Praise

Mirco knows his stuff. He has a way of stating simple choices and framing options that encourage action. His unique perspective on how to transform your IT organization, especially when you have heavily invested in outsourcing, are both thoughtful and practical. After you read his book, make sure to subscribe to his blog. You want to make sure to have Mirco by your side as you go through this transformation.

—**Mustafa Kapadia**, Digital Transformation Leader, IBM

This is a much-needed book on building teamwork to drive technology infrastructure efforts. Mirco introduces a simple set of processes and well-reasoned principles to align the organization, transform technology infrastructure, and deliver value for customers.

—**Eric Passmore**, Partner Director of Commerce at Microsoft

A pragmatic view on IT and DevOps that doesn't just focus on the "what" but the "how," based on firsthand experience.

—**Ajay Nair**, DevOps Architect, Accenture

This is a truly practical companion book to the other leading publications in the DevOps space. Mirco shares real-world lessons from many years of collective experience and provides tried and tested exercises for you to use to help drive insights and improvements in your organization.

—**Emily Arnautovic**, Software Architect, Accenture

DevOps is a hot topic. Pretty much every major organization wants it, and pretty much every major organization is struggling to come to grips with what "it" is and exactly how to get it. Mirco is a rare entity in the space. He's an architect who has grown up in large, complex organizations working with even larger, more complex systems integrators and delivery partners heavily reliant on ERP, CRM, and other COTS applications that seem to challenge much of what DevOps is about. Both pragmatic and practical, he not only gives great tips on how to find your way down the DevOps path in these environments but will help you avoid many a mistake as he shares his own.

—**Mark Richards**, SAFe Fellow at Coactivation

DevOps *for the* Modern Enterprise

DevOps for the Modern Enterprise

Winning Practices to Transform Legacy IT Organizations

Mirco Hering

Foreword by
Dr. Bhaskar Ghosh

IT Revolution
Portland, Oregon

25 NW 23rd Pl, Suite 6314
Portland, OR 97210

First Edition
Printed in the United States of America
24 23 22 21 19 18 1 2 3 4 5 6 7 8 9 10

Cover and book design by Devon Smith
Author photograph by Julian Dolman

Library of Congress Catalog-in-Publication Data

Names: Hering, Mirco, author.
Title: DevOps for the modern enterprise : winning practices to transform legacy IT organizations / Mirco Hering.
Description: First edition. | Portland, OR : IT Revolution Press, 2017. | Includes bibliographical references and index.
Identifiers: LCCN 2017053420 (print) | LCCN 2017056532 (ebook) | ISBN 9781942788201 (ePub) | ISBN 9781942788225 (Kindle) | ISBN 9781942788195 (trade pbk.)
Subjects: LCSH: Computer system conversion. | Operating systems (Computers) | Computer software--Development. | Management information systems. | Information technology—Management.
Classification: LCC QA76.9.C68 (ebook) | LCC QA76.9.C68 H47 2017 (print) | DDC 004.068—dc23
LC record available at https://lccn.loc.gov/2017053420

ISBN TP: 978-1942788195
ISBN ePub: 978-1942788201
ISBN Kindle: 978-1942788225
ISBN PDF: 978-1942788218

For information about special discounts for bulk purchases or for information on booking authors for an event, please visit our website at ITRevolution.com.

Contents

Figures

Preface and Introduction

Chapter 1

Chapter 2

Chapter 4

Chapter 5

Chapter 6

Chapter 7

Chapter 9

Chapter 10

Chapter 11

Chapter 12

Appendix

Tables

Chapter 1

Chapter 2

Chapter 3

Chapter 11

FOREWORD

by Dr. Bhaskar Ghosh

Throughout my many years working in the IT industry, I have encountered numerous disruptions from technology advances, new business models, and even global economic cycles. Among all those waves of change, DevOps in particular stands out. Paradoxically, it stands out not because of the principles it embodies but because of the downstream change it has ushered in.

I like to say that you can do DevOps without Agile, but you cannot do Agile without DevOps. This is just one of many examples of transformations that have been catalyzed by the advent of DevOps. By empowering software developers to do more and, consequently, own more, DevOps is unleashing creativity, which is leading to a feedback loop of continuous improvement in systems delivery.

While I fondly recall the time I spent years ago managing infrastructure operations for large enterprises, the challenges of that job often resulted from the distinction that was drawn between development and operations responsibilities. Of course, this was in an era of largely monolithic systems using Waterfall techniques for large-scale, periodic delivery

of software releases. In those past production environments, such separation of duties was a practical and efficient operating model for the needed pace of system changes.

In the digital era, however, speed is paramount. The distinction of responsibilities is not conducive to the more incremental change delivery approaches that are required to meet the demands of business today.

In this book, Mirco shares much more than just the mechanics of DevOps; he also shares his passion for improving software engineering. Through clever analogies and prescriptive advice, Mirco dispenses practical recommendations for how to embrace DevOps in an enterprise. Whether you are just getting started with DevOps or you are a seasoned professional seeking counsel on how to apply its principles at scale, as you go through *DevOps for the Modern Enterprise*, you will find yourself infected by the enthusiasm Mirco has for DevOps and the benefits it can bring.

Dr. Bhaskar Ghosh
Group Chief Executive—Accenture Technology Services
Bangalore, India
March 2018

Preface

> Learning is not compulsory; it's voluntary.
> Improvement is not compulsory; it's voluntary.
> But to survive, we must learn.
>
> —**W. Edwards Deming**

One of the most rewarding things in my career has been the search to find the most efficient way to deliver meaningful projects and to get as many people as possible to do the same. When we are not working efficiently, we spend time on unnecessary, repetitive, and boring tasks, which is not fun at all. I don't have lofty goals of changing the world by doing my work, but I think everyone deserves to enjoy his or her work. And when workers enjoy what they do, good outcomes are inevitable.

Since joining Accenture as a consultant over ten years ago, I have worked with dozens of teams to increase their delivery capability through increased productivity or increased speed. But even before my time as a consultant, I was driven to figure out ways to bring efficiency to IT. I came into the workforce in the late 1990s when offshoring IT work was still in its early days. I spent my first few years as a developer in research labs for IBM, working on *telematics* and developer tools (e.g., developing languages for custom CPUs and providing the associated *compilers* and *IDE extensions*). When I started working, packaged software was on the rise, but most work was done in custom development and onshore. The

only way to improve productivity was to increase the level of automation, and in all my early projects, we had creative solutions built around *shell scripts*, *Perl* scripts, and other custom tooling to make the life of developers and operators easier. I thoroughly enjoyed building these automation solutions and seeing how projects became easier and more enjoyable for everyone involved.

Then something curious happened. I spent the next five years on two large projects and focused on building the kind of developer tooling that I knew helped projects deliver successfully. When I finished those projects and started to look around across organizations, I realized that somehow I didn't see as much automation as I would have expected. After all, I spent all of my professional life working on automating tasks for delivery teams. When I spoke to others in the industry, it became clear that packaged software and offshore delivery capabilities provided shortcuts to productivity gains and cost reductions that many organizations leveraged rather than investing in good development practices and tooling.* I spent the next few years in what was perceived as a niche market to help organizations implement delivery tooling, but truth be told, it just wasn't sexy for organizations to invest in this.

Larger and larger offshore percentages and reductions of the average cost per developer workday (otherwise known as ADR, *average daily rate*) were targets that could more easily be sold to the organization as success than the somewhat more difficult and less easily measurable activity of developing a good delivery platform that makes everyone in IT more productive. After all, how do you measure productivity in IT in the first place? I am with Randy Shoup and Adrian Cockcroft, who both admitted while presenting at conferences that in their whole career, they have been looking for a good measure of productivity but have not been able

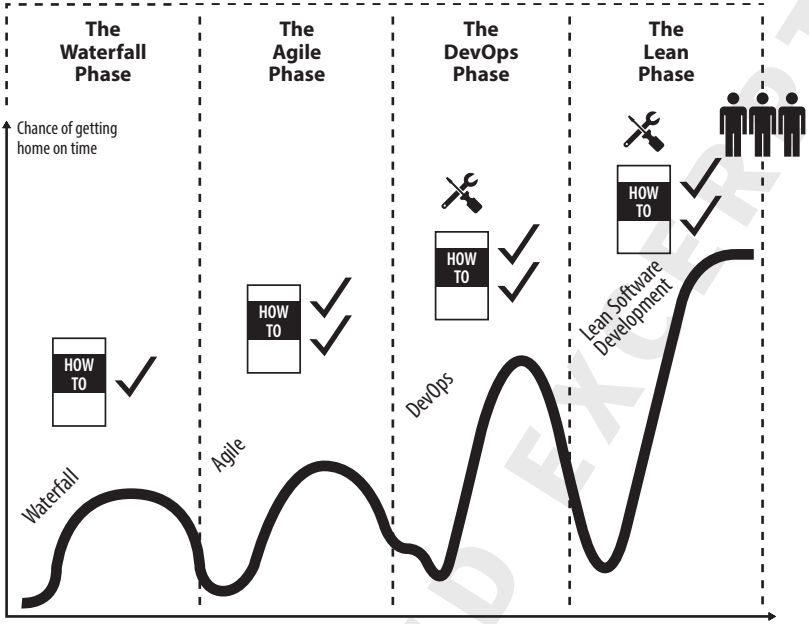
* For more information, see my DevOps.com article "Why We Are Still Fighting with the Same Problems in DevOps as 15 Years Ago."

to find something useful. I elaborated on this in a blog post to describe that productivity is very difficult to measure in IT; instead, measure cycle time, waste, and delivered functionality.[1] It is important to measure some meaningful metrics, as you are otherwise not able to see whether you are improving; it just turns out that productivity in the usual sense is elusive in IT and that we need to look for other measures that help us judge how efficient we are.

I spent the next few years working to understand where the problems in IT are coming from and how to solve them. I was lucky, as my research fell into a time when technology and methodology were evolving to bring out the foundations to a new way of delivering IT: Agile, DevOps, and *cloud*, among others, made it a lot easier to implement the kind of solutions I had built my entire career. The niche that I had worked in became more and more fashionable; today it is difficult to find an organization that is not talking about Agile and DevOps.

Yet when we take a good, hard look at ourselves in the mirror and see where the IT industry is currently, we realize that IT delivery is still not where it should be. We all tend to agree on *continuous delivery* being a good practice to use and to build modern application architectures, but when we look for organizations that have mastered it, they are few and far between. Many organizations are working in ways that have evolved over many years by taking lessons from traditional manufacturing. After all, those practices are well codified in many an MBA curriculum and have hundreds of years of experience behind them. But those practices and ideas are not appropriate anymore.

I have not mastered it all myself and am still learning every day and with every engagement, but I want to make my experiences available to as many people as possible. As you can see from Figure 0.1, I am a developer at heart who looked for technical solutions first instead of considering the people involved. I had to learn the hard way that just having the right methods and tools will not magically transform an organization. The



Waterfall	Agile	DevOps Practices	DevOps Culture
Methods & Process		DevOps Tools	People
Defined Process		Enforced Process	Guided by Principle

Figure 0.1: How Mirco's understanding of organizational change evolved

cultural change that is the hardest is also the most impactful. It took a good amount of failures and near misses to learn what I have to share in this book and to understand that you need to bring all the ingredients together with the right culture to really transform an organization.

Over the last few years, I have developed a workshop that I run with CIOs and other IT leadership from our clients to explore their challenges and help them identify possible ways forward. The fascinating thing when you are in a room with intelligent and visionary leaders is that you learn more each time. I have been running this workshop for a while, and I am extremely thankful for the experiences and ideas that the CIOs have

shared with me, which continue to improve the workshop. (This book contains the accumulated knowledge from those sessions.)

DevOps for the Modern Enterprise is meant to address the different challenges that organizations face as they transform into modern IT organizations, and yes, most organizations today are IT organizations, whether they are car manufacturers or banks, due to the dependence on IT for their core business. We all know that technology is evolving faster and faster. In the meantime, we have *legacy* applications from many years ago. Even new applications that we are building today will be legacy in a few years' time. I actually subscribe to the idea that legacy is any code written before today. Software and technology are transforming the business landscape by enabling new ways for people to connect, share, and collaborate. Furthermore, technology has freed people from constraints and geography. As a result, the world is becoming more complex and even faster, and these new consumption patterns are disrupting well-established business practices. Many organizations are facing the fundamental challenge of modernizing their IT infrastructure. Our old *mental models* and methods are clearly not working anymore. We need new ways of dealing with the need for new solutions. The path toward modernized solutions and improved technology remains mysterious and tricky. Very few legacy organizations have mastered the transformation.

