminates the fragmented approach to applications delivery that has traditionally crippled many organizations in addressing the digital market opportunity. Rather than asking developers to build an application and then charging IT management with ongoing support, creating highly disparate and inefficient dynamics, DevOps brings those specialists together so that engineering is completed with a constant eye toward ongoing management.

Just as critical in boosting organizational efficiency and software quality, DevOps methodology—like the Toyota Way—also promises to increase the ability to change the existing code base and deliver new capabilities to end users, while cultivating internal experimentation. Leveraging automation to do so is another hallmark of the Japanese carmaker's model and is also core to the DevOps approach.

In a May 2014 editorial published in the *Wall Street Journal,* noted technology evangelist and *The Phoenix Project* co-author Gene Kim echoed the words of Deming in framing his view of ongoing DevOps adoption. Titled "Enterprise DevOps Adoption Isn't Mandatory — but Neither Is Survival," Kim, also an established entrepreneur, posits that "the business value created by DevOps will be even larger than was created by the manufacturing revolution."[7]

[7]"Enterprise DevOps Adoption Isn't Mandatory — but Neither Is Survival," by Gene Kim, copyright WSJ, 2014: http://blogs.wsj.com/cio/2014/05/22/enterprise-devops-adoption-isnt-mandatory-but-neither-is-survival/

www.dbooks.org

Responding to some experts' observations that DevOps is only relevant for lean-minded, applications-driven startups such as consumer transportation darling Uber, Kim asserts that even the oldest, most entrenched businesses must wrap their arms around the movement to compete and survive. The author contends in the WSJ piece that this is, "because IT is the factory floor of this century, and not just for manufacturing companies. IT is increasingly how all businesses acquire customers and deliver value to them."

Like the so-called "lean manufacturing" wave that swept the business world in the 1980s—directly related to the success of Toyota in growing to the point of global sales domination—Kim asserts that today's organizations must seek to drive every element of inefficiency out of their software development lifecycle (SDLC).

Just as companies that failed to adopt leaner manufacturing processes in the latter half of the 20th Century lost out to rivals that did, the expert, among many others, maintains that organizations who overlook the need to adopt DevOps in today's Applications Economy will likely disappear.

Evidence that Kim and other proponents are correct in predicting that organizations' willingness to embrace DevOps will directly impact their ability to compete is already mounting. Numerous research reports have reinforced that leading DevOps practitioners already appreciate significant competitive benefits.

In a 2015 study published by Freeform Dynamics, in partnership with CA Technologies, researchers found that the 20 percent of organizations that had broadly adopted DevOps methodologies were 2.5 times more likely to have charted improvements in customer retention.

The report, "Assembling the DevOps Jigsaw,"[8] also found that these early adopters were 2 times more likely to have realized improvements in customer acquisition, and 3.4 times more likely to appreciate growth in market share. Perhaps most notably, those organizations already well into their DevOps transformations were 2 times more likely to have recorded a positive impact on revenue, and 2.4 times more likely to have experienced higher profit growth.

Regardless of the given era and environment, those financial results would seem to speak for themselves. Yet, according to the Freeform Dynamics survey, 80 percent of all organizations have yet to embrace DevOps completely.

[8] "Assembling the DevOps Jigsaw," Freeform Dynamics, copyright 2015: http://rewrite.ca.com/us/articles/devops/assembling-the-devops-jigsaw.html

DevOps as a Critical Requirement

The reality is, whether organizations are ready or not, the requirement to aggressively adopt DevOps methodology has quickly become the new normal in the Applications Economy. At the very least, the notion that such change is inevitable in cultivating continued business growth must be recognized as an inconvenient, yet irrefutable truth.

In an increasingly fast-paced, complex, and ambiguous business world, competing on a landscape dominated by the continued evolution of digital channels and mobile devices, the only tractable strategy is to adapt to survive. Driven by the exponential speed of change, organizations must wrap their arms around DevOps if they hope to defend and expand their market opportunities.

Within this current atmosphere—defined by volatility, uncertainty, complexity, and ambiguity (VUCA)—DevOps offers an unprecedented opportunity for organizations to transform their SDLC to increase efficiency and meet end users' changing expectations. By fundamentally recasting the manner in which they approach every element of software development and management, DevOps represents the broader future of business in general.

Despite the tendency to celebrate industry "unicorns" such as Uber—startups whose technology and business models immediately lent themselves to DevOps adoption—research such as the Freeform Dynamics "Jigsaw" report illustrates that organizations of all sizes and industries must change, or risk potential obsolescence. This conclusion is already being proven out by widespread assimilation of the involved methodologies by everyone from lean startups and centuries-old manufacturers, to nonprofits and government agencies.

Banking on DevOps Practices

Adoption among stalwart names in the banking industry—recognized as one of the most entrenched and "old school" segments in the entire business world—offers further proof of this pervasive need for DevOps transformation.

While notoriously staid in some senses, banks have also long-embraced significant levels of software and IT to diversify their services and increase profitability. Examples include inventions such as automated teller machines (ATMs) and the vast electronic transaction processing systems that allow these companies and their clients to move capital around the world in real time.

With roots dating back over 150+ years into the Dutch banking trade, global corporation ING is one such old world company that has successfully embraced DevOps transformation. Driven by existing inefficiencies in

www.dbooks.org

its 15,000-strong IT workforce and the need to address changing customer demands to increase its (EU) 15 billion annual revenues, the Amsterdam-based company embarked on an aggressive DevOps strategy beginning in 2011.[9]

Aimed specifically at enabling so-called "continuous delivery" of its applications to suit emerging customer preferences around online and mobile banking, company officials sought to remodel SDLC methodologies by invoking a manifesto of legendary Apple co-founder Steve Jobs. Citing a 1989 interview with *Inc. Magazine* in which Jobs famously said, "You can't just ask customers what they want and then try to give that to them. By the time you get it built, they'll want something new,"[10] ING set about on its DevOps initiative.

With the expressed goal of retrenching its SDLC to eliminate onerous, time-consuming waterfall practices and accelerate the pace of applications innovation, ING first moved to create more agile "scrum" teams that brought together development and operations expertise. From a process standpoint, one of ING's tactical goals was to begin testing new applications code in a far more "production-like" environment, such that emerging issues could be identified and resolved faster, accelerating code delivery to end users, while improving quality.

In support of those initiatives, the banking giant also leveraged significant automation to address SDLC requirements, including configuration management and software deployment. By bridging the gap between dev and ops, and employing new levels of automation, the company was able to increase the pace of its applications releases from an average of once every 13 weeks to weekly updates.

In support of this larger continuous delivery effort, ING was also able to increase its pace of underlying mobile applications code deployments from several hundred per month to over 80,000 per month, in less than two years. By leveraging DevOps workflows and supporting automation, the Dutch bank transitioned from release cycles dictated by technical roadmaps, to those focused on strategic business objectives.

Ultimately, customers expressed their approval both in adoption and via public forums, as ING's mobile application reviews on the Apple iTunes App Store climbed from one star in 2011, to four stars in 2014.

---

[9] "ING Bank Case Study," CA World'14 Presentation copyright CA Technologies. https://www.youtube.com/watch?v=9jqY_bvI5vk

[10] "The Entrepreneur of the Decade," by George Gendron and Bo Burlingham, copyright *Inc. Magazine* 1989: http://fortune.com/2009/11/23/decade-steve-jobs-apple/

DevOps: A Key Component of Business Agility

The ING story stands out not only as detailed proof of the manner in which DevOps represents a tremendous opportunity SDLC reinvention, but also as a specific example of how large, entrenched organizations can also emulate the much admired startup mentality. At the end of the day, the company's continuous delivery exercise allowed it to appease the changing preferences of mobile applications users, the ultimate goal of countless new startup ventures.

For years, everyone from management consultants to Wall Street investors have beleaguered the need for such pre-Internet companies to better leverage the "lean and mean" traits of their upstart peers. With the dawn of the DevOps era, driven by the Applications Economy, organizations, regardless of scale and history, have in fact been presented with this specific opportunity.

By completely transforming the manner, and more importantly the speed with which new ideas can be translated into marketable products and services—in the specific form of applications—organizations that have traditionally been slowed by their inherent size and complexity can rapidly accelerate innovation.

Conventional wisdom dictates that large enterprises with sundry customers and products face greater risk in attempting to launch new services and supporting business models. Meanwhile startups, unburdened with existing processes and systems, can rapidly pivot from one guiding approach to another, reinventing themselves whenever the need arises. Enterprises, typically guided by the need to requite shareholder interests, have been prone to err more toward stability, limiting their rapid growth potential.

For example, it is widely recognized that this entrenched, risk-averse mentality has allowed startup companies such as eBay and Airbnb to displace longstanding giants of the global retail and hospitality industries, respectively. If you could turn back the clock 5-10 years and give Sears and Hilton another chance to adopt the same strategies used by their startup rivals, obviously they would jump at the opportunity.

Part of this, in addition to a lack of shareholder reckoning, revolves around the fact that startups have also been able to quickly embrace emerging business technology trends, including cloud computing, mobile apps, open source, crowdfunding, and other means of collaborative economics.

Yet, if the core of this mentality, empowered by technological agility, is that innovation is merely a "good idea made possible," then adoption of DevOps represents just such an opportunity for organizations, regardless of size, to accelerate business via more efficient applications delivery.

www.dbooks.org

As framed by Jez Humble, leading technology consultant, recognized as a founding father of the DevOps movement: "The long-term value of an enterprise is not captured by the value of its products and intellectual property, but rather by its ability to continuously increase the value it provides to customers—and to create new customers—through innovation."[11]

At its core, the premium placed on the startup mentality, as previously referenced, relates primarily to the notion of market disruption. This is perhaps best exemplified, thus far, by those Applications Economy darlings that have successfully introduced new business models supported by web and mobile technologies.

However, leveraging DevOps and its halo effect of increased innovation through more efficient and rapid delivery of differentiated applications, one could easily assert that in the matter of enterprise versus startup, the playing field is being leveled quickly. Specifically, as organizations leverage DevOps to become more agile in addressing the Applications Economy, they are able to better keep pace with their smaller, more innately nimble rivals.

Circling back to the origins of the Toyota Way, the involved tenets of disruption via technological innovation have always been recognized as catalyst in transforming industry; in fact, even the genesis of this specific example has roots predating the auto industry.

In 1896, Sakichi Toyoda invented Japan's first self-powered loom, incorporating numerous revolutionary features, most notably the ability to automatically halt production when threads moving through the devices were broken.[12] Leveraging his invention, Toyoda built his startup into a leader in the Japanese textiles industry.

Those concepts, which became the legendary Toyota Way, were merely carried over when his son Kiichiro launched the automaker Toyota Motors in 1937. As it would turn out, this technology-driven approach to innovation and disruption would also dovetail perfectly with the lean management concepts promoted by Deming as he helped resurrect Japanese manufacturing in the 1940s.

At the time, automakers in other areas of the globe, notably the United States, likely would have never accepted that such a company, far smaller than their own booming operations and literally rising from the ashes of war, would have the opportunity to dominate the worldwide market.

---

[11] Jez Humble, *Lean Enterprise: How High Performance Organizations Innovate at Scale,* (Sebastopol, California, O'Reilly, 2015) p. 39.

[12] "Automation with a Human Touch," Toyota Motor Corporation World site, copyright 2016: http://www.toyota-global.com/company/vision_philosophy/toyota_production_system/jidoka.html

In 2012, after decades of displacing customers from other brands, Toyota was recognized at the world's largest automobile manufacturer, having produced its 200-millionth vehicle and grown into the first automobile manufacturer to ever produce more than 10 million vehicles per year. In 2016, amid larger worldwide economic concerns and a headline-grabbing airbag recall issue, the company remains among the global sales leaders in its industry.

Companies such as Toyota and ING, along with endless agile startups, offer tacit proof that for organizations seeking to increase market relevance and disruption, DevOps offers an attractive template. Yet, citing Freeform Dynamics' findings that only 20 percent of all organizations have achieved any semblance of DevOps maturity, the movement is clearly still in its nascence.

To wit, the researchers found that even though 80 percent of respondents to its survey view DevOps as a "key component of business agility," only 55 percent of organizations laid claim to having created a well-defined DevOps strategy. Furthermore, while 86 percent of those organizations surveyed labeled DevOps-oriented education of business stakeholders and greater alignment of IT and business priorities as important, only 33 percent and 37 percent, had enacted those programs, respectively.

Loosely stated, the concept of disruption, in general, suggests that market incumbents typically become less profitable and open to displacement from more agile peers based on the entrenchment of inflexible business models. It is widely acknowledged that there are two primary models for market disruption in the current era—"new market" disruption and "low end" disruption—at least as defined within the context of so-called "disruptive innovation" theory.[13]

In the case of new market disruption, the landscape is forever altered when a new products arrives that somehow better addresses a segment that incumbents have not yet envisioned or delivered. Whereas with low end disruption, incumbents see business chipped away by products that aim to encompass a smaller subset of perceived value drivers offered at a lower price point.

Looking closer at the promise and outfalls of DevOps—and the ability of adopters to better align themselves with the exploding demand for software and applications—one could easily argue that this ongoing technological revolution directly supports both archetypes of the disruptive opportunity. It would seem obvious that those organizations that have yet to jump in sit greatly imperiled by its continued advancement.

Some may view the Darwinian views of Deming, or Gene Kim, as overstated, particularly as related to DevOps and the Applications Economy. One could imagine that organizations that do not yet supply applications to consumers or business partners would remove themselves from the larger conversation.

[13]"What is Disruptive Innovation?," copyright Harvard Business Review, Dec. 2015.

www.dbooks.org

However, in the following chapters of this book, there is a great deal of evidence—backed by specific technical practices—that make it clear just how sensible, if not necessary, it is for every organization to view the DevOps opportunity as their best chance for success and survival.

DevOps: A Practice for Champions

At the conclusion of the 2015 F1 season, the world champion driver Lewis Hamilton and teammate Nico Rosberg of the Mercedes AMG Petronas Formula 1 Team, also winners of the highly coveted F1 constructor's title, returned to the outfit's UK headquarters to thank the factory workers who made their victories possible.

While most of those people, numbering in the hundreds, never joined the team at the track on a single race day during the globe-spanning nine month season, the champions heaped praise on their colleagues, recognizing the year-round effort that supports the entire operation.

Mercedes made easy work of the rest of the F1 field in 2105, having won a stunning 16 of the 19 total races. At the same time, this was not without constant updates to its racecars, with new components constantly meeting the team around the world as they progressed throughout the grueling race calendar.

"In racing there are always things you can learn, every single day. There is always space for improvement, and I think that applies to everything in life," Hamilton has been quoted as saying.[14]

"With the team, whether it be my guys that are at each Grand Prix, the team back at the factory who work day and night to develop the car, build the parts, and send the parts, the IT team, and the crew cleaning the factory. Everyone. We do it together. We all feel the joy when we are winning and the same pain when we are losing," the champion driver continued.

Without the constant delivery of new innovations, without the continued experimentation and refinement of the involved technologies, and of course all the underlying processes, Hamilton affirms, the likelihood of his success would be scant, if not impossible.

DevOps offers the chance for any organization to run more like a championship team, ever improving performance, going faster, and winning.

---

[14] "Mercedes AMG Petronas Team End Season on Top," By Donny Halliwell, *Inside Blackberry*, copyright2015.http://crackberry.com/mercedes-amg-petronas-formula-one-team

Summary

Only a few years into the Application Economy it's clear that huge benefits are being appreciated by those organizations capable of embracing emerging processes and technologies that enable agile methodologies. DevOps is widely recognized as one of the most influential and important movements that empower that transformation.

At the same time, such a sea change in the manner that organizations approach software development and operation, or any process for that manner, cannot be realized without some growing pains. In the next chapter, we'll take a look at the historic state of disconnect that has existed between developers and IT operations staff, and the approach that organizations embracing DevOps have employed to reinvent the manner that such experts collaborate.

www.dbooks.org

CHAPTER

2

IT Impasse

Faster Software Development Versus Operational Stability

Ever since we flipped the switch on commercial computers back in the 1950s, IT departments have been struggling to keep up with an insatiable demand for software applications and services. Of course many technologies like commercial of-the-shelf software packages, virtualization, and cloud computing have helped along the way, but generally IT delivery has been slow and uncoordinated.

In such an environment where IT generally supported internally-centric business processes, separate teams overseeing discrete technology functions was for many years considered the norm. But in today's rapidly evolving digital world, these practices no longer work.

A World of 'Wicked' Business Problems

Keeping up with demand for changes to internal applications supporting business processes is tame compared to the "wicked" problems facing IT departments today. Like societal problems such as global warming and drug abuse, they're wicked because IT is placed in an unenviable position of trying

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_2

to rapidly deliver solutions before problems are fully understood and where business conditions constantly change. This is due to three transformational forces:

- *Products to services*—Customers now care less about physical things and more about the total experience. This explains why companies wrap physical products in services. Like Tesla, who routinely deliver enhancements to the Model S car via software. Or Bosch, who now provides tailored telematics and analytics as-a-services so that fleet operators can optimize maintenance and cut fuel costs.
- *Efficiency to agility*—Established brands with decades of reputation are constantly being disrupted. Kodak lasted 100 years, Blockbuster less than 20. Even technology stalwarts like Microsoft have felt the ground shift beneath them. Business reputations are forged from digital adeptness and the ability stay ahead of the market—or create new ones.
- *Separation to fusion*—There is no longer separation between physical products and software. Is your smartphone circuitry and plastic, or is it a Spotify music service and digital payment system? Is your Nest home thermostat an aesthetically pleasing appliance, or is it an analytical marvel of energy management?

Operating within this environment, wicked problems now challenge the essence of how business is conducted and how applications should be designed, developed, and supported. Thanks to mobility and social computing, the producer-consumer relationship has been reversed, with businesses placed in a position of having to respond to customer behaviors rather than dictating them. This places many organizations on the back foot because of traditional development practices.

Internal business processes supported by large complex applications have been the lifeblood of business. Supporting processes like logistics, inventory, and financial management, these applications are periodically optimized to eke out operational cost efficiencies. In this context, the focus on IT has been to maintain a steady state; keeping the technical lights and only changing applications over longer cycles, sometimes only once or twice a year.

Even when custom development occurs, the general practice has been to invest heavily in application software to support large-scale projects. Here, the pattern is to define all the requirements and features at the start and progress the application toward production status by developing and testing in a linear fashion. Finally, if many business stakeholders agree the system is what's wanted, the application is handed over the wall to production for IT operations to maintain.

www.dbooks.org

Considering these forces, this "Waterfall" style of development (see Figure 2-1) now has a number of disadvantages. First, by establishing a fully defined set of requirements up-front and then failing to accommodate shifting customer behaviors, organizations risk delivering "white elephant" applications, which are software systems that customers never use. Secondly, in the time taken to release software, business conditions may have changed, meaning departments must change the system radically, or as is often the case, abandon it completely. Finally, since application software and functionality is delivered en masse, the support and maintenance burden on IT operations increases suddenly and significantly.

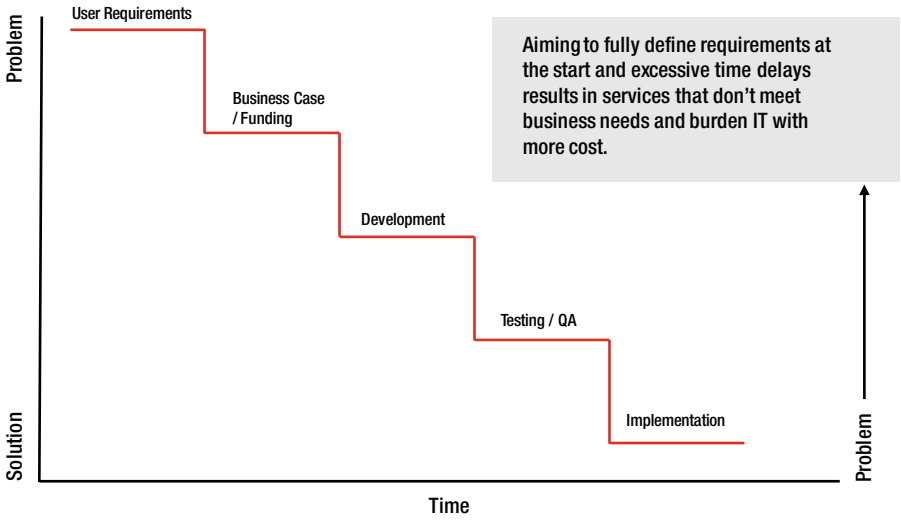


Figure 2-1. Waterfall software development model

The Emergence of Agile Development

In Formula 1 racing, rules change every season. Cars now compete without fuel pit-stops and on sets of tires that degrade more quickly. It's the same in IT, where software must now be delivered as a continuous stream of value that constantly changes to support new business conditions.

Trying to solve wicked business problems (e.g., transforming the businesses operational model with a new mobile sales channel) with existing methodologies like Waterfall can be like trying to corral cats. Problem definitions and requirements can change as new solutions are considered and implemented, so much more flexibility is required across the software development lifecycle. And, with many diverse opinions on what actually constitutes a business problem, there'll be many stakeholders to engage and ideas to analyze.

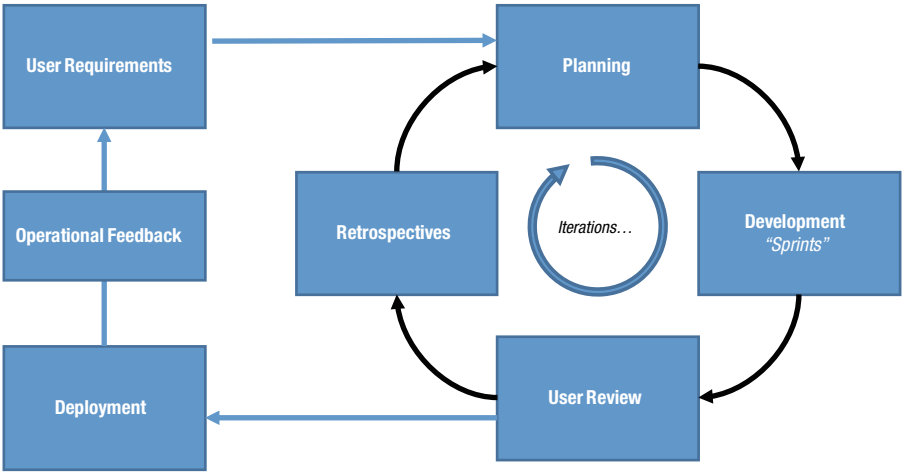


Figure 2-2. Agile software development model

Agile development has emerged as the practice of choice to address these challenges. Agile (see Figure 2-2), which involves iterative problem solving and continuous feedback, is well suited to solving these complex business problems. Some common benefits include:

- Software updates are made in small regular batches, allowing businesses to adapt quickly to new information
- Mistakes or false assumptions are detected much earlier in the software lifecycle, where they are much less costly to correct
- By detecting problems quickly and early, teams have much faster feedback and organizational knowledge improves
- Teams can immediately apply learning and knowledge gained to improve the quality of application software
- By working in parallel across development and testing, agile teams can increase the velocity of application development and delivery

Agile methods help teams manage and run their own projects. Rather than wait for approvals and sign-offs across multiple stages, agile supports the notion that fully autonomous teams should be able to make their own architectural design decisions, even when it comes to tool usage and deciding what's needed to push software code all the way into production.

www.dbooks.org

But agile development is not without its drawbacks. While agile development teams are focused on speed and agility, the traditional mantra of IT operations is maintaining application stability, even if that means slowing things down. To this end, IT operations often employ rigorous change control processes and enforce infrastructure standardization dictates to maintain control. While well suited to supporting legacy internal systems where updates were delivered in large batches, these processes can be too rigid to support newer applications designed to be always changing.

Many of these issues are a direct consequence of how IT operations have traditionally been run. Historically, IT operations have been delivered as a set of shared services, supported by functional disciplines in areas such as data center services, network management, and security. If business users or development needed an enabling service (e.g., provisioning test labs), operations had to be engaged for delivery.

However, this model is changing, with new tools and methods that enable business groups and development teams to bypass the IT operations shared service completely. Many recent technology trends have supported these practices, including:

- *Polyglot programming languages/platforms*—Autonomous teams can select the programming language best suited to their project. This may include older languages like Java and C++, or new platforms and languages to support specific development use-cases or design requirements (e.g., Node.JS, Go, and Rust). In other cases development teams are eschewing traditional relational databases in favor of NoSQL or document style data stores. These could be advantageous in projects where data requirements are indeterminate or evolving and where speed and scalability is more critical than up-front logical design and data integrity. Examples include MongoDB, PostgreSQL, and Cassandra.
- *Programmable Infrastructure*—Also known as Infrastructure-as-code, this allows teams to write code to manage configurations and automate infrastructure provisioning. Rather than using manual methods or scripting to build development ready infrastructure, teams can write code in languages with which they are familiar, together with familiar practices such as version control and automated testing to reduce errors. Unlike scripting, which is primarily used to automate static steps, programmable infrastructure with its descriptive languages allows developers to code processes that are far more versatile and adaptive. Examples include Chef, Puppet, and Ansible.

- *Open source software*—Beyond the more obvious "free to use" benefits, many organizations are embracing open source software (OSS) as a means to increase agility. By adopting OSS, development teams can benefit from community-based and collaborative development, with tools that simplify and automate many tasks (e.g., version control and software build management). By having the eyes of the "community" on the software, bugs can be quickly identified and resolved, with new improvements constantly being developed and delivered.
- *Cloud and platform-as-a-service (PaaS)*—Not too long ago testing a new application or update required development managers to submit a business case for the procurement of new hardware, wait weeks for delivery, then go through a painstaking process of provisioning and configuration. This has changed in recent years—first with the advent of server virtualization and more recently with PaaS providing complete cloud technology stacks for development and testing.
- *Containers*—A recent technology innovation, containers comprise an entire runtime environment, including the application, plus all its dependencies, libraries, and configuration files needed to run it—all bundled into one package. By containerizing the application platform and all dependencies, any differences in OS distributions and underlying infrastructure are abstracted away. Unlike virtual machines, containerized applications run a single operating system, and each container shares the operating system kernel with the other containers. This makes containers more lightweight and resource efficient than virtual machines. Containers can be run on public cloud services and are easily shared, which makes them particularly useful for development and testing teams.
- *Canary releases/dark launches*—This is a release method used to test the performance of software deployments and new functionality in a phased manner. Canary releases are an important aspect of agile development, since, by rolling out new features incrementally and testing them with real users, teams can gather feedback and implement improvements quickly.

www.dbooks.org

- *Design for failure*—As businesses embrace public cloud services, teams are employing design methods to make applications completely independent of the availability of underlying infrastructure. By building resilient systems in which inevitable failures have a minimum impact on service, design for failure allows teams to scale applications quickly while achieving higher levels of uptime, even during major cloud infrastructure outages.

Agile Empowerment Challenges

While many of the technologies and methods described above have empowered teams to deliver software faster, this doesn't guarantee success. While developers are tasked with producing great software code, they often neglect critical issues associated with application performance and supportability. In many cases the adoption of new technologies and practices exacerbates this problem due to some common failings:

Technology for Technologies' sake—It's understandable that developers want to use the latest tool, but this can lead to support problems. For example, opting for a NoSQL database because it helps avoid lengthy schema design issues may address short-term issues, but increases the support burden because IT operations and support have no experience with the technology. Operational cost structures can also be impacted, by way of increased training costs and hiring specialists.

Tip Make new technology tool selection a collaborative exercise. Enterprise architects can coach teams on the importance of placing program or business level objectives above discrete tool requirement or personal preferences.

Misuse of New Technology—It's a fact of IT that new technologies often can and will be abused, unwittingly or intentionally. With modern technologies, lack of experience may entice people to use technologies in ways they were never designed. For example, blindly dictating that every monolithic and legacy application should be containerized whether they're appropriate for the technology or not.

Tip Today's new technology is tomorrow's legacy. Always assess whether the use of technology is appropriate for the business and never underestimate architecture and design requirements.

Metric Misalignment—With developers using techniques like canary releases, dark launches, and split or A/B testing, businesses needs to know whether new features are successful and have led to more customer conversions and sales. Historically, however, IT operations have used technical diagnostics as a means to assess performance, which may be misaligned with the business.

Tip Consider adopting monitoring methods that focus more on achieving desired business outcomes. In a mobile app context, this could include monitoring techniques to analyze app and functional usage by geographic area and user community. By managing to these outcomes, it becomes easier to assess the implications of any application performance issues and feed richer information and insights back to development.

Lost Opportunities—As new development platforms become increasingly abstracted from hardware infrastructure, many complex performance problems can surface. Because developers are shielded from the infrastructure, they can be slow to respond to any issues their code has introduced. This lack of insight also means they can fail to fully exploit the true potential of new cloud-based technologies.

Tip In customer-centric computing, high-performance and low-latency are as important as functionality and design. Consider teaming junior and experienced developers with skilled IT operations and sysadmins to determine how modern hardware architectures can be better exploited to consistently deliver a high-quality customer experience.

Workarounds Due to Constraints—Even as teams look to adopt public cloud services and PaaS to accelerate development, they're still constrained by access to production systems for testing. This can involve delayed access to applications, infrastructure, and perhaps the most time-consuming and resource-intensive task in IT—creating, maintaining, and provisioning accurate, compliant and up-to-date production-like test data. With these constraints, teams may introduce sub-optimal testing practices that fail to detect software defects or compromise compliance with regulatory data protection requirements.

Tip Consider technologies that can simulate unavailable systems across the development lifecycle. This allows developers, testers, and integration teams to work in parallel for faster delivery. Capabilities in test data management, including data subsetting and masking, together with synthetic on-demand test data generation, should also be assessed.

---

www.dbooks.org

Modern Application Architectures

To support the businesses need to innovate, development teams must continuously deliver software services at an increased velocity. This has been recognized by web-native companies such as Netflix and Amazon, who've changed their software architectures to support the need for continuous innovation—essentially using them to redefine the markets within which they operate.

With agile methods, organizations can iterate quickly to support innovation, but teams are also changing application architectures to improve software flexibility and help accelerate deployment. For this reason, older style monolith designs are being supplemented with microservice designs (see Figure 2-3).

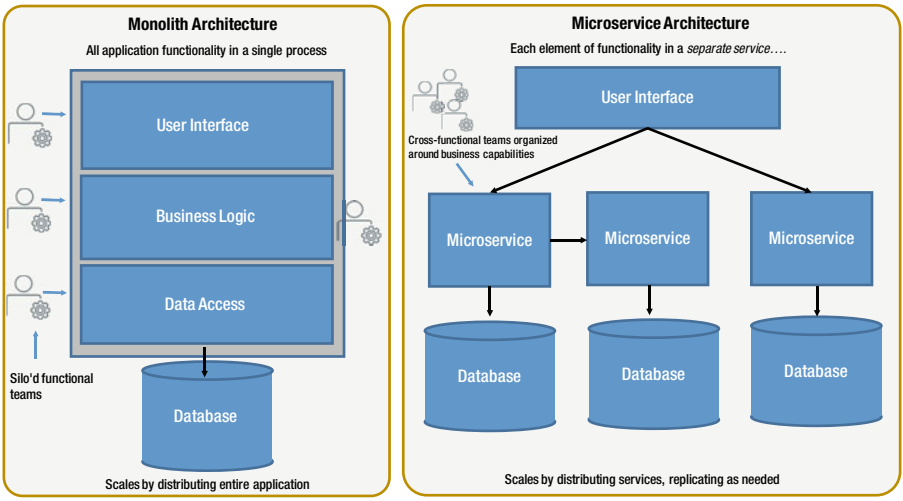


Figure 2-3. Monolithic and microservice application architectures

Until recently, application architectures were monolithic in design and operation. Although consisting of many services, monolithic applications are packaged and operate as a single unit. For IT teams now tasked with faster delivery and deployment, the characteristics of monolithic design have presented a number of operational challenges:

- *Brittleness*—If any single application element of component fails, then the entire application fails. If a task such as payment processing consumes more CPU or memory, then the whole application can degrade.
- *Risk*—Because everything is packaged together (and fails together), teams may be hesitant to change supporting technology stacks and infrastructure. This can explain

operational resistance to increased deployment rates, since even simple application updates to brittle monolithic applications could cause system-level outages.

- *Tightly coupled*—Applications can only be scaled by deploying the entire application on more servers. This can be highly problematic in new mobile application development scenarios when demand is difficult to predict.
- *Dependencies*—Since applications are tied together, developers may have difficulty working independently to develop, test, and deploy their own software components. Development time increases and productivity suffers because they're often dependent on other teams finishing work before they can start.

Microservice designs differ from monoliths in that they involve building applications as a set of small, independent services. In essence, each microservice focuses on a specific element of business functionality. For example, in a web application there could be many microservices supporting everything from login processing and shopping carts, recommendation services and payment processing. Loosely couple in nature, microservices communicate with other services via Application Programming Interfaces (APIs).

Microservices can provide business a number of significant benefits:

- *Independent deployment*—Scaling becomes less problematic. For example, in a web-based shopping application a team can quickly deploy the instances of service it needs to meet demand spikes, while scaling back others.
- *Independent coding*—Teams have more freedom to develop in different programming languages, each optimized for different processing tasks. Microservices can free organizations from being locked into a single technology stack.
- *Fault tolerance*—When a microservice fails, it's unlikely that the entire system will fail. If the recommendation service fails in a web application, shopping cart and payment processing services can continue to function.
- *Increased agility*—Microservices designs can better support continuous delivery. Since systems are built and deployed independently, they can potentially be tested and delivered faster.

www.dbooks.org

Microservices: Small Isn't Always Beautiful

While the simple and elegant nature of microservice style design delivers many benefits, substantial complexity exists in terms of management and operations. If ignored, these issues could increase friction between development and operations teams. Important considerations include:

- *Supportability*—The support burden can increase substantially—especially when IT operations are suddenly faced with managing hundreds of microservices developed in different languages, accessing new data stores, and running on cloud platforms and infrastructure.
- *Monitoring*—Managing a single monolithic application is demanding, but now IT operations has to ensure many more processes remain performant. With highly distributed microservices systems, a whole new set of management considerations surface. These include network latency, fault tolerance, asynchronous messaging issues, and network reliability.
- *Coordination*—Deploying hundreds of microservices demands rigorous deployment processes that can be beyond the capabilities of teams using manual processes and scripting.

Note Modern management approaches to address microservice deployment and monitoring challenges are discussed further in Chapters 6 and 7.

Ending the Technical Impasse

Even with advances in tooling, development methods and software designs, the friction created by teams operating in discrete functional silos can negate all potential benefits. Fractured processes across the application software lifecycle inevitably result in slow delivery, low productivity, and defect-ridden software systems. This has to change.

IT's value proposition isn't just to keep the technology lights on and periodically deliver improvements over long cycles; it's to continuously manufacture business value from a modern high-performance software factory.

DevOps, with its focus on close collaboration between development and other IT groups, while automating essential application delivery processes, is now a critical business imperative.

Summary

The following chapters outline key strategies that can help IT and digital leaders accelerate a successful and business-aligned DevOps initiative.

Starting with Chapter 3, we'll describe how to build a winning DevOps culture and re-energize the IT organization. Here, we'll examine easy and cost-effective ways to increase collaboration, the application of Lean thinking to reduce IT waste, and what constitutes a comprehensive DevOps metrics program.

www.dbooks.org

CHAPTER

3

DevOps Foundations

Culture, Lean Thinking, Metrics

Blink during a Formula 1 pit-stop and you'll probably miss it. But this wasn't always the case. Fifty years ago, a pit-crew would take over a minute to change the wheels and refuel. Today, anything more than three seconds is considered a fail.

It's the same in software development, where teams once tasked with updating enterprise applications at a sedate pace must now deliver new software services as a continuous flow of value to customers.

The problem for today's enterprise, however, is that software teams don't work like Formula 1 pit-crews. Rather than working in tandem, IT teams often work serially—development codes, then QA tests, and finally IT operations monitors. However, with application software released, enhanced, and retired over more compressed timeframes (months and even days), this stop-start method of development falls short. It's as ineffective as each member of a Formula 1 pit-crew replacing a tire and checking wheel nut tension before the next one could start—the race would be over before the car left the pits.

While we can celebrate the heroics and skill of great racing car drivers, what sets successful constructors apart is their ability to build a winning culture irrespective of role and responsibility, be that driver, team manager, telemetry engineer, or aerodynamics chief, everyone is focused on a singular goal—winning races. It's why drivers thank the teams before they spray champagne on the podium.

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_3

Like Formula 1 drivers, technological advancements have improved the efficiency and effectiveness of IT professionals. However, in organizations that traditionally measure and incentivize based on technical specialization within functional areas, relying on tools alone will never build the collaborative culture needed for business growth and profitability.

What Characterizes DevOps Culture?

DevOps is very different from traditional thinking because it places great emphasis on culture. It instills a shared sense of vision across multiple teams, directly aligned to the business and its customers. To this end, maverick behavior, such a cutting corners and allowing defect ridden code to go into production, or blaming operations when a software release fails, is counter to a DevOps thinking. With DevOps unified IT is the hero and no one is singularly to blame for problems.

But this is challenging in IT because of the friction existing between development and other IT teams—especially IT operations. On the one hand, developers are focused on accelerating change by faster delivery of applications, while the operational mantra has been resilience and stability at all costs, even if that means holding back change.

Evidence suggests, however, that while both these goals are equally important, they are not mutually exclusive. For example, the 2016 Puppet Labs "State of DevOps" report illustrated that high-performing IT organizations are well able to achieve faster software delivery along with increased resilience and stability.[1] Clearly, DevOps high performers have ended the divisional "turf wars" by enacting strategies to re-shape entrenched silo thinking and behaviors into a more powerful collective force.

Since DevOps culture involves creating new shared values and behaviors across IT teams, leadership must play an active role in driving these characteristics across the entire organization.

Focusing on Products over Politics

Traditionally, IT teams have been organized in technical silos. Interaction and communication has been conducted through overly engineered and rigid processes. Software changes run the gauntlet of lengthy change-management processes, human intervention, and change review boards. Though not wrong per se, these elements were designed to cater to situations where change was less frequent but occurred in greater volumes, requiring more rigor to ensure operational stability and compliance.

[1] https://puppet.com/blog/2016-state-of-devops-survey-here

www.dbooks.org

A strong DevOps culture, however, is characterized by systems thinking. That is, a collective emphasis on service as a whole, not on discreet functional elements or processes. Rather than persist with technical fiefdoms, DevOps aims to break down barriers—organizing by product over structure and continuously driving improvements in context of a products lifecycle, from the inception of an idea to full production status. Strong leaders recognize this by promoting open communication, using shared metrics, and establishing (even automating) feedback mechanisms within and across teams.

Building Trust and Respect

Over many years, respect has been garnered by individual contributors. Be that superhuman developers who crank out code, or on-call operations staff who fix problems at 4:00am. In a thriving DevOps culture, hero worshipping takes a back seat to collective respect. With DevOps, everyone should respect the contributions of others and no one should be afraid of speaking up for fear of abuse and vilification.

This is critical, because from healthcare to aerospace, studies have shown that bad practices and behaviors can over time become accepted as normal practices—often with disastrous consequences (see Chapter 8 for further discussion on strategies to combat normalized bad practices). In IT this happens all the time due to power games and lack of respect. Even if new staff witness blatantly suboptimal practices, they'll be loath to report it for fear of rebuke and retribution by managers, eventually accepting the situation and practicing it themselves. DevOps leaders should be mindful of this and work across the product lifecycle to identify situations where violations are tolerated because people are afraid to speak up or look mean.

Trust also plays an important role in DevOps culture. Just as the Formula 1 driver trusts his pit-crew to fit four wheels securely, so must cross-functional trust be established across IT. For development, this means trusting that the production performance information from operations can actually help in software refactoring and reducing technical debt. For operations, it means trusting new application design patterns will help the business scale. Everyone from security to enterprise architecture is part of the trust equation, and as the speed of software delivery accelerates, no DevOps program will be successful without it.

Increase Empathy Everywhere

It's well understood how important the role of empathy plays in today's app-centric software design. Without understanding the emotional and physical needs of customers, together with their behavioral patterns, businesses risk substantial losses from their software investments. This explains why many

organizations conduct rigorous design experiments before any full software release. This is illustrated in the extreme by Google's "50 Shades of Blue" user interface testing exercise.[2]

Yet despite this, empathy is lacking within many enterprise IT departments. Teams usually operate in separate locations, so development and operations teams have few face-to-face opportunities necessary to share each other's pains, surface concerns, or raise issues.

There are many simple but effective strategies leaders can use to build empathy. Not the least this should include building closer ties between development and support. Even with the greatest software and delivery processes it's important to understand their perspective and what they experience when dealing with customers.

Tip When developing products or new features, put yourself in the position of the customer and support staff by examining all the situations where they might need help.

Staff (including developers) should understand the importance of enabling a great customer experience. To that end, consider working directly with customers directly in the field in a variety of adverse situations. What happens when as a customer you're trying to board an airplane and the scanner breaks? Or what's the impact when a mobile app crashes or bad network coverage means you can't call road side assistance?

Obviously it's not always practical or feasible for developers to work this closely with clients; however, with analytics tools, staff can put themselves in the "shoes" of the customers and gain realistic insights into the customer experience.

Open Communication Channels

In 1968 a computer programmer, Melvin Conway, postulated that organizations that design systems are constrained to produce designs which are copies of the communication structures of these organizations.[3] In a DevOps context, what's now dubbed Conway's Law has great relevance, especially in situations where organizational structures and closed communication channels prevent developers and operations from agreeing on IT performance objectives (e.g., increased change frequency and improved reliability). In such cases, it's possible for team-based activities to be prioritized over more important cross-functional improvement strategies.

---

[2] https://www.theguardian.com/technology/2014/feb/05/why-google-engineers-designers

[3] Melvin E. Conway, "How do Committees Invent?," *Datamation* 14 (5) (April 1968): 28–31.

www.dbooks.org

There are many possible solutions to this problem. The technology teams at Netflix and Amazon structure themselves around small teams, with each one responsible for a small part of an overall system. Spotify promotes collaboration across team boundaries via agile development squads, chapters, and guilds, with a separate IT operations team providing all teams the support needed to release software themselves.

Looking beyond technology-centric companies, there are other examples of businesses thriving because they've modified communication structures. Take Zara for example.

Year after year in the fickle world of retail fashion and apparel, Zara continues to increase revenue and profit. For Zara, flexible responsiveness to customer demand, backed by a tightly integrated supply chain, is fueled by teamwork and collaboration. In retail stores, managers use real-time intelligence to place orders and feedback information directly to the point of manufacture. Different teams (including design, product management, and merchandising) use shared spaces and work closely together. With increased emphasis on communication from initial garment design to distribution to the shop floor, Zara has shortened product lead times and doesn't unnecessarily commit large volumes of product in advance of a fashion season.

Considering approaches like these, cross-functional IT collaboration could be improved when:

- Leaders apportion budget to practical co-location strategies. This should be more than simple staff re-housing and include shared work spaces and lounges, together with team huddle areas and common wall whiteboards.
- IT operations team members regularly participate in agile standup meetings in order to appreciate the value of deploying code quickly and how their current activities help or hinder release processes.
- Development teams attend operational post-mortems or workshops to gain better insights into the problems caused by poorly performing or insecure software.
- IT operations works with developers to establish performance monitoring in pre-production so as to detect problems earlier where they are easier and less costly to fix.
- Developers are placed on the after-hours support roster to better appreciate the impact of problematic software code on users and customers.
- Support specialists share critical application experience analytics obtained during mobile app engagements with customers.

Additional Factors

Changing IT culture isn't easy. Right or wrong, people have pre-conceived notions and firmly entrenched ideas. Any sudden shift in workplace practices and it's only natural that people will feel threatened and push back. Addressing this means patiently working with people to enact the necessary behavioral change or move people to different jobs.

With DevOps and cultural change, it's critical to start with a clean state. Rather than dive head first into massive IT workforce transformation programs, leaders should first assess the cultural landscape from a business perspective. This involves understanding the primary goal of the business and then analyzing whether prevailing behaviors support it. Although seemingly obvious, many organizations miss or neglect this critical step. If, for example, a company defines its goals too broadly, people working in different IT teams will interpret them in different ways and shape activities accordingly, often to the detriment of each other and the business.

In a broader sense, culture will also be influenced by an organizations' business model and operational perspective. There are three classes to consider:

Run the business—In this model the overarching strategy of the business is on continuing operations in much the same way—only better, faster, and cheaper. Here the focus for IT is operational excellence—adopting new technologies, yes, but using them to pound out efficiencies across processes like warehousing and logistics.

For these organizations, IT culture is characterized by discipline and rigidity—all fine to support efficiency improvements, but inadequate in a world where old business rules are constantly being challenged by disruptive technology.

Grow the business—The business strategy shifts from doing more of the same to conducting the same business in radically different ways. Netflix presents a good example of this model. Five years ago, Netflix delivered DVDs through the mail, now they stream entertainment over the web—even creating their own content. Customers still turn to Netflix for entertainment, but the way in which Netflix is addressing that need has fundamentally changed.

For organizations in this category, the culture also needs to change. For Netflix, their DVD-delivery model required strong operational oversight over processes like inventory management and distribution to drive

www.dbooks.org

efficiencies and increase customer satisfaction. Now however, their streaming model and content generation programs requires teams rapidly delivering new services based on quickly analyzing customer preferences and optimizing web performance and throughput. Obviously if a company's cultural behaviors and values are still skewed toward driving efficiency in one operational area, pivoting to the new model will be difficult.

Transform the business—This model carries the most promise and risk because it involves changing the very fabric of a company. Businesses don't just conduct the same types of business in new ways, they reinvent themselves completely. For Amazon that's meant moving to being a mega cloud computing provider from selling books. For Walgreens, it's meant going from selling medicine over the counter to treating illnesses in stores.

With strategic transformation examples like these, the business is introduced to new dynamics and competitors. Now, IT performance will not only be judged on growing the business of today, but also on creating the core business in the future. To this end, a DevOps culture built on open communication and collaboration, trust, respect, and empathy isn't just important for short-term growth, it's essential for long-term business sustainability.

Once business models and goals are clearly understood and communicated, any required IT behaviors and values needed to support them can be influenced through the development of fully aligned IT goals and metrics. These could include shortening lead times for new products to support faster time-to-market, increasing mobile app customer conversions to support increased revenue objectives, or helping the current business scale by making better use of cloud infrastructure.

Lean Thinking to Reduce Waste

Fuel strategies play a significant role in Formula 1 racing. A car with a half-full tank can be as much as three seconds faster than a rival vehicle with a fully loaded fuel cell. Extra fuel equals extra weight, so teams go to great lengths to calculate the exact amount of fuel needed under full race conditions.

Though beautifully engineered and continuously refined, racing car engines are still flagrantly wasteful. Like your car at home, the dynamics of internal combustion still mean that only a certain percentage of fuel stored in the tank is converted into useful energy. The rest is lost as heat and friction and explains why teams constantly refine chassis designs to reduce aerodynamic drag.

Beyond the frenzied world of Formula 1, many elements contribute toward engine waste in the cars we lesser mortals drive. Running an air conditioner consumes fuel without contributing to motion. Friction in engine pistons wastes fuel, as does tire pressure and extra luggage. All told, as little as 14 percent of passenger car fuel is converted into useful energy. Clearly, this gas guzzling engineering throwback is rife for disruption from electric cars and advances in battery technology—time will tell.

In many ways software development is as wasteful as the internal combustion engine. Friction between development and operation causes delays. Manually assembling multiple components and configurations within our own software factories leads to lost time. Carrying excess inventory in the form of unneeded infrastructure capacity adds extra cost. Add to this constraints preventing access to critical systems and data during testing, and defects can accumulate across the software lifecycle.

Lean and Value Creation

The traditional view of IT value has been internally-shaped. For decades, systems and applications have been designed, built, tested, and released to customers, citizens, and end users where they hopefully influenced behaviors. All this has changed. With the advent of cloud, mobility, and social computing, consumers rather than producers call the shots. This means businesses find themselves in the position of having to respond to the behaviors and desires of their customers.

For IT, this redefinition of business value means teams must focus on two essential strategies. First, they must continuously reexamine the software services they deliver from the perspective of the customer, and second, they must constantly strive to minimize any interference or waste across the entire software factory. This includes everything that impedes the flow of value to customers and incurs more cost.

This notion of value being "pulled" by customers and waste elimination is not new. Lean pioneers and practitioners such as Toyota, Motorola, and Xerox redefined manufacturing by applying these principles; understanding that many forms of waste exist across production processes. And, because they add no value to customers, must be clinically removed.

www.dbooks.org

But can Lean principles be applied in IT, where, unlike traditional manufacturing, waste isn't visible across a factory floor through telltale signs like excess physical inventory or idle machinery? Often due to its intangible nature, waste in IT can be hard to identify yet alone eliminate.

Interestingly, the delivery of software bares many similarities to a manufacturing process. In IT, we have the means to respond to value triggers from our customers by quickly designing, developing, and releasing software services. And, since the software delivery lifecycle represents a manufacturing production line within a software factory, we have the guiding context upon which identify all elements of waste that add no value to the business and its customers.

Eight Elements of Waste

As illustrated in Table 3-1, there are eight elements of waste or "Muda" (using Lean terminology) that severely impact the IT group's ability to increase the value of software services.

Table 3-1. Eight Elements of Waste (D.O.W.N.T.I.M.E)

Type of Waste	Examples	Business Outcomes
Defects	Badly designed and poor quality code	Lost customers and revenue; negative brand impact
	Non-functional performance issues	
Overproduction	Delivering features customers don't need or want	Delays, cost over-runs, and budget problems
	Procuring extra capacity due to unanticipated performance requirements	
Waiting	Excessive release backlogs and bottlenecks	Slow time-to-market and value; lost opportunities
	Infrastructure and data not available for testing	
	Change reviews; security and compliance audits	
Non-value added processing	Lengthy problem resolution and fire-fighting	Morale issues; high-staff turnover
	Team-based activities prioritized over program-level objectives	
Transportation	Frequent release rollbacks	Application launch delays; increased cycle times
	Development/QA handoffs	

(*continued*)

Table 3-1. (*continued*)

Type of Waste	Examples	Business Outcomes
Inventory (excess)	Underutilized resources Partially completed work and excessive work in progress	Increased capital and operational costs
Motion	Developers constantly switching Relearning and rework	Lost productivity; talent erosion
Employee knowledge (unused)	Closed retrospectives and stand-up meetings No feedback established from service management (e.g., call center/ service desk)	Missed opportunities to drive improvements

Examining this table it should be noted that there are close relationships and linkages between elements. For example, undetected code *defects* resulting in performance problems may result in an organization purchasing additional hardware capacity, which leads to excess *inventory*, which increases the support burden. In situations like this, waste begets waste and technical debt accumulates to such an extent it becomes difficult to pay off. The result is that essential development is tied up on maintenance and support activities.

Originally coined by Ward Cunningham in the Agile Manifesto, technical debt has tended to be reviewed from a development perspective[4]. After all, if software defects can be identified and eradicated during early stages of development, then production related problems (which could be significantly costlier to fix) can be avoided.

But technical debt can also be created in IT operations. For example, failing to document or visualize business services (and supporting applications an infrastructure) means teams could take longer triaging problems. Here the waste is *non-value added processing*, which again (because of linkages) results in more waste. In this case, increased *transportation* because a release has to be rolled back.

Using the eight elements of waste list, DevOps practitioners can begin a process of identifying waste elements across the software lifecycle. It's important to understand that "toxicity" levels will vary, so mechanisms must be developed to continuously reveal new situations and conditions that can potentially introduce more waste.

[4]The Agile Manifesto: http://www.agilemanifesto.org/

www.dbooks.org

It's also critical not to restrict the exercise to new development. These may become the debt burden of the future, but could only represent a small part of the portfolio. Legacy infrastructure and production applications should also be included because, even though they change less frequency, they often incur significant management costs and overheads.

Finally, debt and the associated waste should be reviewed as a continuum, with special attention paid to integrations between new customer facing apps and essential back-end business processes. For most enterprises, multi-channel engagement creates tremendous opportunity for value creation, but will introduce more waste if they're not integrated and coordinated with existing back-end systems, applications, and call-center services.

Waste Removal Strategies

Today's mobile and API-centric forms of service delivery mean that customers assess value based on extremely high levels of functional and operational quality. They also expect businesses to deliver additional value in the form of continuous change. With customer experience so important, it's critical to begin waste identification from the perspective of the customer; monitoring and analyzing the usage and behaviors of applications and determining what elements impact the total experience. This is especially important for mobile applications, since factors beyond the control of IT departments (e.g., carrier network latency and cloud service performance) can quickly erode value, however good the functional quality.

Some immediate practices cross-functional teams can apply to help identify and eliminate waste, include the following.

Prevent Defects by Removing Constraints

When development and testing is constrained due to lack of access to dependencies (e.g., middleware, web services, and test data), defects can quickly work their way into the code base. This is illustrated in a Service Virtualization survey, which identified that on average, participants require access to 52 dependent elements for development or testing, yet have unrestricted access to only 23 of these.[5]

To circumvent these issues, many development teams often attempt workarounds by hand coding (mocks and stubs), but this doesn't provide for realistic application behavior, causing test validation errors and the late

[5]VOKE Market Snapshot™ Report: Service Virtualization; https://www.ca.com/au/register/forms/collateral/voke-market-snapshot-report-service-virtualization.aspx

discovery of defects. Discussed further in Chapter 5, a more scalable approach is to incorporate Service Virtualization into parallel development and test activities.

Focus on Value to Prevent Overproduction

New application features don't necessarily mean more customer conversations and increased revenue. Unnecessary features can result in additional maintenance overheads and cost. There are many methods DevOps practitioners can use to reduce this form of waste, including:

- Incorporating application experience analytics into monitoring strategies to identify mobile app functions and features that are not used
- Split or A/B testing and funnel or cohort analysis
- Refactoring code elements to reduce complexity, remembering that the cheapest and most reliable components are those that don't exist!

Smoothing Flow to Reduce Wait Times

Like waste element #1, this waste can eventuate due to delays waiting on dependencies during development and testing. In the VOKE report mentioned above, 81 percent of participants identified development delays of waiting for a dependency in order to develop software, reproduce, or fix a defect. Additionally, 84 percent of participants identified QA delays of waiting for a dependency in order to begin testing, start a new test cycle, test a required platform, or to verify a defect.

Caution Never underestimate the wait times associated with accessing test data, since a massive 20 percent of the average software delivery lifecycle is wasted waiting for data, locating it, or creating it manually when none exists.

Excessive wait times may also be due to problems managing highly complex release and deployment processes. However good the code, its ultimate value will be determined by how quickly it can be deployed into production. Manual processes and fragile scripting not only compromise these goals, but also increase the potential for defect code being released. These issues can be addressed by:

- Ensuring all key stakeholders possessing the knowledge to move a service swiftly across lifecycle are involved early and often

www.dbooks.org

- Using smaller batch sizes so that value is delivered to customers at regular intervals
- Developing and automating reusable and repeatable processes to simplify and streamline application releases

Limit Non-Value Added Processing Through Data-Driven Insights

Fixing application problems provides limited value to customers. Rather than wait for problems to occur in a production, IT operations should be involved much earlier in the development lifecycle.

Using tools to share information is especially valuable. By leveraging application performance change impact analysis during a build process, for example, developers can quickly determine any adverse performance conditions their code is introducing.

Reduce Transportation Cost by Automating Deployments

When work is manually handed off from one team to another (e.g., developer to test/QA, QA to operations), critical knowledge can be lost. This could lead to additional delays or the highest transportation cost of all—release rollbacks.

There are many strategies to address these issues, including:

- Reducing the number of handoffs by automating standard tasks and activities
- Ensure release automation tools provide an extensive set of action packs and plug-ins so as to fully deploy at an application level, while also integrating key supporting processes (e.g., configuration management)
- Build more knowledge as releases progress (e.g., establishing application performance management in pre-production)

Eliminate Excess Inventory Across the Software Factory

Minimizing inventory is the hallmark of Lean thinking. As in traditional manufacturing, there are many waste indicators in IT's own software factory. In development, partially completed work can become obsolete before it finds its way into production and should be exposed to ensure it doesn't degrade or corrupt the code base. In operations, excess on-premise server infrastructure acquired as a fail-safe to address unanticipated performance problems could be avoided by establishing monitoring in pre-production.

Note Costs can accumulate substantially when agile teams acquire specialist tools. Work collaboratively to assess whether the additional cost (training or support) offsets the value delivered to one team.

Prevent Unnecessary Motion with Parallel Development

While transportation waste is associated with the unnecessary movement of software, motion waste involves the unnecessary movement of people. A good example is task switching, where an API developer might shift focus to a new project rather than wait for testing dependencies to become available.

Apart from adding more waste (e.g., delays), task switching can introduce many more problems, especially related to the productivity of developers due to constant interruptions.

Some simple strategies to reduce this waste, especially task switching, include:

- Try to ensure teams have all of the knowledge, tools, and data needed to complete their assigned work
- Simulate and virtualize all dependencies so that development teams can code and test in parallel
- Since as much as 50 percent of testing is wasted by teams trying to locate test data or create it manually, consider supplementing constraint-removal strategies with test data management (see Chapter 5)
- Aim to eliminate unimportant work, meetings, and interruptions. If it isn't delivering value, ask why your team is doing it!

Incorporate Employee Knowledge Using Feedback Loops

While the feedback of production information is important to drive software improvements and improve supportability, it isn't the only place where knowledge can be transferred.

Service desks and call-center processes should also include mechanisms to deliver (to development) important information gained from customers on their usage and response to new application features and functions. Knowledge transfer should also be bi-directional. For example, application experience analytics could (when integrated with incident management processes or even social media) become an early warning mechanism to trigger coordinated responses in the event of mobile app usage problems.

www.dbooks.org

DevOps Metrics

With any IT-driven methodology or program, measuring the effectiveness in a business context is critical. But since DevOps isn't a formal framework, organizations have little guidance in determining what metrics should be used.

This can be problematic and lead to a number of suboptimal practices:

- *Efficiency status-quo*—The IT team falls back to metrics traditionally used to demonstrate technical proficiency in meeting stability and resilience goals. Although these are not necessarily wrong, DevOps metrics should also demonstrate how new processes and automated technologies are impacting the business—for example by speeding time-to-market and reducing lead times.
- *Outputs over outcomes*—Organizations gravitate to metrics that are commonly used in assessing team-level productivity. These can include output-based metrics like number of features delivered or servers provisioned. Metrics in this class can be counterproductive unless balanced with outcome-centric indicators that show results achieved against desired quality levels.
- *Low-hanging fruit*—Organizations select metrics that are easily obtained but not necessarily useful. Since DevOps success is predicated on cultural change, businesses must also measure what's harder to determine but potentially more valuable—namely, how the adoption of DevOps behaviors and values at an organizational level is impacting the business.

Anti-Pattern Metrics

Before embarking on a metrics refresh, organizations should consider all existing measures and their applicability in a DevOps context. Particular attention should be given to carefully review those metrics and incentives that are counter to DevOps principles, as illustrated in Table 3-2.

Table 3-2. Problematic Metric Classes

Metric Class	Examples	Adverse Effects
Vanity Metrics	Lines of code produced Function points created	May be counterproductive since they reward the wrong types of behavior—especially if incentives are linked to the metric. Producing more code and features without validation can inhibit other valuable activities such as refactoring and design simplification.
Intra-Team Metrics	Agile team leaderboards Deployments/ changes prevented	Beware of metrics that pit-teams against each other and use vanity metrics as scoring mechanisms. Strike a balance with metrics and rewards that influence positive inter-team behaviors—such as code sharing, peer reviews, and mentoring. Pay particular attention to metrics that promote an anti-DevOps culture, such as rating operational effectiveness on the ability to prevent releases and deployments.
Traditional Metrics	Mean-time-between-failure (MTBF) FTEs: Servers	With faster delivery of services, some failure is to be expected. Always consider that improving responsiveness can be more important (and less costly) than trying to prevent failures.

Suitability Checklist

When reviewing and developing DevOps metrics, it's also important to consider each against a general suitability checklist:

- *Obtainable*—Culture and behavioral improvements are important to measure, but metrics may be difficult to obtain or quantify. Seek out other related data points to help expose —e.g., staff retention rates/transfers as an indicator of employee morale.
- *Reviewable*—Every metric must stand up to rigorous scrutiny in a business context. Carefully review metrics that can be easily collected, but add no tangible value—e.g., lines of code produced per developer.
- *Incorruptible*—Determine whether each metric can be influenced by team and employee bias. Seek out any associated incentives that can work against a collaborative DevOps culture—e.g., existing SLA bonuses inhibiting change.

www.dbooks.org

- *Actionable*—Any metric must support improved decision making. Exposing A/B testing results can for example be a valuable way to quickly determine the effectiveness of new functionality.

Wherever possible, metrics should also be shareable and have relevance across the software lifecycle to both development and operations. For example, generating security scores at a cross-functional team and divisional level can be used to inform teams about the risks of their actions.

Metrics that Matter

Having determined what not to measure, the next stage is to develop a candidate list of metrics supporting the DevOps program. One common mistake is to measure too many elements, falling back to what's easily collectable. Additionally, metrics applicable to DevOps may be new to organizations (e.g., the speed of deployment, rate of change, and customer responsiveness), so it's important to think broadly how changes to work practices, process and technology can support these goals.

- *People*—Staff related metrics can be the most difficult to collect but are still powerful change indicators. Strong consideration should be given to internal metrics like staff retention rates and training, together with mentoring and knowledge building (e.g., open source contributions and wiki development).
- *Process*—It's important to consider how existing practices will help or hinder new targets being achieved, paying special attention to existing bottlenecks (e.g., security audits only conducted after testing will impact deployment rates).
- *Technology*—Good metrics are those that help teams drive improvements, even after failures (e.g., what is the percentage of failed releases and what percentage of these were due to code defects, manual processing, configuration errors, etc.).

When developing metrics, it's important to maintain balance. Defaulting to metrics skewed toward one particular area (e.g., operational or development efficiencies) can have a negative effect in terms of behavioral improvement

Figure 3-1 illustrates four dimensions and sample metrics that can be used to measure the effectiveness of a DevOps initiative.

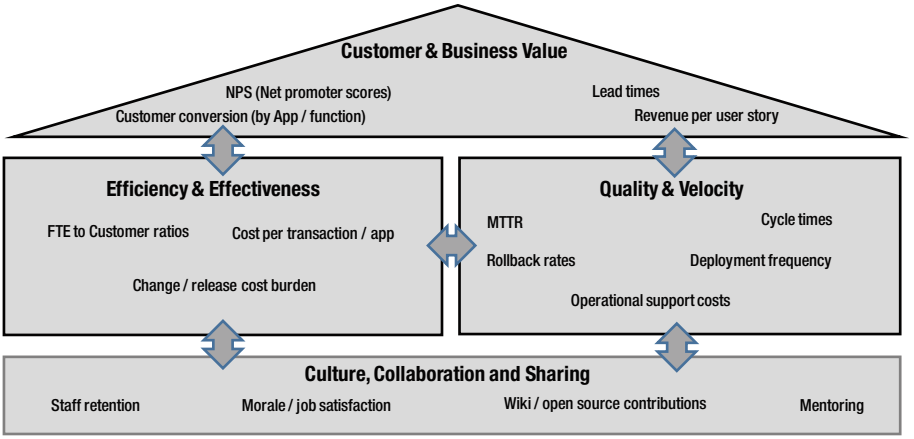


Figure 3-1. DevOps metrics dimensions

Culture, Collaboration, and Sharing

Metrics in this category are especially valuable because they provide an ongoing indicator of acceptance/resistance to DevOps. Some metrics in this dimension will be easier to collect (e.g., staff retention rates/turnover) than others (e.g., employee morale). It's important therefore to look at measures across other dimensions to understand how they impact this area. For example, are mean-time-to-recover (MTTR) improvements positively impacting staff morale, absenteeism rates, and responsiveness to change? Consideration may also be given to automated surveys and employee feedback, as long as these are fully transparent and actionable.

Efficiency and Effectiveness

Metrics here normally focus on elements of development capacity and operational capabilities. While traditional metrics such as server to sysadmin ratios have been used, many organizations are now adopting more customer-centric ratios like full-time-equivalent (FTE) to customers.

Examining full costs on a transactional or application basis is another good candidate metric, as it's focused on improving data center efficiencies (e.g., energy and cooling). Other metrics such as cost of release are also good since these can expose inefficiencies associated with acquiring, preparing, and maintaining physical infrastructure for development, testing, and production.

www.dbooks.org

Quality and Velocity

This dimension looks to measure data points with respect to service delivery. For organizations starting on a DevOps initiative, many indicators (e.g., percentage of deployments rolled-back due to code defects/outages/negative user reactions) could initially be high. This may be a result of the extra time needed to adopt new processes, combined with remediating existing technical debt and waste elements these metrics expose. However, with DevOps' focus on establishing quality right from the start of development, this should reduce over time.

When paired, these metrics also provide additional insights. For example, if the *rate of rollbacks* still increases during periods of low *change volume* it could be indicative of serious problems, e.g., errors due to manual/scripted release processes, task switching, and excessive handoffs.

Other useful metrics in this dimension include:

- *Cycle time*—Measures the length of time it takes to complete a stage or series of stages in a release operation. This can be extremely valuable in exposing any bottlenecks.
- *MTTR*—This can be broken down into detection, diagnosis, and recover phases. MTTR is a great indicator of how effective teams are in handling changes. For complex deployments, there will be spikes, but this metric should be trending down as DevOps becomes established.

Customer and Business Value

This category of metrics are externally focused and help measure how DevOps supports business goals—like increased customer loyalty and faster time-to-market. The manufacturing concept of lead time provides DevOps practitioners with an analogous metric (time taken from when code starts development to successful production deployment) and determines how well DevOps is at meeting the need for rapid delivery of high-quality software services. This metric is especially important to scrutinize because long lead times could be indicative of code defects or testing constraints.

Another interesting candidate is Net Promoter Score (NPS), which is a simple management method to measure customer loyalty. While this metric has traditionally been used in other areas of the business (e.g., marketing), its inclusion is valid since the loyalty of customers is increasingly determined by how quickly high-quality software services and updates can be delivered to via web sites and/or mobile apps.

Additional Methods and Techniques

With metrics developed across each of the four dimensions discussed previously, teams can begin a process of determining the relationships between them. This is important so teams gain insight into what processes enhancements and tools are needed to meet targets or address capability gaps.