Over the years, I've worked with some of the largest technology organizations in every industry vertical. And whatever excuses you may have, whether they're about technology, complexity, or culture, I claim that I've seen worse. And yet, these organizations have been able to radically transform and improve their outcomes. I want to share some of their learning and achievements with you. IT should not be a place where we spend the majority of time solving the same old problems again and again.

Everything in this book is supported by Agile, DevOps, and Lean principles. In fact, I start pretty much all my client discussions by asking "What do these principles mean to you?" because they are all so ambiguous. I tend

to use the overview picture in Figure 0.2 to make sure everyone understands exactly how I use these principles.

The mission I've set for myself as an advisor to my clients is to make myself redundant. Once my clients are leveraging the principles of Lean, DevOps, and Agile successfully and are working in an efficient way, I can go off and spend more time on implementing exciting solutions. I cannot think of a better goal in life than to make myself redundant and go on to focus on one or two projects instead.

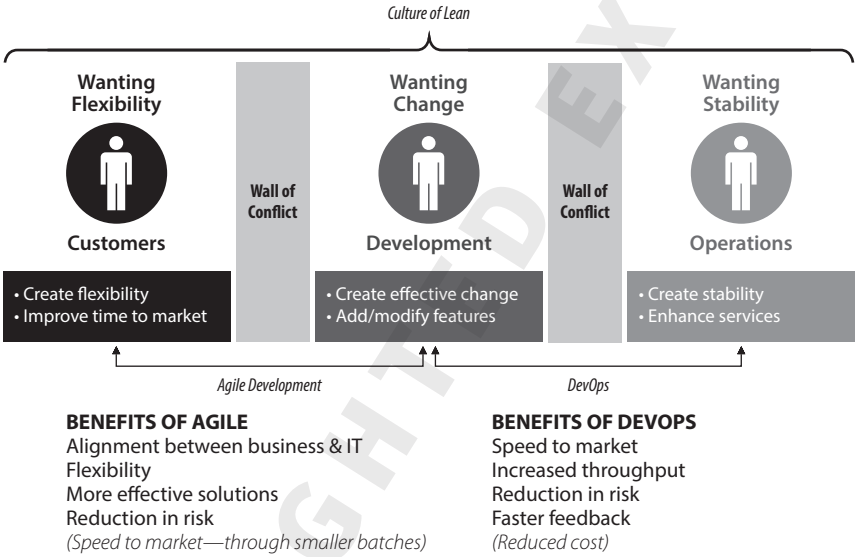


Figure 0.2: Relationship between Agile and DevOps: How the principles Lean, Agile, and DevOps relate to each other

But as long as IT continues to be a place where too many things go wrong and where people struggle every day, I will help make IT a better place to be. I will aim to help the transition with this book, as my team cannot be everywhere. There is more than enough work for all of us. Like all honest answers to complex problems, this is just a starting point. You should feel free to experiment with the recipe, add your own ingredients, and shake it up. Each journey of transformation is contextual, and there is never only one recipe for success.

I hope that with this book I can make it easier for all of you to navigate the challenges ahead. I am sharing my experience and practical exercises that I run with my clients and that you can use in your organization (whether small or large, old or new) to progress in your journey. I am looking forward to seeing you along the journey: at a conference talking about our successes and failures, at a meet-up over drinks, or perhaps on one of my consulting engagements as we solve some challenges together.

I am a developer at heart, and I want to develop cool new applications. Let's transform our industry so that we can all spend more time on the creative side of IT.

Mirco Hering

INTRODUCTION

How We Got Here

> The greatest danger in times of turbulence is not the turbulence;
> it is to act with yesterday's logic.
> —**Peter F. Drucker**, *Managing in Turbulent Times*

Before we get into the meat of the book, I want to take a moment to explain why it is so important to change the way we approach IT delivery. We obviously didn't choose to be in the situation we are in today, where most organizations are struggling to deliver IT in a way that truly supports the business. IT is either too slow or too expensive, or does not deliver the quality that business stakeholders expect, but it's not because we have consciously made bad decisions. When you look around, we are also pretty much in agreement as to what good looks like—the examples of Netflix or Google are used in literally hundreds of presentations about IT. Pretty much every organization is talking about Agile and flexible delivery, automation across the *software delivery life cycle* (SDLC), and leveraging modern architecture patterns like *cloud native application*, *twelve-factor applications*, and *microservices*.

Yet if we look around, we still struggle to find organizations that have mastered this new way of working in IT. Many of the organizations with good examples are relatively young organizations that have grown up as cloud or *internet natives* (companies that were founded with the internet

as the target platform). We even have a term for them: DevOps unicorns. We call them unicorns because they are rare and seemingly unattainable for the average organization with a legacy IT architecture.

One could conclude that the challenge must be with this legacy architecture that organizations try to transform. While this is to some degree correct, I think it has even more to do with the mind-set. Many technology leaders are using ideas that were adopted from a traditional manufacturing context even though IT is inherently different*—they leverage the mental model of manufacturing for an inherently creative process. In traditional manufacturing, we follow a predictable process to produce the same outcome (a product) again and again. In contrast, we never do the same project twice in IT. To support this more creative nature of IT, we need to address three areas to achieve a shift in mind-set: the organizational ecosystem we find ourselves in, the people we work with, and the technologies that support our business. I have structured the book around these three dimensions. Here in the introduction, I will make the case for why the old mental model is not appropriate anymore. I will then spend the body of the book, which is organized into three parts, helping you transform your organization and adapt your mental models.

The three parts of the book are broken down as follows:

1. Part A: Creating the right ecosystem for success (chapters 1–4): How do you create an organization in which modern IT is possible? How do you work with other organizations such as software vendors and system integrators to support a new way of IT delivery? The first "meaty" part of the book goes into these kinds of questions and is really addressed to the CIO and other leaders in IT who set the strategy of the organization.

* Some DevOps books, like *The Phoenix Project*, leverage a modern manufacturing model that is different to the traditional model I am referring to.

2. Part B: The people dimension (chapters 5–8): Nothing happens in organizations without people. How little we sometimes consider this is reflected in our usage of the term "resources" instead of people. Part B of the book will focus on people and what we can do to empower people to bring their best to work each day, which will, in turn, make your organization more successful.
3. Part C: Technology in the driver's seat (chapters 9–12): There is no doubt in my mind that the recent advances in technology have made some of the new ways of working possible and are forcing leaders in the industry to rethink the way to deliver IT solutions. I will talk about some of the key trends and how best to leverage these. Technology is evolving ever more quickly, so I will refrain from using specific names of tools, vendors, or methods. For some help navigating this space, I will, however, provide a list of resources that I hope remains available and updated over time on the internet.

For each chapter in the main section of the book, there is a discussion of the topic in which we explore the problem space and what possible solutions and approaches can be used. I aim to keep the chapters short so that you can consume them in bite-size pieces.

At the end of each chapter, I provide exercises in the spirit of "how to try this at home," in which I present some practical steps for you to take to leverage what I have said in the chapter for your organizations. This will sometimes include templates, some questions to ask yourself or your organizations, or a step-by-step guide. It will not replace having someone with experience as an advisor but will provide you with some first steps to help you along the transformation journey.

I will also (somewhat self-indulgently) provide links to some of my blog posts along the way, usually when I have written more extensively

about a topic but don't think it fits into the flow of this book. Feel free to check out those posts.

I've included a conclusion to wrap up the book and leave you with some general recommendations on how to take the next step. In addition, I've added my personal recipe for dealing with the demands of the fast speed in our industry. In all, the information found in this book should leave you feeling prepared to begin the transformation into a modern IT organization.

Finally, I've included an appendix that goes into more detail of the factory analogy I discuss throughout the book. If you're not familiar with it, you may want to read this appendix first, before moving into the meat of the book. For those who are well versed in the analogy, the appendix will provide a deeper understanding of it and help to further your thinking in this area. Let's quickly look at why a traditional manufacturing mind-set stands in the way of achieving the speed and reliability of IT delivery we are aiming for with DevOps.

Manufacturing Principles Don't Apply Easily to IT Delivery

In recent years, modern manufacturing has been the inspiration for many positive trends, like Lean, systems thinking, and theory of constraints. The Toyota manufacturing system, in fact, was one of the inspirations behind Agile and DevOps ideas. Yet many managers operate with a mental model that is more inspired by a legacy view of manufacturing—the image of manufacturing from the Henry Ford era. This factory model is based on the idea that we create a production process that allows us to put the intelligence into the process; then we can use less-skilled people to achieve a high-quality outcome. This is not true for IT work, where collaboration and creativeness are required to achieve the best outcomes. I believe that the principles of legacy manufacturing are causing many of the problems we currently see in legacy IT: rigid processes with a command-and-control

mind-set; bloated, functionally specialized organizational structures; and large amounts of handovers that reduce the flow of work and increase the opportunities to introduce defects.

I chose the phrase "not a factory anymore" for my blog as a rallying cry for the change in thinking that IT executives need to make if they want to successfully transform their organization. The word "factory" in this phrase stands for the kind of factory Henry Ford represents, where mass manufacturing takes place with highly specialized people working on assembly lines to do a specific job and produce a mass product with little to no customizations.

Legacy manufacturing was based on the invention of the assembly line at Ford and the work that Frederick Taylor did on scientific management.† Building a factory was an expensive and time-consuming business. The work itself was optimized so that workers could become highly specialized at one specific task and do that task extremely efficiently. To reduce cost, a manufacturer had few choices: he could automate more by investing in better machines, he could change the material going into the product, and he could move his factory somewhere where the worker is paid less. Most changes to the product or production process would be a significant endeavor, as they required the purchase of new machines, reconfiguration of machines, teaching of new processes to all the workers, or a change in supply chains for the product material.

The product itself was not customizable (Henry Ford allegedly said about the Model T, "You can have any color as long as it's black"), so it was very easy to compare output and cost of the product. If you changed the production process or used different material for the product, you could evaluate the result scientifically (which is the idea that Taylor's work was

† Frederick Taylor is considered the founder of scientific management, which he implemented especially in the steel industry. He published *The Principles of Scientific Management* in 1911.

inspired by). One can argue that most of manufacturing today still allows you to do these kinds of measurements of cost and productivity, as most products are mass produced. This is the kind of manufacturing example that most economic education and, therefore, management is based upon. I went to university over fifteen years ago and did an MBA recently, so I know firsthand that most of the teaching of IT processes is still influenced by this model. It is, after all, relatively easy to understand and to control, which makes scientific treatment of the process possible.

You will see that IT in the early days had a lot in common with a manufacturing business. Let's put ourselves back in the 1990s at the IT department of a car company. (I like this scenario because I used to work in such a department during my school holidays and, based on that experience, was inspired to pursue a career in IT.) IT worked pretty similarly and had undergone similar trends as manufacturing had. Building a new IT system was an expensive and time-consuming business. The work itself was optimized so that workers could become highly specialized as testers; Java developers, SAP configurators, or operations engineers. Each one could focus on one specific task and do that extremely efficiently within an overall *Waterfall*-inspired SDLC process.

To reduce cost, the IT department had a few choices: they could automate more by investing in better tools, they could change the kind of software being used (for example, from custom Java to SAP to leverage existing functionality instead of building it), and later on, they could shift development or other functions somewhere where the worker is paid less. Most changes to the product after the design phase would be a significant endeavor, as this required changes to the contract with software vendors or significant rework for the existing team.

The product itself was pretty standard: an enterprise *resource planning* (*ERP*) *system* or a *customer relationship management* (*CRM*) *system* using "best practices" coming from packaged solutions. And while it was never really the same product twice due to some customization, the over-

all effort was relatively comparable (e.g., an SAP implementation at one car company to the SAP implementation of another).

You see, all in all, while IT was somewhat different, there was enough similarity that practices borrowed from manufacturing worked. And they worked for a long time. But IT has changed a lot, and we haven't updated the way we manage it accordingly.

Using the factory analogy in IT conjures up images of people sitting in their cubicles and actioning work packages, with one person translating a requirement into a design, then handing it over to another person who writes a bit of code, then on to the next person who tests it—and in all of this, no one talks to each other. It's all done mechanically, as with people on an assembly line. (This is reminiscent of Charlie Chaplin in the movie *Modern Times*.) The world has changed, and a significant change enabler has entered the stage: the reduction in transaction, or setup, costs for IT.

The Concept of Transaction Costs and Batch Size Is Central for the Shift in Approach

Transaction costs (sometimes called setup costs) are the costs required to make a change in the production mechanism to either produce a different product or make changes to the way the same product is being produced. Transaction costs determine the optimal batch size of the product to achieve an economic outcome, as the holding costs are difficult to influence. How transaction costs influence the optimal batch size is illustrated in Figure 0.3.[‡]

The optimal batch size is determined by the holding costs and the transaction costs (higher holding costs drive smaller batch sizes and higher

[‡] Stefan Thomke and Donald Reinertsen have probably written the best explanation of this in their May 2012 *Harvard Business Review* article, "Six Myths of Product Development." Although they talk about product development rather than IT explicitly, for all means and purposes, IT delivery is the delivery of an IT product.[1]

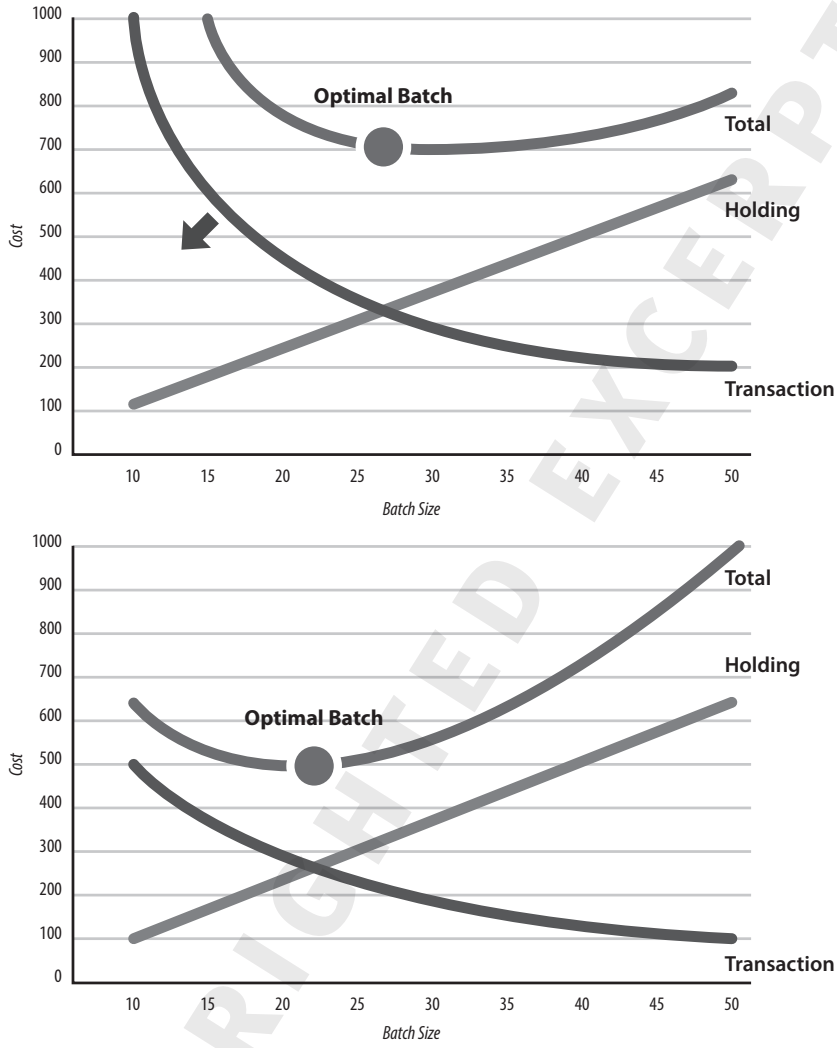


Figure 0.3: Relationship between transaction costs and batch size: Reducing transaction costs allow for smaller batch sizes

transaction costs drive larger batch sizes). In IT, the holding costs are a combination of the increasing cost of fixing a problem later in the life cycle and the missed benefit of features that are complete but not in production yet. These two factors don't change much with modern DevOps practices; what changes are the transaction costs.

As manufacturing has changed over the years, there has been a focus on reducing the transaction costs so that more differentiated products can be produced. This has cumulated in mass customization and 3-D printing. In IT, both Agile and DevOps have tried to achieve the same reductions.

In the past, the transaction costs were high for IT projects. You had to order hardware and install all the *middleware*, and the development of your customized software product included a lot of manual effort–intensive tasks, such as deploying code changes, regression testing the solution, making changes to the environments (for example, an MVP takes less than three months rather than 6–12 months; incremental releases take days/weeks instead of months).

This has changed now that the transaction costs have been significantly reduced by cloud-based infrastructure, anything as a service *(XaaS)*, and DevOps-inspired automation across the SDLC, which have made much smaller batch sizes possible.

And there is a real business benefit in small batch sizes. Small batch sizes allow us to invest smaller amounts to validate a business idea and make fewer large bets. The best analogy for this comes from long-time product developer Don Reinertsen:[2] Imagine a lottery where you get a prize if you guess a three-digit number correctly. One option is to give me five dollars and guess the three-digit number; I then tell you whether you have won the $1,000 prize. Alternatively, you can give me two dollars and guess the first digit. In return, I tell you the first digit of the winning number, and you can decide to pay another $2 for the next digit and again the same for the third digit. Would you choose the first game or the second game? I think it is clear that the second game provides you with a better outcome.[§] The feedback from the smaller batch

[§] Scenario 1: -3.995 = (-5×999/1000)+1/1000×1000. Scenario 2: -1.214 = (-2×9/10) + (4×1/10×9/10) + (-6×1/10×1/10×9/10) + 1000×1/10×1/10×1/10.

size provides you information to decide on further investment or not. This is exactly what customer feedback will allow you to do when you work on smaller increments of functionality and smaller batch sizes in your IT projects.

We also want smaller batches for a couple of other reasons. Smaller batch sizes reduce the complexity in IT delivery, as there is a smaller amount of changes to consider in testing and during *go-live*. In manufacturing, you can scale to larger batch sizes more easily, as you can run the same process for larger batches without the inherent complexity penalties that appear when trying to scale your IT solution. Smaller batch sizes in IT allow us to test the direction the product is developing, with customer feedback leveraged to direct the creative nature of IT products.[‖] In manufacturing, it is much more difficult to iteratively create and validate a product; just imagine building one product in a factory (e.g., a car), selling it, and then using feedback to build the next one, and so forth. The delays and costs would explode quickly. Eric Ries's Lean Startup method[3] has been adapted for physical goods by FastWorks with great results (reduced cost, increased speed),[4] but the preference for larger batch sizes will remain a feature of manufacturing (at least until 3-D printing of everything has become the norm and reduces the setup/transaction costs).[5]

With smaller batch sizes comes a very different economic model for governance to minimize the required overhead from management. In other words, the technical transaction cost has been reduced, but the management-related and architecture-related transaction costs have not been reduced at the same time. This is because we still follow the same ideas and principles that are inspired by manufacturing. With this book, I want to help you change this.

[‖] In IT, due to the trial-and-error nature of product-development work, smaller batches have the benefit of reduced risk and faster feedback. This is different from manufacturing, where efficiency and productivity are measurable.

In the appendix, I provide a closer look at some ideas and principles that have been adopted by IT from manufacturing and whether or not they are still applicable.

I hope this book will help you all to create IT delivery organizations that don't resemble Charlie Chaplin's *Modern Times* but are engaging workplaces where IT people work with business stakeholders to build meaningful solutions for customers. As my Accenture colleague Mark Rendell has said, "DevOps is not about making IT efficient. It's about making business effective through IT."[6]

Let's jump right in and get started with the transformation.

PART A

Creating the Right Ecosystem

In my role as a consultant, I have to explain to technology leaders frequently that knowledge of what good looks like and the best intentions unfortunately don't always result in the best outcome. The ecosystem that the company leadership creates plays a huge role in how successful a transformation can be. The ecosystem you are likely working in has been created within the context of your legacy—the things that have been in the past and the systems that were built previously.

The concept of legacy is not just tied to your technologies and applications. As was explained in the introduction, there is a legacy mind-set inspired by a command-and-control culture, which I am trying to change with this book. So, what does it take to define a roadmap to guide this shift? How do you find the right technologies to work with? And what makes a good partner who can join you on this journey away from legacy toward the new?

With the toolkit provided in part A of this book, you will be a significant step closer to creating an ecosystem that allows IT to thrive and transform. It describes the high-level considerations for creating an ecosystem that supports the transformation to a modern IT development organization. I will cover the transformation journey, multispeed IT, your application portfolio, working with legacy, choosing software packages, and finding the right delivery partners. I will also discuss how my team works together with our clients at the leadership level to explore how the ecosystem needs to change for the benefit of both organizations, I will provide some of that analysis toolkit with activities that you can run yourself. Together, this information will help you understand how to leverage your legacy as an enabler rather than as a hindrance.

CHAPTER 1

The Roadmap to Transformation

> Alice asked the Cheshire Cat, who was sitting in a tree, "What road do I take?" The cat asked, "Where do you want to go?" "I don't know," Alice answered. "Then," said the cat, "it really doesn't matter, does it?"
>
> —**Lewis Carroll**, *Alice's Adventures in Wonderland*

Many challenges in your IT delivery process are caused by bottlenecks, which lengthen the time before you get meaningful feedback that you can, in turn, use to improve your process and products. Making work visible is one of the most powerful ways to identify bottlenecks, yet IT is mostly dealing with invisible work: there is no visible stock showing how much product a team or facility has created, there is no warehouse that indicates how much product is available but not in use, and there is no physical process that you can follow to see how an output is being created from the inputs. This leads to an interesting situation: while most people working in manufacturing have a rough idea of how their product is being created, in IT, the actual process is a lot less known. And I mean the real process, not the one that might be documented on some company web page or in some methodology. Yet without that visibility it is difficult to improve the process. So, one absolutely crucial task for any IT executive is to make the process visible, including status and measures like quality and speed. In this chapter, we leverage value stream maps to make work

visible and jump-start the transformation with an initial roadmap and governance approach.

Making the IT Process Visible

I like to start any DevOps consulting activity with a *value stream mapping exercise*. The reason is quite simple: it is the most reliable exercise to align everyone in the organization and my team to what the IT process looks like. You could look at the methodology page or some large Visio diagrams for the IT delivery process, but more often than not, reality has evolved away from those documented processes.

I have outlined the process to run such an exercise at the end of the chapter so that you can try this too. In short, you are bringing representatives from all parts of the organization together in a room to map out your current IT delivery process as it is being experienced by the people on the ground and, perhaps more importantly, reveals areas within the system that can be improved. I suggest that you engage an experienced facilitator or at least find someone unbiased to run the meeting for you.

Ideally, we want to be able to objectively measure the IT process in regard to throughput, *cycle time*, and quality. Unfortunately, this is often a work-intensive exercise. Running a value stream mapping exercise every three to six months (depending on how quickly you change and improve things) will give you a good way to keep progress on the radar while investing just a few hours each month. It will highlight your current process, the cycle time, and any quality concerns. You want to make the result of the exercise visible somewhere in your office, as that will help focus people on improving this process. It will act as a visible reminder that improving this process is important to the organization.

Once you have a good understanding of the high-level IT process and the areas that require improvement, you can then create a first roadmap for the transformation.

Creating a First Roadmap

Roadmaps are partly science and partly art. Many roadmaps look similar at the high level, yet on the more detailed level, no two people create the exact same roadmap. The good news is that there is no *one* right answer for roadmaps anyway. In true Agile fashion, it is most important to understand the direction and to have some milestones for evaluating progress and making IT visible. Many things will change over time, and you will need to manage this. There are a few guidelines on how to create a good roadmap for this transformation.