One simple and effective approach, as illustrated in Figure 3-2, is business impact mapping. This involves determining which DevOps processes will be needed to support a business or customer experience goal, together with the underpinning metrics, targets, and initiatives/tools across multiple dimensions that support this outcome.

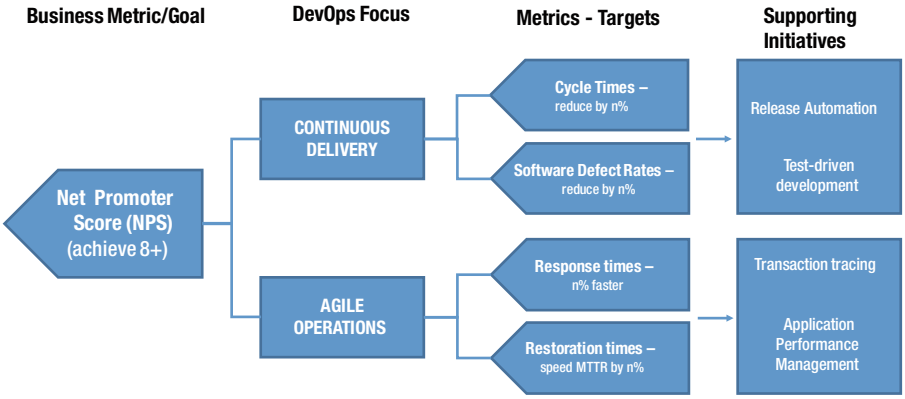


Figure 3-2. Metrics, targets, and initiatives linked to business outcomes

Figure 3-2 illustrates that an organization is seeking to achieve a Net Promoter Score of 8+. To support this goal, IT needs to deliver software releases and new functionality faster, together with ensure a high-quality customer experience. Metrics and targets have therefore been set within the quality and velocity dimension, together with targets and supporting process/tool initiatives.

As DevOps metrics programs develop, practitioners should also:

- Have regular and ongoing target reviews to ensure that goals are not completely unrealistic or that existing processes and tools are not delivering improvements.
- Consider removing persistent "green light" metrics when targets have been consistently achieved.
- Avoid having every metric focused on velocity without paying attention to customer satisfaction and loyalty.

www.dbooks.org

- Strive to prevent vanity metrics and operational or team-centric bias creeping back into the program and distorting the true performance picture.
- Beware of ranking teams based on targets—the best way to compare teams is to measure things like customer loyalty (as described) and how successful teams are in meeting their commitments.
- Give strong consideration to metrics, targets, and initiatives that foster peer review and openness.
- Carefully build incentives and reward programs that reinforce the value of a strong collaborative culture.
- Involve business counterparts right from the start to ensure customer and business data is held to the same standard as operational/efficiency metrics.
- Match tools to the DevOps program, especially those that can monitor and respond to real-time conditions (such as transaction times, response times, and mobile app crashes), but can also proactively detect and prevent adverse conditions (such as code defects and release bottlenecks) that impact performance.

Summary

At its heart, DevOps is about building a generative organizational culture where business improvement is placed above everything else. But as this chapter has illustrated that won't always be straightforward, especially in organizations beset by divisional friction and lack of direction. By leveraging this chapter's guidance, especially with regard to building high-trust teams, an outcome-based metrics program and Lean thinking, organizations have a solid foundation upon which to guide their DevOps programs.

In Chapters 4-7, we'll look closely at the automated tooling needed to support this goal and how businesses can refit and re-engineer their own software factories to manufacture high-quality software innovations, at speed. In these chapters, we'll examine critical tooling strategies across the software lifecycle continuum—Build, Test, Deploy, and Manage.

PART

II

Essential DevOps Tooling

www.dbooks.org

CHAPTER

4

Build

APIs for the Connected Business

In the digital economy, building application programming interfaces (APIs) is essential for executing ideas quickly and seizing new business opportunities. APIs are the building blocks of digital transformation, enabling organizations to deliver exceptional customer experiences, create new revenue streams and connect employees, partners, apps, and devices to data—anytime, anywhere. APIs are not necessarily a new technology, but in today's digital world, they have risen in prominence and become important to every facet of the enterprise. This in turn has increased the demand for effective API management.

But what does an effective API management look like? A good place to start answering this question is by examining a case study.

Case Study: IceMobile

IceMobile helps leading food retailers across the globe boost footfall and basket size. Ralph Cohen founded the then creative agency in 2002 after recognizing that mobile devices offered a great potential for increasing customer engagement, loyalty and, ultimately, spend. [1]

In 2012, IceMobile took two steps that transformed its business into the global mobile loyalty program provider it is today. First, it merged with BrandLoyalty, extending its reach beyond the Netherlands. Secondly, it built the Bright Loyalty

[1] Full story: http://www.ca.com/content/dam/ca/us/files/case-studies/icemobile-cuts-rollout-times-with-ca-api-gateway.PDF

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_4

Platform, which helps food retailers engage with shoppers on a personal level, driving turnover, increasing basket size and bringing in new customers.

At the solution's front-end, the Bright Stamps product makes loyalty campaigns digital. Shoppers can easily collect stamps and redeem using their mobile phone, resulting in increasing spend and more people joining the campaigns. At its back-end, the Bright Loyalty Platform analyses shopper data from retailers' back office point of sales (POS) and customer relationship management (CRM) systems to ensure the shopping experience resonates with customers.

Located in offices across 14 countries, IceMobile's 120 developers, designers, and engagement experts offer retailers a range of loyalty services. These include concept and strategy, user experience design and research, solution design, implementation, project management, and data analysis.

When IceMobile talked to retailers regarding its product Bright Stamps, despite commercial departments being enthusiastic about adding the Bright Stamps solution to their loyalty programs, retailers' legacy IT systems were often a hurdle to overcome. "Retailers were concerned their IT departments, which were often starved for time and budget, would not have sufficient resources available for integrating IceMobile with their various legacy systems," reveals Jeroen Pietryga, IceMobile's Chief Executive Officer.

The company needed to find a safe and manageable way of accelerating and simplifying the integration between its Bright Stamps solution and retailers' back office systems. "We wanted to take some of the IT headaches and risk away from retailers' IT departments," adds Pietryga. "We wanted to make implementation faster, cheaper, and a better experience." Application Programming Interfaces (APIs) gateway provided IceMobile with the perfect solution. An API gateway solution would only require limited changes to the food retailers' back office systems while reducing the amount of time IceMobile engineers needed to spend on implementing the integrations.

By simplifying integration, IceMobile has cut implementation times from 14 to 8 weeks while minimizing impact on retailers' busy IT departments.

The solution also ensures IceMobile meets security standards for inter-operating with retailers' IT systems; ensuring the digital stamps IceMobile collects and stores for customers are safe.

The API gateway also enables IceMobile to:

- Minimize impact on its own and retailers' IT departments.
- Integrate with almost any technology used by a retailer, safeguarding sales. To date, the solution has simplified connections with multiple back office systems for various food retailers. These back office systems include POS systems, such as Wincor Nixdorf, NCR, and IBM, and CRM systems, such as Siebel, eFuture, and SAP.

www.dbooks.org

- Standardize the way it delivers its services as it expands across the globe.

From Little API Acorns Big Things Grow

Like IceMobile, many companies have wisely invested in an API strategy and are reaping the benefits. Of course, this includes some pioneers, like Netflix and the streaming of content to over 200 devices types, thanks to APIs, or Twitter, who receives a staggering 13 billion API calls to its platform each day. But it's more about big business than big numbers. APIs enable Expedia to conduct $2 billion of business through its partner affiliate network, and 60 percent of eBay's listings are made through its API.

While software-driven companies have much to benefit others in the "physical product" world have realized the API carpe diem moment. Withings have transformed the humble bathroom scale into a hub for a fitness ecosystem, while Nest Labs' thermostat is a marvel in intelligent home energy management.

So irrespective of whether business sell sedans, soda, or sneakers, executing a successful API strategy can deliver numerous advantages.

- *Customer satisfaction*—No matter how fast the new features are released, customers still expect more. By exposing applications and data to external partners, developers, and customers, APIs can significantly ease the feature and release pressure on IT teams.
- *Scale and reach*—With API enabled networks, business can work with partners to build new sales channels without relying on existing functions to generate and support them.
- *Business efficiency*—Customers look to the provider for specific customizations. If each of these requests had to be addressed, development could be distracted from delivering more strategic features across the customer base. By exposing functionality through APIs, implementation teams and partners could provide this as a differentiated service, which is another revenue generating opportunity.

APIs provide the essential glue that binds together internal business departments, allowing them to operate as a seamless whole. They enable resources and information to be decoupled from functional silos and made available for greater business benefit. So if sales needs access to marketing data or marketing needs customer analytics, the secret sauce will be an API.

This is extremely beneficial from a DevOps perspective. Since APIs help unify people, process, and technology across the software lifecycle, every group

becomes a partner of the other. If development needs access to application performance information from production monitoring tools, it's an API that'll service the request. If a production-based API needs to be tested against changes to a back-end system, it'll be another API (together with service visualization software) that helps removes the constraints.

API Management: Stakeholders and Requirements

Because different stakeholders perceive APIs in different ways, they will require specific management capabilities in context of their role.

- *Business executives*–APIs and management must be a strategic enabler for launching new innovative products, forging new business partnerships and improving the customer experience. These stakeholders will be especially interested in planning and tracking API revenue and ROI.
- *API owners/product managers*—This group will have responsibility for creating and owning an API strategy and roadmap. As such, they be subject matter experts and oversee how APIs are released and managed. They'll also review API roadmaps with key stakeholders to determine API priorities, including new enhancements.
- *Enterprise architects*—Being responsible for translating digital initiatives into an optimum technology infrastructure, they'll regard APIs as the connective tissue. API management will provide tools to help them model, design, shape, and optimize critical integrations across both legacy and modern infrastructure.
- *Application developers*—Developers build front-end applications, while discovering, acquiring, and consuming APIs as their gateway to enterprise data and capabilities. For this group, API management represents stable, secure, and scalable access to both cloud and on-premise systems, with tools that help them leverage APIs quickly by removing development and testing constraints.
- *Security managers*—Because APIs "open the enterprise," they create new security challenges and opportunities. For this group, API management is about providing advanced threat protection, data encryption, and authentication, all without compromising the goal of increasing connectivity and convenience.

www.dbooks.org

- *API support group*—Regardless of which team supports APIs (development or operations), advanced monitoring will be needed to ensure consumers have optimum access to the API and related services (e.g., back-end systems and cloud infrastructure).

■ **Note** Because of the importance of APIs, many companies organize teams around their development and ongoing support. Rather than engage multiple stakeholders on an as-needed basis, dedicated cross-functional API teams are established. This can be a very effective DevOps practice since it removes functional silos and aligns teams around desired business outcomes—versus managing APIs in technology silos.

APIs Are Products

APIs are the building blocks of great products. With APIs, modern applications can be developed faster and integrating existing systems becomes easier. And because APIs can help facilitate new partnerships and business opportunities, they can in fact be considered actual products in their own right.

As with any software product, APIs will go through a lifecycle where they are designed, created, secured, managed, and optimized. With revenue and reputation increasingly riding on APIs, this will need to be conducted on an enterprise scale. API management provides organizations the ability to achieve this, as long as capabilities embody the principles of DevOps and continuous delivery. While many individual tools address the discreet needs of the individual stakeholder groups, the goal of API management should place overall business improvement above everything.

Managing the API Lifecycle

API management embodies the principles of DevOps by managing the API lifecycle; providing the platform, visibility, and tools that architects, developers, security professionals, and administrators need to drive continuous delivery. Without effective API management, organizations will increasingly struggle to deploy, control, measure, and optimize the growing volume and variety of APIs needed to support digital transformation initiatives.

Since API management inherently addresses the DevOps lifecycle for APIs, it should address all elements (see Figure 4-1) and empower every stakeholder described in this chapter.

It should be noted from Figure 4-1 that the definition, design, deployment, promotion, protection, and optimization of APIs is not limited to just the development team. IT operations and business stakeholders also need to be involved to ensure APIs are aligned to the overall program initiatives.

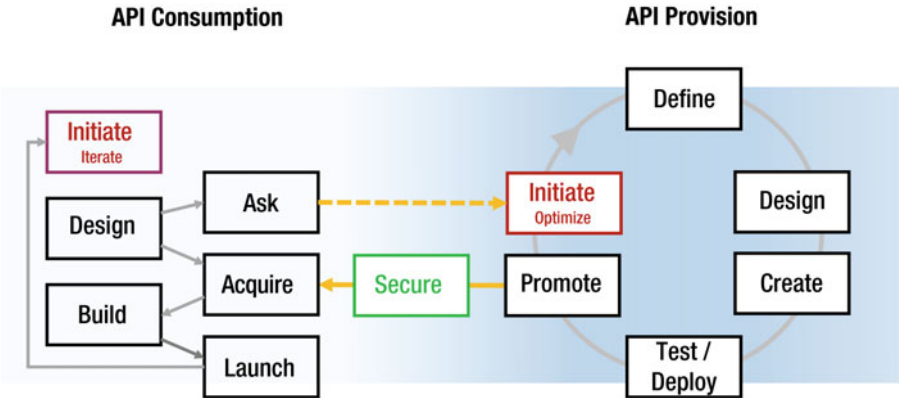


Figure 4-1. API management lifecycle

Ongoing measurement and monitoring is often overlooked in the API lifecycle. For a DevOps-driven API strategy to succeed, organizations need to be able to evaluate API utilization and performance as applications evolve; answering questions, such as who is consuming APIs and how often are they doing so? Which APIs are meeting business goals? Where are the latency issues? Will the API scale to support demand spikes?

Analytics play a critical role in answering these questions. In a DevOps context, they are also especially valuable in serving essential information and feedback to cross-functional team members in order to drive API improvements. This includes:

- *Performance analytics*—Providing developers with real-time insight into operational performance, such as transaction speed, availability, and API latency. This helps ensure apps are meeting their users' performance expectations, particularly during peak usage periods.
- *Business analytics*—Help organizations monetize APIs by tracking usage and consumption by external partners and developers. This means they can better respond to market demand and prioritize future development investments in APIs.
- *App experience analytics*—Report on metrics, such as app usage and revenue generation over the lifecycle of the app; providing complete return on investment visibility both in the short and long-term.

Automation and integration are essential for capturing such insights with minimum effort. Effective API management is founded on intelligent and integrated tools that simplify different stages of the lifecycle for different stakeholders.

www.dbooks.org

For developers, API management solutions need to provide a simple point and click interface that can automatically generate scalable enterprise-grade APIs from data sources as diverse as RDBMS, NoSQL, existing APIs, and JSON.

This is particularly important as legacy monolithic systems are supplemented with dozens of microservices, each fronted by an API. While microservices decrease the size of each deployment, they increase the number of components deployed. API management tools are vital for helping developers navigate the microservices maze: first by auto-generating APIs and then by managing and providing simple yet secure access.

Access and discovery can also be simplified by leveraging an API gateway, which frees up time for developers to focus their efforts on delivering what the customers want—useful and engaging apps that work on multiple devices.

Essential API Management Plays

While API management provides capabilities to many stakeholders, the stronger methods will be those that unify teams toward continuously improving business outcomes from the API strategy as a whole. By reviewing their strategic API objectives (e.g., accelerate mobile development), organizations can assess API management capabilities using a number of tactical initiatives, or "plays".

Create and Integrate APIs

In the past, organizations have relied on APIs from SQL calls and hand-coded business logic—often developed on an ad hoc basis. However, new digital imperatives now dictate adopting scalable and sustainable approaches to creating API-based apps faster and unlock the value from legacy applications and disparate data stores. And with DevOps processes needing to support the development of microservices that can be changed faster, this pressure will only intensify.

To address this, API owners, enterprise architects, and business executives should consider adopting methods and API management solutions that expand the scope of the API lifecycle management beyond gateway enforcement control toward the creation of APIs closer to business information.

The value proposition here is a significant reduction in the time and cost associated with manual-based API creation. Combine this with immediate API performance monitoring and security (discussed next) means quality and compliance becomes established as APIs are created.

As illustrated in Figure 4-2, the API management needed to facilitate rapid API creation and integration should include:

- *Connect SOA, ESB, and legacy applications*—The ability to streamline integrations across disparate systems, middleware, and databases by providing protocol adaptation, mediation, and transformation.
- *Aggregate data from multiple sources*—Assist engineers integrate SQL and NoSQL databases with fine-grained data access controls and the flexibility to deploy and scale to current and future architectures.
- *Connect cloud services*—Enable the creation of performant, yet cost-effective API-centric digital platforms that integrate and orchestrate on-premise systems and cloud solutions.
- *API creation from reactive logic*—Instantly generate enterprise-grade REST APIs from multiple data sources, with logic-based processing to apply business rules to API calls at runtime.

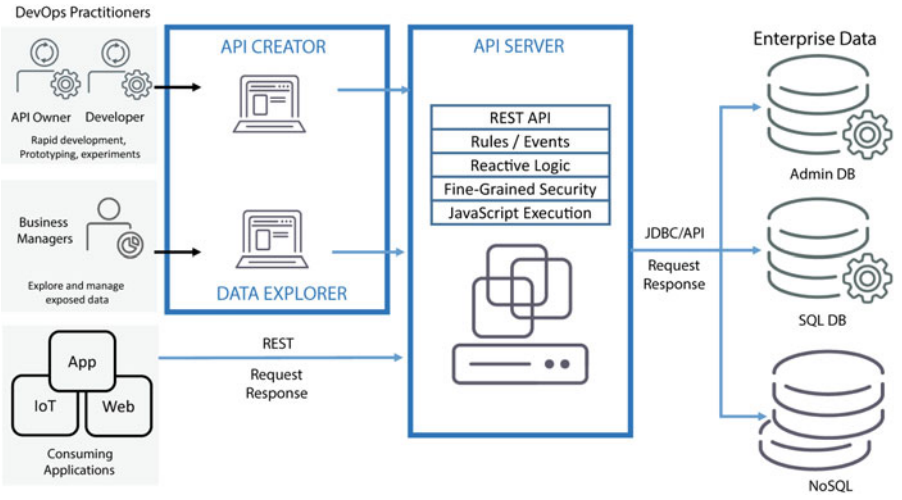


Figure 4-2. API Management for API creation and integration

Note Using reactive logic models to assist in the creation of APIs can be significantly faster than hand-coding. These techniques are valuable to DevOps practitioners since concise application logic is easier and less costly to support.

---

www.dbooks.org

Secure the Open Enterprise

As more information assets and engagement channels become digitized, security is a major concern across the API lifecycle—from the business owner and the developer to the user.

For developers, API management represents stable, secure, and scalable access to back-end systems and information stores, as well as a source of tools and utilities to help them obtain and leverage APIs more efficiently. API management allows them to simply consume existing secure encrypted APIs, for example, without needing to scribe a unique security module for each.

In a DevOps context, information security professionals will work with developers to identify and neutralize critical threats, enable robust and workable policies, offer consistent and repeatable security for mobile apps, and provide the capabilities needed to deliver features such as single sign-on and privileged user access. The goal shouldn't be to wait for security issues later, but rather to help teams establish strong security as they design, create, and test APIs. This may include

- *Protection against threats and vulnerabilities*—API management will provide threat detection and neutralization for key Open Web Application Security (OWASP) vulnerabilities such as SQL injections, cross-site scripting, and denial-of-service attacks.
- *Controlled access with SSO and identity management*—Securing apps and their connections, while maintaining or enhancing the all-important user experience.
- *Providing end-to-end security for apps, mobile and IoT*—API management will protect the digital value chain from front-end app to back-end systems. It should extend controlled access to all touch points —from web apps to IoT, while supporting convenient features such as social login and risk-based authentication.

Unlock the Business Value of Data

Competitive pressure, rising customer expectations, and the increasing pace of change mean that applications—especially for mobile and the IoT—must be delivered faster and more efficiently. Developers look for API management to help them discover, acquire, and consume APIs quickly, while also providing tools that speed up or eliminate the "dirty work" of repeatedly building

core functionality to handle data and security. Therefore, comprehensive API management should support:

- *Simplified and controlled access to data*—API management will provide a controlled way to access data that shields developers from unnecessary complexity. It should aggregate and orchestrate data, while ensuring compliance through authorization, shaping and policy management.

Tip To protect customer privacy and ensure compliance during API testing, consider integrating with management solutions that provide synthetic test data.

- *Support a wider partner/public developer ecosystem*—Solutions will empower internal and external developers by streamlining API consumption lifecycle tasks such as discovery, acquisition, design, and collaboration.
- *Reduce mobile app delivery time*—Developers will need access to reusable services in the form of SDKs and APIs that provide security, messaging, and offline storage.

Accelerate Mobile and IoT Development

Digital transformation initiatives that leverage APIs create new business opportunities and channels. Businesses look to API management as a central launch point for their digital strategies, with a range of capabilities that will support their efforts to build a robust digital ecosystem by expanding partnerships, nurturing developer communities, monetizing data, and leveraging digital connections to improve operations and efficiency.

- *Monetize APIs to generate revenue*—Advanced API management will provide the functionality needed to package, price, and sell data products or services via any combination of free, freemium, purchase, subscription, or consumption-based models. It should also simplify integration with analytics and billing services.
- *Build digital ecosystems to enhance business value*—This involves providing granular control, compliance, security, and reporting mechanisms needed to support the expansion of digital value chains across a wide range of platforms, apps, devices, partners, and third parties.

www.dbooks.org

- *Create efficiencies through analytics and optimization*—By providing instrumentation and analytics that allow them to optimize technical and business performance, API management will encourage developers to build more efficient, performant, and scalable digital ecosystems.

API Management: Essential Integrations

When considering API tooling, it's important to seek out integrated capabilities that help establish information feedback loops that cross-functional teams need to drive API improvements.

API Performance Monitoring

API performance monitoring provides a huge opportunity to increase quality, but has proved challenging. This is primarily because development and operations teams use two classes of tools to address specific performance issues.

API Management Gateways offer tremendous detailed insight into APIs from the client-side perspective (e.g., API throttling, policy, and routing), but lack the depth of analysis offered by Application Performance Management (APM) solutions—especially clear insight into the impact of API performance on critical business services.

Application Performance Management (APM) solutions provide powerful analysis of application performance (e.g., transaction tracing and differential analytics), but lack inclusion of API-level data for related monitoring and root-cause analysis, as well as specifics on the gateway itself.

As a result, many problems can arise:

- An *operations team* receives calls that API gateway services are underperforming, but this is reactive and after the fact because they've received no alerts in APM that indicate latency issues.
- An *application support team* is called to field problems with mobile transactions using an API gateway, but because APM visibility ends at the firewall or proxy and they can't see the actual gateway services themselves, they struggle to identify the root cause.
- An *API development team* has gateway insight, but because they lack visibility into back-end application performance, they cannot easily determine overall service impact.

Bridging the gap between API management and APM solutions via a dedicated integration point (see Figure 4-3) can help teams collaborate effectively.

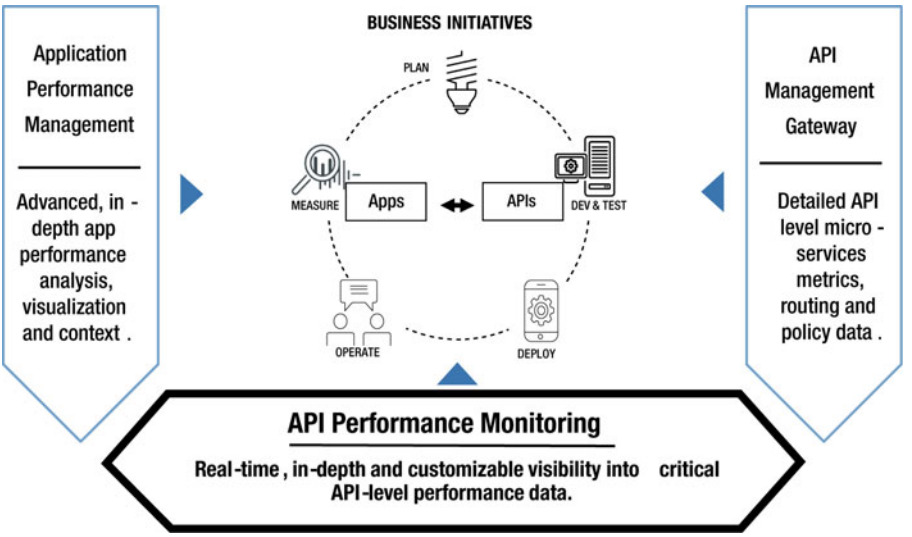


Figure 4-3. API performance monitoring extends the functionality of API management and APM solutions

By monitoring detailed API metrics and data traversing gateways in combination with APM analytics, teams can detect slow growing problems and fast acting acute ones. This allows faster response to problematic API service latency conditions before customers are impacted.

When including detailed analytics at the API layer (e.g., front-end latency, policy violations, and routing failures), root cause analysis becomes more thorough and conclusive.

Through real-time and historical analysis of API performance information, developers and operations have a clear understanding on what API design improvements are needed.

■ **Note** Integration between API gateways and APM solutions is especially valuable in pre-production testing. With complete end-to-end API performance visibility before a system goes live, teams are better positioned to remediate complex performance issues before they impact the business.

API Development and Testing

A 2015 Freeform Dynamics report indicated that it is 2.8 times more likely that digital disrupters are utilizing APIs in development.[2] However, teams remain constrained by circumstances impeding success. In a Voke report, for

[2]Freeform Dynamics, *Exploiting the Software Advantage: Lessons from Digital Disrupters*, October 2015: http://transform.ca.com/rs/117-QWV-692/images/457955-Exploiting-the-Software-Advantage-2015.pdf

www.dbooks.org

example, 56 percent of respondents stated that critical resources were not available when development and testing teams needed them.[3]

API management solutions simplify the process of developers gaining access to APIs, but if back-end systems aren't available for testing connectivity, there can be delivery delays.

An elegant solution to this problem is the integration API management with service virtualization solutions. Described in more detail Chapter 5, service virtualization emulates the behavior of dependent systems; enabling development and testing to be conducted free of constraints.

Service virtualization is a great pairing for API management. It allows developers access to a developer portal where they can quickly discover, learn to use, and integrate APIs into their apps. Then, with virtualized services, these apps can be tested against virtual back-ends, loads, latency, etc., to validate that apps under development apps will perform optimally in a variety of conditions.

When combining service virtualization with API management, teams should seek out integrated capabilities that all allow developers and testers to:

- Secure access to a directory of virtual services, enabling faster service discovery and consumption.
- Switch automatically between real and virtualized environments, improving development velocity, increasing contract and data fidelity, and lowering defect counts.
- Faster creation and easier maintenance of virtual services by leveraging API management data.

Note According to the aforementioned Voke report, simultaneously accessing APIs and virtual services accelerates application development by 97 percent.

Virtualizing the behavior of APIs and services and adding API management makes it easy for developers to access APIs that have been published and secured. As illustrated in Figure 4-4, working prototype APIs can be published and connected to virtualized back-end web services, so developers can begin using them as soon as they are available, rather than having to wait until the process is complete. This also allows teams to operate in parallel, where different development teams can work on different features at the same time.

---

[3]Market Snapshot™ Report: Service Virtualization by Voke Research: https://www.ca.com/au/collateral/industry-analyst-report/voke-market-snapshot-report-service-virtualization-iar.register.html

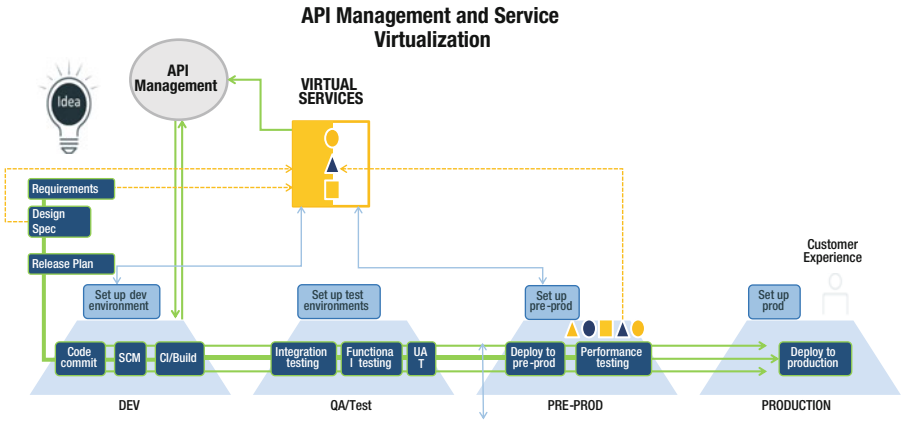


Figure 4-4. Eliminating API development constraints to enable parallel work streams

Taking a Strategic Approach

To fully embrace the APIs opportunity, organizations will need more than just tools, they will need a strategy that will help engender an understanding that APIs are more than just a technology; they are a digital enabler.

Program managers (or API owners) need to be responsible for creating this strategy, communicating it to executive-level decision makers as well as the architects and developers on the front line.

The first step should involve establishing a clear business objective and a vision statement, which needs to be aligned with the company's overall mission and goals. Making this link is often quite easy.

For example, a retailer's vision might be, "Give our customers a compelling shopping experience while offering the best possible quality, service, and value". While the API vision might be, "Leverage APIs to build a high-quality and engaging shopping experience that seamlessly crosses all of customer engagement channels."

The next step should be to build a business model around the API vision, outlining the details of:

- *Costs, resources, and efficiencies*—The systems, relationships, activities, and other resources the program will leverage and how the program will empower the enterprise to deliver on its strategic goals.
- *Value, revenue, and innovation*—The customers, markets, and channels the program will target and how technical innovation will make it possible to generate new revenue.

www.dbooks.org

- *Operational processes*—The tools and approaches needed to effectively control, measure, optimize, and deploy a large number of APIs throughout the development and operations lifecycle.

At the core of this business model, there should be a value proposition that clearly outlines the real, measureable outcomes that the API program will deliver to the business.

API owners or product managers should also engage the business when defining the API vision and delivery model. This will ensure:

- Alignment with core business processes
- Business value realization
- Effective measurement criteria.

Taking a strategic approach to APIs will enable organizations to overcome existing challenges around application integration, data silos, and fragmented omni-channel experiences.

Building an API Future, Faster

Digital disruptors have already recognized the value that APIs and effective management can deliver in terms of customer engagement, satisfaction, and retention. They're already using APIs both internally and externally: more than 65 percent of digital disruptors use APIs internally for building their web applications, mobile apps, and back office systems and externally for integrating third-party services or enabling external developer access to their systems.[4]

Having the right management strategy and tooling can help organizations advance their API business goals by increasing maturity levels (see Figure 4-5).

- *Digital services*—Organizations develop ad-hoc APIs that digitize simple transactions and make them available and secure for third-party use. These APIs bring traditional services to the Web and mobile devices.
- *Partner integrations*—Organizations expand their business model by developing a top-tier API with numerous capabilities, and integrate it with select and trusted partners. While the API itself enables the connection, the requisite SLAs, agreements, documentation, and support have to be managed "offline".

[4]Freeform Dynamics, *Exploiting the Software Advantage: Lessons from Digital Disrupters*, October 2015

- *Partner Infrastructure*—After success with some integrations, organizations look to roll them out to a larger community of partners via an API portal or marketplace. They offer base capabilities for anyone who wants to consume them or create entirely new use cases. Here they not only provide the back-end for existing user experiences, but infrastructure for new ones as well, with standardized processes around SLAs and support.
- *API ecosystem*—As organizations continue to monetize APIs and learn, they make deeper investments into building innovative, new capabilities that create competitive advantage. Here, APIs push the limits into a broad ecosystem, creating entirely new sources of revenue and enabling the adoption of entirely new technologies.

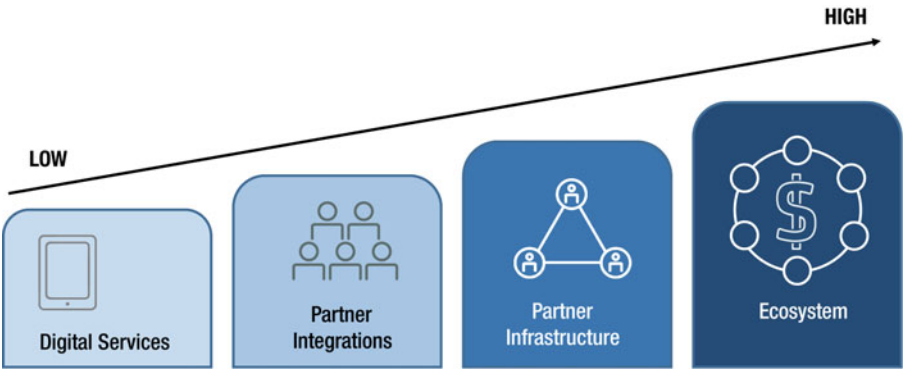


Figure 4-5. API maturity Levels

Summary

Effectively managing APIs is a critical business differentiator. As Mohan Subramanian, associate professor of strategy at Boston College's Carroll School of Management, confirmed: "APIs provide the key to unlock new growth opportunities at an unprecedented scale in our digitally connected economy. The capability to manage APIs will soon become a primary driver of competitive advantage."[5]

Having the right API management tools and processes in place, organizations will be able to build the connections, digital services, and user experiences they need to succeed.