Based on the value stream map of your IT delivery process, you will be able to identify bottlenecks in the process. As *systems thinking*, *theory of constraints*, and *queuing theory* teach us, unless we improve one of the bottlenecks in the process, every other improvement will not lead to a faster outcome overall. This is important, as sometimes we spend our change energy on "shiny objects" rather than focusing on things that will make a real difference. One good way to identify bottlenecks is to use the value stream mapping exercise and let all stakeholders in the room vote on the problems that, if addressed, will make a real difference to overall IT delivery. The wisdom of the crowd in most cases does identify a set of bottlenecks that are worth addressing.

There are two other considerations for your roadmap to be a success: flow and speed of delivery rather than cost and quality. A focus on flow is the ultimate systems thinking device to break down silos in your organization. In the past, the "owner" of a function, like the testing center of excellence or the development factory, ran improvement initiatives to make its area of influence and control more effective. Over time, this created highly optimized functions for larger batch sizes to the detriment of the overall flow of delivery. Flow improves with small batch sizes.

There are usually three ways to evaluate IT delivery: speed, cost, and quality. Traditionally, we focused our improvements on cost or quality,

which, in turn, often reduced the speed of delivery. If you evaluate your IT delivery by just looking to improve quality, you often introduce additional quality gates, which cost you more and take longer to adhere to. If you evaluate your IT function based on reduced cost, the most common approaches are to push more work to less experienced people or to skip steps in the process, which often leads to lower quality and lower speed due to rework. Focusing on cost or quality without considering the impact on flow is therefore an antipattern for successful IT in my experience.

In contrast, focusing on speed, specifically on bottlenecks that prevent fast delivery of really small batches, will bring the focus back to the overall flow of delivery and hence improve speed of delivery in general (even for larger batches), leading to improvements in quality and cost over time. It is impossible to achieve higher speed if the quality is bad, as the required rework will ultimately slow you down. The only way to really improve speed is to automate and remove unnecessary steps in the process. Just typing faster is unlikely to do much for the overall speed. So, speed is the ultimate forcing function for IT. I have been in transformations with clients where cost was reduced but the overall delivery experience continued to be bad for business stakeholders. I have also seen a lot of quality improvement initiatives that stifled IT delivery and nearly ground it to a halt. I have yet to see the same problem with improvement initiatives that evaluate based on speed.

Two words of caution when it comes to speed: The first one is really not that bad of a problem. You can obviously "game" the speed evaluation criteria by breaking work down further and delivering smaller batches, which can be delivered faster. While this does not result in a like-for-like comparison of the speed between batches, it is still a win for the organization, as smaller batches are less risky. The second warning is that people might look for shortcuts that increase risk or reduce quality. To prevent this, you need to continue to look for quality measures on top of speed to make sure that quality is not dropping as speed increases. To evaluate

for speed, you will look at work coming through your delivery life cycle, and the process of measuring it will make it more visible to you. Good measures for speed are cycle time for work items (cycle time = time from work item approved to work item completed and available in production) or volume of work delivered per time period.

As Figure 1.1 demonstrates, your overall transformation roadmap will likely have milestones focused on different functions and capabilities (e.g., automated regression testing available, lightweight business case introduced), which makes sense. However, there is another dimension, which is the coverage of applications and technologies. In the next chapter, I will explain how to do an application portfolio analysis that allows you to identify sets of applications that you uplift as part of the transformation. Your roadmap should include prioritized sets (often called waves) of applications, as any organization at scale will not be able to uplift absolutely everything. You shouldn't anyway, as some applications might not be worth the effort and cost of an uplift.

One last comment on the transformation roadmap: many capabilities and changes require a significant amount of time to implement. Unfortunately, organizations are not very patient with change programs, so you need to make sure that you build in some early and visible wins. For those early and visible wins, all other rules do not apply. They can be with applications that are not critical for the business or in areas that are not part of a bottleneck. The goal of those wins is to keep the momentum and allow the organization to see progress. You should see these as being part of the change-management activities of the transformation. Of course, ideally, the early and visible wins are also in one of the priority areas identified earlier.

Transforming your IT organization will take time. In Figure 1.2, you can see a transformation blueprint that is common among clients I work with. The pattern shown here is something I have seen again and again. It starts with adopting Agile and then realizing that Agile without DevOps

practices (like test automation, deployment automation, etc.) does not allow you to speed up as much as you were hoping for. As you adopt DevOps and become faster, you start to realize the organizational bound-

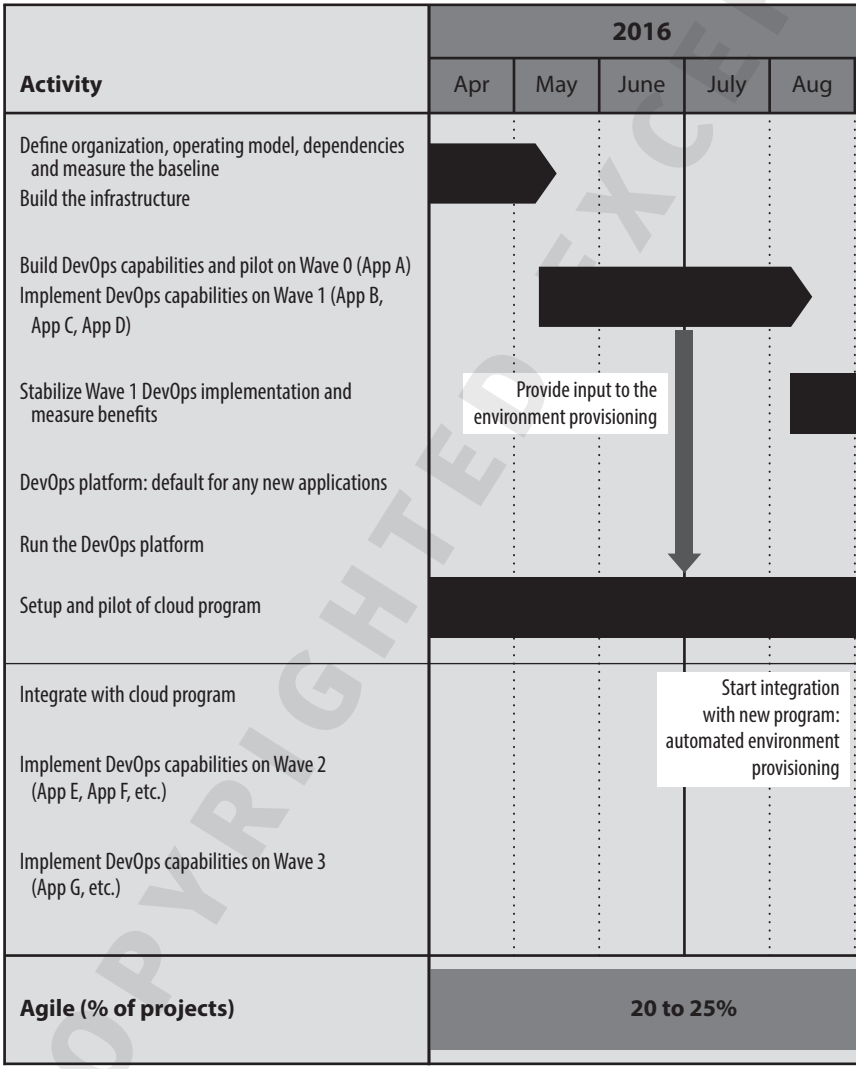


Figure 1.1: Transformation roadmap: Example showing waves of applications and capabilities

aries and speed bumps that are embedded in the operating model, which require some real organizational muscle and executive support to address. Don't be discouraged if things don't change overnight.

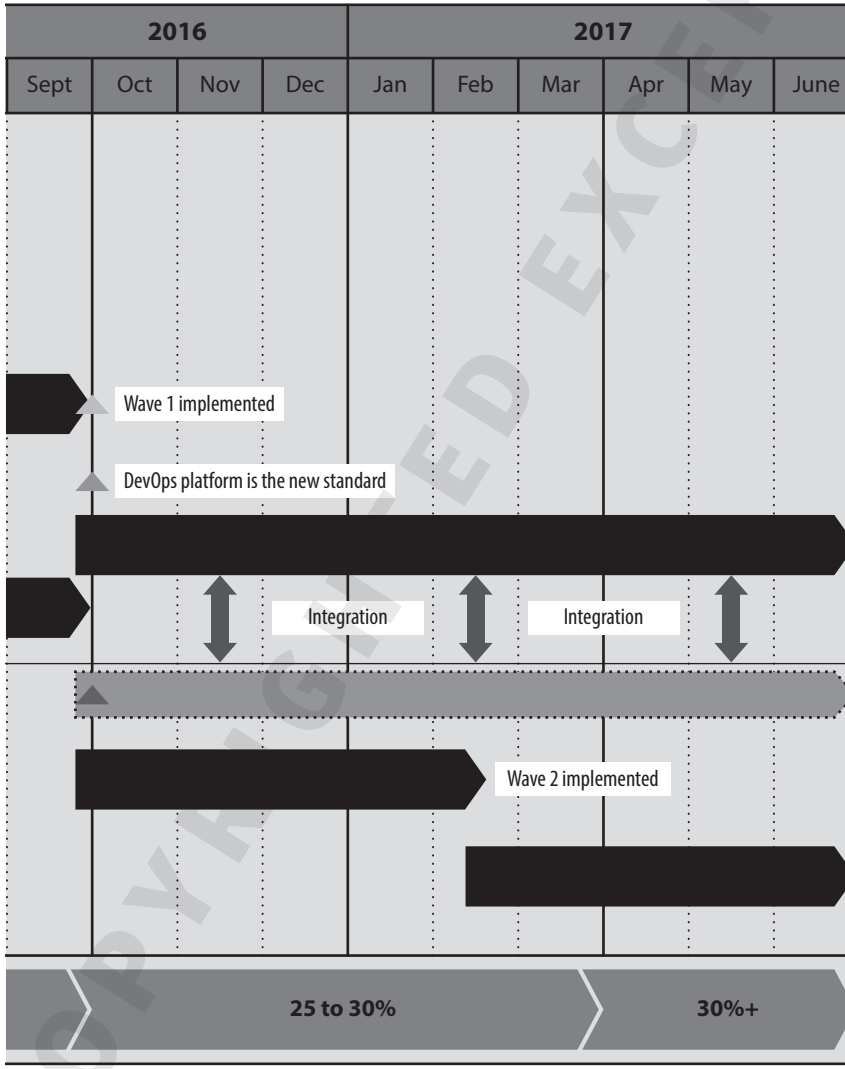


Figure 1.1, cont.

Governing the Transformation

As mentioned earlier, the roadmap is important, but without appropriate transformation governance, it is not going to get you much success. All too often, transformations get stuck. It is not possible to foresee all the

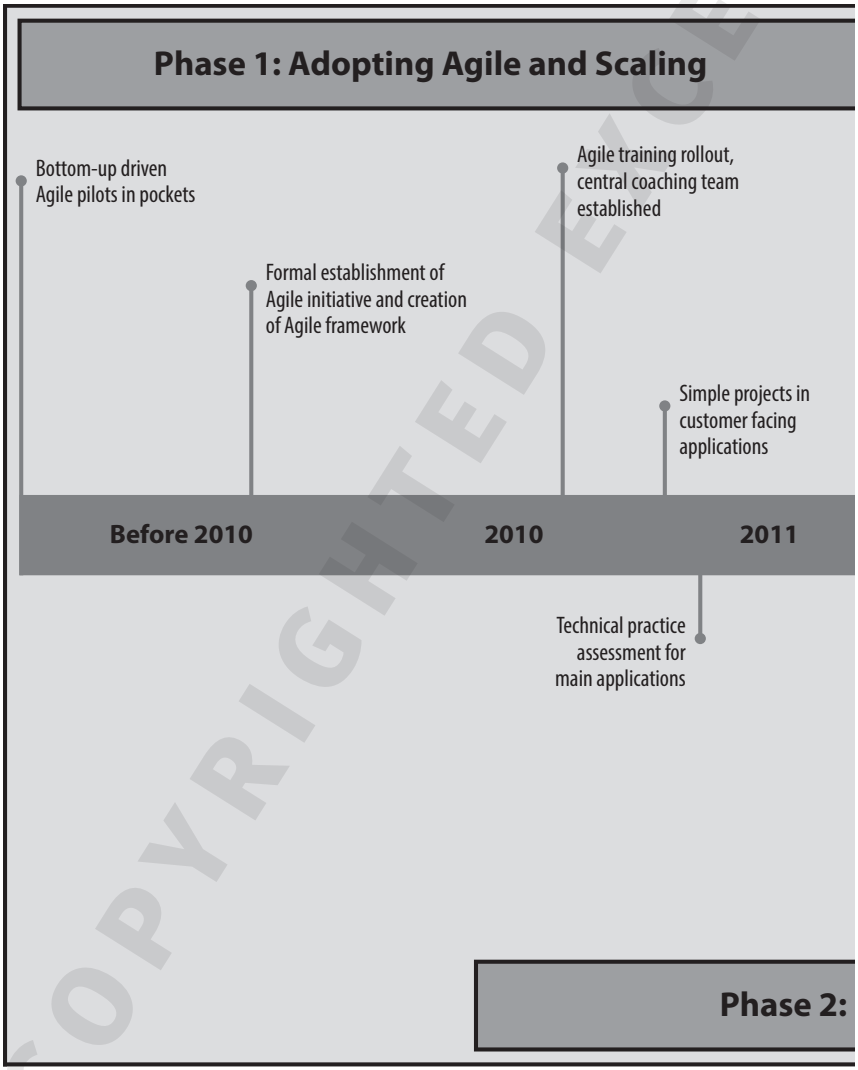


Figure 1.2: Common transformation blueprint: Changing your organization takes time as you adopt different methods

challenges that will hinder progress, and without appropriate governance that finds the right balance between discipline and flexibility, the transformation will stall. Transformation governance makes the progress of the transformation visible and allows you to steer it. It's different from the normal IT delivery governance that you run for your delivery initiatives (e.g.,

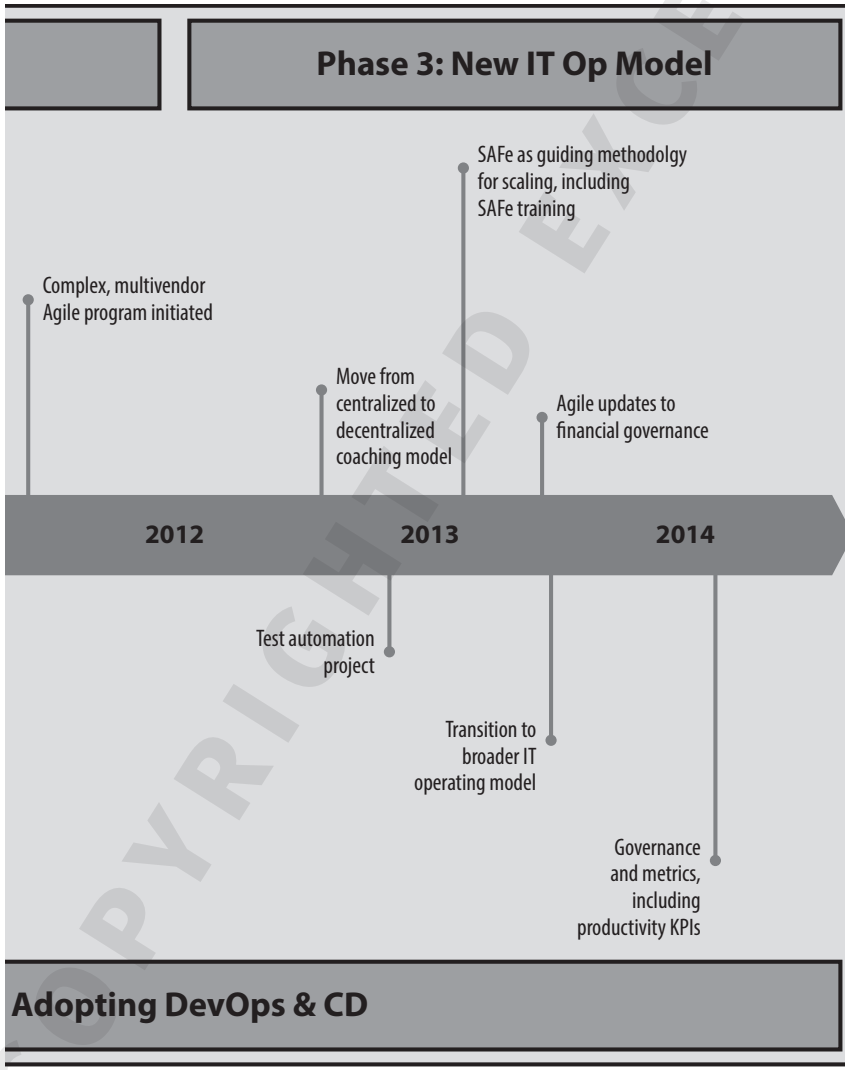


Figure 1.2, cont.

change review boards). In a meeting with a number of transformation change agents and consultants at the 2015 DevOps Enterprise Summit, we tried to identify what it takes to be successful when adopting DevOps. We all had different ideas and were working in different organizations, but we could agree on one thing that we believed was the characteristic of a successful organization: the ability to continuously improve and manage the continuous improvement process.

This continuous improvement and the adaption of the roadmap are the largest contributors to success in transforming your IT organization. DevOps and Agile are not goals; hence, there is no target state as such.

What does successful transformation governance look like? Governance covers a lot of areas, so it is important that you know what you are comparing against as you make progress with your transformation. This means you need to establish a baseline for the measures of success that you decide on before you start the transformation. Too many transformations I have seen spent six months improving the situation but then could not provide evidence of what had changed beyond anecdotes such as "but we have continuous integration with *Jenkins* now." Unfortunately, this does not necessarily convince business or other IT stakeholders to continue to invest in the transformation. In one case, even though the CIO was supportive, the transformation lost funding due to a lack of evidence of the improvements.

If you can, however, prove that by introducing continuous integration you were able to reduce the instances of build-related environment outages by 30%, now you have a great story to tell. As a result, I strongly recommend running a baselining exercise in the beginning of the transformation. Think about all the things you care about and want to measure along the way, and identify the right way to baseline them. (I've provided some examples in Table 1.1.) I will talk a bit more about how to measure metrics when we talk about delivery governance later in this chapter.

Metric	Definition	Measurement
Release Cycle Time	The average time it takes to approve a work package (user story, feature, set of requirements) and release it	Usually measured as the time difference between work item states in your work tracking system
Cost of Release	The effort it takes to release new functionality, measured as effort for all release activities performed for go-live (a variation of this only counts effort outside of business hours)	Typically based on timesheets
Regression Duration	Time it takes to validate that a change has not caused regression	The time between deployment and the "all clear" from either an automated or manual validation
Production Availability	Percentage production is available to perform the right service	Measured by a percentage of time production is functionally available or percentage of successful transactions
Mean time to Recovery	Time it takes to rectify any production issue	Measured from time of occurrence until full user functionality is achieved
Longevity of Teams	The average duration teams stay together	Measured as months before teams get disbanded and restructured for new projects

Table 1.1: Baseline metrics: These metrics have proven to be successful in guiding transformations

The other important aspect of transformation governance is creating flexibility and accountability. For each improvement initiative, as part of the roadmap, you want to leverage the scientific method:

- Formulate a hypothesis, including a measure of success.
- Baseline the measure.
- Once the implementation is complete, evaluate the result against the hypothesis.

Some things will work, some won't; and during governance, you want to learn from both. Don't blame the project team for a failed hypothesis (after all, it should have been we, as leaders, who originally approved the investment—so, who is really to blame?). You should only provide negative feedback where the process has not been followed (e.g., measures were not in place or results were "massaged"), which prevents you from learning.

As you learn, the next set of viable improvement initiatives will change. Your evaluation criteria of the initiatives you want to start next should be guided by:

- previous learnings,
- the size of the initiative following a *weighted shortest job first* (WSJF) approach,
- and how well the team can explain the justification for the initiative.

Don't allow yourself to be tempted by large business cases that require a lot of up-front investments; rather, ask for smaller initial steps to validate the idea before investing heavily. You should keep an eye on the overall roadmap over time to see that the milestones are achievable. If they are not anymore, you can either change the amount of improvement initiatives or, when unavoidable, update the roadmap.

In the transformation governance process, you want representation of all parts of the organization to make sure the change is not biased to a specific function (e.g., test, development, operations). Governance meetings should be at least once a month and should require as little documentation as possible. Having the transformation team spend a lot of time on elaborate PowerPoint presentations for each meeting is not going to help your transformation. Ideally, you will look at real-time data, your value stream map, and lightweight business cases for the improvement ideas.

Making IT Delivery Visible

Talking about making things visible and using real data, it should be clear that some of the DevOps capabilities can be extremely useful for this. One of the best visual aids in your toolkit is the deployment pipeline.* A deployment pipeline is a visual representation of the process that software follows from the developer to production, with all the stages in between. This visual representation shows what is happening to the software as well as any positive or negative results of it. I have provided an example in Figure 1.3 from one of the solutions that I often work with. You can see how the different stages of the life cycle and the associated activities are represented in the pipeline view.[1] This deployment pipeline provides direct insights into the quality of your software in real time. You might choose to provide additional information in a *dashboard* as an aggregate or to enrich the core data with additional information, but the deployment pipeline provides the core backbone. It also creates a forcing function, as all the steps are represented and enforced, and the results can be seen directly from the dashboard, which reduces the chance of people doing

* Gary Gruver wrote a whole book, *Starting and Scaling DevOps in the Enterprise*, about the deployment pipeline as a means to drive the transformation.

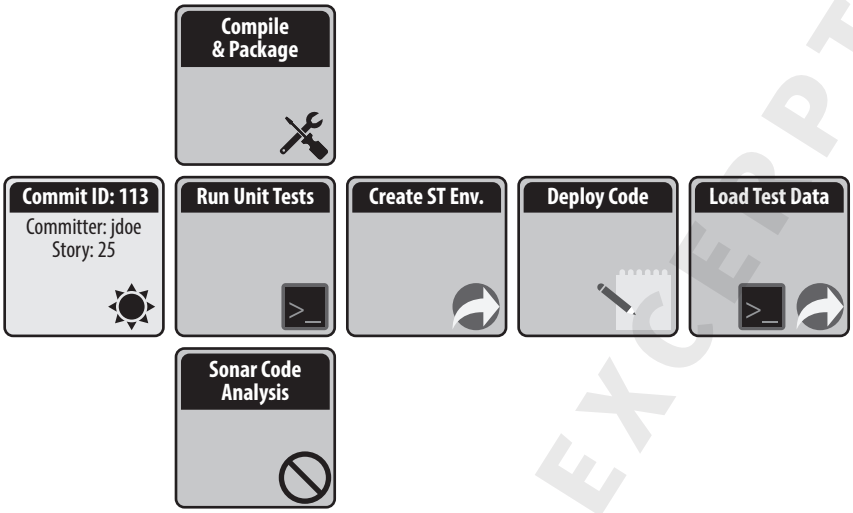


Figure 1.3: Deployment pipeline example: Accenture DevOps platform provides a look into the deployment process

things that are not visible. Any improvements and process changes will be visible in the deployment pipeline as long as it remains the only allowed way to deliver changes. Where you don't have easy access to metrics you can also add steps to each stage to log out metrics for later consumption in your analytics solution.

Having an analytics solution in your company to create real-time dashboards is important. Most companies these days either use a commercial visualization or analytics solution or build something based on the many *open source* options (like Graphite). The key here is to use the data that is being created all through the SDLC to create meaningful dashboards that can then be leveraged not only during the transformation governance but at any other point in time. High-performing teams have connected their DevOps tool chain with analytics dashboards and

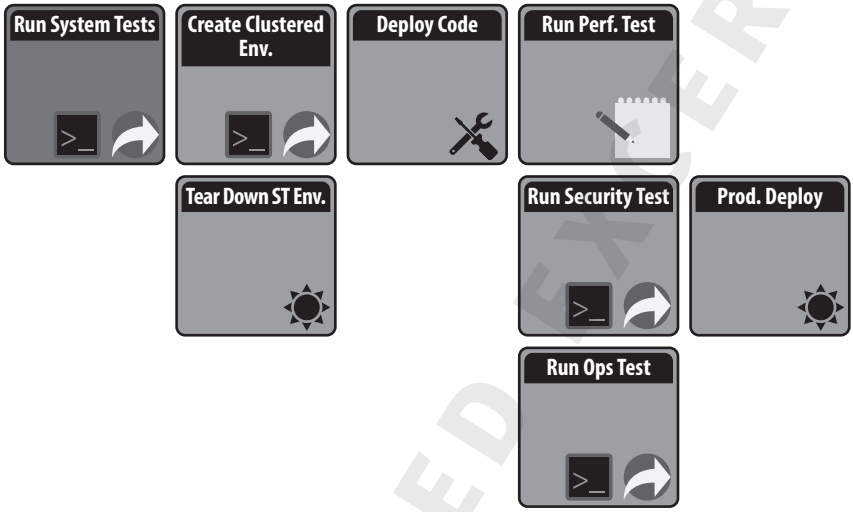


Figure 1.3, cont.

it allows us to see important information in real time. For example, we can see how good the quality of a release is, how the quality of the release package relates to post-deployment issues, and how much test automation has improved our defect rate in later phases of the SDLC.