[5]The Chief Digital Offer's Guide to Digital Transformation: https://www.ca.com/au/collateral/ebook/the-chief-digital-officers-guide-to-digital-transformation.register.html

www.dbooks.org

As discussed in this chapter, this requires taking managing the entire API lifecycle so that organizations can consistently deploy, measure, and optimize the growing volume and variety of APIs needed to support digital transformation initiatives.

In the next chapter, we'll continue our exploration of the DevOps-enabled software factory by examining all the testing constraints facing practitioners today. We'll also describe some essential automated testing strategies needed to ensure that optimal quality and compliance is consistently achieved in the most cost-effective way possible.

CHAPTER

5

Test

Continuously Championing Quality

The volume and velocity of software innovation afforded by DevOps is perhaps the biggest driver of IT's shift to this new method of delivery. But when organizations neglect quality in the head-long rush to DevOps glory, the glass can only ever be half empty.

The business-technology landscape is littered with many examples of what happens when software speed has been pursued at the expense of quality. Perhaps the most extreme is Knight Trading, where a software update accessed outdated code (8 years old) that made more than $440 million in bad trades in less than 30 minutes.[1],[2]

DevOps principles and practices are therefore not only intended to improve the tempo of software releases but also increase quality—and as with delivery, this must happen continuously!

However, with DevOps' focus on an automating the software pipeline, it's clear that traditional methods for ensuring quality must now be questioned and reviewed. Separate teams working in silos, working with centralized polluted test data, and performing manual tasks late in the software development cycle is no way to sustain quality.

[1] http://www.bloomberg.com/news/articles/2012-08-02/knight-shows-how-to-lose-440-million-in-30-minutes
[2] https://www.sec.gov/litigation/admin/2013/34-70694.pdf

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_5

www.dbooks.org

Progressive businesses understand implicitly the connection between speed and quality; increasing the cadence of releases, yes, but championing quality through the application of advanced DevOps automation. One illustrative example is AutoTrader.com, the online marketplace for car buyers and sellers.

Case Study: AutoTrader.com

Every month more than 18 million people use AutoTrader.com to search for a used car. The site does more than host advertising for sellers; it also helps people research and compare cars and trade in their old vehicle.[3]

Based in Atlanta, Georgia, AutoTrader's goal is to make exchanging vehicles simpler and more secure than ever before, while maximizing value for private and trade buyers and sellers.

When AutoTrader's services were mainly available through a browser, it saw itself as a media company. The emergence of smart mobile devices and the application economy has changed all that.

"Right now we're a technology company. The technology aspect is very important to AutoTrader as a business," reveals Adam Mills, Senior Manager of Application Development at the company.

Ten years ago AutoTrader released just four web services updates a year; today it expects to release one almost weekly. As Mills explains, "We have to keep up with changes to current operating systems and devices as well as evolve our own offerings. Customers expect us to deliver great new functionality in weeks rather than months."

Customers also expect the same excellent experience whether they are accessing AutoTrader via an app or a browser on a mobile, desktop, or laptop.

In a highly competitive market, this excellent experience is a key differentiator for AutoTrader. "Our app has to be the best," explains Mills. "All it takes is a couple of bad customer experiences then everybody's talking about it on Twitter and we lose market share."

As the functionality of AutoTrader.com and the number and variety of devices it supported grew, software testing became complex, costly, and time-consuming. "We had to set up huge emulation environments, buy the licenses, and ensure all the services were talking to each other," recalls Mills. "But because there were so many interdependencies, we couldn't complete all the different tests in the same timeframe. We then had to find all the physical devices, plug them in, and test our code on every one."

---

[3]Full Story: http://www.ca.com/content/dam/ca/us/files/case-studies/autotrader-avoids-300000-in-testing-costs-with-ca-service-virtualization.pdf

With plans already underway to adopt a DevOps approach to bring together its disparate development teams, AutoTrader realized that virtualizing different services would enable even greater unity.

AutoTrader searched the market for a service virtualization solution that supported the DevOps approach, inviting a select number of vendors to demonstrate the solutions.

After implementing a solution in six weeks, AutoTrader began using the solution to simulate apps behaving normally and performance issues. "Teams can test how resilient their services are and answer those key 'what if' questions, like 'what happens if the database crashes?'" comments Mills.

Mills envisages that soon the last human interaction with a piece of code will be when a developer checks it in. Test, build, and deployment will be automated, reducing processes that previously took weeks to just minutes.

AutoTrader has been able to accelerate testing, while improving quality and freeing up resources. As Mills confirms, "The solution means we can complete testing in hours rather than weeks. Previously we would have needed hundreds of testers to check performance on every device, but now we can test all devices automatically while our team focuses on higher value activities."

The time taken to set up a new testing environment has also been cut from two weeks to two days, with costs dramatically reduced. AutoTrader.com has been able to:

- Cut integration time from three days to three hours
- Save and average of 567 man-hours—or 2.5 people—per release
- Avoid $300,000 in test hardware and software costs
- Decrease software defects by 25 percent

As Mills concludes, "By getting new releases and services out the door quickly, we can provide a better experience to millions of car buyers and sellers and continue to differentiate in a competitive market."

Testing Times

Apart from illustrating the importance of software quality, the AutoTrader story shows that this doesn't have to slow things down. As the forward-thinking Mills suggests, automation will be key for testing to become established within both continuous integration and continuous delivery processes.

Testing is essential to DevOps because it brings the discipline smack-bang into the development processes and avoids the problems (e.g., release delays and quality issues) created by leaving QA as a gate or rubber-stamp function only performed at the very end of the cycle.

www.dbooks.org

This isn't to say that the role of tester will be subsumed with development, but the discipline will change. Rather than providing a transactional service to developers (e.g., executing tests and handballing the bad news), the focus of testing will shift toward a more consultative role that will help developers learn how to write better tests and improve their approaches to scanning for quality. Developers aren't necessarily hard-wired to look for quality issues, and even though the vast majority do care about quality code, they are still going to miss issues and opportunities for improvement.

Tip To establish testing expertise DevOps style, leaders should consider positioning their teams in a way that can add the most value across the software development lifecycle. This may involve embedding specialists within agile product teams or even creating a center of excellence.

As advanced automation becomes more pervasive, QA and testing professionals will need to become better skilled at fully leveraging it. This involves providing a comprehensive and elevated test discipline rather than just executing a series of day-to-day tasks.

Some new skills include:

- *Thinking beyond pass or fail*—Helping the business understand what the customer actually experiences and how that can be best simulated during testing. Essentially supplying the right data and real-world conditions needed to better support and enhance a quality experience.
- *Intimate understanding*—With the complexity surrounding applications today, QA, and testing staff need to become far more proficient at understanding all the intricacies. At a minimum, this means visualizing all dependencies and being able to remove constraints.
- *Assurance to analytics*—QA has traditionally been focused on documenting defects and reporting back to development. This must shift toward collecting and aggregating data from a broad range of automated tests to determine the actual cause of defects and where more rigorous testing is needed.
- *Early and thorough testing*—With agile increasing the volume of user stories, it makes perfect sense to incorporate testing into the acceptance criteria. At this early stage any progression into the sprint should be dependent on reviews involving QA, but also security and IT operations too.

Note With agile and DevOps, quality is baked into the SDLC, not bolted on at the end. This requires establishing ownership at a cross-functional level, not devolving to one team. Automation to support this goal should be available to all stakeholders, not just QA/testing teams.

- *Mentorship over conflict*—Rather than constantly being called in to address fragile developer-written tests, DevOps focused QA will work closely with their coding colleagues to continuously improve testing resilience.
- *Ambiguity to clarity*—Vague requirements stored in multiple formats leads to defective software and a sub-optimal customer experience. Teams should seek out methods to map changing requirements to visual models and eliminate ambiguous requirements and the costly defects they create.
- *Quality over quantity*—Having many redundant, duplicate tests guarantees nothing but cost overruns and delays. QA and testing teams should consider advanced automation methods that generate the smallest number of test cases needed for 100 percent functional coverage—all linked to the right data and expected results.

Agile Testing Trifecta

A key goal of DevOps should be making testing an accelerator, not an obstacle to fast application delivery with the highest levels of quality. To support this, more advanced testing tools are needed, equipping QA and testing teams with three essential capabilities needed to support agile and continuous delivery methods. This "testing trifecta," as illustrated in Figure 5-1, includes:

- *Test automation* to create test cases right from requirements
- Generating *synthetic test data* to be used on demand
- *Test constraint removal* by virtualizing every environment that needs to be accessed

www.dbooks.org

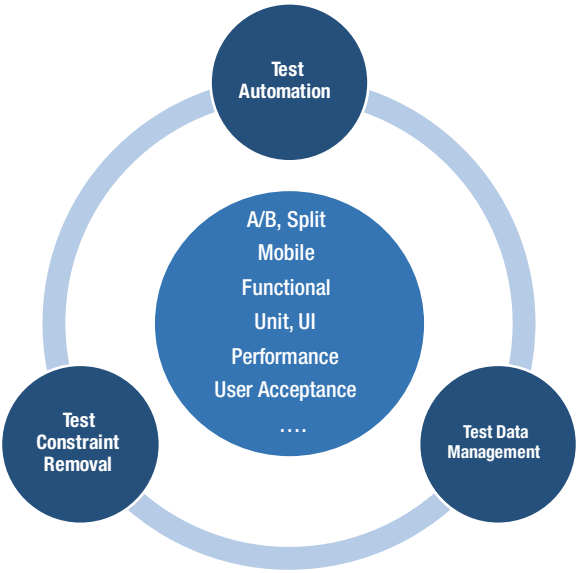


Figure 5-1. Testing trifecta for agile and DevOps

Test Automation

Today, nearly every company is in the software business. Although an organization may sell a tangible product, their use of software to streamline and enhance the customer experience means they must place higher importance on quality application delivery. Frequently, applications that are rushed through the development cycle without adequate testing often encounter costly defects that impact the customer relationship.

Just like building a house, the foundation is key to successful software construction. If the foundation has issues, there is a high likelihood for expensive delays further into the process. Using the right development tools at the onset will help ensure that the software foundation is properly defined, constructed, and tested while keeping quality and end user goals top of mind.

Incomplete Requirements Equals Faulty Software

Many quality problems eventuate during the requirements design phase. This is because software requirements are typically ambiguous, incomplete, and stored in many different formats by numerous people within the organization. Test cases are then manually defined from incomplete requirements and thus the stage is set for foundational cracks to appear even before the application has been built.

Further, the manual definition of test cases is a slow and unsystematic process that leads to perhaps 10-20 percent functional test coverage. Testers end up testing the same features over and over again without knowing for certain the results. As a consequence, defects are detected later much in the development cycle, leading to costly rework.

An Automated and Agile Approach

If testing is going to keep pace with continuous delivery goals, it needs to become much more automated and agile. Adopting a requirements-driven (or customer centric) approach is the first step and may require software solutions to force the change. With the advanced tools, testers can generate the right test cases needed for maximum coverage. Test assets can be derived directly from the design and updated automatically to reflect changing user needs.

Tools in this category allow user stories to be imported and modeled as an active flowchart (see Figure 5-2).

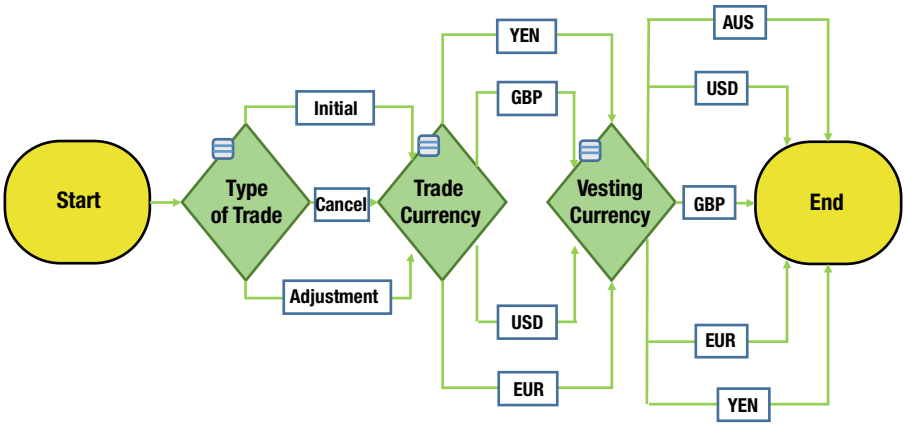


Figure 5-2. Agile requirements design allows user stories to be verified with end users

Active flowcharting helps eliminate requirements ambiguity and reduce defects early in the design phase. This class of tool will also generate the smallest set of automated tests needed for maximum coverage. Importantly, and to support the drive to testing as a discipline becoming much more proactive, these tools also help testing teams know which features should receive the most rigorous testing based on analytics and metrics gathering capabilities.

www.dbooks.org

Achieving Complete Test Coverage

As applications become more complex and distributed, business logic is no longer found only in the user interface (UI) and the database (as with client/server models), but extends across multiple tiers and technologies. This becomes further complicated when applications consume underlying services from cloud providers or third-parties, or use highly interactive presentation layer technologies.

Organizations are also implementing more agile development methods from distributed teams, yet the use of shareable, reusable test assets between these teams is limited or non-existent. Traditional tools designed for more linear style waterfall development are often employed, but lack extensibility, only supporting the needs of one group. For example, code-based unit testing tools for developers that are unusable by QA and functional user interface (UI); failing to translate errors into repeatable defect identification needed by developers to catch bugs earlier.

This requires a much higher degree of test automation and collaboration among stakeholders. As testing efficiency and effectiveness become paramount, a new continuous testing model supported by advanced automation technologies should be the goal. Only through this coordinated approach can organizations build the scale needed to meet future demands.

Meeting these goals can only be ensured when every layer of the application and the complex interactions between components is automatically tested and verified throughout the software lifecycle. This involves providing complete test coverage with the ability to invoke and verify the behavior of each component, singularly or as an end-to-end service. Solutions in this class must therefore provide industry-leading standards support, with native integration to J2EE servers, integration suites, and ESBs. To help strengthen the DevOps toolchain, solutions will also integrate popular open source tools (e.g., Selenium Builder for UI testing), thereby enabling end-to-end testing from user interface all the way to back-end systems.

Case in Point: Mobile Testing

True extensibility means one tool coordinating and running functional tests, test UIs, and APIs on multiple mobile devices under various conditions. Traditional approaches to testing fall down in the mobile world because it's no longer sufficient to just to test the "function" of the application. Code needs to be tested using the same conditions that the app will run under when in the hands of a user, with experience-based metrics and test reports reviewable by both the user acceptance teams and development to further improve quality and expedite defect resolution.

To support the goal of ensuring high-quality during continuous integration (critical for mobile apps where changes updates occur more frequently), such solutions should provide unattended automation coverage. This involves executing tests against real mobile devices connected locally or from the cloud (see Figure 5-3) immediately code is committed.

Virtualized services described in more detail later in this chapter address the common mobile testing challenge of testers needing access to dependent systems for end-to-end analysis. Tests should also allow for different profiles simulating network conditions, location, background applications, and device orientation. This way teams can report and benchmark the user experience of different personas at various points in the app workflow.

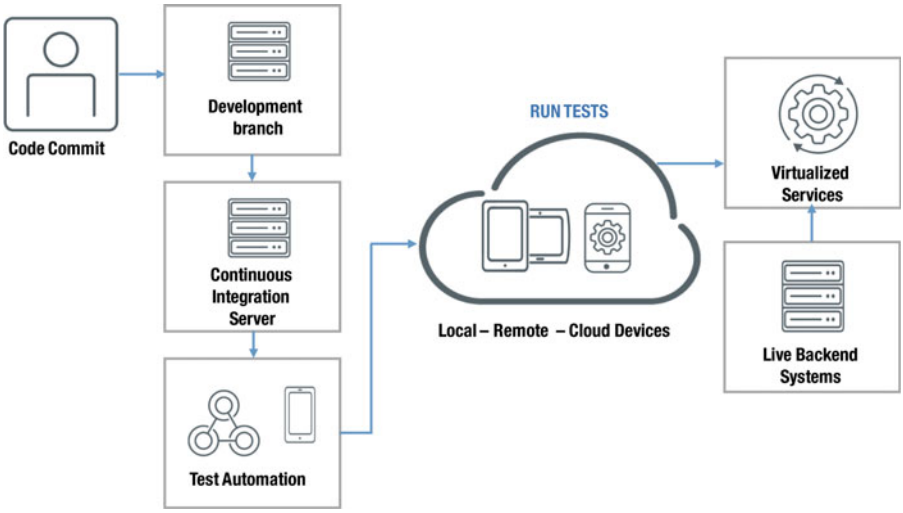


Figure 5-3. Automated mobile tests on smartphones, using multiple OS versions on multiple carrier networks and in different locations worldwide

■ **Tip** When mobile apps are in full production, consider using app experience analytics tools for continued insight into both usage and performance. Results can be valuable for determining where functional and performance improvements are needed.

Test Data Management

The second part of the testing trifecta and an area rife for improvement in the software development lifecycle is in test data management. Every tester needs quality test data and quickly. The challenge is getting the right data to

www.dbooks.org

match their tests when they need it. As companies have improved their development processes, moving from Waterfall to agile, testing has lagged behind. Again, manual processes cannot keep pace with a company's test data need, with companies relying heavily on teams of people constantly creating and maintaining test data.

Another major challenge when managing test data is ensuring compliance with legal and regulatory requirements. Many organizations apply the necessary rigor when protecting personal and sensitive customer information in production, but neglect to consider the implications when working with data in non-production environments. In the event of non-compliance, this can mean significant consequences, not the least brand reputation, but also financial loss due to fines and penalties.

Many industry-specific regulations come with their own unique sets of test data challenges, and some introduce new complexities. Take the General Data Protection Regulation (GDPR) for example. The GDPR is designed to protect the rights of European Union (EU) citizens where the processing of their personal data is concerned.

Although many companies will have already adopted privacy processes and procedures consistent with the directive, the GDPR contains a number of new protections for EU data subjects and threatens significant fines and penalties for non-compliance (up to 4 percent of annual global turnover or 20m euros, whichever is greater) once it comes into force in May 2018.

GDPR introduces many new obligations in areas such as data anonymization, breach notification, and trans-border data transfers, to name just a few. Many have implications for test data management. One example is the "right to erase," where individuals may notify businesses processing their data what they may or may not use that data for, including testing.

Complying with obligations like this carries a huge overhead. If customers state they don't want their data used (even it is masked), then testers will need to acquire subsets of data and apply filtering rules. They will also need to ensure their methods can track every record not approved for testing and be fully auditable.

Many businesses might pursue programmatic solutions to these test data problems, but this only increases the development burden and potentially introduces additional fragility. One alternative, of course, is to use more modern synthetic test data generation to avoid these problems completely.

Facets of a Gold Standard Solution

To address the complex issues involved with the acquisition of quality data and regulatory compliance, modern test data management solutions will provide:

- *Synthetic test data generation*—Synthetic data contains all of the characteristics of production but none of the sensitive content. This ensures teams are provisioned with secure, realistic data that maintains referential integrity as part of a move toward a "Live Data Exclusion" model for testing. In addition to addressing compliance issues, and as illustrated in Table 5-1, synthetic test data generation can address other constraints.

Table 5-1. Removing Constraints with Synthetic Test Data

Constraint	Resolution
Regulations and compliance	**Lower risk as data is generated**
Data functional coverage	Measure and get 100 percent coverage
	Capacity to identify data "holes" by comparison between environments or directly identify data from test cases
Test database size	**Only stores most efficient set of test data**
Provisioning delays	Provisioning in minutes, on-demand, through a web portal
	Capacity to book data for each tester

- *Scalable end-to-end platform*—Tools should clone subsets of data into target environments and be capable of securing millions of data rows in minutes using automated data profiling and advanced masking engines.
- *Test data allocation*—Tools must facilitate automated data discovery for testers to receive exact datasets, linked to their test cases.
- *Test data warehouse*—The ability to store pools of test data as reusable assets in a central repository and test multiple versions and releases in parallel.

The following checklist can also be useful in assessing the efficacy of test data management solutions:

- Provides a standard set of data to test
- Is "*production-like*"

www.dbooks.org

- Covers all possible tests that need to be run, including future and negative scenarios
- Contains *just enough* data to test repeatedly
- Is up-to-date, while also containing and supporting all previous data
- Contains absolutely *no* sensitive data

Combining with Test Automation

The *test automation* methods described in the first part of this section, especially the ability to create test cases right from requirements, are powerful capabilities in their own right. However, combine them with test data management and testers can move beyond just executing tests to proactively driving quality improvements.

By way of example, consider a two-way integration between test data management and agile requirements definition. Here, test matching functionality should be available to locate or create the data needed to execute the optimized test that has been built straight from requirements. The test data itself would be stored in a central test data warehouse where it can be provisioned on demand and used in parallel with development efforts.

Through dynamic building, testers can request the data they need based on specific criteria and receive it in minutes from a self-service web portal. The provisioned data is cloned and version controls are applied to update data to immediately reflect any changes in requirements.

Using the integrated approach, teams benefit in many ways:

- Distributed test teams can work with multiple application versions with matching test data
- Automatically locate or create test data based on specific testing needs
- Test for outliers, unexpected results, and negative scenarios
- Significantly reduce the time and resources required to provision test data
- Generate synthetic data (data from scratch) without the need to mask production data
- Create test data quickly for use in service virtualization to speed testing and increase quality; feeding data directly to service virtualization engines and linking test data with virtual end-points

Implementing a test data management strategy is crucial to realizing the goal of continuous application delivery. By making test data accessible during requirements design, teams can streamline and eliminate the bottlenecks associated with test case creation and locating the right test data.

Test Constraint Removal

There is a fundamental shift in the way enterprises build applications today. In the early days of mainframe and client/server applications, you had a much more limited scope of applications—all of the components from the database to the UI could be under one development and testing team's control.

After the Dot-Com days of the early 2000s, a new style of composite applications arose. The new approach to developing software, including agile, created two new challenges for organizations

- Constraints created by the highly parallel development efforts
- Dependencies on consistent behavior of the components in the system

These complications increased the complexity and cost of developing and maintaining composite applications.

Applications today are the result of many decades of "building systems on top of systems," which creates huge chains of dependencies. These complex architectures mean software development is more difficult, more costly, and more complex than ever before.

Many large organizations now find that many of systems they depend on such as mainframes, databases, and external services are constrained and not accessible by developers and testers when they are most needed.

For instance, a needed mainframe may be off-limits, a system of record could have bad data, or a third-party service may still be under development. Attempts to reproduce these environments—by manually coding stubs and managing test data—are costly and inconsistent.

One customer with constraint issues put it this way, "I can't do anything until I have everything… and I never have everything!"

In a recent Voke Market Snapshot Report on Service Virtualization (January 2015), over 500 companies validated that constraints are a major hurdle to innovation in the software development lifecycle.[4] The report mentions that:

- 80 percent of teams experience delays in development due to constraints everywhere across the SDLC

[4] https://www.ca.com/au/collateral/industry-analyst-report/voke-market-snapshot-report-service-virtualization-iar.register.html

www.dbooks.org

- 56 percent of critical dependencies are unavailable when development and test need them
- 70 percent of teams face prohibitive restrictions (delays, time, and fees) when needing to access third-party systems

Service virtualization solutions can solve these constraint issues by capturing and modeling dependent systems. As virtual versions of the real thing, these services simulate the constrained components in any environment, providing low-cost, 24/7 available models.

When developers and testers use service virtualization, the services behave and perform similar to the real thing, but without the underlying hardware and software complexity of a physical system. Development and testing continue just as they always have, but less constrained, and without contention between teams for environments, labs, test data, and so on.

Although service virtualization solves many different development problems, four common ones are seen repeatedly:

- "Shift left"—Enabling parallel software development, testing, and validation for faster time-to-value with earlier defect resolution (see Figure 5-3)
- Infrastructure availability—Eliminating much of the concurrent demand for environments and hardware that agile development creates
- Performance readiness or solving the challenging problems of properly evaluating the scalability of applications
- Scenario and data management—Often eliminating the need for complex test data management, system setup, and other complexities

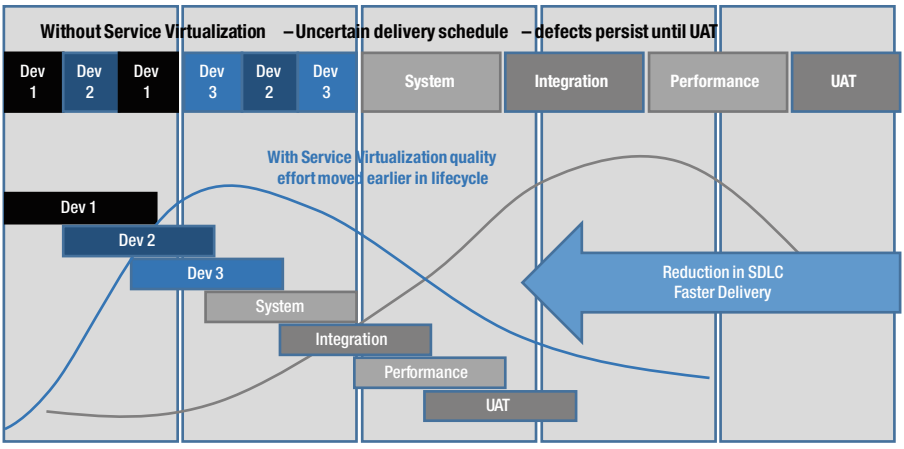


Figure 5-3. "Shift-left" testing with service virtualization

One common problem that service virtualization solves is in the area of integration. Customers buy companies, they provide service to third parties, or they are updating applications for functionality, compliance, or architecture. Each of these challenges presents an opportunity for service virtualization to improve the software development process. Integration teams and customers have the resources they need for software development and testing, without the added expense of acquiring additional hardware and software.

When time-to-market matters, service virtualization offers an excellent opportunity to shorten development lifecycles. Service virtualization reduces the constraints of software development, allowing more teams to effectively work in parallel, without underlying dependencies. Typically, service virtualization users experience a 25-50 percent reduction in release times.

A huge opportunity for service virtualization is in the area of performance engineering. Creating a lab capable of handling and testing to production capacity loads is difficult and resource intensive. Furthermore, ready access to systems such as mainframes and transaction servers may be impossible. All this makes performance testing expensive, unreliable, and inconsistent. Some teams might have a small window for testing, while others will have to wait for an entire application architecture to be assembled before any testing can commence. But by using service virtualization, teams can performance test each individual component, identifying many performance problems earlier in the lifecycle, and reduce, even eliminate, the amount of final performance testing needed in a production-like lab.

Using traditional Waterfall methodologies for developing software, much of the activity of development and testing of the application happens in a series of steps, one after another. But by eliminating constraints common in typical software development practices, service virtualization enables much of the SDLC to operate in parallel and the steps within it become less time consuming.

Using service virtualization, developers can have their own private environments for coding, directly from the laptop. They don't share environments and don't need to wait for other developers to finish their work.

With service virtualization, much of the testing at a component level can "shift left," or be moved earlier in the SDLC. Because each component can be tested individually (instead of waiting for a complete assembly), unit and regression testing happens sooner and is more complete, and defects are identified long before integration or user acceptance testing. Finding defects earlier means developers fix issues at the point in time they incur the lowest cost. This avoids defects leaking into later stages or even into production and become harder to resolve because developers have been moved onto other projects.

www.dbooks.org

As teams increase service virtualization maturity, regression and individual component testing become increasingly automated. Now validation as early as code check-in is possible, making defect detection a consistent and repeatable process. Again this is possible because service virtualization allows component level testing in isolation, without underlying dependencies.

Once automation is implemented, you can easily make it a continuous process. Using this approach, any change breaking interfaces, contracts, or use patterns are easily detected before the code disrupts other services or applications.

The deployment of service virtualization at one large bank solved two critical challenges. First, by eliminating system dependencies, testing began far earlier in the development cycle. Defects in code no longer lurked until UAT, but were found much earlier.

Additionally, the bank's formally serial processes were set in parallel, dramatically reducing release times.

Using virtual services reduces the demand for physical hardware and test labs. This approach is distinctly different, and complementary to, hardware virtualization. With service virtualization, you virtualizes services and business functionality instead of hardware.

When demands for hardware decrease, so do costs. The challenges and costs of provisioning labs and equipment, software and configurations disappear. The physical hardware demand decreases dramatically, freeing budgets for application and business investments instead of capital assets. Demand for data center rack space, power, and storage also decrease.

In performance testing, service virtualization helps customers reduce cost and increase quality and flexibility in several ways. Customers can load test at the component level. Instead of waiting until the application is complete, components are tested for volume and capacity independently, locating bottlenecks and issues early.

Production-only systems such as master databases, mainframes, and third-party systems not normally available for load testing are virtualized, creating an always-ready, highly scalable virtual back-end immune to traditional load testing constraints. This benefits users both in convenience and by reducing the cost associated with replicating expensive back-end systems.

Customers may also face third-party access or software license fees. Service virtualization eliminates the need for highly scalable versions of these systems by virtualizing their behavior. For organizations selling services, virtualized versions of their entire platform are made available to customers in virtual form. Validating against a virtual back-end ensures production readiness without the complexities of full production-style dev/test implementations.

Using service virtualization addresses many thorny issues associated with test data management. For example, organizations struggled to set up just the right scenarios, only to "burn" them with a test cycle. Or, find it difficult to construct test scenarios for edge conditions and business logic. In such cases, it often becomes more expensive to set up the test harness than do the test!

Virtualizing behaviors such as edge conditions, negative test scenarios, and error handling are easily configured in the behavior of the virtual service and are never "burned" since the virtual service is simply playing back behavior responses.

With service virtualization, "test data" and scenarios are easily versioned and changed for each new requirement. In addition, when two test cycles or teams have differing needs for test data, they will not collide in the test lab.

Summary

With the advent of agile development, testing as a discipline is changing radically. Using the approaches described in this chapter, testing can move beyond being a separate siloed function employed at the end of cycles, to becoming a more proactive, continuous, and analytical discipline that firmly establishes quality, whatever the pace of the delivery.

While achieving this goal may require changes in mindset and organizational structure, what's indisputable is the need to adopt a comprehensive testing approach to address end-to-end automation needs, manage test data, and remove all constraints.

In the next chapter, we'll examine the software releases strategies organizations should consider as they move to a more continuous method of delivery.

www.dbooks.org

CHAPTER

6

Deploy

Building an Agile, Resilient, and Scalable Continuous Delivery Pipeline

To keep pace with demands for new features and application updates, digital transformation must be driven by continuous delivery—the ability to rapidly and reliably release software across the pipeline at any time.

By almost every metric, companies that address this imperative create a competitive advantage over those that lag. Yet few companies have actually developed the process maturity and scalable automation needed to deliver applications at the volume, velocity, and quality levels now required to remain competitive.

Before looking the challenges and strategies needed to increase maturity, let's examine how advanced DevOps thinking backed by release automation has helped technology giant Citrix significantly cut deployment times and reduce errors during the release process.[1]

[1]Full story:http://www.ca.com/content/dam/ca/us/files/case-studies/citrix-boosts-business-agility-and-accelerates-devops-adoption-with-ca-release-automation.pdf

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_6

Case Study: Citrix

Citrix provides a range of virtualization, networking, and cloud solutions to around 400,000 customers worldwide. To help create more productive workspaces both for its customers and own users, Citrix is continually looking at ways to improve its business operations and enable innovation. As part of this drive, Citrix created an Office of IT Delivery Optimization in December 2014, which is tasked with evaluating and improving all aspects of IT. As part of its optimization efforts, the team is embracing DevOps principles. As Eugene Lehenbauer, Worldwide IT Delivery Optimization Group Manager at Citrix, explains, "Adopting a DevOps approach to IT delivery will help us achieve better cross-team collaboration, faster delivery, and greater quality."

Although individual development, architecture, and design teams at Citrix had already embarked on their own DevOps journeys, the company wanted to take a more centralized approach to maximize results and share best practices. "Supporting innovation and free-thinking is really important at Citrix, so we didn't want to impose a specific toolset," says Lehenbauer. "We did, however, want to give teams the option of using a proven enterprise platform for automating application deployments to help free up their people from repetitive and mundane tasks."

During the proof of concept exercise, Citrix moved from manual release processes to fully automated release processes, reducing deployment time by 80 percent. But that was not enough for the Citrix team. Development was inspired to re-architect the one large "MyCitrix" application into many smaller pieces, which, along with release automation, enabled the application deployment time to be reduced further to 94 percent.

"Achieving such impressive and immediate quantifiable results has really helped accelerate the adoption of DevOps principles across the business and inspired development innovation," explains Lehenbauer.

A central dashboard permits everyone involved to view the status of all releases, giving teams the information they need to act quickly and providing an audit trail for development and operational teams alike.

The weekly updates to MyCitrix are managed via a release automation solution. "Since deploying Release Automation, we've achieved faster delivery times and fewer issues. As a result, we now have more developers focused on innovating, rather than reacting," added Lehenbauer.

Release automation has been a catalyst for Citrix's adoption of DevOps principles. It has enabled Citrix to take an enterprise-level approach to application delivery by automating application release tasks and orchestrating its continuous delivery toolchain.

www.dbooks.org

Citrix has been able to significantly accelerate its application delivery and reduce the errors and time required during the release process. This has helped the company to be more responsive to customer needs, ensure compliance and auditability, and focus on innovation instead of repetitive tasks.