Governing IT Delivery

IT governance is, in my view, one of the undervalued elements in the transformation journey. Truth be told, most governance approaches are pretty poor and achieve very little of the outcome they are intended to achieve. Most governance meetings I have observed or been part of are based on red/amber/green status reports, which are subjective in nature and are not a good way of representing status. Furthermore, while the

criteria for the color scheme might be defined somewhere, it often comes down to the leadership looking the project manager in the eyes and asking what she really thinks. Project managers from a *Project Management Institute* (*PMI*) background use *cost performance indicator* (CPI) and *schedule performance indicator* (SPI), which are slightly better but rely on having a detailed and appropriate project plan to report against. I argue that most projects evolve over time, which means that if you're preparing a precise plan for the whole project, you plan to be precisely wrong.[†]

Additionally, by the time the status report is presented at the meeting, it is—at best—a few hours old. At worst, it's an unconscious misrepresentation, because so many different messages needed to be aggregated and the project manager had to work with poor inputs. Too often, a status report remains green over many weeks just to turn red all of a sudden when the bad news cannot be avoided anymore. Or the status that moves up the chain of command becomes more and more green the higher you get because everyone wants to demonstrate that he is in control of the situation. Remember, one of our goals with status reports is to make IT work visible, and we're not doing that in a meaningful way if the information we're presenting isn't a factual representation of our processes and progress.

What you should use in your governance of delivery are objective measures, such as number of working features, time to recover from an incident, cycle time for features to be delivered, and stories/features delivered and accepted per iteration. Those provide more meaningful ways to evaluate progress and quality. This information as well as other metrics should not be manually collected; you should be able to get the information from a real-time system or dashboard. Some metrics can be taken from your delivery pipeline, but many require additional data points from other systems (e.g., your Agile life-cycle-management tool). The "color commentary" is provided by the project team, and it can

[†] Inspired by Carveth Read: "It is better to be vaguely right than exactly wrong."[2]

either be overlayed as a discussion thread or annotated snapshots can be created for your governance meeting (see the example of an annotated burnup in Figure 1.4).

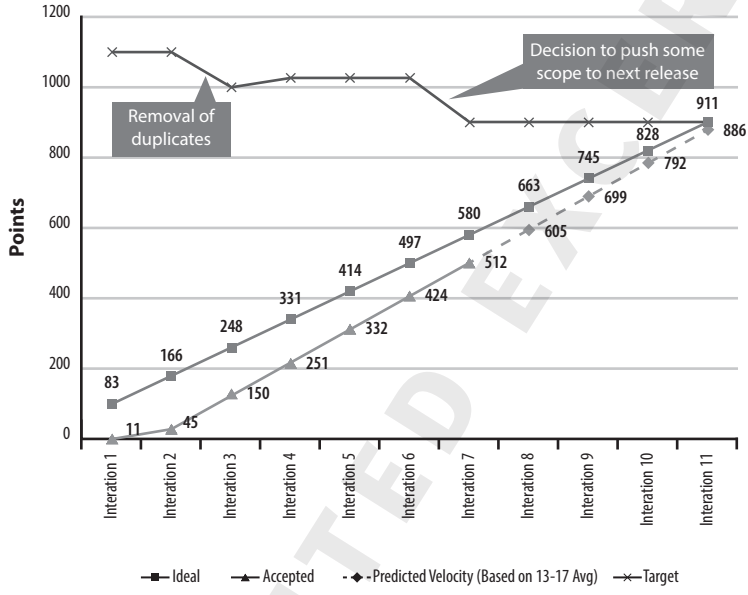


Figure 1.4: Annotated burnup chart: Burnup charts provide an annotated status of the project

The same is true for the metrics that you use during delivery governance. There is really no excuse for not having all the data available for each stage of your IT delivery process. It surprises me that we use IT to build great analytics solutions for our businesses, yet we don't utilize the same powerful solutions within IT to improve our area of the organization. Manual gathering of metrics is not acceptable when it is so easy to build steps into your process that automatically log the data. In the very worst case, you build a little bit of automation for each step that dumps the data out into a common logging format. I've had to do this many times, as most tools used during the SDLC do not expose the captured data in ways that they can easily be consumed. Rather, most tools assume that you can

rely on the built-in reporting functionality, which often is not the case. You want to be able to relate data from one tool to another (and probably across tool vendors), so you will have to do some custom tooling to make sure all the data from your SDLC is available for your analysis. This relatively small investment will pay back in spades over time.

With all this data, you can easily be overwhelmed, as the "data exhaust" of IT delivery and operations is huge and comparable with other big data scenarios. The key here is to focus again on the bottlenecks. Find the metrics that represent the bottlenecks and keep a close eye on them as you try to improve. Once the primary problem has improved sufficiently, your focus will shift, and new metrics will become important. Thankfully, if you create a metric measurement framework well (including dashboard and data preparation), you will have all the needed information at your fingertips. Dashboards, as described earlier, are a powerful way to aggregate information and make it consumable.

The Lean Treatment of IT Delivery

In transformations, our focus is often on technologies and technical practices, yet a lot can be improved by applying Lean to IT delivery governance. By IT delivery governance, I mean any step of the overall IT delivery process where someone has to approve something before it can proceed. This can be project-funding checkpoints, deployment approvals for test environments, change control boards, and so on. During the SDLC there are usually many such governance steps for approvals or reviews, which all consume time and effort. And governance processes often grow over time. After a problem has occurred, we do a post-implementation review and add another governance step to prevent the same problem from happening again. After all, it can't hurt to be extra sure by checking twice. Over time, this creates a bloated governance process with steps that do not add value and diffuse accountability. I have seen deployment

approval processes that required significantly more time than the actual deployment without adding value or improving quality. I find that some approval steps are purely administrative and have, over time, evolved to lose their meaning as the information is not really evaluated as it was intended. The following analysis will help you unbloat the process.

I want you to take a good, hard look at each step in your governance process to understand (a) how often a step actually makes an impact (e.g., an approval is rejected), (b) what the risk is of not doing it, and (c) what the cost is of performing this step.

Let's look at each of the three aspects in more detail:

1. When you look at approvals and review steps during the SDLC, how often are approvals not given or how often did reviews find issues that had to be addressed? (And I mean serious issues, not just rejections due to formalities such as using the wrong format of the review form.) The less often the process actually yields meaningful outcomes, the more likely it is that the process is not adding a lot of value. The same is true if approvals are in the high ninetieth percentile. Perhaps a notification is sufficient rather than waiting for the approval, which is extremely likely to come anyway. Or perhaps you can cut this step completely. I worked with one client whose deployment team had to chase approvals for pretty much every deployment after all the preparation steps were complete, adding hours or sometimes days to the deployment *lead time*. The approver was not actually doing a meaningful review, which we could see from the little time it took to approve once the team followed up with the approver directly. It was clearly just a rubber-stamping exercise. I recommended removing this approval and changing the process to send information to the approver before and after the deployment, including the test results. Lead time was significantely reduced, the approver

had less work, and because a manual step was removed, we could automate the deployment process end to end.

2. If we went ahead without the approval or review step and something went wrong, how large is the risk? How long would it take us to find out we have a problem and correct it by either fixing it or withdrawing the change? If the risk is low, then, again, the governance step might best be skipped or changed to a notification only.
3. What is the actual cost of the governance step in both effort and time? How long does it take to create the documentation for this step? How much time does each stakeholder involved spend on it? How much of the cycle time is being consumed while waiting for approvals to proceed?

With this information, you can calculate whether or not the governance step should continue to be used or whether you are better off abandoning or changing it. From my experience, about half the review and approval steps can either be automated (as the human stakeholder is following simple rules) or changed to a notification only, which does not prevent the process from progressing. I challenge you to try this in your organization and see how many things you can remove or automate, getting as close as possible to the minimum viable governance process. I have added an exercise for this at the end of the chapter.

First Steps for Your Organization

There are three exercises that I find immensely powerful because they achieve a significant amount of benefit for very little cost: (1) value stream mapping of your IT delivery process, (2) baselining your metrics, and (3) reviewing your IT governance. With very little effort, you can get a much better insight into your IT process and start making improvements.

Value Stream Mapping of Your IT Delivery Process

While there is a formal process for how to do value stream mapping, I will provide you with a smaller-scale version that, in my experience, works reasonably well for the purpose that we are after: making the process visible and improving some of the bottlenecks.[‡] Here is my shortcut version of value stream mapping:

1. Get stakeholders from all key parts of the IT delivery supply chain into a room (e.g., business stakeholders, development, testing, project management office (PMO), operations, business analysis).
2. Prepare a whiteboard with a high-level process for delivery. Perhaps write "business idea," "business case," "project kickoff," "development," "testing/QA," "deployment/release," and "value creation" on the board to provide some guidance.
3. Ask everyone in the room to write steps of the IT process on index cards for fifteen minutes. Next, ask them to post these cards on the whiteboard and work as a group to represent a complete picture of the IT delivery process on the whiteboard. Warning: you might have to encourage people to stand up and work together, or you may need to step in when/if discussions get out of hand.
4. Once the process is mapped, ask one or more people to walk the group through the overall process, and ask everyone to call out if anything is missing.

[‡] It is worthwhile for you to pick up *Value Stream Mapping* by Karen Martin and Mike Osterling if you want to formalize this process further.

5. Now that you have a reasonable representation of the process, you can do some deep dives to understand cycle times of the process, hot spots of concerns for stakeholders due to quality or other aspects, and tooling that supports the process.
6. Get people to vote on the most important bottleneck (e.g., give each person three votes to put on the board by putting a dot next to the process step).

In my experience, this exercise is the best way to make your IT delivery process visible. You can redo this process every three to six months to evaluate whether you addressed the key bottleneck and to see how the process has evolved. You can make the outcome of this process visible somewhere in your office to show the improvement priorities for each person/team involved. The highlighted bottlenecks will provide you with the checkpoints for your initial roadmap, as those are the things that your initiatives should address.

Baselining Your Metrics

Because having a baseline of your metrics is such an important part of the transformation governance, I want you to spend a few minutes filling out your own Table 1.2. Identify the metrics you care about now and in the future, and identify the mechanism you will use to baseline them. There are a couple of ways to identify the baseline. The baseline approach can be based on surveys, time-in-motion studies, or, ideally, existing and historical data. Where this is not possible, you should think about investing in an automated way to measure this metric. Where that fails, you can run a manual investigation and measuring process (e.g., time-in-motion studies), but those are less reliable and more time consuming.

Metric	Definition	Measurement Mechanism	Baseline Approach	Baseline Value
Release cycle time	The average time it takes for a story to go from a "ready" state to being deployed in production	Extract of date and time from Agile life-cycle-management system	Historical analysis of the last six months of user stories that were successfully deployed in production	168 days

Table 1.2: Metrics definitions example: Metrics should have definitions, measuring mechanisms, and baseline values

Reviewing Your IT Governance Process

There is a lot of talk about automation to help improve the IT delivery process when it comes to speed of delivery and quality. One thing that people underestimate is how much they can influence by just improving their governance process. Here is a short checklist that you can use to review your governance process. Ask these questions to guide where IT delivery governance is really required. Based on the answers, you can evaluate the impact and risk of removing the process step, ideally even with an economic model reflecting monetary impact and risk probability.

IT governance checklist:

- How often has someone rejected a submission to the checkpoint based on reasons other than process compliance?
- What would really happen to the process if an incorrect decision was made?
- What value is being added by the person approving this checkpoint that a computer could not provide automatically based on a number of inputs?

- How much time and money are being spent on this governance process (including the usual wait time that initiatives encounter while waiting for approvals)?
- Is this governance step based on objective measures or a subjective measure? How do you know?

CHAPTER 2

Accepting the Multispeed Reality (for Now)

> If everything is important, then nothing is.
>
> —**Anonymous**

Clients I work with often have a thousand or more applications in their portfolio. Clearly, we cannot make changes to all of them at the same time. This chapter looks at how to navigate the desire for innovative new systems and the existing web of legacy applications. We will identify *minimum viable clusters* of applications to start your transformation and perform an application portfolio analysis to support this.

One of the trends in the industry that has caused the increase in interest in Agile and DevOps practices was the arrival of internet natives, as I mentioned in the introduction. Those companies have the advantage that their applications are newer than most applications are in a large-enterprise context. "Legacy" is often used as a derogatory term in the industry, but the reality is that any code in production is really legacy already. And any new code we are writing today will be legacy tomorrow. Trying to differentiate between legacy and nonlegacy is a nearly impossible task over time.

In the past, organizations tried to deal with legacy through transformation projects that took many years and tried to replace older legacy systems with new systems. Yet very often, many old systems survived for

one reason or another, and the overall application architecture became more complicated. These big-bang transformations are not the way things are done anymore, as the speed of evolution requires organizations to be adaptable while they are changing their IT architecture.

I think we all can agree that what we want is really fast, flexible, and reliable IT delivery. So, should we throw away our "legacy" applications and build a new set of "fast applications"? I think reality is more nuanced. I have worked with dozens of organizations that are struggling with the tension between fast digital applications and slow enterprise applications. Some of these organzations just came off a large transformation that was trying to solve this problem, but at the end of the multiyear transformation, the new applications were already slow-legacy again. A new approach is required that is more practical and more maintainable, and still achieves the outcome.

While we want everything to be fast, we have to accept that the architecture of some applications and the years of accrued *technical debt** might not allow every application to be delivered at the same speed. There are discussions about *bimodal IT* (using two methods of delivery, such as Waterfall for predictability and Agile for exploration)[1] or *multimodal IT* (using several different methods, such as several Agile and Waterfall flavors), which classify applications by type (*systems of engagement* for customer interaction and *systems of record* for internal

* I speak about technical debt quite a bit in this chapter, so I wanted to leave you with a few thoughts on how to measure it. There are, of course, the static code-analysis tools that provide a view of technical debt based on coding issues. I think that is a good starting point. I would add to this the cost to deploy the application (e.g., how many hours of people does it require to deploy into a test or production environment), the cost of regression testing the product (how many hours or people time does it take to validate nothing has broken), and the cost of creating a new environment with the application. If you are more ambitious, you can also look for measures of complexity and dependencies with other applications, but I have not yet seen a good repeatable way for measuring those. The first four I mention are relatively easy to determine and should therefore be the basis for your measure of technical debt.

facing processes).[2] I think this hard classification is somewhat dangerous when it comes to speed; if your core business relies on systems of record to differentiate yourself, then those should be delivered as fast and as reliably as possible. Many organizations use the classification as an excuse to not improve some applications, which is wrong from a business-value perspective.

In the rest of the chapter, I will propose an alternative method that I use with my clients to help shape a multispeed approach with the ultimate goal of everything being as fast as is feasible and economically sensible, which might mean one or more different delivery speeds in the future.

Analyzing Your Application Portfolio

Large organizations often have hundreds if not thousands of applications, so it would be unrealistic to assume that we can uplift all applications at the same time. Some applications probably don't need to be uplifted, as they don't change often or are not of strategic importance. In the exercise section of this chapter, I provide details so that you can run your own analysis.

With this analysis, we can do a couple of things: we can prioritize applications into clusters (I will talk about that a little bit more later) and gather the applications into three different groupings that will determine how we will deal with each application as we are transforming IT delivery. The groupings will determine how you will invest and how you will work with the software vendors and your delivery partners.

The first group is for applications that we want to divest from or keep steady at a low volume of change. Let's call this *true legacy* to differentiate it from the word "legacy," which is often used just for older systems. In the true legacy category, you will sort applications that are hardly ever changing, that are not supporting business-critical processes, and in which you are not investing. I think it is pretty obvious that you don't want to spend much money automating the delivery life cycle for

these applications. For these applications, you will likely not spend much time with the software vendor of the application, and you will choose a low-cost delivery partner that "keeps the lights on" if you don't want to deal with them in-house. And you really shouldn't invest your IT skills in these applications.

The second group is for applications that are supporting your business but are a little bit removed from your customers. Think of ERP or HCM systems—these are the "workhorses" for your applications. You spend a bulk of your money on running and updating these systems, and they are likely the ones that determine your overall speed of delivery for larger projects. Improving workhorses will allow you to deliver projects faster and more reliably, but the technologies of many of these workhorses are not as easily adaptable to DevOps and Agile practices. It is crucial to these systems that you work closely with the software vendor to make the technology more DevOps suitable (I will talk about that in the next chapter). If you choose to get help maintaining and evolving these systems, make sure the partner you work with understands your need to evolve the way of working as well as the system itself.

The third group are your "innovation engines" applications. These are the customer-facing applications that you can use to drive innovation or, on the flip side, that can cause you a lot of grief if customers don't like what you are presenting to them. The challenge here is that most of these will rely on the workhorses to deliver the right experience. My favorite example is the banking mobile app, which you can experiment with but only in so far as it continues to show accurate information about your bank accounts; otherwise, you will get very upset as a customer. Here, you will likely use custom technologies. You should work very closely with your software vendor if you chose a *commercial-off-the-shelf* (*COTS*) *product*, and the delivery partner should be a co-creator, not just a delivery partner.

Now this grouping of applications is not static. As your application architecture evolves, certain applications will move between groups;

that means your vendor and delivery-partner strategy evolves with it. Active application portfolio management is becoming increasingly more important as the speed of evolution increases and application architectures become more modular. Continuing with the chapter 1 theme of making things visible, the best way to represent the analysis is by using an application radar, which has "innovation engines" in the middle and true legacy on the outside.

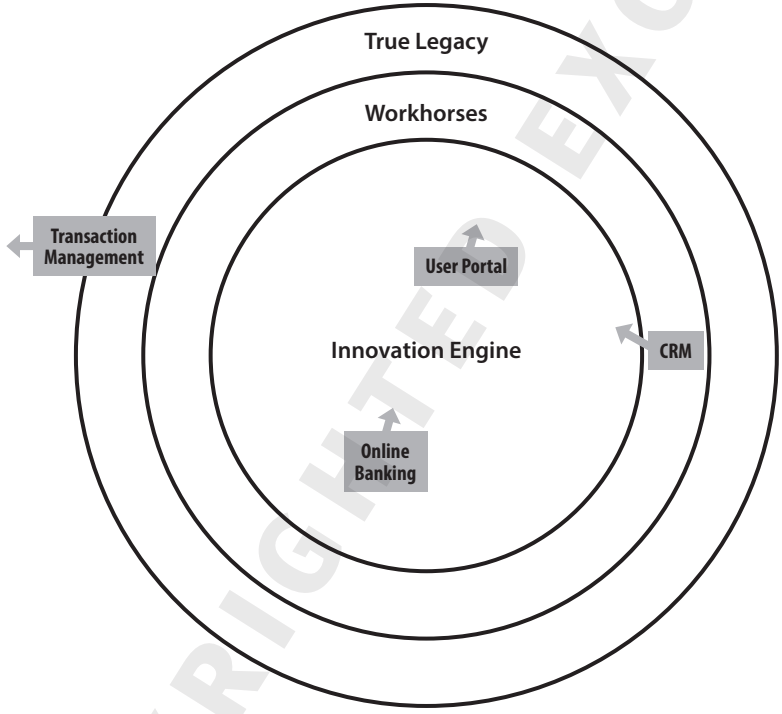


Figure 2.1: Application radar: Makes the status of each application visible

Finding a Minimum Viable Cluster

The Agile principle of small batch sizes applies for transformations as well. We can use the information from the application portfolio analysis

above to guide us. It is very likely that the categories of workhorses and innovation engines contain too many applications to work on at the same time. Rather than just picking the first *x* applications, you need to do a bit more analysis to find what I call a minimum viable cluster.

Applications don't exist in isolation from each other. This means that most functional changes to your application landscape will require you to update more than one application. This, in turn, means that even if you are able to speed up one application, you might not be able to actually speed up delivery, as you will continue to wait on the other applications to deliver their changes.

The analogy of the weakest link comes to mind; in this case, it is the slowest link that determines your overall delivery speed. What you need to determine is the minimum viable cluster of applications. The best way of doing this is to rank your application based on several factors, such as customer centricity and volume of change. The idea of the minimum viable cluster is that you incrementally review your highest-priority application and analyze the dependencies of that application. You look for a small subset of those applications in which you can see a significant improvement of delivery speed when you improve the delivery speed of this subset. (Sometimes you might still have to deal with further dependencies, but in most cases, the subset should allow you to make significant changes independently with a little bit of creativity.)

You can continue the analysis for further clusters so that you have some visibility of the next applications you will start to address. Don't spend too much time clustering all applications. As you make progress, you can do rolling-wave identification of the clusters.

I want to mention a few other considerations when thinking about the prioritization of applications. First, I think it is important that you start to work on meaningful applications as early as possible. Many organizations experiment with new automation techniques on isolated applications with no serious business impact. Many techniques that work for those

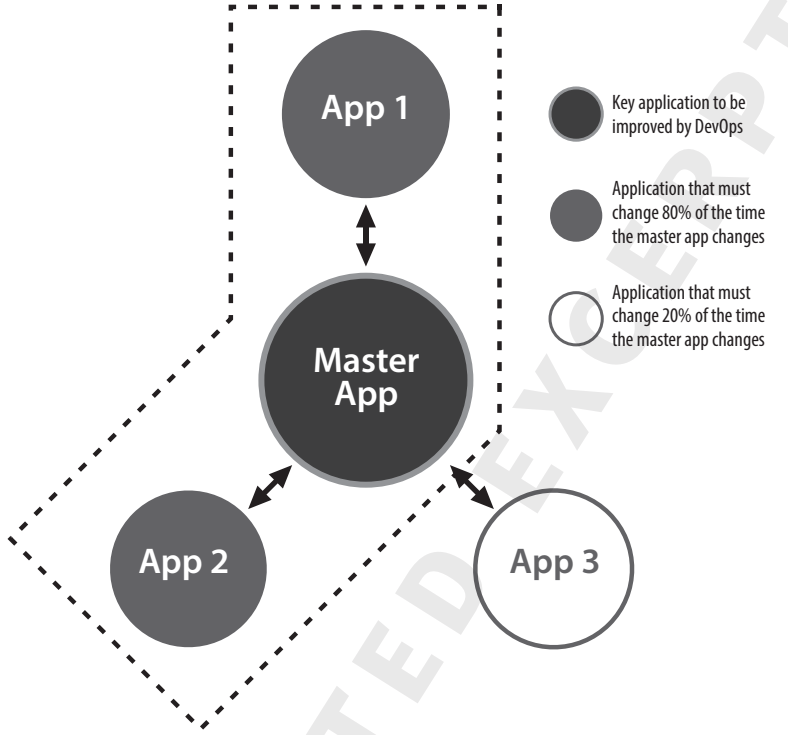


Figure 2.2: Minimum viable cluster: Applies system thinking to application analysis

applications might not scale to the rest of the IT landscape, and the rest of the organization might not identify with the change for that application. ("This does not work for our real systems" is a comment you might hear in this context.)

Because the uplift of your minimum viable cluster can take a while, it might make sense to find "easier" pilots to (a) provide some early wins and (b) allow you to learn techniques that are more advanced before you need to adapt them for your first minimum viable cluster. The key to this is making sure that considerations from the minimum viable cluster are being proven with the simpler application so that the relevance is clear to the organization. Collaboration across the different application stakeholders is critical to achieving this.

How to Deal with True Legacy

We have spoken about the strategy that you should employ for the applications that continue to be part of your portfolio, but what should you do with the true legacy applications?