Obstacles to Continuous Delivery

As the Citrix story demonstrates, continuously delivering software is an extremely collaborative process that spans multiple departments, from development to test, to release management to operations. With so many stakeholders and motivations, the challenges faced by development-focused teams can be very different than those confronting operations professionals.

Development Challenges

With an emphasis on increasing throughput, major obstacles are delays and release bottlenecks. For example:

- Manual, time-consuming, error-prone environment provisioning and release processes
- Numerous errors happening throughout the application release cycle and lots of detective work to find the source of problems
- Inefficiencies caused by the uncoordinated adoption of open source tools, leading to duplication of effort, redundant solutions, and disjointed integration
- Slow response to customer feedback and market needs impacting customer retention and acquisition

Operations Challenges

With an emphasis on ensuring stability, major challenges involve guaranteeing resilience as the volume and velocity of deployments increases. For example:

- Fractured release processes; managing with spreadsheets, scripts, and tools
- Difficulty managing/tracking the volume of releases as more agile development ensues
- Long weekends, low staff-morale and stress due to problems when finally deploying to production

- Double-digit application outages or downtime happening each month and needing an "all hands on deck" approach to resolve
- Loss of customers and revenue due to downtime/outages or errors in application deployments

Finding Common Ground

Regardless of the issues facing each team, it's important that common ground and consensus is reached by tracking all issues preventing successful business outcomes. This is a shared exercise and involves all stakeholders collectively working to determine where the organization as a whole is on the path to automating software releases that drive a continuous flow of value to the business and its customers.

Tip Consider organizing a continuous delivery "current state" workshop that brings all stakeholders together. These may include application owners, developers, enterprise architects, security managers, change managers, release managers, operations, and support.

To facilitate open discussion, some good conversation-starters include:

- In terms of continuous delivery, what are our agreed business goals and metrics?
- How are we managing and executing application deployments? What elements are heavily scripted and rely on manual intervention?
- How are we configuring environments from development through to production? Are different teams using different processes?
- Across the software pipeline, what are the readily visible bottlenecks in the application release process?
- What automation tools are currently leveraged (e.g., continuous integration and configuration management)? Are teams using different tools?

By jointly answering questions like these, teams can develop a structured understanding of where there may be weaknesses across the entire release pipeline and identify opportunities to automate and improve processes.

www.dbooks.org

Tip Try not to restrict analysis to release teams and processes only. Look for opportunities where automation can help drive improvements in development and testing. Careful attention should be given to how the "current state" affects the work of others. For example, if there are release delays, how does this impact development? What processes are they using to circumvent?

For example, are development teams being pulled off important refactoring work because of delays? Is testing being pushed late in the cycle or neglected because teams think they have time to do it later?

Continuous Delivery Maturity

As with all new technologies and best practices, organizations will be at different points on the journey to continuous delivery (see Figure 6-1). Some will have already begun, often by adopting facets of agile or even DevOps, while others will just be starting out. In fact, it's not uncommon for different teams across IT to be at different points in adoption.

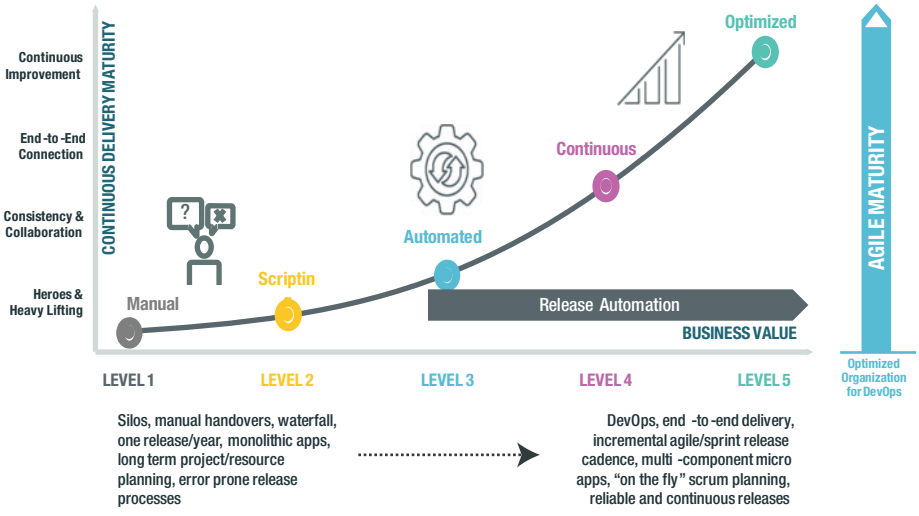


Figure 6-1. Continuous delivery maturity levels

Level 1: (Manual)

At this level, success depends on the competence and heroics of the people doing the delivery. Teams are very much operating in silos. Application releases are error-prone and infrequent and the business is badly positioned to act quickly on new opportunities, defend market position, or retain customers.

Level 2: (Scripting)

Deployment processes are planned per release, and status is managed and tracked. Automation may exist for some deployment capabilities (e.g., scripts). Teams are probably using version control/repositories (e.g., Nexus) and doing automated builds using tools like Jenkins/CloudBees. It's likely that provisioning or configuration management tools like Chef or Puppet are used to help with delivery, but no application-centric end-to-end release orchestration is employed.

Level 3: (Automated)

Here, there are common, reusable, automated application delivery processes established across environments and releases. Release processes, release metadata, and release artifacts are monitored and tracked under full lifecycle control. Delivery automation exists at the environment release level, which may include leveraging existing provisioning and deployment automation capabilities (which may not be scalable).

Level 4: (Continuous)

At this stage, release automation orchestrates application release promotion, enabling predictable, monitored, and measurable continuous delivery from development to production. Organizations can deploy applications consistently across different types of environments and releasing software is a routine and relatively low-risk event.

Level 5: (Optimized)

Now all elements are working in a fully orchestrated fashion to provide a zero-touch deployment—from planning to production. Continuous optimization of end-to-end application delivery processes through feedback loops, with deployment patterns, scenario simulation, and analysis of operational release data are used to continuously improve cost-performance of application delivery. Teams manage multiple applications (multi-services) through the continuous delivery pipeline, which becomes a single point of control. There is also a strong focus on resilience and continuous availability.

No matter where you sit on the continuous delivery maturity curve, one thing is clear—every new level provides tangible benefits. Processes become more automated and standardized. And teams become more productive, focusing on delivering differentiating features rather than managing unplanned work and maintenance tasks. They can handle the growing tempo and complexity of applications, while still ensuring quality and resilience.

www.dbooks.org

Accelerating Maturity: Three Ways

As Figure 6-1 illustrates, the adoption of automated release processes and tools typically drives major inflection points in a continuous delivery journey. There are three important considerations.

The First Way: Connect End-to-End Release Management

Scripting to Automated

Taking an end-to-end release automation approach is essential in order to execute a successful continuous delivery strategy. Key to this is the ability to automate and standardize application releases all the way from development through to production, combined with capabilities to plan, manage, and optimize the release pipeline to improve quality and processes. Rather than act in isolation, release automation must easily integrate with other processes and tools (e.g., continuous integration, provisioning, and configuration management) across the continuous delivery toolchain; seamlessly scaling as the volume, velocity, and complexity of applications grow.

Taking this step to end-to-end release automation also supports DevOps adoption. It becomes easier for teams to have the release transparency, communication, and consistency needed for more purposeful collaboration.

More importantly, cross-functional teams gain control and visibility of the entire release pipeline, looking at the release process systematically versus in silos.

At this and any stage it's important to measure how improvements are helping support the business goals of continuous delivery that were identified before any toolset implementation. For a large Fortune 100 financial services company participating in a release automation ROI study, this involved increasing application release rates across the software lifecycle. As stated by the manager of DevOps enterprise release and deployment, "One of our core application deployments was done twice a week due to lack of automation, intensive manpower, and complicated deployment procedure. After automating this application deployment with release automation, the application is being deployed at least 50 times in a week, all the way from continuous integration to production."[2]

[2]The Total Economic Impact™ of CA Release Automation, December 2015: http://www.ca.com/content/dam/ca/us/files/industry-analyst-report/the-total-economic-impact-of-ca-release-automation.pdf

■ **Tip** Never underestimate the people impact when introducing new automated release methods. Rather than enforcing enterprise adoption, consider small but important projects where benefits can be quickly demonstrated. This can become the catalyst for wider support.

The Second Way: Operationalize Feedback Loops

Automated to Continuous

While automation is essential for continuous delivery, it's only the start of the journey. As automated end-to-end release processes become firmly entrenched, many new release challenges emerge. Taken individually or as a whole (as illustrated in Table 6-1), these pressure points drive a shift to better pipeline management.

Table 6-1. Pressure Points Increase the Need for Advanced Release Automation

Application Content Complexity	Infusing releases with feedback more quickly
	Prioritizing deployment of the right content
	Demonstrating implementation against business requirements
	Preventing "polluted" content from reaching production
The Pipeline Multiplier Effect	Planning, tracking, and prioritizing many complex multi-level applications and independently developed services
	Managing dependencies and avoiding conflicts
	Sharing resources between multiple teams, projects, and timelines
Pipeline Tooling Expansion	Juggling a growing breadth of open source, home-grown, and third-party commercial tools used across the enterprise by different teams

As these pressure points intensify, organizations need to consider processes for executing multi-team, cross-app, composite releases, while ensuring all dependencies are handled. The proliferation of moving parts requires a "big picture" view of the pipeline to maintain throughput, contain issues, and ensure fast feedback.

More importantly, the continuous delivery pipeline is becoming the single control point and application delivery is becoming streamlined, predictable, and risk-free. At this stage, release automation is orchestrating tools and processes beyond deployment, including application lifecycle management (ALM) and service management processes (e.g., change management). This is essential for DevOps, since it strengthens feedback loops and better informs decision-making.

www.dbooks.org

Many more teams within in the enterprise should now be running apps through the single control point. If they are, they are better equipped to establish a framework of continuous delivery best practice that's valuable across the organization.

■ **Note** According to the 2016 State of DevOps report, high-performing IT organizations deploy 200 times more frequently than low performers, with 2,555 times faster lead times.[3]

The Third Way: Optimize the Continuous Delivery Pipeline

Continuous to Optimized

Although few departments are operating at this level, it is the pinnacle toward which all teams should aspire.

With the continuous delivery pipeline being too important to fail, attention should become focused toward making the pipeline (so many teams depend upon) as efficient, stable, and resilient as possible.

This involves shifting toward mastering the art of releasing multi-app, cross-app, multi-team applications and making deployments more predictable and efficient. Improving business execution through accelerated feedback loops will be another benefit, and by establishing a culture of continuous improvement, teams will embrace a "fail fast" culture and then apply the lessons learned within their release processes to prevent future problems.

This notion of continuous improvement is well illustrated by a director of DevOps tools management at a leading Fortune 100 financial services company, who stated, "Agile and continuous delivery can be nothing but a journey. You are never done; you are constantly moving the needle. There is always something you can do."[4]

Essential Toolchain Integrations

While it's important to review functional aspects of release automation solutions, what's more important is examining a solution in terms of how it helps organizations increase continuous delivery maturity.

---

[3]2016 State of DevOps Report: https://puppet.com/resources/white-paper/2016-state-of-devops-report

[4]The Total Economic Impact™ of CA Release Automation , December 2015: http://www.ca.com/content/dam/ca/us/files/industry-analyst-report/the-total-economic-impact-of-ca-release-automation.pdf

No release automation tool will work in isolation. More advanced solutions will serve as an integration hub, orchestrating many activities across the pipeline. At a simple level this could involve application-centric release automation to configure all the resources needed to support a new build (e.g., allocating server and storage capacity and ensuring an appropriate platform is in place to receive the build).

Beyond addressing immediate operational requirements, advanced solutions will work in concert with many other processes to build a continuous delivery ecosystem that helps IT achieve the most advanced levels of maturity. What distinguishes capabilities here isn't just strong integration, but the flexibility needed to support a more adaptive toolchain - one where new technologies can be quickly and easily incorporated to strengthen the continuous delivery model.

Tip To avoid vendor lock-in, ensure release automation tools provide an open and scalable platform, integrating easily with any continuous delivery toolchain for end-to-end visibility and orchestration of releases.

Figure 6-2 and the section that follows illustrate and describe essential release automation toolchain integrations needed to optimize continuous delivery.

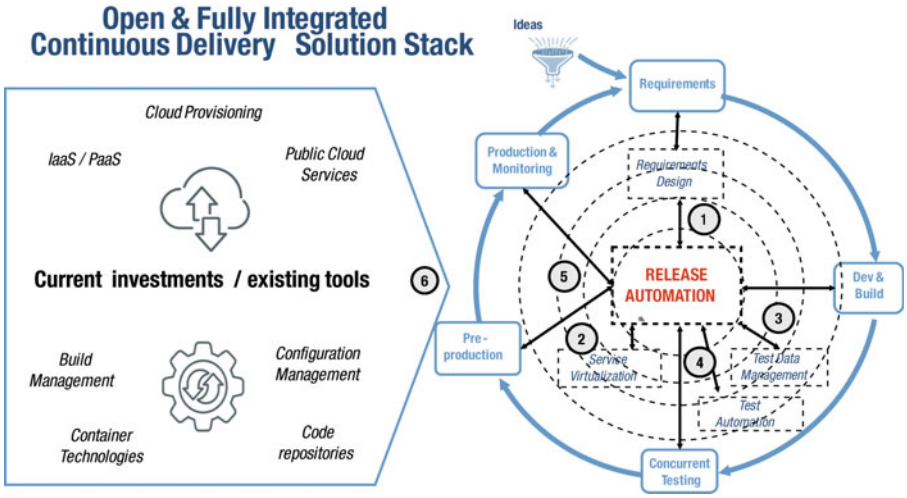


Figure 6-2. Release automation: toolchain integration

www.dbooks.org

1. Requirements Design

This integration allows agile teams to track multi-application release content through the software lifecycle and establish critical feedbacks loops for faster problem resolution and application delivery.

With a real-time dashboard for managing and monitoring multi-application release content (user stories, features, and bug fixes) through the release pipeline, agile teams gain complete visibility of release progress, more easily reconcile dependencies, and can map to business requirements.

Without this integration, agile teams would have to manually track and report on business-level user stories, features, or fixes to specific application releases moving through the pipeline.

2. Service Virtualization

This integration automates the launch of virtual services as part of a deployment to optimize resources and speed testing.

Here, DevOps practitioners can provision virtual services and execute test suites across multiple virtual environments directly within a deployment workflow. By deploying into any testing environment, teams are freed from constraints (e.g., waiting for physical hardware environments to be built and made ready for testing). This improves productivity and speeds time-to-value.

Without this integration the release process could be interrupted. Manual requests would be needed to provision physical systems and virtualized services separate from the automated deployment workflow. This impedes the flow of value and ties up resources on repetitive and error-prone tasks.

3. Test Data Management

This integration automates the generation of accurate test data based on proper test cases within a release workflow.

Without this integration, manual requests are needed to generate the proper test data separate from the automated deployment workflow. Again, this results in release interrupts, delays, and slower delivery.

Note Integrating test data management with release automation should be considered a DevOps automation best practice. Not only does it ensure teams have ready access to accurate test, it also helps establish compliance (e.g., generating synthetic data to protect customer information) into the release process itself and avoid the delays associated with lengthy auditing checks at the end of each cycle.

---

4. Test Automation

This integration automatically starts the test case process and ties the results back into the release to determine and confirm readiness for promotion.

Here the test case process would be automatically initiated with the results linked back to the release. This is essential in order to determine go/no for automated promotion—enabling faster, higher quality deployments.

Without this integration it would be necessary to manually determine if the application has sufficiently passed a testing stage in order to move forward to the next stage and then manually promote the application. Again, this is time consuming.

5. Performance Monitoring

This integration establishes monitoring earlier in the software lifecycle in order to feedback critical information needed to improve quality.

Release automation can coordinate the installation and activation of monitoring in pre-production. The technique of "shift left" monitoring (discussed in Chapter 7) enables teams to see the performance impact of releases and compare it against production baselines. This provides development with earlier warning on code-related performance issues and operations earlier guidance on service-level requirements.

6. Existing Toolchain Investments

A fully integrated continuous delivery toolchain solution will be open and scalable, coordinating the application of any existing products within standard and reusable release processes. Some important integrations include:

- *Continuous integration*—Automatically kick off an application deployment upon the immediate completion of a software build in Jenkins.
- *Configuration management*—Combine release automation with solutions like Chef and Puppet to solve the problem of attempting a deployment when the target environment is not in a good known state. Integration here can be used to enforce specific environment configurations prior to deployment and manage configuration drift.
- *Cloud provisioning*—Enable users to build workflows that provision, configure, and tear down cloud environments within a deployment workflow.

www.dbooks.org

Release Automation: Capability Checklist

With release automation playing such a central role in integrating tools and processes across the toolchain, solutions in this category should at a minimum deliver a deployment engine capable of supporting:

- *Artifact management*—The ability to deploy many different components and configurations of applications on physical, virtual, and public or private clouds.
- *Configurable deployment options*—A powerful, visual workflow engine to easily create standard, reusable deployment processes to promote apps from one environment to the next.
- *Reusable deployment best practices*—Shared components, allowing teams to leverage and reuse deployment logic across different projects and applications.
- *Orchestration of preferred tools*—As discussed, solutions should leverage existing tool and technology investments to automate deployments by using out-of-box action packs or through a software development kit.
- *Deployment remediation and auditing*—Using a visual dashboard teams can track and record configurations, artifacts, and release progress for improvement and auditing.

To develop, plan, manage, and optimize the continuous delivery pipeline, release automation should also scale to helping teams:

- *Design a shared pipeline*—Orchestrate manual and automated tasks within the continuous delivery pipeline.
- *Execute many complex releases*—Run through all the release phases—development to production for multi-app, multi-team releases. Iterate and improve failed content.
- *Plan and manage the timeline*—Schedule and manage apps through multiple phases using a visual calendar. Provide immediate notification of conflicts and maintenance windows.
- *Improve collaboration*—Assign owners to tasks and use an activity feed to share comments.

- *Manage and track content*—Track features as they proceed to production. Provide full insight when prioritizing and ensure the business implications of delays are clear.
- *Optimize releases*—Detect problems in real-time, recognize bottlenecks, and improve processes and team activities.

As maturity increases, release automation should cater to more advanced requirements. This may include:

Dependency Management

When building multi-component/multi-application systems, there will be complex dependencies between applications or different versions of an application. This may include a mix of release, content, application, and application version level dependencies. The knowledge of which application version depends on which is critical and often only known to a small number of experts within the department. Systems should be able to establish the definition of these dependencies with automatic alerts when dependency conditions are not met.

Pipeline Visibility with Notifications

Systems should provide a clear view of the release pipeline, including all phases and all tasks within each phase. Each phase should show list of tasks, the order of their execution, and whether it is to be run sequentially or in parallel. To support continuous delivery, each release should trigger a new build and promote this build through the pipeline, from the test phase and all the way to production. Automation should reiterate phases until all tasks pass predefined criteria. If something goes wrong with a release, delays and idle time should be reduced through automated notifications.

Flexible Approval Processes

For sensitive phases such as production deployment, full governance may be required. To support these cases, systems should prevent mistakes by allowing only the permitted users to approve the execution of these phases.

Recommendations and Action Plan

To attain the continuous delivery best practices described in this chapter, organizations need to ensure they apply due diligence when adopting an automated approach.

www.dbooks.org

As suggested, the best way to start is by assessing processes, culture, and tools currently in use. This way a clearer picture emerges of where businesses are today in comparison to where they need to be to support agreed goals and objectives.

At the start of its journey, City Index used manual processes to deploy application code from development to production. Value chain analysis showed that moving code through development environments to quality assurance, then pre-production before finally going live, made up 50 percent of the delivery effort.[5]

Tip When assessing capabilities, don't limit analysis to one element (e.g., test lab provisioning or configuration management). Take a system-level approach to understanding the flow of value and inhibitor across every stage—involving people, processes, and technology.

Demonstrate Business Benefits and ROI

Stakeholders, influencers, and decision makers need to understand the underlying business benefits of adopting release automation tools to support continuous delivery.

Two key metric categories that are used to indicate IT performance and can be useful in supporting a case include the speed or throughput with which applications are delivered and the quality or stability of the releases.

Tip Seek out real-world customer examples from companies that have achieved significant improvements in both release throughput and quality. ING is one such example. They increased release frequency to over 12,000 a month, achieving faster time-to-market with less than six weeks cycle time, but with a greater than 50 percent reduction in incidents.[6]

Any solution must also demonstrate positive economic impact to the department and business—both short term and long term. To support this, CA Technologies commissioned Forrester Consulting to conduct a Total Economic Impact™ (TEI) study and examine the potential return on investment (ROI) that enterprises may realize by implementing CA Release Automation.[7]

---

[5]Full story: http://www.ca.com/content/dam/ca/us/files/case-studies/city-index-bets-on-ca-release-automation-for-it-operations.PDF
[6]http://www.slideshare.net/CAinc/case-study-ing-builds-highly-available-continuous-delivery-pipeline-with-microservices-and-containers
[7]http://www.ca.com/au/collateral/industry-analyst-report/the-total-economic-impact-of-ca-release-automation.html

To better understand the benefits, costs, and risks associated with an implementation, Forrester interviewed five organizations that had implemented this solution in their enterprise. Taken as a whole, this composite company reported a 389 percent return on investment, $8.44 million net present value, with a 2.8-month payback. The study also illustrated that the composite organization's configuration management and testing team saved time and effort on deployments, with savings of six FTEs quantified at $1.22 million over three years.

Note To help determine business benefits and ROI, seek out tools that calculate the full economic impact of release automation. Comprehensive tools provide total benefit analysis, assessing metrics such as increased staff productivity, reduced release errors, improved time-to-value, and reduced auditing and compliance costs.

Execute Tactically, Grow Strategically

Starting small is okay: It's not in anyone's interests to embark on a lengthy company-wide committee to investigate introducing release automation. It's often easier to showcase business value through a pilot.

With this is mind, consider selecting a suitable project to act as your pilot. Many departments start small with a low-risk application that is important but not business critical. The aim is to start a groundswell of support, gather compelling metrics, and then apply lessons learned across larger teams and projects. The Western Union Shared-Service Enterprise IT Operations team took a grass roots approach to DevOps adoption, starting small and measurable. The team used release automation tools to release software into production and then used this as a lever to open the door for broader conversations with its development partners.[8]

Summary

In this chapter, we discussed the automated methods needed to advance continuous delivery maturity—taking teams from manual, scripted processes to more automated, standardized, efficient and agile methods, all while continuously improving both the quality of releases and the applications they deliver.

---

[8]Presentation - https://www.youtube.com/watch?v=JW2eukJuOqw

www.dbooks.org

We also described how as the lynchpin in a continuous delivery ecosystem, release automation solutions must be capable of orchestrating many processes and tools. The ultimate goal is complete continuous delivery optimization across the enterprise and zero-touch deployments, from planning to production.

In the next chapter, we'll examine the DevOps strategies needed to build more agile operations—strategies that extend beyond basic monitoring of applications and infrastructure toward optimizing the all-important customer experience.

CHAPTER

7

Manage

Agile Operations: Powering the Modern Software Factory

In traditional enterprise IT, developers code and operations manage what comes "over the wall" to production. While DevOps regards this as the ultimate divisive anti-pattern, this practice has still been conducted for decades—but why?

The model generally persists because of the nature of customer engagement. Applications have been generally designed "inside-out," with customer interaction through a single channel. Even if the channel is digitized, the focus is on improving business efficiencies, with customer benefits only considered as an afterthought. For IT operations, supporting this model has been difficult but manageable.

But all this is radically changing. Now businesses understand that customer needs "outside" their organization must be brought "inside" and supported via omnichannel engagement. Omnichannel is all about continuity of experience, regardless of where, when, and how a customer interacts with a business.

From a commercial perspective, omnichannel provides an opportunity to enhance the all-important customer experience via new digital touch points; however, this increases IT operations complexity, with teams now faced with managing increased volumes of rapidly changing software services, delivered over modern and legacy applications and infrastructure.

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_7

www.dbooks.org

With companies digitally "upshifting" from business efficiency to business model transformation, the value proposition of IT operations must change—from being good at managing the technology status quo, to becoming more *agile* and integral to driving successful business outcomes.

This notion of an agile operation is highly synergistic with DevOps since it involves teams working collaboratively to establish a high-quality customer experience across the software lifecycle. Rather than wait until production and retrofit performance, an agile operations team works closely with development using new monitoring approaches to "bake" or craft quality into applications—as they're engineered, tested, and released.

Before examining new challenges and agile operations monitoring strategies, let's examine a case study where DevOps style practices have been used to great effect.

Case Study: ANZ Bank

Today, the only source of competition in the banking world is an obsession with serving and delighting customers. This is something Melbourne, Australia-based ANZ Bank had in mind recently when it launched a new application performance management (APM) program with the hopes of extending it across the entire IT infrastructure, which includes four data centers, mainframes, and more than 10,000 servers.[1]

"The primary goal of all my teams is 100 percent availability of services for our customers. It's simple as that," explains Adam Cartwright, head of IT Security and Operations at ANZ Bank.

"That means *whatever channel it happens to be*—whether it's a corporation doing a payroll transaction or a private user transferring money using Internet banking—it's got to be up. It's got to be seamless. It's got to perform to their expectations. That is the primary mission of operations within technology."

Unfortunately, the complexity of modern distributed applications means that this doesn't happen all the time, says Cartwright. Applications and systems go down, adversely impacting end users. In 2012, it became apparent that infrastructure monitoring alone—which focused on platform and event monitoring at the infrastructure level—was not enough to give ANZ Bank the insight it needed to fix and prevent incidents within the organization.

"What you really need to do is to understand the transaction flow within the application context you're going to identify the root cause, or if you're going to get early signaling of potential problems before a customer actually has [an incident]," he explains.

---

[1] Full story: http://www.ca.com/us/rewrite/articles/management-cloud/customer-obsessed.html

For example, the bank has a distributed payment application that it used to transact billions of dollars for its highest value customers. The application comprises more than 120 distributed servers and 60 separate Java applications. Whenever there was an issue with that application, it was nearly impossible to determine the root cause of the stability issues. The worst part was that many times ANZ Bank IT didn't realize there was a problem until its customers notified them about the issue.

These challenges led to many of the goals of ANZ Bank's Application Performance Management (APM). First and foremost, IT wanted to reduce the number of incidents caused by application releases. This would increase the quality and confidence of application deployment or changes while improving the overall lifecycle of new ANZ applications.

In addition, Cartwright wanted a way to perform deep dives into the application layers to obtain code-level visibility to measure performance and availability. He also wanted to measure and analyze transactions as they moved across the distributed and highly diverse ANZ Bank infrastructure technologies. These efforts have helped the bank minimize customer downtime.

Using APM, IT is now proactively alerted when there's an issue before it affects service. Since APM allows the company not only to look at transactions flowing through an application, but also to identify the business user behavior attached to those transactions, outages can be found more quickly, and applications can be redesigned so outages don't repeat themselves.

"Getting that insight in production is fantastic, but what is perhaps more brilliant is getting that *insight in the development-and-test environment*," he says. "We have been able to show that by putting APM into development and test—making it part of the process in those areas—you can stop defects from getting into production. And that's probably the most surprising thing to people new to APM, but the most critical thing from a production support point of view."

Indeed, says Cartwright, this is one of the more astounding benefits of APM. For example, one project team was able to prevent 10 high-severity incidents from happening, saving more than a dozen hours of investigations. Another project team reduced recovery time from more than four hours to less than 30 minutes with no impact to service. "While it's good to have APM and to use it to diagnose problems once they've occurred in production, you've got a customer that has been affected," Cartwright explains. "It's much better to stop that customer effect from occurring in the first place by not letting that sort of design issue propagate into production under transition."

One of ANZ Bank's first projects in the test–and-development realm was a payments application that caused serious production issues. The bank had five or six major releases each year, and every time a new version went live there was "a raft" of high-severity incidents. "No matter what we did with testing, reviewing, traditional sorts of approaches, or performance and volume testing,

www.dbooks.org

we always managed to end up in a situation where we had a release go live and we'd have problems in production," says Cartwright.

Using APM, ANZ Bank eliminated between 10 and 15 high-severity incidents, which often stretched out three or four weeks following a release. "Now, we're down to one or two or, in some cases, zero [incidents]," says Cartwright. IT employees are thrilled with the change.

"Since APM has been in, we have been able to pinpoint the exact impacted servers and restart these without impacting business and payments processing," explains Joseph Rocco, Support Transition Analyst at ANZ. "Before APM, the LMS [Limits Management System] recovery was four-plus hours. Post APM, it has been around 30 minutes total to restart impacted servers with no outages."

This is an example of the fact that, while ANZ's main focus was on customer satisfaction, a positive side effect of reducing incidents is the boost it gave the IT organization as a whole. "If you can stop having incidents, you track capacity and headcount within the organization to do other things, more proactive things," Rocco says. "APM essentially means your organization is going to continue to grow as your system footprint grows and the complexity grows." IT employees are freed up to be proactive rather than reactive.

Following on from this success, ANZ extended its *shift-left* approach to more than 20 applications. As a result, it has code-level visibility of performance and availability issues, which not only stops defects from getting into production but also increases confidence in application development and deployment at the bank.

Both development and operations teams undertake performance and load testing and correlate their data. As a result, pre-production efficiency has increased with a 60 percent reduction in time spent on solving software problems, which equates to savings of AU $300,000.[2]

More Change, More Complexity

Progressive organizations like the ANZ Bank understand how customers expect rapid software iterations of new functionality together with high levels of performance. This fact was illustrated in an Enterprise Management Associates (EMA) report, which indicated that two-thirds of organizations who have embraced continuous delivery release code weekly or even more frequently.[3] But this is not without its problems, with the EMA report also suggesting that development now spends as much time supporting production as it spends writing new code, while operations spends more time on application support than on any other single task.

---

[2] https://www.brighttalk.com/webcast/7819/134027

[3] "Omnichannel, Microservices, and Modern Apps," January 2016: http://www.ca.com/content/dam/ca/us/files/white-paper/ema-omnichannel-microservices-and-modern-applications.pdf

This support now extends to managing modern microservice style architectures. Designed to be deployed as discrete elements (or services) performing a specific set of tasks and running as its own process, microservices break down specific functions into small components connected via APIs.

While this approach potentially allows services to be updated more regularly without impacting other elements supporting a business process, there are major operational challenges. Not the least:

- *Increased diversity*—With microservices, developers can code in multiple languages and work with databases best suited for their service. For operations, this means maintaining performance and availability over unfamiliar technologies like Node.js and MongoDB.

Tip Always remember that with modern digital systems applications supported by microservices and newer technologies, application performance management solutions must be more resilient and scalable than whatever they are monitoring!

- *Massive complexity*—One monolithic application can become thousands of microservices. Unlike monoliths, microservices make visualizing application topologies and transactional flow using traditional tools extremely challenging. Add the prospect of potentially running multiple service versions in parallel and monitoring complexity increases exponentially.

  Much of the microservice complexity lies in the relationship and API-centric communication between services. With distributed systems like these, teams must consider a whole range of new issues, including network latency, asynchronous messaging, and load balancing, not to mention end-to-end performance issues when microservices connect with back-end applications.

- *Increased noise*—With mobile apps, microservices, and containerized environments, the volume of alarms and alerts can grow significantly. Trying to find filter out noise and find the root cause of problems using traditional rules-based approaches becomes much more difficult.

www.dbooks.org

■ **Tip** Consider periodically assigning developers to review the alerts and log messages their code is producing. This could be a useful way of identifying where refactoring work or improving application supportability is required.

- *Ephemeral nature*—With the pace of change necessitating far shorter application lifespans, triage teams no longer have the luxury of capturing and analyzing historical data using a plethora of tools. Teams require methods to better understand real-time performance across modern architectures (including containers), which in more dynamic environments might be changing in a matter of minutes, even seconds.

New IT Operations Imperatives

IT operations as a discipline will no longer be judged on how effective it is at fixing application problems, but rather on the ability to improve business outcomes—detecting and fixing issues, yes, but working collaboratively with other teams across the software factory to establish quality.

What's admirable in the case of ANZ is how managing to business outcomes has become an established part of the IT operations mantra. True, the team is still responsible for maintaining stability and resilience, but by establishing performance monitoring in areas beyond their traditional control, quality and confidence have increased substantially.

Rather than reactive break-fix approaches to monitoring, a DevOps focused operations function will leverage advanced tools to proactively ensure a quality customer experience before a system reaches production. In this sense, practitioners will become uber sysadmin *craftsmen* and as agile as their development colleagues. And with developers increasingly empowered to make operational decisions, a move toward the *agile operations* approach will become more important.

So what new skills will agile operations teams need to acquire? There are a few important ones to consider, covered in the following sections.

Proactive Engagement

The IT operations landscape changing is dramatically with many advances in technology, but not necessarily changes in values or thinking. With the democratization of operational functions, a good agile operations focused

practitioner will be one that strives to intimately understand the behavior of applications, nurture production systems, and feedback information and knowledge. To this end, agile operations craftsmanship will be less about pulling out router cables and console watching and more about analyzing app behavior and the customer experience to drive improvements.

Designing for Failure

The traditional approach of measuring operational effectiveness in terms of preventing failure doesn't work anymore. With cloud applications, there are many moving parts, in terms of technology and process. There will be mobile apps and APIs supporting new digital channels, but also back-end data and systems. In these environments, failures are inevitable, so the objective should be to design for them—containing problems, but still keeping the business running.