Obviously, the best thing to do would be to get rid of them completely. Ask yourself whether the functionality is still truly required. Too often, we hang on to systems for small pieces of functionality that cannot be replicated somewhere else, because the hidden cost of maintaining the application is not visible; not enough effort is being put into decommissioning the system.

Assuming this is not an option, we should use for architecture what software engineers have been using in their code for a long time, the *strangler pattern*.[3] The strangler pattern in this case means we try to erode the legacy application by moving functions to our newer applications bit by bit. Over time, less and less functionality will remain in the legacy application until my earlier point comes true: the cost of maintaining the application just for the leftover functionality will become too high, and this will serve as the forcing function to finally decommission it.

The last trick in your "dealing with legacy" box is to make the real cost of the legacy application visible. The factors that should play into this cost are as follows:

- the delay other applications are encountering due to the legacy application,
- the defects caused by the legacy application,
- the amount of money spent maintaining and running the legacy application, and
- the opportunity cost of things you cannot do because of the legacy application being in place.

The more you are able to put a monetary number on this, the better your chances are to overcome the legacy complication over time by convincing the organization to do something about it.

I said before that every application you build now will be the legacy of tomorrow. At the increasing speed of IT development, this statement should make us nervous, as we are creating more and more legacy ever faster. This means that, ultimately, the best way to deal with legacy is to build our new legacy with the right mind-set. There is no *end-state architecture* anymore (well, there never was, as we now know—in spite of what enterprise architects kept telling us). As a result of this new architecture mind-set, each application should be built so that it can easily be decommissioned and to minimize its dependency on other applications. I will cover this in more detail in chapter 10.

Governing the Portfolio and Checkpoints

Your application portfolio is always evolving, and the only way to be successful in such a moving environment is to have the right governance in place. Governance was hard in the past; in the new world, it has become even more difficult. There are more things to govern, the overall speed of the delivery of changes has increased, and without a change in governance, governance will either slow down delivery or become overly expensive.

There are four main points of governance for any change:

- Checkpoint 1 (CP1): this answers the question of whether or not the idea we have for the change is good enough to deserve some funding to explore the idea further and come up with possible solutions.
- Checkpoint 2 (CP2): this answers the question of whether we have found a possible solution that is good enough to attempt as a first experiment or first release to validate our idea.

- Checkpoint 3 (CP3): this answers the question of whether or not the implemented solution has reached the right quality to be released to at least a small subaudience in production.
- Checkpoint 4 (CP4): this answers the question of whether or not the experiment was successful and what we will do next.

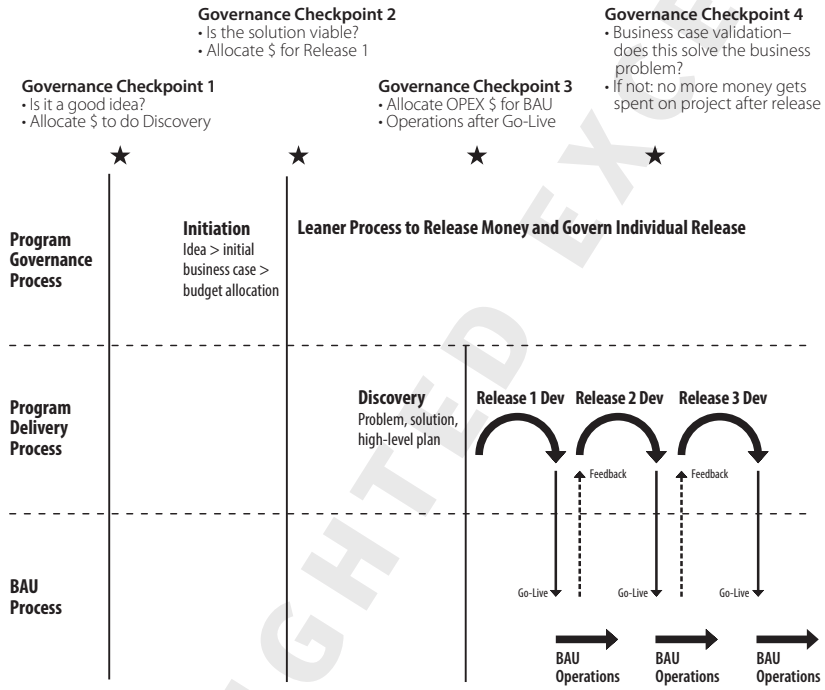


Figure 2.3: Governance checkpoints: An Agile governance process with four checkpoints

Checkpoint 1 (CP1)

At CP1, we are mostly talking about our business stakeholders. Somewhere in the organization, a good idea has come up or a problem has been found that requires fixing. Before we start spending money, our first checkpoint is to validate that we are exploring the right problems and opportunities that have a business impact, are of strategic importance, or

are our "exploratory ideas" to find new areas of business. This checkpoint is a gatekeeper to make sure we are not starting too many new things at the same time and to focus our energy on the most promising ideas.

Between CP1 and CP2, the organization explores the idea, and both business and IT come together to run a *discovery* workshop that can take a couple of hours or multiple weeks depending on the scale of the problem. You can run this for a whole business transformation or for a small change. The goal of discovery really falls into three important areas: (1) everyone understands the problem and idea, (2) we explore what can be done with support of IT, and (3) we explore what the implementation could look like in regard to schedule and teams. This discovery session is crucial to enable your people to achieve the best outcome.

Checkpoint 2 (CP2)

After discovery, the next checkpoint is validation that we now have discovered something that is worth implementing. At this stage, we should check that we have capacity to support the implementation with all parties: IT, business stakeholders, the operations team, security, and anyone else impacted. This is a crucial checkpoint at which to embed architectural requirements, as it becomes more difficult to add them later on. Too often, business initiatives are implemented without due consideration of architectural aspects, which leads to increased technical debt over time.

It is my view that every initiative that is being supported by the organization with scarce resources such as money and people should leave the organization in a better place in two ways: it better supports the business, and it leaves the IT landscape better than it was before. This is the only reasonable way to reduce technical debt over time and deal with legacy. CP2 is the perfect time to make sure that the improvement of the IT landscape / down payment of technical debt is part of the project before it continues on to implementation. This has to be something that is not

optional; otherwise, the slippery slope will lead back to the original state. It is quite easy to let the necessary rigor be lost when "just this once" we only need to quickly put this one temporary solution in place. I learned over the years that there is nothing more permanent than a temporary solution.

Between CP2 and CP3 is the bulk of the usual software delivery that includes design, development, and testing work being done in an Agile fashion. I am confident that Agile is the only methodology we will need going forward but that we will have different levels of rigor and speed as part of our day-to-day Agile delivery. Once the solution has matured over several iterations to being a release candidate, we will have CP3.

Checkpoint 3 (CP3)

At CP3, we will confirm that the release candidate has reached the right quality for us to release it to production. We will validate that the architecture considerations have been adhered to and technical debt has been paid down as agreed, and we will not introduce new technical debt unknowingly. (Sometimes we might consciously choose to accrue a little more debt to test something early but commit to fixing it in the next release. This should be a rare occasion, though.) This checkpoint is often associated with the change control board, which has to review and approve any changes to production. Of course, we are looking for the minimum viable governance here, and you can refer to the previous chapter for more details on general governance principles to follow at CP3.

Between CP3 and CP4 the product is in production and is being used. If we follow a proper Agile process, the team will already be working on the implementation of the next release in tandem with supporting the version that has just gone live. Internal or external stakeholders are using the product, and we gather feedback directly from the systems (through monitoring, analytics, and other means) or directly from the stakehold-

ers by leveraging surveys, feedback forms, or any other communication channel.

Checkpoint 4 (CP4)

Checkpoint 4 is the checkpoint that is extremely underutilized in my experience. It's one of those processes that everyone agrees is important, yet very few have the rigor and discipline to really leverage it to meet its full potential. This checkpoint serves to validate that our idea and the solution approach are valid. Because projects are temporary by definition, the project team has often stood down already and team members have been allocated to other projects. CP4 then becomes a *pro forma* exercise that people don't appreciate fully. If we have persistent, long-lasting product teams, the idea of learning from the previous release and understanding the reaction of stakeholders is a lot more important. Those product teams are the real audience of CP4, though, of course, the organizational stakeholders are the other audience that needs to understand whether the money was well invested and whether further investment should be made.

CP4 should be an event for learning and a possibility for celebrating success; it should never be a negative experience. If the idea did not work out, we learned something useful about our product that we have to do differently next time. You can combine CP4 with a post-implementation review to look at the way the release was delivered and to improve the process as well as the product. It is my personal preference to run the post-implementation review separately to keep improving the product and the delivery process as two distinct activities.

With this governance model and the four checkpoints in place, you can manage delivery in several speeds and deal with the faster pace. Each checkpoint allows you to assess progress and viability of the initiative, and where required, you can move an initiative into a different delivery model with a different (slower or faster) speed.

First Steps for Your Organization

To support you in adopting what I have described in this chapter, I will provide two exercises for you to run in your organization. This time, both of them are highly related: the first is an analysis of your application portfolio and the second is the identification of a minimum viable cluster of application for which a capability uplift will provide real value.

Application Portfolio Analysis

If you are like most of my clients, you will have hundreds or thousands of applications in your IT portfolio. If you spread your change energy across all of those, you will likely see very little progress, and you might ask yourself whether the money is actually spent well for some of those applications. So, while we spoke about the IT delivery process in the chapter 1 exercises as one dimension, the application dimension is the second dimension that is important. Let's look at how to categorize your application in a meaningful way.

Each organization will have different information available about its applications, but in general, an analysis across the following four dimensions can be done:

- Criticality of application: How important is the application for running our business? How impactful would an issue be on the user experience for our customers or employees? How much does this application contribute to regulatory compliance?
- Level of investment in application: How much money will we spend in this application over the next 12–36 months? How much have we spent on this application in

the past? How many priority projects will this application be involved with over the next few years?

- Preferred frequency of change: If the business could choose a frequency of change for this application, how often would that be (hourly, weekly, monthly, annually)? How often have we deployed change to this application in the last 12 months?
- Technology stack: The *technology stack* is important, as some technologies are easier to uplift than others. Additionally, once you have a capability to deliver, for example, Siebel-based applications more quickly, any other Siebel-based application will be much easier to uplift too, as tools, practices, and methods can be reused. Consider all aspects of the application in this technology stack: database, data itself, program code, application servers, and middleware.

For each of the first three dimensions, you can either use absolute values (if you have them) or relative numbers representing a nominal scale to rank applications. For the technology stack, you can group them into priority order based on your technical experience with DevOps practices in those technologies. I recommend using a table with headings much like the one in Table 2.1. On the basis of this information, you can create a ranking of importance by either formally creating a heuristic across the dimensions or by doing a manual sorting. It is not important for this to be precise; we are aiming only for accuracy here.

It's clear that we wouldn't spend much time, energy, and money on applications that are infrequently changed—applications that are not critical for our business and on which we don't

intend to spend much money in the future. Unfortunately, just creating a ranking of applications is usually not sufficient, as the IT landscape of organizations is very complex and requires an additional level of analysis to resolve dependencies in the application architecture.

#	Application	Technology	Strategic Application	Frequency of Change	Size of the Application in the Investment Portfolio
95	App A	Java, .NET, Oracle	4–Critical	9	4–Very High

Table 2.1: Application analysis example: A table like this will help you structure the application analysis

Identifying a Minimum Viable Cluster

As discussed above, the minimum viable cluster is the subset of applications that you should focus on, as an uplift to these will speed up the delivery of the whole cluster. Follow the steps below to identify a minimum viable cluster:

1. Pick one of the highest-priority applications (ideally based on the portfolio analysis from the previous exercise) as your initial application set (consisting of just one application).
2. Understand which other applications need to be changed in order to make a change to the chosen application set.

3. Determine a reasonable cutoff for those applications (e.g., only those covering 80% of the usual or planned changes of the chosen application).
4. You now have a new, larger set of applications and can continue with steps 2 and 3 until the application set stabilizes to a minimum viable cluster.
5. If the cluster has become too large, pick a different starting application or be more aggressive in step 3.

Once you have successfully identified your minimum viable cluster, you are ready to begin the uplift process by implementing DevOps practices such as test automation and the adoption of cloud-based environments, or by moving to an Agile team delivering changes for this cluster.