Accepting this reality, agile operations will work closely with development to mature the software engineering and monitoring practices needed to optimize modern cloud-based systems—such as, for example, establishing infrastructure monitoring with every release or exposing performance diagnostics with every application build.

Tip Make it a rule rather than the exception to establish infrastructure and performance monitoring in every environment—right from the moment something is provisioned or placed in maintenance mode.

Moving Beyond Resilience

Like the mythical Phoenix, modern cloud systems and microservices should be designed to bounce back from every situation. And they must, because when a business uses these approaches to engage customers at scale and deal with unknown demand, there'll be much more complexity—at the very least ensuring hundreds (perhaps thousands) of services continue to be available and performant.

Throw in complex architectural issues like asynchronous messaging and API latency and there's a whole new world of pain. It's analogous to cutting a head off the mythical Hydra. Just when you've addressed one problem, two more grow in its place. This is why resilience doesn't go far enough.

When considering new digital systems-of-engagement, applications can't only be resilient, they too have to be Hydra-like. Rather than fixing problems, agile operations methods work to improve applications both technically

www.dbooks.org

and commercially after every significant event (positive or negative). This is especially important for mobile apps and why experience-based analytics is becoming increasingly important.

Making Support a Top Design Issue

Rather than working as downstream production traffic cops, agile operations will engage upstream with developers in an advisory capacity. Of course this requires development to be fully involved; having the desire to learn from IT operations in order to design and build systems that are much easier to support once in production.

But taking the time to learn is challenging if operations provides no useful information or the tools used only address monitoring from one perspective. It'll be difficult too, if in the rush to meet project level goals, development teams select their own point tools and methods at the expense of overall system-level resilience and performance.

Combatting these issues requires ending the "toe-to-toe" battles with development at various checkpoints across the software lifecycle (which never end well). It involves demonstrating how the knowledge and expertise everyone has acquired over many years not only helps improve application supportability, but also makes peoples jobs easier and more rewarding. Some examples include:

- In a mobile app scenario, presenting live usage crash analytics that developers can use to identify where functional improvements may be needed.
- Sharing APM toolsets that enable developers to review issues from their perspective. Serving up information in their terms, about their world, with their code.

Tip Consider organizing joint dev and ops workshops where teams openly discuss and share the "cool" things they've learned over many years. This includes operations sharing information about improving resilience and development explaining the methods used to release software updates in small batches.

- Openly discussing how older style alert and static baselining make less sense in monitoring modern dynamic environments and only increases the support burden.

- Jointly conduct triage scenarios, simulating what's really involved when attempting to fix application issues at 3:00am (clinically review tools to determine whether any of the "noise" actually warrants getting support staff out of bed in the small hours)!

Tip Use workshops and tools to: 1) help developers understand the "on-call" support implications of their designs, and 2) help operations understand what information developers need to make performance improvements.

Active Monitoring

Teams need to embrace active monitoring methods to build an understanding about issues before they affect customers. Part of this involves finding better ways to remove misleading alarms and false-positives.

Traditionally, monitoring solutions have dealt with false-positive alerts using performance baselines. Although this has helped, these approaches typically look at only one part of the issue: severity. A different way to look at the issue is to analyze both *severity and duration*. For example, a minor issue occurring over a long period of time could eventually escalate into a larger issue that teams need to investigate. Alternatively, a medium issue that occurs some of the time should raise an alarm because it could become a larger problem very quickly.

Tip Once techniques to remove irrelevant noise and alerts have been applied, look for ways to combine active alerting with methods to capture more detailed information. For example, automatically initiating a diagnostic transaction trace.

Toward Agile Operations

The 2015 Freeform Dynamics report, produced in association with CA Technologies, indicated that with DevOps, 63 percent and 61 percent of advanced adopters, respectively, were better able to help the business act swiftly on digital opportunities and attack and defend more quickly—with 77 percent and 72 percent also indicating improvements in achieving customer acquisition and retention goals.[4]

---

[4] https://www.ca.com/us/rewrite/articles/devops/assembling-the-devops-jigsaw.register.html

www.dbooks.org

This suggests organizations can move fast without sacrificing high quality.

Achieving this requires IT operations adopting many practices and processes familiar to their agile development colleagues. Of course, this still requires monitoring the performance of production applications, but it also means ensuring information gained is fed back and incorporated into agile development processes (e.g., agile sprints). See Figure 7-1. But, and as we've described, IT operations must ensure that the information being shared with development is of higher value than that provided through traditional system alerting.

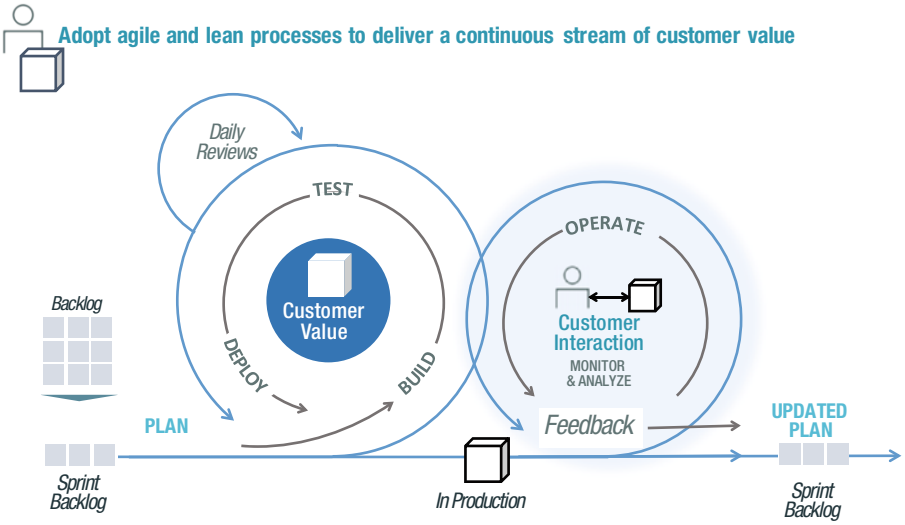


Figure 7-1. Agile development and operations

From a development perspective there are many new tools and techniques that support shifting-left phases so teams can work in parallel to meet the goals of continuous delivery. For example, and as described in Chapter 5, by simulating dependent systems, service virtualization solutions help teams remove constraints, thereby providing teams immediate and realistic test environments.

IT operations has to a lesser extent been slower adopting this approach. This is often due to keeping watch over a myriad of diagnostic tools—meaning less time is spent delivering information that's actually useful to their colleagues.

But failing to embrace an agile operations shift-left approaches can lead to cost increases and missed business opportunities. For example, hurriedly purchasing additional capacity due to unexpected performance problems, or hiring more contractors because of increased staff burnout and turnover after excessive after hours support.

Shift-Left Monitoring

Adopting agile operations methods like shift-left can be a daunting prospect when people feel ill-equipped to cope with any type of change. According to an Enterprise Management Associates (EMA) report, fewer than 50 percent of IT professionals are confident that their application management solutions can adequately meet the monitoring requirements of modern IT environments.[5]

It's important therefore to resist the urge to add another tool (be that commercial or open source) to a growing arsenal of monitoring tools. This approach can further increase team fragmentation, as confirmed in an Infrastructure & Operation Trends Survey, where 80 percent of respondents agreed that disjointed, cross-platform management leads to lost opportunities.[6]

As the survey alludes, this can be suboptimal from a business perspective, especially when incomplete (or incomprehensible) performance signals result in bad decisions. For example, snap purchasing servers due to an unexpected performance condition. This might have addressed the immediate issue, but the organization has just reduced its profit margins and the real problem still lurks-somewhere.

This suggests that monitoring approaches should be conducted from an "avoidance" perspective. That is, avoid problems by detecting them early and often, before they can impact the business. Do this with fewer tools or specialists and cost encumbrances are avoided too. Achieve and demonstrate repeatedly, and IT operations becomes seen as less of cost-center and more as a value-generator, adding to the bottom line.

Continuous High-Quality Feedback

Enabling a continuous cycle of feedback is fundamental to an agile operations shift-left approach and the success of a DevOps program. In its simplest form, feedback from operations should help development toward reducing the volume of code defects, while feedback from development should guide operational service-level requirements before an application goes into production.

That's easy to say but difficult to achieve in practice. Applications are more diverse, distributed, and ephemeral, meaning feedback must be faster, richer, useable, and useful. To support this, modern approaches must clinically remove noise, distill intelligence, and then (and this is the really important part) put it in context of the people having the most to benefit from it. This way feedback is more valuable because it's actionable.

---

[5]"Omnichannel, Microservices, and Modern Apps," January 2016: http://www.ca.com/content/dam/ca/us/files/white-paper/ema-omnichannel-microservices-and-modern-applications.pdf

[6]Settling the Settling the Breadth vs. Depth: http://www.ca.com/content/dam/ca/us/files/ebook/settling-the-breadth-vs-depth-debate-mfinfrastructure.pdf

www.dbooks.org

Take for example a mobile shopping app update with new social network API that's going to be released to take advantage of Black Friday. The new API is potentially a great way to increase revenue, but with demand difficult to predict, how do developers know that their code will handle back-end latency and load? We could take the "suck it and see" approach or stack the data center with more capacity, but with brand reputation on the line, is that a risk worth taking?

Alternatively, we could use agile operations thinking—serving up developers' critical insights into API latency, back-end load issues, end-to-end transaction times, mobile device performance impacts, every time they commit their code. This way, feedback is provided exactly when they have the most to gain from it—as they develop. And since it's placed in context of the business goals they're looking to support, it's immediately actionable. This enables the entire team to act with more purpose and urgency.

Scenarios like this play out all the time in the real world, with agile operations approaches providing the cross-functional glue needed to build a shared understanding of both problems and opportunities for improvement. For example, by using monitoring tools in performance benchmarking, teams can better address capacity and scalability issues; gaining confidence, new applications under development can cope with production load. This was a significant benefit that Danish Retailer Dansk Supermarked realized when implementing an APM solution to monitor development, test, and production environments.[7]

By facilitating the sharing of performance information, agile operations help teams move beyond making small incremental improvements in operational efficiency toward becoming a business differentiator. This is evidenced by a Techvalidate report, which indicated that more than half of organizations using APM stated it has helped them proactively manage user experience to create competitive advantage.[8]

Intelligence and Analytics

As operational functions become fused with development, advanced analytics and statistical methods will play a key role in serving DevOps practitioners the essential information needed to drive improvements.

Rather than attempting to process a mountain of alerts to gain clarity over the current state of performance, cross-functional teams will newer techniques to better predict application performance and usage. Armed with these insights, teams will make smarter and faster decisions, not just in a production context, but across the software lifecycle (see Figure 7-2).

---

[7] http://www.ca.com/content/dam/ca/us/files/case-studies/dansk-supermarked-group-safeguards-online-sales-with-ca-apm.pdf

[8] https://www.techvalidate.com/product-research/ca-application-performance-management

Some valuable methods include:

- *Mobile app analytics crash reports*—Help ensure any reliability issues impacting user experience are quickly resolved. With device and platform specific analytics, developers can optimize code for uniform performance across all mobile endpoints.
- *Behavioral analytics*—By understanding how mobile apps perform under real-world conditions (e.g., slow wireless networks), developers can optimize the user experience (UX) designs.
- *Usage and performance analytics*—These enable developers to improve design with every new iteration, ensuring what they build stays performant as usage and conditions change.
- *Business analytics*—These can increase the chances of an app meeting business targets when it moves into production. Analytics can also be used to measure ROI from different applications to help prioritize future development (e.g., revealing how newly developed features and functions are helping increase customer engagement and preventing churn).

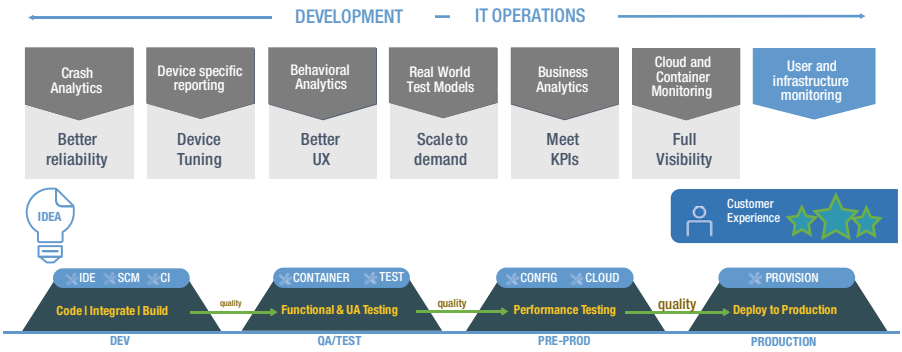


Figure 7-2. Agile operations analytics—quality and resilience across the software factory

With the digital-business landscape shifting continuously, predictive analytics will be supplemented with prescriptive techniques that can actually guide practitioners toward finding the best course of action in any given situation. And with less and less time to conduct protracted root-cause exercises, techniques like these will become essential for agile operations.

www.dbooks.org

In an APM context, it's easy to see immediate usefulness for complex triage—prescribing solutions as problems emerge. But as these technologies mature, they'll also become more business-centric. Consider for example having the ability to *predict* a missed online sales target due to a slow developing performance problem and then quickly *prescribing* necessary changes in both application (design, function, etc.) and infrastructure to prevent the problem from occurring.

Providing business outcome-based capabilities like these will become the ultimate litmus test of successful agile operations and a shift-left strategy.

Agile Operations Tooling

To deliver agile operations and shift-left approaches we've described, organizations need to ensure they have the right application performance management strategies and tooling. Before looking at the tools themselves, and as illustrated in Figure 7-3, let's outline some major considerations:

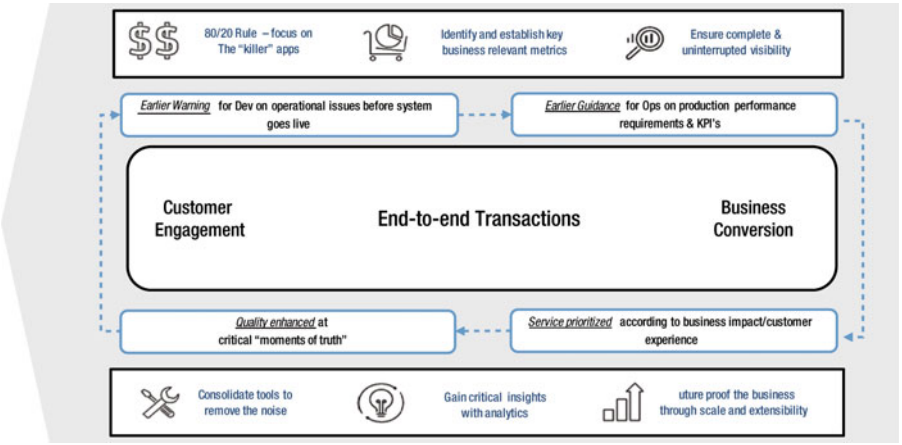


Figure 7-3. Agile operations and shift-left monitoring—essential strategies and tools capabilities

Early Warning for Business and Development

Agile operations tools should provide key insights into the performance of business transactions before the system goes live. With business traversing APIs, mobile apps, web infrastructure, and back-end systems-of-record, comprehensive views into performance across the entire application and infrastructure fabric become essential. This enables development teams to quickly understand infrastructure dependencies, how functional changes impact business performance, and where refactoring is needed.

By way of example, let's consider how integrating APM with continuous integration tools (e.g., Jenkins) can help developers. By providing APM data during a build process, developers can quickly determine the impact their code will make on performance (the early warning) and then leverage it to enact improvements. For example:

- Immediately see API usage increases between builds. *Payback:* Avoid increased costs in API pay-per-use scenarios.
- Identify authentication service overuse in new mobile app development. *Payback:* Improve customer experience.
- See how upstream functional changes cause downstream performance issues. *Payback:* Determine risks associated with the build.

The APM data provided should also be extremely granular (e.g., methods and transactions) and allow developers to roll back to see the impact of any code-level or environmental changes on performance. Additionally, by incorporating APM with continuous integration pass-fail build tests, high quality is established early (see Figure 7-4).

■ **Note** Using bi-directional integration, this process also updates APM, allowing operations to establish a complete performance "system of record" before production.

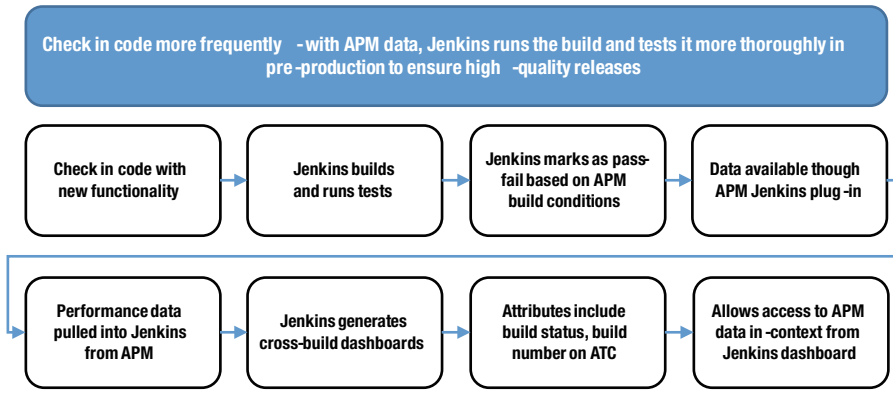


Figure 7-4. APM and Jenkins integrated workflow

In a mobile app context, agile operations tools should incorporate capabilities that help business and development teams better understand and respond to customer behaviors during mobile engagement. Here, end-to-end transaction

www.dbooks.org

performance monitoring should be supplemented with analytics that reveal how new functions are driving increased engagement levels, activity, and customer retention—data that is critical when aligning development efforts to business initiatives.

Early Guidance on Operational Impact

This involves employing tools that continuously track business transactions and customer engagement to guide operations on production system requirements and KPIs. This requires discussion and shared agreement on what's important and why, collaboratively developing a shared set of metrics against which the system as a whole (comprising application and infrastructure components) can be monitored.

Difficulties may arise when architectural complexity and new emergent application behaviors make it difficult to establish accurate baselines against which application performance can be measured. Too often, best-guess performance baselining results in intermittent but acceptable performance spikes (caused by new application functionality) flooding monitoring systems with alerts. In other cases, longer-lived problems indicative of serious code defects go unnoticed because they fall within tolerance levels. To address this, teams should consider mechanisms that rely less on subjective or best-guess baselining, preferring instead statistical methods such as differential analysis where patterns of performance are built and monitored using multiple weighted criteria. Then, when thresholds are breached, deeper analysis (such as transaction tracing) can be initiated to pinpoint the problem and provide immediate and accurate feedback.

Prioritize by Business Impact and Customer Experience

Today's applications rarely function in isolation. In an online flight booking system, the overall customer experience can be delivered by 10 or more discrete elements delivered across multiple channels—from selecting a seat online and checking luggage at a kiosk, to making a payment and scanning a boarding pass with a mobile app.

The elements supporting this "experience" could be transactional, contextual, or a mixture of both. Applications and infrastructure will be equally rich, including APIs, mobile apps, wireless networks, sensors, and kiosks, along with legacy booking and CRM applications. It's essential therefore that agile operations tools deliver deep diagnostics across all these technologies, plus the ability to unify and monitor information at a higher service level. In the case of a flight booking system, for example, being able to prioritize a seemingly unimportant API latency problem because of its significance in supporting a new revenue generating service (e.g., a car rental booking service from a partner).

Feedback at Key Moments of Truth

In all environments, agile operations tools should provide teams fast feedback on software code effectiveness and problem components. This feedback is most critical at key points impacting customer experience (moments of truth). By leveraging live app experience analytics, for example, operations can provide invaluable information to developers and business analysts about the effectiveness of new code in improving user engagement, activity, and retention.

In cloud environments where application resources are invoked according to demand, cost, time, or other business metrics, feedback may be more difficult. Often, when performance problems arise the first question asked is "what changed," which can lead to finger pointing and conflict. While many tools can detect changes, they still require lengthy analysis to determine if the change caused a problem. More modern solutions address this by placing changes in context of application performance, allowing teams to roll back to a point in time to determine what changed and the knock-on effect on performance.

To support these strategies, modern monitoring solutions should deliver the following capabilities:

- *Noise removal*—Distill and simplify complex application topologies into role-based views based on elements such as application component, location, and business unit. Using these and other attributes, developers and support analysts can quickly reorient toward addressing issues and enacting improvements in context of the task at hand.
- *Analytical insights*—As discussed, provide all stakeholders actionable information across the software lifecycle. With app experience analytics, for example, advanced geo-spatial services allows business analysts to build performance and usage patterns, while developers can use video app playback to better determine the impact of newly introduced functionality on customer experience.
- *Uninterrupted visibility*—Deliver end-to-end transaction level visibility from the mobile app to the mainframe. This visibility should be provided in from a customer experience perspective with full and immediate traceability into problem areas across the supporting applications and infrastructure (e.g., network response time and API calls).
- *Scale and extensibility*—The only way to effectively engage customers and transact more business at scale is through mobile and cloud. Monitoring therefore needs to be equally scalable—future-proofing business by seamlessly supporting new technologies as they're introduced.

www.dbooks.org

- *Manage to outcomes*—Strong consideration should be given to advanced tools that better determine what customer experience improvements (business outcomes) can be achieved by aggregating technology diagnostics and optimizing performance (outputs). This is key to ensuring the actions of cross-functional teams are fully aligned toward driving business improvement—not managing technology for technology's sake.

Summary

The days of IT operations working in the closed confines of a network operations center and just keeping the technology "lights" on are numbered.

As software releases grow in volume, variety, and velocity, agile operations will emerge as a key DevOps enabler. It's a collaborative discipline where practitioners work across the software factory to help business achieve the best outcomes from a high-quality customer experience.

Practitioners of agile operations will become true DevOps craftsmen; employing new methods, modern tools, and advanced analytics to deliver superior application performance without increasing the support burden.

In the next chapter, we'll discuss how DevOps can coexist with existing methodologies and best practices. We'll also examine its impact on other IT functions, including enterprise architecture and security.

PART

III

Tuning and Continuous Improvement

www.dbooks.org

CHAPTER

8

Practical DevOps

Leveraging Existing Reference Sources, Roles, and Practices

In enterprise computing, DevOps will never operate in a vacuum. Over many years businesses have invested, adopted, and adapted many other methodologies and practices. For DevOps to be successful, this means many practices and existing roles (beyond development and operations) should be carefully reviewed and, if necessary, adjusted to drive improvements across the DevOps-enabled software factory.

DevOps and Enterprise Architecture

For many years, organizations have understood the dangers of technical debt. That is, the additional overhead arising when badly designed, poorly tested, and defect-ridden software is accepted for short-term gain. Analogous to financial debt, too much technical debt and the associated interest can cripple an organization to a point where they're constantly putting right previous wrongs at the expense of delivering new innovations.

Architectural debt is similar to technical debt and equally problematic. Poor architectural decisions can severely limit an organization's ability to move toward more agile styles of delivery. The result can be lower levels of innovation, slower time to market, or more cost and effort consumed rebuilding applications. These poor decisions are the consequence of conflicting architectural

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_8

perspectives. On the one hand, scant regard for architecture and unconstrained development leads to software that is hard to integrate, support, and enhance. On the other, overly defined architecture is difficult and complex to implement, leading to delays in software delivery.

Some DevOps practitioners feel that they don't need to be guided Enterprise Architecture (EA). Conversely, many Enterprise Architects believe that rigorous dictates, methodologies, and practices must be adopted fully by DevOps programs. In truth, both agile development and DevOps cannot succeed without EA. However, the scope and application of the discipline must change to accelerate DevOps benefits without burdening the business with additional risk.

Without Good Architecture IT Builds Software Slums

The UK high-rise tower blocks built circa 1950-1980 were heralded as architectural wonders. Large towers housing the same population as the smaller pre-war houses they replaced. With large rooms, excellent views, and surrounding open spaces, they were initially lauded as the low-cost future for urban housing. However, there were many problems.

Due to cost cutting, substandard materials, demanding deadlines, and rushed construction practices, these modern-age housing miracles quickly became the new slums of the day. And by building tower-blocks across the country, town planners unwittingly replicated bad designs everywhere. The result: undesirable housing, elevated crime levels, and urban decay. Even the surrounding open areas and playgrounds were neglected because no one had ownership over maintenance or supervised them.

In many ways, this is analogous to many architectural design calamities made by IT in the past.

Built over many decades, rigid monolithic application designs have become difficult to scale and require costly maintenance. They're also tough to police and secure, with vandalism and theft being a constant problem. Add to this integration issues (IT's own unsupervised open spaces) and we're left with systems that are inadequate to meet the needs of modern digital business. That's not to say, however, that modern design approaches are the panacea. Web and microservices introduce new architectural issues, not the least, dependency management and monitoring complexity.

Enterprise Architecture Must Adapt to the Times

DevOps, with its focus on collaboration across the entire service lifecycle, is now seen as the answer to many of these issues. This is fine in principle, but without flexible EA, the end result could be IT developing application "slums," only more of them at an increased pace.

www.dbooks.org

With some justification, many DevOps practitioners argue that heavy EA practices and frameworks haven't evolved to match the pace of agile. Iterative style development essentially means an end product very different from the initial idea, which necessitates a transition from set-in-stone architectures to a more fluid mix based on continuously evolving decisions. And if architects can't adapt, autonomous agile teams with easier access to open source and cloud-based resources are increasingly empowered to make architectural calls. But this can be problematic.

Great developers don't necessarily make great architects; they're not hard-wired that way. Charging down a development path without architectural guiderails, or tactically making team-based decisions, could address short-term development requirements but compromise broader program-level objectives. For example:

- Scalability and performance issues needed to address the business strategy of tripling the customer base and increasing satisfaction scores are neglected because development only concentrates on delivering more functionality.
- Developers acquire public cloud resources to accelerate application testing, but don't consider masking personally identifiable customer details when moving test data. As a result, organizations (especially those in heavily regulated industries) risk compliance breaches and heavy financial penalties.
- Because of team-based technical bias, a development group chooses one NoSQL database over the technology already used by another team. The existing technology would have met their needs (albeit with compromises), but their technology bias has significantly increased the support burden on IT operations.

It's important not to lay the blame squarely on development. Architects should recognize that any misalignment between existing processes (particularly the rigid ones) and the vision of an increasingly software-driven business will only further isolate the practice to the eventual detriment of the business.

New Fluid Guidelines and Principles

By incorporating the experience of enterprises architects within DevOps teams, organizations can maintain the pace of software delivery without introducing chaos. What's key, however, is to limit architectural over-engineering and provide development groups a minimum set of EA so as to avoid technical and architectural debt.

Even in minimal-mode, EA can still help developers quickly identify critical software design issues. The aim being to guide developers into understanding what architectural facets make up a successful application, and more importantly, how new ways of thinking can support more sustainable software innovation.

Interestingly, these practices are analogous to those that could be used to avoid building "housing slums":

Avoid substandard or unsupported materials—Development tools will most likely comprise both commercial and open source software and utilize both on-premise and cloud-based infrastructure. Teams must ensure application supportability becomes a key consideration, especially as more modern applications comprising multiple components move into production.

Note Just as the cost of maintaining UK housing tower-blocks reached unsustainable levels, so too will support burden on the organization when open source software isn't maintained and enhanced by the broader software community or lacks commercial backing. Enterprise architects should also work closely with development to understand where constraints lead to testing compromises—the "construction shortcuts" syndrome, resulting in compliance and quality issues.

New technologies shouldn't rehouse old problems—Enterprise architects should avoid rigid and inflexible edicts on technology usage. For example, mandating that all applications must be containerized, even legacy applications. This could be extremely problematic since the runtime-independent nature of containerized applications will extend the life of "problems" and provides no incentive to clear existing technical debt.

Note As UK high-rise housing tower-blocks degraded and vandalism increased, many local authorities tried to contain the situation by housing "problem groups" in the same units. Enterprise architects should avoid the same trap as they adopt technologies such as public cloud services and containers.

Monitor and manage sub-contractors—Whatever cloud model is adopted, abstracting away the infrastructure or application stack frees up development to focus on coding. But this shouldn't mean relinquishing visibility and control over application performance and the end user experience. To this end, enterprise architects must act as cloud-brokerage advisors. For example, working to ensure that cloud service providers incorporate open APIs with their offerings so that monitoring can be seamlessly incorporated into existing tools.

www.dbooks.org

Building new digital services with DevOps is only part of the role of EA. The innovations of today may become legacies of the future, while many new apps must also integrate with existing systems that have strict compliance and risk controls. It's essential therefore that architecture covers both bases, working in minimal mode to ensure fast construction of quality services, but also applying standards and governance when needs dictate and systems change.

Actions to Establish EA in DevOps Programs

Some important steps needed to ensure EA becomes a sustainable contributor to a DevOps initiative include:

- *Communication*—To impress the importance of flexible architecture practices and standards. For example, API instrumentation shouldn't be mandated with rigid rules, but by carefully outlining how the practice improves software quality and supportability. This involves building closer ties with developers to ensure the right tooling decisions are being made. Enterprise architects should always emphasize that their participation isn't to slow down one particular team, but to ensure that team decisions (especially tooling) support broader program goals and objectives.
- *Collaboration*—In dynamic agile environments, it's natural for small teams to only focus on their own project and not consider the wider business context. Enterprise architects must apply flexible governance to ensure all stakeholders are involved in decision making without the system becoming overly bureaucratic. To this end, the governance approach must outline the need for big-picture strategy at an overall application portfolio level (complete with funding, commitment and risk management), together with support at a project level so to drive better outcomes in a business context. This support should start at the requirements phase and extend across the software development lifecycle.

DevOps and Information Security

It's a common misnomer that DevOps only involves closer collaboration between development and IT operations teams. In actuality, DevOps programs must also involve other disciplines that have traditionally been engaged late in the software development lifecycle. This is especially important with regard to information security.

At first glance it appears that the goals of DevOps and security are at odds. Whereas DevOps calls for increasing the delivery of high-quality software, security and compliance seeks careful and deliberate oversight to ensure the business isn't opening itself up to vulnerabilities. And with a mountain of rules and regulations to support, it's not surprising that security could easily become being regarded as another bottleneck in release and deployment processes.

All teams must accept that security is a key facet of "high-quality" software, which again can be established without slowing down development. There are four essential practices to consider:

Make everyone accountable for security—DevOps impresses the need shared responsibility and accountability. Therefore, security professionals should seek to build relationships with dev and ops teams and engage them as active stakeholders and participants in driving security improvements. As with enterprise architecture, this doesn't mean continually enforcing rigid and inflexible policies, but actually working collaboratively to assign security responsibilities to the team's best positioned to act on them. For example, during every application security incident, developers responsible for the actual code implicated should really be the first group called to help address the problem. These teams will be much more familiar with the software workings, plus the lessons they learn will help harden application security.

Demonstrate how DevOps improves security and vice versa—As organizations increasingly embrace DevOps, there'll be many new automated tools and practices introduced. As with everything new, these elements could introduce new threats and risk. Rather than see this as a problem, highly collaborative teams should work proactively to identify where additional guidance and controls are needed and can be applied without causing friction.

Take the development of a new mobile application for example. Here security experts can provide critical guidance on new threat surfaces, API governance requirements, and vulnerability testing. It's also important to consider that many new tools introduced (especially in areas like configuration management and release automation) also provide an opportunity for teams to build and improve security within the continuous delivery pipeline. To this end, it becomes less about making DevOps more secure and more about using DevOps (and especially automation) to improve security. This could include:

- Invoking techniques such as static code analysis during every application build, or providing development teams with comprehensive and fully automated security testing services that can be used repeatedly.
- Automatically creating the minimum set test cases with maximum security test coverage, right from the earliest stages of software development: the requirements phase.

www.dbooks.org

- Reducing security test cycle preparation time by requesting and reserving accurate and compliant data from a test data repository.
- Generating realistic synthetic test data and incorporating directly into virtual or emulated services so as to improve testing quality while avoiding compliance exposures.

Shift security "left"—As with the traditional development to operations code handballing, the tendency has been to engage security very late in the development process. Too often, security teams are seen as the bottleneck police, holding up deployment with snap code audits and lengthy compliance checks. DevOps practices, however, enable security to be established during parallel development and testing. As code is developed, automated tests can be automatically invoked to continuously check and demonstrate compliance controls. This could include separation-of-duties and privileged user access controls, or masking personally identifiable customer information during cloud-based testing to demonstrate compliance.

Note By shifting security controls left into development and continuous delivery, it becomes easier to demonstrate compliance against a broad range of regulations (e.g., Federal Security Information Management Act—FISMA and General Data Protection Regulation—GDPR). High costs and delays resulting from auditors coming late into the process and finding the system isn't compliant may also be avoided.

As applications become increasingly complex and threats more pervasive, highly skilled security specialists will become highly prized and critical element to the success of any DevOps initiative. Organizations shouldn't make the mistake of assuming that developers themselves with a smattering of web application security experience can take on a full time security role (or will even want to), or that security staff (more used to maintaining security in legacy applications that infrequently change) can suddenly think like an agile developer. Over time, these skills will need to be developed by leveraging DevOps style collaboration. This could include agile teams inviting security to participate in user story development, stand up meetings, and retrospectives. For security professionals, it also means gaining credibility with a more detailed understanding of modern coding practices, providing faster feedback, and becoming an active voice in all security related discussions.

Rethinking Security Practices for DevOps

For many organizations, embedding security professionals into DevOps teams isn't practical. There just aren't enough of them and security operations may have problems scaling to handle a sudden influx of software changes. Addressing this requires a radical rethink on how to best apply security practices. This can involve:

- *Using security as a guiderail*—Security must take a lead in developing solutions and policies that all development teams can adopt. However, if teams gain management approval to bypass a policy because it slows them down, or the business unit accepts the trade-off is well worth the risk, security shouldn't stand in their way. Rather, security should measure their security capabilities and continue to inform teams about the risks of their actions.
- *Building closer collaboration with suppliers*—With applications moving to the cloud, organizations must work closely with software and cloud service providers to instruct what additional security controls and methods are needed in order to develop, test, and store information, without carrying additional risk. Businesses operating in different industry verticals will have specific compliance and data protection mandates, meaning providers must be willing to act in a more enterprise-friendly manner. This involves service providers including customers in development roadmaps and a willingness to support enterprise specific security requirements.
- *Making security a whole-of-business issue*—The role of security in DevOps should be to make security everybody's issue, not just the responsibility of the highly specialized security team. One effective way to do this is to develop a hierarchical security scoring system. If for example a deficient security practice is detected during the provisioning of a test environment, then the team responsible should be rated accordingly. That score should also bubble up to a group or divisional level. In this way, everyone (including senior management across business and IT) becomes more accountable.
- *Proactively involving security*—While it might not be practical to embed security specialization into every team, the security group can establish small teams charged with continuously testing security across the software

www.dbooks.org

development pipeline. At regular times this team will focus on particular services (even groups of people) and use their expertise to hunt out vulnerabilities and log via established ticketing mechanisms with the appropriate classification and priority. During these exercises, no one should be immune from investigation or the activity limited to static systems. Even if teams are in the middle of a large important release, any severe problems detected must be addressed immediately.

Essential Characteristics of Security-Minded DevOps

As illustrated in Table 8-1, a security-minded DevOps program transcends beyond reacting and fixing security problems to protecting the business as applications are designed, developed, and tested.

Table 8-1. DevOps and Security: Organizational Mindset

Customers first	Mindset that enables security to be continually tailored according to customer needs and business outcomes.
Team alignment	Flexible organizational structures that enable security expertise to be embedded within development, testing, and operational functions.
Proactive engagement	Constantly assessing the security readiness across the software development lifecycle by introducing unplanned security events and threats.
Continuous investigation	Thorough analysis of external attempts to attack a business so teams can remediate security issues quickly and effectively.

DevOps and IT Service Management

While DevOps as a movement is relatively new and many organizations are in the early stages of adoption, most have heavily invested in more established methodologies and practices—especially ITIL®[1]. Since its inception in 1994, ITIL has been positioned as the most complete approach to IT management, with the exception of project management and enterprise architecture. It's not surprising then that upwards of two million people had some form of ITIL training (from foundational to expert) and that most enterprises have adopted many of the processes as detailed across the five ITIL volumes (service strategy, service design, service transition, service operations, and continuous improvement). For some, this starts and ends with service operations

[1]ITIL® is a (registered) trademark of AXELOS Limited. All rights reserved.

processes (especially the service desk function together with incident, problem, and change management problems), while others have embraced a fuller lifecycle-based approach to adoption.

DevOps and ITIL

Despite common misconceptions, ITIL is not specifically opposed to agile and DevOps thinking. For example, the service strategy volume promotes the notion of continuous improvement via feedback across the service lifecycle, while service design mentions agile and iterative design. However, despite the synergies, the general philosophy behind ITIL is one of rigorous sequential planning and control via process; opposite of the fast iterative design approach of agile development. ITIL also suggests that silos will continue to exist (albeit aligned around 26 processes), whereas the idea of smaller cross-functional style product teams, fast feedback, managing work in process, and small batch sizes is not well supported.

This appears to suggest that ITIL is becoming increasingly irrelevant with the practices under increased enterprise scrutiny. While this is perhaps true, it's important to appreciate where coexistence is practical and the immense value established ITIL processes can still deliver a DevOps program—especially core ITIL processes (e.g., incident and problem management).

Take problem management for example. This can provide considerable insight into the behavior and performance of a particular application, which can inform developers of needed non-functional improvements. This can also help teams avoid getting information (often conflicting) from a variety of sources. Additionally, a standard incident/problem ticketing method across development and operations may help improve teamwork and collaboration.

Overcoming Resistance

The biggest problem with respect to DevOps adoption is the resistance to change by established ITSM practitioners within the organization. Very often many roles are aligned around ITIL processes (change managers, problem managers, etc.), so it's natural people may resist DevOps if they feel their careers are threatened. To this end, DevOps and ITSM leaders must actively work to better understand how existing roles and practices can be enabler of successful DevOps capabilities and where refinements are needed. This could include:

Developing a better understanding of change—The adoption of ITIL has led many organizations to create the often-maligned Change Advisory Boards (CABs). Meeting infrequently, these groups have the unenvious task of approving production changes—essentially becoming the control point for delivery.

www.dbooks.org

DevOps' approach to change is radically different. With DevOps, all change is encouraged unless it introduces greater risk and the increased probability of adverse customer and business impact. To this end, change isn't policed at the end of the cycle, but managed at the start of development. This is supported by many automated methods, including:

- Automatically establishing tests early, even when requirements are being established
- Maintaining consistent environmental configurations across development, test, and production
- Automated construction and invocation of testing during development, application builds, promotion, etc.
- Visibility and automation of complete release workflows

In dynamic agile environment, the rigor associated with CAB style change management may be too inflexible. It's important that ITIL roles readjust processes to accommodate the more continuous introduction of change. This could involve CAB leadership determining which DevOps-related changes do not require the normal process rigor and may bypass the traditional controls.

Integrating existing ITIL processes with DevOps—DevOps practitioners should collaborate with ITIL process owners to determine where integration with existing systems can drive improvements. Even if changes have to go through an approval process (e.g., for systems subject to compliance controls), this can be streamlined by integrating release automation with the change request process (normally maintained in a help desk system). Rather than rely on manually entering a change request separately, this activity would be incorporated in the actual release automation workflow itself, together with all necessary approval requests and escalations. In another example, security managers could work closely with DevOps practitioners, establishing technologies such as privileged access management to better accommodate the needs of developers without compromising strict compliance controls.

Participating in DevOps discussions—With many years of experience, ITIL practitioners can provide DevOps practitioners essential knowledge needed to drive improvements. An experienced IT operations manager could, for example, demonstrate how application performance management tools can be used in pre-production to identify any performance related problems undetected during earlier stages of testing. On the flip-side, ITIL process owners may benefit from newer tools introduced by agile and DevOps practitioners. Examples include configuration management and release automation solutions.

Beyond discussions and as illustrated in Table 8-2, there are many other practices that teams can adopt to reinforce the value of more collaborative behaviors.

Table 8-2. Example Behavioral Practices for DevOps and ITIL Teams

Practice	Example
Celebrate successes	Service managers present the business outcomes from a successful release together with lessons learned.
Improvement indicators	Service management team attends daily stand-up meetings and presents how data sharing has improved the supportability of a new application.
Recognition and rewards	At a joint weekly meeting, DevOps and service management teams jointly recognize individuals who have demonstrated strong collaboration and a shared commitment to driving improvements.
Reinforcing vision	At a weekly meeting, IT operations leadership reinforces the vision and goals of the DevOps program.
Satisfaction snapshots	Process owners are regularly engaged to assess the current level of engagement and support for a DevOps program.

DevOps and Lean Startup

While DevOps will need to coexist with and leverage traditional IT practices such as ITIL, there may be occasions when DevOps practices are very appropriate to help drive the adoption and success of newer business-driven methodologies. One such example is Lean Startup, a method for developing businesses and accelerating product development through a combination of hypothesis-driven experimentation, iterative product releases, and validated learning.

Originally proposed in 2008 by Eric Ries[2], the Lean Startup promise is to help businesses iteratively develop products to meet customer needs, while reducing market risks and avoiding heavy project funding and expensive product launches and failures.

DevOps practices have many synergies with Lean Startup, including:

- *Customer focus*—Lean Startup places great emphasis on validated learning, which is basically the process for understanding quickly what a customer actually needs or wants so that useful (and profitable) products can be developed. Agile and DevOps can help achieve this through the delivery of smaller units of work or iterations, after which results are validated.

[2]"The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation To Create Radically Successful Businesses," Eric Ries, September 2011

www.dbooks.org

- *Fast feedback*—Lean Startup success depends heavily on continuous customer feedback during product development. This is to ensure that business don't invest unnecessary time and money developing features that customers don't want. DevOps supports this through two practices. First, by teams aligning activities to business outcomes and developing actionable metrics (discussed in Chapter 3), and secondly by employing continuous delivery processes to quickly release and test prototypes, experiments, etc.
- *Team collaboration*—With Lean Startup promoting the continuous running of experiments and validation, it is critical that all cross-functional teams operate in unison. In such fast-paced environments, any form of waste like long cycle times and delays hampers validated learning and inhibits the flow of value. It's why DevOps practitioners supporting a Lean Startup model place great emphasis on reducing all elements of waste across the software lifecycle (see Chapter 3).

Summary

Being myopically focused of DevOps to such an extent that existing bodies of knowledge and best practices are shunned or ignored is a recipe for disaster. That said, organizations must also be ready to adjust existing methods, roles, and practices. As this chapter illustrates, this includes enterprise architecture and information security.

In the next chapter, we'll describe important tangible business benefits and strategies needed to accelerate DevOps ROI.

CHAPTER

9

DevOps and Real World ROI

During the 2016 Formula 1 season, it's not unusual for pit-stops to hover somewhere around the three-second mark. In rare instances, such as for the Williams F1 team at the 2016 European Grand Prix, the feat has amazingly been accomplished in less than two ticks of the clock.

Wrap your mind around that idea for a moment. It will likely take you as long to digest the concept, and it certainly took longer to type out this sentence, than the amount of time required for a 15-20 member pit-crew to pounce on a stopping race car, lift it off the ground, swap four wheels and tires, and then drop it down to be sent on its way. Two seconds![1]

Quality Is Everyone's Responsibility

Of course, this larger performance represents a complex chain of lesser events, mapped out and practiced relentlessly by highly-paid, full-time professionals—pit-crews whose sole job is to save their teams and drivers precious seconds that could result in a change in position, and in some cases even a victory.

[1] "Williams Data Shows Baku Pit Stop a New Record," Motorsport.com, June 22, 2016: http://www.motorsport.com/f1/news/williams-data-shows-baku-pit-stop-a-new-record-790776/

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_9

www.dbooks.org

As one crew member pirouettes away with a used tire, another immediately spins into place and mounts a new wheel on the car's hub, with a pneumatic air gun operator barely removing the apparatus as one wheel is removed and another arrives in its place. This process is carried out in concert simultaneously on all four wheels of the car. Suffice it to say, to pull this act off successfully, execution must be flawless, every time.

Consider that over the 70-plus years of F1 history, at least ten individual races have been decided by less than two-tenths of a second, and many more by a matter of mere seconds; the impetus to strive toward such lofty speed and precision quickly comes into focus.[2]

Making a Case for DevOps ROI

How do these F1 crews perform such a precise ballet? By using teamwork, communication, and near constant review meant to spur process improvement. Countless hours are spent reviewing pit-stop video looking for even the smallest opportunities for potential refinement, followed by an endless cadence of hands-on practice wherein these movements are repeated hundreds of times.

Meanwhile, revised wheel, hub, and pit-crew equipment designs are introduced in the name of further speeding this process. The drive for perfection is continuously evolved as part of a cycle that never ends.

The return-on-investment (ROI) for teams that master this craft is evidenced not only in race results and unofficial pit lane bragging rights, both of which carry weight, but more importantly in hard dollars and cents. While races are often decided by seconds, the difference in prize money between the first place and tenth place finishers in the 2016 F1 Constructors Championship is estimated at roughly $150 million.[3]

Pit-stops represent only a small element of the overall competition, but when it comes down to it, every single aspect of F1 racing is aimed at shaving seconds, and even milliseconds, in the name of victory.

Inside the modern software factory, workers are engaged in a similar high-stakes race aimed at addressing an elusive opportunity, which, like a race or season win in F1, can disappear in the blink of an eye. This is crucible —succeeding in the applications economy also demands constant review, improvement, and acceleration of velocity.

---

[2]"Who Won it? 10 of F1's Closest Finishes," FOX Sports, September 1, 2014: http://www.foxsports.com/motor/story/who-won-it-10-of-f1-s-closest-finishes-090114

[3]"Formula 1 Prize Money 2016 (Constructors Championship)," Total Sportek, March 17, 2016: http://www.totalsportek.com/f1/formula-1-prize-money/

As outlined in the proceeding chapters, it's widely accepted that in order to grow and survive in the current business environment, organizations must embrace digital transformation, thereby advancing their ability to evolve software and services to meet changing customer expectations. Beyond the growing emphasis on this theme by numerous business and technology leaders, and certainly speculation on the part of technology vendors, hard data has begun to emerge which serves to cement the requirement for DevOps adoption.

For instance, in a 2015 study published by CA Technologies and Zogby Analytics, 68 percent of consumers noted that they will completely abandon a particular brand based on mere seconds of delay in application loading times.[4] Specifically, respondents indicated an expectation that applications load in six seconds or less, or else they simply drop the app, and often the related provider. Further, more than half these respondents indicated that they actually expect applications loading times of less than three seconds before they run out of patience.

Based on this reality, the need for broader adoption and advancement of DevOps culture, process, and tooling to speed and improve applications delivery is thrown into stark relief. An organization's ability to wrap its arms around DevOps has become a key differentiator in competing for end users' attention and dollars; in contrast, failing to meet changing customer expectations related to applications quality and performance often means losing out to more agile competitors.

In short, improvement of software delivery has become a prerequisite for survival. Based on that conclusion, beyond the vague notion of advancing general viability, measurement of DevOps ROI impact is clearly an increasingly critical practice.

The Challenge of Measuring DevOps Success

Measuring the impact of a process as far-reaching and transformational as DevOps within the context of ROI may seem impossible to some observers, but organizations of all kinds—from practitioners to industry analysts—have begun to attempt such calculations.

For starters, creating the necessary framework for such metrics requires closer consideration of the very reasoning behind DevOps adoption. Loosely, this rationale is accepted as the need to increase the pace of software development and release, all while improving the quality of resulting applications. Just as importantly, DevOps also brings with it the promise of speeding the rate of response to emerging performance issues, all while better addressing changing customer expectations.

[4] "Software: The New Battleground for Brand Loyalty," CA Technologies and Zogby Analytics, March 4, 2015: http://rewrite.ca.com/us/articles/application-economy/research-study-software-the-new-battleground.html

www.dbooks.org

At first glance, this would seem a set of processes and concepts easily tied to straightforward notions of measurement, such as the rate of software deployments. However, that in itself may be too simple a viewpoint. The reality is that such measurements highlight some of the immediate benefits of DevOps advancement, while many other intrinsic results are certainly harder to quantify.

Consider that some experts, such as DevOps industry pundit and former research analyst Michael Cote, have posited that attempting to gauge the ROI of DevOps, in this dualistic sense, is "simultaneously absurd and important".[5]

The reason being, the expert maintains, that there are too many variables involved in DevOps transformation to make any overarching estimate of impact a solely believable value.

Writing for FierceDevOps in 2015, Cote specifically noted that "DevOps is an immeasurable process with respect to ROI, because its value is nearly impossible to measure independently and precisely."

That said, Cote relents that while the larger import of DevOps may be difficult to define in terms of pure ROI, it is possible and practical to measure the success of specific projects—or products developed using DevOps processes—to establish some baseline for related assessment. This is where calculations such as software deployment rates and so-called mean-time-to-repair (MTTR)—which measures the ability of organizations to identify and repair applications issues that arise in production—infer some guidelines for assessing DevOps returns, if not larger success.

As noted by industry watcher Christopher Null of *TechBeacon* in his seminal article, "How to Measure DevOps ROI,"[6], such measurement of impact is most practical within the most basic context of DevOps adoption, namely, saving any amount of time previously spent building, deploying, and maintaining applications.

"One of the biggest benefits, if not the biggest benefit, of DevOps is the promise of speed; DevOps enables more and faster code deployments, which means a decreased time to market and more opportunities to capture revenue from customers," Null observes.

Using such a framework, one can also begin to tie delivery of mission-critical, or even revenue-impacting functionality to applications lifecycle improvements garnered via DevOps processes. For these reasons, the most practical forms of DevOps ROI measurement available today do in fact revolve around factors such as software release cycles, MTTR, and applications code change

[5]"DevOps ROI," Fierce DevOps, July 21, 2015: www.fiercedevops.com/tags/devops-roi
[6]"DevOps ROI," Fierce DevOps, July 21, 2015: www.fiercedevops.com/tags/devops-roi-how-measure-guide

failure rates. These are metrics that may not conclusively outline the larger impact of the overall movement, but that can infer some notion of ongoing improvement.

While additional forms of DevOps ROI assessment are being created, including those focused on measurement of financial impact, enhanced productivity, and other indicators of organizational efficiency—for now, the most accurate metrics are those largely focused on applications lifecycle excellence and these figures do provide a compelling case for DevOps adoption and advancement.

As Null points out, Patrick Debois, himself a sysadmin, notably stated regarding examination of overriding DevOps value: "We should be thinking about [ROI] in terms of … accelerated benefits realization, and shortening that [cycle]. It's really not the ROI of DevOps, it's really more that the ROI of your original project can be realized sooner."

A Real-World Model for DevOps ROI

As widely accepted that applying specific ROI calculations within the domain of IT, in general, has always been a tricky science. This certainly applies to the domain of DevOps.

This is largely related to the fact that technology has been largely perceived as a cost-center versus a driver of profits, making it difficult to convince management to increase investment with the specific goal of improving the bottom line.

Even with scads of industry analyst calculations aimed at addressing this very issue (IT ROI), there's been little concrete evidence establishing that IT investments can in fact provide significant returns. Most experts would more readily accept that IT may become a larger channel for, or catalyst to growth of the business, but any notions or promise of massive ROI is often overshadowed by requirements for sizeable upfront and ongoing investment.

Further, technology vendors notoriously muddy the waters by proffering self-serving calculations that often inflate the proposed impact of their products in the name of boosting sales and competitive prospects. That said, it's also recognized that, especially in recent years, technology paradigms such as mobility, the cloud, and virtualization have delivered economies of scale and new business opportunities so massive as to make them difficult to fully quantify.

So, how does one go about making a believable ROI case around DevOps, a process that, like many other movements, promises to completely redefine the manner in which organizations create and monetize their value?

www.dbooks.org

The 2016 State of DevOps Report, the industry's leading research project—conducted and prepared by the DevOps Research and Assessment (DORA) initiative—offers an avalanche of data providing concrete testament to the proposed, and oft-considered fuzzy, benefits of the overall movement.[7]

For starters, the report posits that so-called "high-performing organizations," those already deeply engaged in the use and maturation of DevOps practices, are "decisively outperforming their lower-performing peers in terms of throughput." This finding not only builds on similar conclusions in previous iterations of the same report, but also maintains that the espoused gap between DevOps adopters and laggards continues to widen.

Specifically, DORA contends that such high performers deploy applications code 200 times more frequently than organizations that have not yet closely embraced DevOps. So, practically speaking, such organizations have seemingly become far more capable of engaging the benefits of agile computing; namely, responding adeptly to changing customer demands.

In addition, the State of DevOps Report also finds that high-performing organizations are currently achieving 2,555 times faster lead times, serving as even greater proof of their abilities to remove obstacles to software innovation and improvement.

Perhaps even more importantly, in particular as it relates to the matter of agility and the ability to cater to evolving performance conditions, the report estimates that leading DevOps practitioners lay claim to 24 times faster recovery times and three times lower change failure rates. These figures formulate a highly compelling narrative when it's recognized that reduction of MTTR represents one of the most critical differentiators in today's applications economy.

The State of DevOps Survey, completed by roughly 4,600 technology practitioners, also found that high-performing teams spend far less time on so-called "unplanned work and rework"—most often required to respond to emerging issues related to existing applications code. This allowed them to spend nearly 50 percent more of their time on new work. The latter brand of effort is typically aimed at adding or improving feature functionality in the name of increasing business opportunities.

Lastly, in terms of pure applications delivery ROI metrics, the State of DevOps Report suggests that leading practitioners are suffering far fewer applications and services outages—those timeframes when systems are offline and truly prevent the flow of end user traffic and business.

---

[7] "2016 State of DevOps Report," DORA and sponsors, June 22, 2016: https://puppet.com/resources/white-paper/2016-state-of-devops-report

Based on a framework established by research analysts at IDC, which concludes that hourly application downtime costs can range from $1.25 to $2.5 billion for a Fortune 1000 firm, and that the average cost of a critical application failure is $500,000 to $1 million per hour,[8] the State of DevOps Report allows for the conclusion that high-performing organizations, compared even to those with some level of DevOps adoption, save an average of over $91 million per year.

Much of this advantage is found, the researchers note, in the form of lower applications change failure rates, or those updates to software code that result in subsequent performance issues or outages. While high-performing DevOps practitioners experience an average change failure rate of 7.5 percent, medium performance organizations see that figure rise dramatically, to a whopping 38 percent.

If the undeniable adage that "time is money" applies in today's applications economy as much, if not more than ever, then these ROI calculations carry substantial meaning.

Beyond such metrics aimed at measurement of improvement appreciated across the applications lifecycle, the 2016 State of DevOps Report also sheds light on another area of the movement's impact that would appear to transcend immediate calculation, but which can certainly be quantified in general terms. Organizations deeply engaged in DevOps transformation tend to have happier, more dedicated employees.

Given the wide recognition that it is far more expensive to hire and train new employees than retain existing staff, and that highly skilled IT workers in particular are difficult to find and hold on to, this less hands-on ROI benefit of DevOps may prove as even more critical support for adoption than some of the proceeding metrics.

Specifically, DORA's researchers found that employees of high-performing, DevOps-centric organizations are 2.2 times more likely to recommend their organization as a great place to work, and 1.8 times more likely to recommend their team as a great working environment. The figures were calculated using the employee Net Promoter Score (eNPS), which is designed to measure overall employee loyalty.

As a follow on, it's worth noting that other research, notably a report published by experts at Bain & Co., has found that companies with highly engaged workers typically produce revenues 2.5 times greater than those with low engagement levels. In addition, Bain submits that publicly traded stocks of companies with a high-trust work environment outperformed market indexes by a factor of three.[9]

---

[8]"DevOps and the Cost of Downtime: Fortune 1000 Best Practice Metrics Quantified," IDC, December 1, 2014: http://www.idc.com/getdoc.jsp?containerId=253155

[9]http://www.bain.com/publications/articles/the-chemistry-of-enthusiasm.aspx#2

www.dbooks.org

Simply stated, these conclusions, both process-specific and more general in nature, highlight the specific effect that DevOps is having in terms of creating the most critical form of ROI measurement—the ability to innovate faster, build organizational momentum, reduce costs, and attempt to differentiate offerings within the competitive environment.

It only follows naturally that organizations spending more of their time creating new value by translating ideas into product offerings are likely outperforming those struggling to play catch-up by addressing existing shortcomings.

Measuring the Impact of DevOps Automation

As previously noted, success in every element of the F1 racing arena is measured in the smallest of increments, in this era most often in the form of tenths, or more typically hundreds of seconds. For F1 pit-crews, refinement of each tiny physical movement or adaptation of time-saving technology can result in a measurable benefit.

The same can be said of DevOps adoption, which can be extrapolated to span nearly every aspect of software design, delivery, and support, along with a multitude of tools aimed at automating some involved process.

To isolate any one or two specific processes or tools only provides a small window into the broader impact of DevOps automation as a whole; however, leveraging these as a starting point can provide perspective which can then be expanded across numerous domains.

One such toolset that has been so examined is the utilization of CA Agile Requirements Designer solution, which seeks to automate test case design and execution based on changing requirements—to automate one of the most time-consuming and resource-intensive problems in applications delivery—namely, delays created by manual testing.

By allowing engineering teams to automate the process of creating and applying software testing requirements, the solution promises to greatly reduce related cost and complexity, specifically by directly cutting down the number of required test cycles.

According to a calculation designed to forecast the potential ROI of CA Agile Requirements Designer related to cost reduction and avoidance, along with related revenue enhancement, immediate benefits of employing this process—representing only one minor aspect of DevOps adoption—immediately come into focus.

Based on a real-world implementation of the solution, in addition to gains related only to software testing, one organization found that it was able to drive far closer collaboration between business and IT teams, and better

estimate the time and cost of development projects. In addition to the following ROI calculations, these gains map closely to core benefits of broader DevOps adoption.[10]

In terms of specific ROI, the involved organization found that its worst case appreciation of benefits over a three-year period would result in a 168 percent return on its existing investment, or a period of 11 months before 100 percent realization of payback on initial costs; the study projected a best case scenario of 319 percent ROI, or 8 months for complete payoff.

Among the underlying benefits driving this return were the organization's ability to reduce software defect remediation (a similar domain to reduction of MTTR), lowering of test creation and maintenance costs, and, most importantly within the overall scope of strategic investment, improved time to value for application releases.

At the production end of the software lifecycle, another important DevOps tooling aspect that warrants ROI analysis is Application Performance Management (APM). APM seeks to simplify the triage of performance problems so teams can address issues fast, before there's an impact on end users and customers.

By quickly analyzing performance across a complex array of microservices, APIs and containers across mobile, cloud, and on-premise applications, these monitoring tools provide IT operations an essential mechanism for maintaining expected levels of service together with a high-quality application experience.

In a DevOps context and as described in Chapter 7, these tools are especially valuable because they help establish feedback loops between operations and development teams. By gaining insights into application performance before committing software updates to production, developers can quickly pinpoint and remediate any issues related to their code. In this way, APM solutions not only help ensure operational stability, but also assist teams in building quality into the applications.

According to a 2015 Total Economic Impact study conducted by Forrester Research, the potential ROI of CA APM is related to significantly reducing downtime costs and improving productivity.[11]

[10]"CA Test Case Optimizer ROI Business Case Detailed Report," CA Technologies, July 1, 2015: https://km.ca.com/sales/presales/roi/ROI%20Portal/Dashboard/CA%20Test%20Case%20Optimizer%20Reference%20ROI%20Business%20Case%20V1FY16REF.pdf

[11]The Total Economic Impact of Application Performance Management: http://www.ca.com/content/dam/ca/us/files/industry-analyst-report/the-total-economic-impact-of-apm.pdf

www.dbooks.org

Interviews with four customers (across financial services, banking, and health insurance sectors) and subsequent financial analysis revealed that a composite organization based on the interviewed organizations experienced benefits of more than $7.2 million versus implementation costs of just over $1.7 million, adding up to a net present value (NPV) of $5.4 million and an ROI of 306 percent.

As would be expected, all the interviewed organizations experienced a positive financial impact by improving the availability and performance of customer-facing applications. For one company, this extended to reducing service-level penalty fees because it was able to prove the cause of problems to a key commercial customer, reducing the flow of more than $800,000 that it was paying yearly to a large, global vendor.

Beyond availability gains, and importantly from a DevOps perspective, there were other benefits. The study found that by applying CA APM within the development process, developer productivity increased. When the development team used CA APM to test new code, the organization was able to accelerate development cycles by 15 percent in the first year. As a representative of one company said, "In essence, we're issuing less code or less fixes into production, which allows our developmental organization to focus more on maintaining a competitive advantage in the marketplace of what we deliver in our online solutions."

Leveraging CA APM provided insight into performance from mainframes through mobile devices. Using CA APM allowed the organization to eliminate uncertainty about the cause of problems in complex environments. CA APM also reduced the average time to resolution and it eliminated the finger-pointing.

Avoiding blame games and a hero culture is of course the essence of DevOps, but application complexity and needing to support multiple stakeholders means tools must be capable of delivering uninterrupted visibility. The study illustrated that by providing performance insight from mobile to mainframe, CA APM allowed the organization to eliminate the uncertainty about the absolute cause of problems in complex environments. As one performance analyst from a regional consumer bank stated, "APM gives us micro-visibility into the customer experience."

The Whole Is Greater than the Sum of its Parts

Given that these aspects (agile requirements design and APM) of applications delivery represent only parts of the overall lifecycle and areas for potential DevOps impact, one can easily justify adoption based on the ROI they singularly deliver. However, to be truly reflective of DevOps impact, tools should be examined in unison. This way, organizations can begin to extrapolate how combinations across the automated software factory can create the massive gains highlighted in the State of DevOps Report.

Figure 9-1 and the notes that follow illustrate a simple but powerful example of the cumulative benefit of combining automated tools across the software lifecycle.

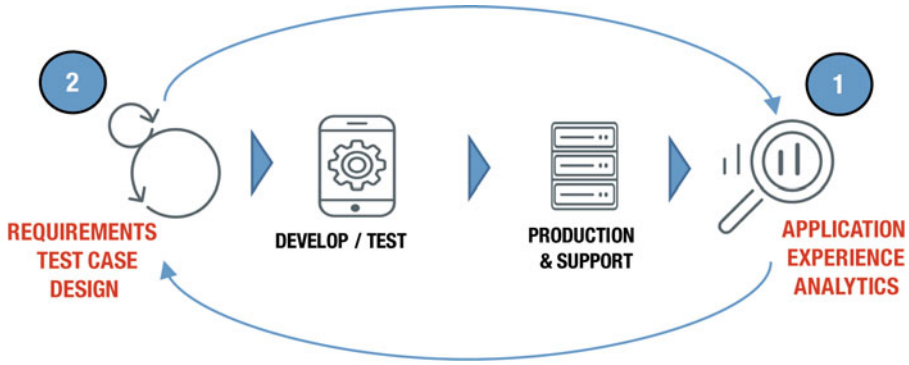


Figure 9-1. Cumulative benefits of combining multiple tools example: application experience analytics and agile requirements design

Step 1: Application Experience Analytics

In this step, tools feed information back to agile teams, which is used as input for requirements changes and iterative development.

Expected benefits include constant cycling of intelligence gained through analytics (e.g., app and functional usage and live performance) that helps developers, operations, and the business improve applications with every release.

Step 2: Requirements and Test Case Design

As requirements change, agile requirements design tools automate test case design and their execution.

Expected benefits include maximum functional coverage with the smallest number of tests, as well as reduction in test case cycles by automatically removing any outdated, redundant, or duplicate test cases.

When other tools are integrated, the benefits will continue to accumulate. For example, by automatically providing synthetic data from a test data warehouse and matching this directly to the test cases created in Step 2, an organization can:

- *Improve quality*—Providing the right data for testing can reduce defect creation
- *Improve testing efficiency*—Eliminating data constraints can reduce time and resources needed to provision data

www.dbooks.org

- *Reduce costs*—Creating accurate subsets of data can reduce infrastructure cost

Summary

As you've seen, current measurement of DevOps ROI remains somewhat nascent, and difficult. However, real-world use cases and industry research clearly highlight the massive impact that the adoption of DevOps practices is having among today's early adopters. The future looks bright for those who get on board, while those left behind may struggle to compete and survive.

In the next and final chapter, we'll look at some additional concepts, techniques, and emerging trends that have begun to populate the DevOps landscape and examine how these may be used to enhance a DevOps program.

CHAPTER

10

DevOps Finetuning

Additional Considerations, Concepts, and Practices

Congratulations, you've read this far and are ready to floor the DevOps accelerator. Like a Formula 1 driver in pole-position at the start of a race, you're eager to get the green light and hit the gas. Hopefully, you're part of a team with a winning culture, managing to business outcomes, and building the strategies needed for continuous improvement.

But this is no time to be complacent. Like Formula circuits and race day conditions, business changes constantly. Just as the Monaco circuit demands a different race strategy than Monza, DevOps programs must adjust to different circumstances. For a business focused on operational efficiency (doing more of the same, only faster and cheaper), or government agencies with shrinking budgets, Lean aspects of DevOps might take precedence. But if a business is adjusting its operating model (doing the same things, but in radically different ways), agile methods and continuous delivery will come more to the fore. Whatever the case, be prepared for change and never underestimate the importance of strong culture and building and cultivating great teams.

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6_10

www.dbooks.org

Cultural Recalibration

Formula driver Daniel Ricciardo drove brilliantly at Monaco in 2016, but when his pit-crew wasn't ready for a tire change, he lost precious seconds and the race. And while ten seconds or so doesn't seem like much, it made the difference between Ricciardo spraying champagne and finishing a miserable second.

Incidents like these reaffirm that people make mistakes all the time—it's what makes us human. Even a finely-tuned DevOps program won't be immune to people-related issues and why culture can't be overstated.

Unlike process, culture can't be automated (not yet anyway), which is why organizations must constantly work on fine-tuning. Yes, this involves building a strong collaborative workforce, but also understanding what conditions exist that cause people to persist or revert back to behaviors counter to DevOps thinking-also known as "falling off the DevOps wagon".

The Normalization of Deviance

How many times have you witnessed a sub-optimal IT practice that everyone else thinks is okay? Then over time have you accepted the behavior and started practicing it too? We all have—it's quite normal.

Regardless of whether you lead a startup or work in an established business, we all have a tendency to accept suspect behaviors. Even if outsiders see them as wrong, our IT teams are so accustomed to using them (without any adverse consequences) that they're quickly established as "normal" and accepted.

Studies into what's commonly referred to as the "normalization of deviance" have been conducted in areas from healthcare to aerospace, with evidence showing that many serious errors and disasters occur because established standards have been bypassed and bad practices "normalized".

While examining this phenomena is critical in the context of safety, it's equally applicable in how we develop, secure, and operate software applications. With the boundaries blurred between the digital and physical world, any adverse behavior leading to security and reliability issues could have dire consequences. And when software becomes infused into long-lasting products (from light bulbs to limousines), it's not so easy to discretely exit markets.

As businesses increasingly rely on software innovation for market expansion, faster time-to-market and a high-quality customer experience become essential differentiators. Unfortunately, both can be compromised if rigid change controls or "speed at all cost" mandates lead to a bad behaviors.

Interestingly, guidance from other fields can help IT identify and eliminate poor practices. In the healthcare field, for example, studies have identified seven factors that lead to a normalization of deviance.[1]

All of these are extremely relatable to IT.

The rules are stupid, dumb and inefficient—In healthcare, accidents can occur when medical practitioners disable equipment warning systems because alarms are seen as distracting. This happens in all the time in IT with dire consequence. Like when IT operations staff miss major problems because they filter out noise and alerts on monitoring consoles they regard as irrelevant. Or when testing is deliberately skipped because of lengthy manual processing and provisioning delays.

Knowledge is imperfect and uneven—Employees might not know that a rule exists, or they might be taught a practice not realizing it's sub-optimal. In IT this persists because many new employees feel uncomfortable asking for help, or the application of new technologies distorts logical thinking.

The work itself, along with new technology, disrupts work behaviors—To support goals of more continuous software delivery, organizations are introducing many new technologies and methods, such as microservices and containers. Along with the new tech, new work practices and learning demands may lead staff to poorly implement the technology or use it to perform a function it was never designed for. For example, containerizing a legacy system just because it's "technically feasible".

We're breaking rules for the good of the business—Staff may bypass rules and good practice when they're incentivized on faster delivery times or delivering new functional software enhancements. For example, procuring additional (but unnecessary) hardware capacity to rush through an update, rather than addressing the root cause of persistent performance problems.

The rules don't apply to us…just trust us—Empowered agile teams are fantastic, but with great power comes great responsibility. Teams might slam through more code, but poorly governed open source access or bypassing compliance policies when generating test data can derail a program or lead to massive security breaches. Unfortunately in today's fast-paced digital business, talented professionals often feel completely justified in playing the "trust us, we know what we're doing" card.

Employees are afraid to speak up—Violations become normal when employees stay silent for fear of admonishment. How many times have bad software code practices been tolerated because junior staff is afraid to speak up? Even in IT organizations with a strong blameless culture, people can remain silent for fear of appearing "mean" to their colleagues.

[1] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2821100/

www.dbooks.org

Leaders withhold or dilute findings on application problems— Whether you work in healthcare or IT, no one wants to look bad to managers. Rather than present ugly and unpleasant realities, many will distort the truth; presenting diluted or misleading information up the command chain. In IT, this behavior is easily normalized, especially if teams get away with reporting technical vanity metrics over business outcome-based performance indicators.

Tip Staff will never admit to anything they don't see as wrong. Get out of the office and "gemba," or walk your own software production line. Look for all the warning indicators that might be causing people to do the wrong things.

Combatting Sub-Optimal Behaviors

No sudden cultural reawakening across the IT organization will eliminate ingrained bad practices, but DevOps and Lean thinking can help identify the warning signals and can become the catalyst to drive behavioral improvement. This can involve:

- *Fostering open discussion*—It's easy to stand back and do nothing if a sub-optimal practice has yet to cause problems. But over time, poor behaviors become more difficult to remove, so catching early is critical. Ongoing education and open discussion can help with both development and operations participating. Some of this might be uncomfortable and peer-level critiques are never easy, but it's better than conducting blame games, post outage or major incident.
- *Managing to outcomes*—People are reluctant to change behaviors if they perceive the probability of a major incident or outage to be low. Part of this is normal human behavior, but can be exacerbated when the staff downgrades the likelihood of problems eventuating because diagnostics point only to minor technology issues. Therefore, it's important for staff to clearly understand the business impact (worst case scenarios) of poor practices.
- *Finding visual clues*—This includes visualizing the flow of value delivered by software applications and pinpointing all the bottlenecks and constraints. Analogous to unsafe stepping stones across a stream, these are the value interrupts. When examined, they reveal all the process and

technology issues causing staff to do the wrong things. Immediate candidates are software release and testing processes, but analysis shouldn't be restricted to development. Everything from enterprise architecture, stakeholder engagement, and information security, to vendor management and customer support can be a choke-point.

DevOps and Talent Management

While DevOps promotes the collaboration of software development and other IT groups, there is nothing stopping other departments across the organization from becoming involved in the program. Even the HR team can be included, especially with regards to talent acquisition and development.

In the application economy, demand for technology professionals in areas such as IT architecture, big data/analytics, and cloud computing is growing fast. Having DevOps skills is also becoming financially attractive. As was indicated in a 2015 survey from the cybersecurity firm Incapsula, which revealed that high school graduates in entry-level jobs requiring DevOps skills are seeing a median starting salary of $106,000.[2] But in advanced economies, demand is outstripping supply. In the UK, for example, the *Tech Nation* report, from Tech City and innovation charity Nesta, indicated that 40 percent of digital entrepreneurs say they face challenges finding skilled digital workers.[3]

The source of skills is also changing. The *Tech Nation* report also revealed informal or in-house training as the most common source of skills, with traditional universities ranking relatively low in IT activities. This is in no small part due to easier access and cheaper technology. For example, and as of July 2016, GitHub (the open source development platform) has 15 million people collaborating across 38 million repositories, while the credit card-sized raspberry PI processor has reached 5 million sales.

So You Think You Can DevOps?

With major supply challenges, HR can work collaboratively with IT to enrich the organization with much-needed DevOps skills. This will involve sourcing by traditional mechanisms (e.g., online job boards), but since DevOps requires no formal qualifications or certification, careful attention during candidate screening and selection is needed.

[2]DevOps Salary Survey, 2015: http://lp.incapsula.com/rs/incapsulainc/images/Report%20-%20DevOps%20Salary%20Survey%202015.pdf
[3]Tech Nation, 2016: http://www.nesta.org.uk/sites/default/files/tech_nation_2016_report.pdf

www.dbooks.org

Ideally, DevOps practitioner should exhibit the following skills:

- *People*—DevOps practitioners need to be good communicators, which is a skill that's not normally associated with introverted technologists. Look for people who can exhibit evidence of written and verbal communication—within, across, and external to organizations. For example, has the candidate presented at conferences or tech meet-ups?

Note HR can help IT develop good communication skills through the introduction of formal courseware and "lunch and learn" sessions.

  Careful consideration should be given to candidates who have experience working with business stakeholders and can clearly articulate strategy, priorities, and goals. DevOps practitioners should be able to describe situations where they collaborated and shared information to achieve better business outcomes.

- *Process*—While it's possible for DevOps to work with Waterfall development, success is more likely in environments where agile methods have been applied. Strong candidates will demonstrate understanding and experience in one or more agile approaches (e.g., Kanban, Scrum, or SAFe).

  Value should also be placed on candidates who can demonstrate knowledge in Lean manufacturing concepts such as Just-In-Time and Theory of Constraints. Extra marks if they can illustrate examples of where and how they applied these principles in an IT context.

Note Working with IT managers, HR could introduce an online library of DevOps, agile, and Lean related books. Some great examples include *The Phoenix Project* and *The Goal* and *Lean Enterprise.*[4] Don't limit access to IT; make the resources available across the organization.

---

[4]*The Phoenix Project,* Gene Kim, Kevin Behr, George Spafford, ISBN 978-0988262591
The Goal: A Process of Ongoing Improvement, Eliyahu M. Goldratt, ISBN 978-0-88427-178-9
Lean Enterprise, Jez Humble, Joanne Molesky, Barry O'Reilly, ISBN 9781449368425

- *Technology*—DevOps practitioners will have acquired a range of skills across the software development lifecycle, including, but not limited to, agile project management, build management, and release automation. Strongly consider candidates who are experienced integrating these to build a fully automated DevOps toolchain.

■ **Note** Look for individuals who have worked in cross-functional teams and have broadened their skills by learning new technologies. Individuals who not only use technology but have contributed to its development are also good candidates.

Rather than present resumes, strong technologists will emphasize their achievements and expertise via actual contributions within the agile and DevOps community. This can include technology blogs, and open source contributions (code, scripts, and commentary). Ideally, sysadmin candidates will have coding skills across a variety of programming languages, while developers will be familiar with scripting methods.

Various Talent Management Hacks

Age shouldn't be a factor when selecting DevOps talent. Many of the best practitioners have years of experience adopting new technologies and practices. In the enterprise, this is especially valuable since many legacy systems will need to be integrated with cloud applications, which requires a comprehensive understanding of the inner workings of many technologies and all their limitations. That said, however, it'll be necessary to replenish DevOps programs with fresh talent. And since supply is problematic, HR can assist technology teams with innovative acquisition and development strategies. This could include the following.

IT Apprenticeship and Graduate Intake Programs

Many organizations have graduate intake programs, some of which don't require applicants to have formal IT qualifications. They encourage anyone to apply who has a passion for technology. One example is the ICT Apprenticeship program run by the Department of Finance in Australia.[5] After a series of interviews, successful applications are employed by a sponsoring government agency. Upon assignment, they combine four days of work with one day of formal IT study, leading to certification.

---

[5] http://www.australia.gov.au/information-and-services/jobs-and-workplace/australian-government-jobs/ict-apprenticeship-programme

www.dbooks.org

Such programs are a great way to infuse talent within an organization, but there are challenges that HR teams must address. Often in these types of initiatives, apprentices are rotated through different areas of IT—the aim being to familiarize new staff with all aspects of the IT business. This is fine in theory, but in a DevOps context (where cultural improvement is so important), it can actually be counterproductive. If, for example, the organization is functionally siloed, divisive, and averse to change, apprentices could become isolated, and rotating passionate and eager technologists through highly process-centric areas can be the fastest way to dampen their spirit and enthusiasm.

Tip Consider organizing graduates or IT apprentices in self-managing teams or "pods" with full program oversight by experienced leaders and mentors. Always assign work that is meaningful, valuable, and measurable. Stagger the arrival of new intakes and place them in the same workplace as existing apprentices.

DevOps Hackathons and Coding Days

Hackathons are short events where all kinds of specialists come together to work intensively on small technology projects. Hackathons can be run as a contest (e.g., prize for best mobile app innovation), hosted internally or externally, or purely social. In some cases, they even have the lofty goal of creating commercially viable software solutions.

With DevOps, hackathons can be valuable for instilling the value of collaboration and teamwork within the IT organization. HR can take the lead in helping to organize and arrange these events, but should work with IT leadership to ensure maximum DevOps benefit. For example, conducting the hackathon away from the workplace where participants could be distracted. Other factors to consider include:

- *Balancing teams*—Including developers, sysadmins and project managers, enterprise architects, and security specialists.
- *Providing broader deliverables*—Don't restrict hackathons to new apps. Look for other opportunities to improve quality. For example, give prizes for the fastest way to find the root cause of a complex application performance problem; the most innovative application of security to increase resilience; and the best API to securely expose enterprise data.

- *Managing expectations*—Don't expect clean and robust code from a short hackathon; you'll get alpha at best. Consider longer online developer challenges if the goal is to fuel business with viable business prototypes (e.g., integrating valuable enterprise data with third-party systems).

Involving partner developers in a hackathon is a great opportunity to nurture innovation but requires additional oversight and support. Providing badly formatted open data without metadata, or poorly documented and unmanaged APIs, should be avoided. When involving third-party developers, consider methods to simplify API discovery, registration, support, and engagement, including providing an API catalog, sample code, and mobile device SDKs.

DevOps and the Internet of Things

At the 1990 Interop computer conference, two guys connected a kitchen toaster to the Internet. This early attempt at appliance networking paved the way for what's now called the *Internet of Things* (IoT).[6]

As of 2016, it's estimated there could be more than 10 billion Internet-connected elements, ranging from watches and cars to just about anything in the "thing" category, including light bulbs, fitness wearables, fridges, and, yes, even pets and diapers.

But beyond the big numbers, marketing spin, and hype, the business potential of IoT is being increasingly exploited by early movers. Perhaps not surprisingly, this includes some major businesses and brands.

Bosch, for example, the German multinational engineering and electronics company, announced an IoT cloud. This is the latest initiative as the company transitions from traditional industrial company to a blended manufacturer and technology provider.[7]

While connected home and car use cases grab our attention, many industries are exploiting standards, unified architecture (IT and industrial), and innovative software solutions to connect device and sensors to IT systems.

Take for example manufacturing. When plant production systems can be integrated with customer demand analytics, maintenance windows can be optimized. Similarly in energy and utilities, integrating smart meters with back-end applications and analytics could become a system to predict energy demand across the grid.

[6] http://www.livinginternet.com/i/ia_myths_toast.htm
[7] http://www.wsj.com/articles/robert-bosch-launches-own-cloud-for-internet-of-things-1457528014

www.dbooks.org

This convergence will require new approaches to developing and operating applications, and DevOps with its focus on shortening delivery cycles while improving software quality is absolutely essential. However, IoT project success depends on DevOps practitioners addressing some new challenges:

- *Bridging another cultural divide*—The development and operations divide is nothing compared to the cultural chasm spanning traditional IT and industrial operations. While IT speaks of Java, C++, NoSQL, and the IP stack, plant and industrial engineers talk of programmable logic controllers and supervisory control and data acquisition (SCADA).

■ **Note** Putting all IT "ego" aside, DevOps teams need to build a good understanding of IoT related terms, technologies, and limitations. To assist, try to involve product specialists (e.g., industrial engineers, product managers, and designers) in DevOps/IoT initiatives.

- *Solving complex security problems*—Connecting previously isolated physical devices to networks creates additional threat surfaces. At best this could mean service interruption, at worst physical injury or loss of life. This consequence was disturbingly illustrated in the Jeep security exploit, where hackers demonstrated wireless access to vehicle infotainment and controls.[8] These could be expensive IoT lessons in the automotive industry where fleets of new cars can leave a dealership containing up to 70 specialized computers and more than 20 million lines of code.

■ **Note** Considerable investment will be needed to address IoT security issues. DevOps and security teams must review entire IoT processes (device to cloud or back-end application), thus ensuring that embedded software applications and data can be verified and encrypted.

- *Managing privacy*—Internet privacy issues could be nothing compared to the challenges introduced by IoT. As the physical and digital world becomes blended, smart devices have the potential to track behavioral patterns everywhere—as we sleep, during exercise, when we drive, consume energy—you name it!

---

[8] https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

Tapping into this information means DevOps practitioners must pay careful attention to issues around data ownership, compliance, data access, and sharing arrangements.

■ **Note** As more customer data is obtained from IoT devices and consumed by complex analytical, and big-data systems, careful consideration to accurate and compliant data management (e.g., during testing) becomes more critical.

- *Guidance on IoT interoperability*—Many industrial operations and IoT systems work in isolation and use proprietary networks (e.g., Modbus and BACnet) and protocols. In many cases, what works in one industry vertical won't work in another. While this problem can't easily be addressed, DevOps teams involved in IoT projects must become knowledgeable on emerging IoT standards and interoperability initiatives. Examples include The Open Connectivity Foundation and the Allseen Alliance.[9]
- *Understanding physical device limitations*—Depending on business context, IoT devices characteristics and usage will impose many limitations on software development, testing, and operations. Not the least, constraints on what programming languages can be used for IoT development and strict limitations on runtime size. Since software won't be operating in the confines of a 24/7 data center, there'll also be design issues to factor in (e.g., device battery life, environmental/weather conditions). Consider too, challenges making software updates across sub-optimal networks, the mass migration of sensor data to the cloud for analysis, and determining embedded software security risks (especially in OEM devices and third-party components).

DevOps for the Public Sector

You can be forgiven for thinking that DevOps only works in the private sector. It's simply not true.

Governments across the world continue to search for better ways to deliver citizen services through new technologies and practices. Agile and DevOps are gaining traction, with federal U.S. departments, such as U.S. Citizenship and Immigration Services (USCIS) and the Environmental Protection Agency, becoming adopters.

---

[9] https://allseenalliance.org/https://openconnectivity.org/

www.dbooks.org

But adoption isn't limited to the United States. In Australia, the recently formed Digital Transformation Office (DTO) leverages agile methods and DevOps principles to support its mission of leading the digital transformation of government services.[10]

But as of 2016, the public sector hasn't fully embraced DevOps thinking. This was well illustrated in a Vanson Bourne study, which indicated that public sector organizations are a third less likely to adopt DevOps compared to their private sector counterparts.[11]

Yet those who are starting a DevOps journey are seeing benefits. As indicated in the study, more than one-third of public sector DevOps adopters have experienced application quality improvements, while almost half have seen reductions in IT spend. This last point is especially pertinent for budget-constrained government agencies.

DevOps and Lean to Improve Government IT Outcomes

Government agencies are under increased scrutiny to ensure money is spent wisely. Unlike their private sector counterparts, who can justify mega IT spending as the means to increase revenue and market share, government agencies can only think in terms citizen-centric service improvement, which given the overarching political context and climate, might not be a top priority.

Even when budget is available, the track record in government IT delivery hasn't been great. According to Standish Group, just over 94 percent of all large government IT projects (2003-2012) were over budget, behind schedule, fell short of user expectations, or had to be abandoned completely.[12]

With a 6 percent success record, it's perhaps not surprising that the DevOps mantra of "failing fast" doesn't fully resonate in government departments. Apart from the cost factor, no IT manager would want to front a congressional committee when critical government services have been disrupted.

With increasing budget constraints, probably the best place to start a DevOps program in a government agency is to focus on the Lean principles that underpin much of the thinking. By identifying all elements of "waste" that incur greater cost and negatively impact delivery, agencies can improve cost structure while speeding delivery. To this end, and before any investment is made in tools,

[10] https://www.dto.gov.au/
[11] https://www.ca.com/us/rewrite/articles/devops/research-report--devops-the-worst-kept-secret-to-winning-in-the-application-economy.register.html
[12] http://www.brookings.edu/blogs/techtank/posts/2015/08/25-yaraghi-government-it-projects

mature government agencies will carefully examine the entire end-to-end software delivery cycle and remove any cultural, process, or technology related inhibitors that add no value. The intention of course is to increase speed and quality, but a very important "by-product" for the budget constrained agency is cost reduction.

Many wasteful practices traditionally associated with manufacturing are applicable in a government software development context.

- *Long waits*—Some government agencies remain fixated on performing lengthy change review processes. But with new technologies now making it possible to deploy fully certified stacks to the cloud, only application changes need to be checked. This significantly reduces release times and administrative costs. Of course, there will be occasions when teams require access to production systems and data for testing, but cycle time can be shortened with technologies that virtualize or emulate dependent systems.
- *Excess inventory*—Over years, agencies have acquired a complex array of technology. These include mainframes, applications, and databases. Each servicing agency requirements in everything from massive transactional processing to data matching and analytics. In order to manage this complexity, agencies have become reliant on contractors who generally shun agile and DevOps concepts, preferring instead contracts with fully defined requirements and longer delivery cycles. To address this, agencies should immediately review the IT culture and skills gaps issues this causes, plus consider engaging nimbler agile and DevOps minded contractors via simpler procurement processes.
- *Slow motion*—As with large commercial enterprises, government agencies can be inhibited by bureaucracy and controls. As governments look to adopt DevOps, it's important to analyze every unproductive process across the software development lifecycle. A large part of the magic in DevOps comes from "doing rather than procrastinating," so agencies should start familiarizing themselves with smaller and iterative approaches to software development combined with fully automated release processes—a state where everything moves with purpose.

Even in situations where rigorous compliance testing is seen as a tough but necessary control point, DevOps and automation can pay dividends. By encapsulating testing within development for example, DevOps practitioners can enact the necessary speed and quality improvements, while demonstrating reduced costs in auditing, risk management, and compliance. This is again very attractive to cash-constrained government departments.

www.dbooks.org

- *Untapped skills*—When things are slow and hard to change, everything atrophies, including staff and their skills. If DevOps is to help slow moving government agencies, it's important to identify all issues that prevent change. Over years many staff will have become accustomed to working in a certain way, with incentives linked to these practices. It's important to maximize the motivational impact of new DevOps programs, while also ensuring staff to stay the course. This may involve changes in organizational structures, new talent acquisition strategies (discussed earlier), and the adoption of shared goals, incentives, and metrics.

Summary: The Seven Point Action Plan

As we've discussed in this book, DevOps aims to break down the traditional barriers between development and other IT groups. By unifying people, technology, and processes across a modern software factory, new applications and updates can be built, tested, and deployed more quickly. Together with continuous feedback, businesses can enhance customer experiences, accelerate time to value, and better leverage IT as a competitive differentiator.

But while interest in DevOps continues to grow, many implementations can fall short of expectations or stall completely. In part, and as we've described, this is because DevOps challenges conventional IT wisdom—placing less emphasis on process rigor and more on removing organizational barriers and continuous adjustment.

As such, DevOps doesn't come pre-packaged with implementation blueprints; however, there are steps any organization (irrespective of DevOps maturity) can follow to optimize a DevOps initiative and we present these as a summary.

Understand the Business Goal

A program like DevOps is going take lots of work. It's great to get carried away with the "DevOps buzz," but DevOps comes with a price tag, which includes new tools and processes to learn, people to cajole (and console), and teams to restructure. As with any business decision, all this cost and work must be carefully weighed against the expected benefits.

Doing DevOps for the sake of DevOps is no way to start. Fall into this trap and customer value and business benefits take a backseat and become less important than protracted technology and process discussions. That's a DevOps disaster before you've even started.

It's important therefore to get business and IT on the same DevOps page. Focusing first on the needs of the business and then aligning DevOps, including metrics, tools, and processes.

By focusing first and foremost on the needs of the business and clearly articulating concrete goals across IT, you'll ensure a consistent approach and avoid misinterpretations or DevOps misfires.

As we like to say—don't fall in love with DevOps, fall in love with the business problem and let DevOps help you solve it!

Cultivate Senior Level Sponsors

If you're going to be changing things like organizational structures, work practices, and incentives, you'll need some support. In large organizations, change takes time, so seek out a sponsor to facilitate decisions and make the tough calls.

Executive level sponsors add substance to the DevOps sauce. They give teams a sense that there's real purpose behind the initiative and that it's not another process "wild goose chase". Remember too that senior managers are great at PR, which is very useful when you want to spread the good word about DevOps, build momentum, and gain more support.

Note There's no such thing as a "free lunch". Executive sponsors expect real returns from DevOps, so try to make sure the goals being set can help them meet theirs.

Select Your Pit-Crew

People will make or break a DevOps initiative, so choosing the right staff is important. There are no hard and fast rules, but seek out employees who can:

- *Work well in teams*—Look for people who have demonstrated a willingness to collaborate across organizational boundaries.

Caution Be careful building a dedicated "DevOps team," as this might become another silo. However, recognize that temporary teaming could be a good way of seeding DevOps goodness across the wider organization.

---

www.dbooks.org

- *Show resilience and flexibility*—Great DevOps practitioners quickly learn from failure and embrace new ways of thinking to challenge the status quo. With developers for example, this can involve taking full ownership of production rollouts, not just the code they're working on.
- *Empathize with colleagues*—Seek out staff across the organization who consistently place themselves in the "shoes" of their colleagues. If a developer takes time to work with on-call staff to determine where supportability improvements are needed then they're a great candidate.

Note Generalists are preferred over specialists, but strong consideration should be given to staff who have experience with newer practices, including continuous integration and testing, release automation, and toolset integrations.

Revisit Chapter 3 and the section on DevOps culture for more pointers.

Choose an Immediate Deliverable

Select an application that's small enough to remain manageable with new DevOps approaches, but has enough business visibility to stimulate wider adoption. There's nothing to stop applying a DevOps approach to any type of application, but it's generally better suited to mobile and web facing applications, which are architected to take advantage of aspects of DevOps tooling described in this book, including API management, continuous delivery, and "shift-left" monitoring.

Legacy applications and systems-of-record can also benefit hugely from DevOps and shouldn't be ignored. Newer customer-facing mobile apps may need to integrate with back-end applications hosted on mainframes providing essential horsepower. Unfortunately, the practice of managing mainframes in silos can severely impact software development and adversely impact customer experience.

Take, for example, a situation where a mobile development team needs access test data residing on a mainframe or insight into the performance implications of newly released code. These teams must have fast unconstrained access to this data to meet schedules together with end-to-end performance insights—mobile to mainframe. If teams work in silos using point tools, this will be difficult to achieve.

Many advanced tools, together with in agile-style methods, can help align mainframe management with a DevOps initiative. This includes:

- *Extreme performance visibility*—These tools provide deep visibility into complex interactions and transactions across mobile, web, and mainframe applications. They help teams quickly understand and troubleshoot even the oldest and most complex code bases.
- *Constraint removal*—By simulating mainframe infrastructure dependencies and providing synthetic test data, developers and testers can maintain the pace of DevOps delivery without compromising quality or compliance goals.
- *Unified infrastructure management*—Enables mainframe and non-mainframe experts alike to better understand infrastructure interdependencies, thereby detecting and fixing problems faster and without the need for specialist tools.
- *Extensible analytics*—By extending analytics into areas such as *workload automation*, teams can perform real-time forecasting of critical back-end services. Using advanced statistical methods to calculate job duration, administrators can fine-tune workload schedules and processing to cater to changing conditions.
- *Exposing rich data sources*—Allows developers to rapidly create application back-ends for internal applications, mobile apps, standalone microservices, data as a service, and partner integrations. These systems can help decompose large mainframe applications into self-contained units with everything needed for app delivery, including data integration, business logic, and a robust API layer.

Revisit Chapters 4-7, where these capabilities are described in greater detail. Always consider that tooling shouldn't be limited to managing one particular platform or supporting a single group.

Build a Comprehensive Metrics Program

Traditionally, and as we described in Chapter 3, goals and objectives have been set at a departmental level, with methods for measuring IT capability skewed toward resolving problems and rewarding the "heroes" who fix them. With DevOps, objectives should be set at a "singular team" level and aligned to the desired business end-state—digital transformation via faster release of high-quality software.

www.dbooks.org

With this in mind, work collaboratively to build shared objectives and metrics aligned to business objectives. Avoid vanity metrics that distort the truth or incentives that reward siloed behaviors. Next, set some improvement targets and assess the capability of existing tools in meeting and surpassing your goals.

■ **Tip** Just because a target is constantly being achieved doesn't mean improvements can't be made. Don't get complacent. Just like code, DevOps metrics need constant care and attention!

Integrate DevOps with Existing Practices

As we discussed in Chapter 8, DevOps doesn't preclude leveraging other methodologies and best practices, including ITIL.

To avoid methodology "turf wars," it'll take teams working collaboratively to identify where existing processes need to refining to cater to DevOps-style delivery. Obvious places to start are with existing release and change management, but be prepared for "give and take" on both sides. Never forget that DevOps itself can benefit from well-established processes, such as incident and problem management to strengthen feedback loops and accumulate organizational knowledge.

■ **Tip** Consider establishing a single workflow-based communication process between developers and other groups, including support staff. Integrate these with monitoring, ticketing, and help desk solutions to make problems and solutions visible across the entire organization.

To be fully effective, DevOps practitioners also need to work with many other stakeholders, including enterprise architects and security. Once again, and using the guidance described in Chapter 8, be prepared to adjust existing methods and take other perspectives on board.

Automate the Pipeline; Strengthen the Toolchain

Reexamine the "eight elements of waste" introduced in Chapter 3. It's a useful framework to better identify all the existing process and technology constraints across your own software factory.

While pushing code all the way to production via fully automated releases might be the holy grail of DevOps, it could be better to start by reviewing those areas experiencing the greatest pain or where integrated toolsets can

help fill in the release pipeline "whitespace" (for examples, revisit the release automation integration use cases illustrated in Chapter 6 and the automated testing trifecta presented in Chapter 5).

In other cases, reviewing waste and constraints may indicate that operational functions have the most to benefit from automation in the short term. This could include the adoption of modern configuration methods (infrastructure as code), or establishing performance monitoring and analytics to build critical bidirectional feedback loops between development and IT operations (for an illustrative example, loop back to case study and discussion on "shift-left" monitoring presented in Chapter 7).

Tip As in traditional manufacturing, many wasteful IT practices and constraints inhibit the flow of value/throughput. Identify the greatest painpoint and potential for human error, be it managing test data, accessing infrastructure dependencies, or conducting manual and error-prone handoffs.

In Salute of DevOps

In a world where customers and citizens expect so much more from technology, the race is on to develop a finely tuned high-performance IT function. Hopefully by reading this book you're in a better position to see how DevOps supports this imperative and can develop the strategies needed to reignite your business.

In Formula 1 racing, winning one race doesn't win a championship. It's the same in software development. DevOps teams understand this implicitly. They are smashing silos, collaborating fully, and leveraging advanced automated solutions to eke out business performance improvements everywhere, as they build, test, release, and manage….

…. superior high-quality software solutions, continuously delivered from a modern DevOps-enabled software factory.

….start your engines!

www.dbooks.org

I

Index

A

Agile methods
  autonomous teams, 18, 19
  empowerment, 21–22
  iterative development, 149
  sprints, 114
  standup meetings, 31

Agile operations
  craftsmanship, 111

Agile requirements, 75, 80, 146, 148, 149

Analytics
  App experience, 56, 77, 121
  business, 56, 117
  customer, 53, 159
  performance, 56, 117

Application
  microservices, 23–25, 57, 109, 111, 126, 147, 153, 167
  monolithic, 21, 23–25, 109, 126
  supportability, 21, 25, 110, 112, 128, 136

Application performance management (APM)
  alerts and alarms, 61, 109
  baselining, 113, 120
  experience analytics, 56
  transaction tracing, 61

Application programming interfaces (APIs)
  API gateways, 52, 57, 61, 62
  creation, 57, 58, 63
  deployment, 55, 57, 58, 67
  lifecycle, 53, 55–61, 65, 67
  mobile and SDK's, 60, 159
  monitoring, 54–57, 61–62, 128, 147
  security, 54, 55, 57, 59, 60, 63, 65

B

Bad practices
  normalized deviance, 152
  warning signs, 154

Business
  models, 8, 10–12, 17, 32, 33, 64, 65, 106
  wicked problems, 15–17

C

Cloud Computing, 10, 15, 33, 155

Constraint removal, 40, 73, 81–85, 167

Containers, 20, 21, 109, 110, 128, 147, 153

Continuous delivery
  maturity levels, 91

Continuous integration
  build management, 90

Culture
  collaboration, 44
  empathy, 29–30, 33
  integrity, 19, 79
  respect, 29, 33
  trust, 29, 33

© CA 2016
A. Ravichandran et al., *DevOps for Digital Leaders*, DOI 10.1007/978-1-4842-1842-6

D

DevOps
- Action Plan, 100–102, 164–169
- in government, 8, 151, 161–164
- pipeline, 69, 95, 130, 132, 168–169
- toolchain, 76, 88, 95–98, 157

E, F, G

Enterprise architecture, 29, 122, 125–130, 133, 137, 155

H

Human resources (HR), 155–158

I, J, K

ITIL
- Change Advisory Board, 134
- change management, 134, 135, 168
- co-existence with DevOps, 134
- incident management, 40
- problem management, 134, 168

L

Lean Startup, 8, 136–137

Lean thinking
- value creation, 34–35

M, N

Mainframe systems, 81, 83, 84, 106, 121, 148, 163, 166, 167

Metrics
- anti-patterns, 41–42
- domains and classes
  - culture, collaboration and sharing, 44
  - customer and business value, 45
  - efficiency and effectiveness, 44
  - quality and velocity, 45
- lead time, 41, 45
- mean-time-to-recover (MTTR), 44, 45
- Net Promoter Score (NPS), 45, 46
- outcomes and actions, 137
- programs, 41, 43, 46, 47, 167–168
- targets and initiatives, 46, 47
- vanity, 42, 47, 154, 168

Mobile testing, 76–77

O

Open source software, 20, 128

P, Q

Platform as a Service (PaaS), 20, 22

Programming languages, 19, 24, 157, 161

R

Regulatory compliance
- General Data Protection Regulation (GDPR), 78

Release automation
- integrations, 96, 97, 99, 135, 169

S

Security
- privileged access, 59, 131, 135
- static code analysis, 130

Security-minded DevOps, 133

Service management, 36, 94, 133–136

Service virtualization, 37, 38, 63, 71, 80–85, 97, 114

Shift-left
- application monitoring, 98, 115–118, 166, 169
- security, 131

T, U,V

Talent management
- DevOps skills, 155–159

Test automation, 73–77, 80–81, 98

Test data management
- data masking and sub-setting, 79
- synthetic test data, 78, 79

Test data warehouse, 79, 80, 149

Testing trifecta, 73–85, 169

www.dbooks.org

W, X, Y, Z

Waste–elements of
defects, 35–38
employee knowledge (unused), 36, 40
inventory, 36, 39
motion, 36, 40
non-value added processing, 35, 36, 39
overproduction, 35, 38
transportation, 35, 36, 39
waiting, 35, 38–39
Waste-removal, 37–40
Waterfall development, 17, 76, 78, 156