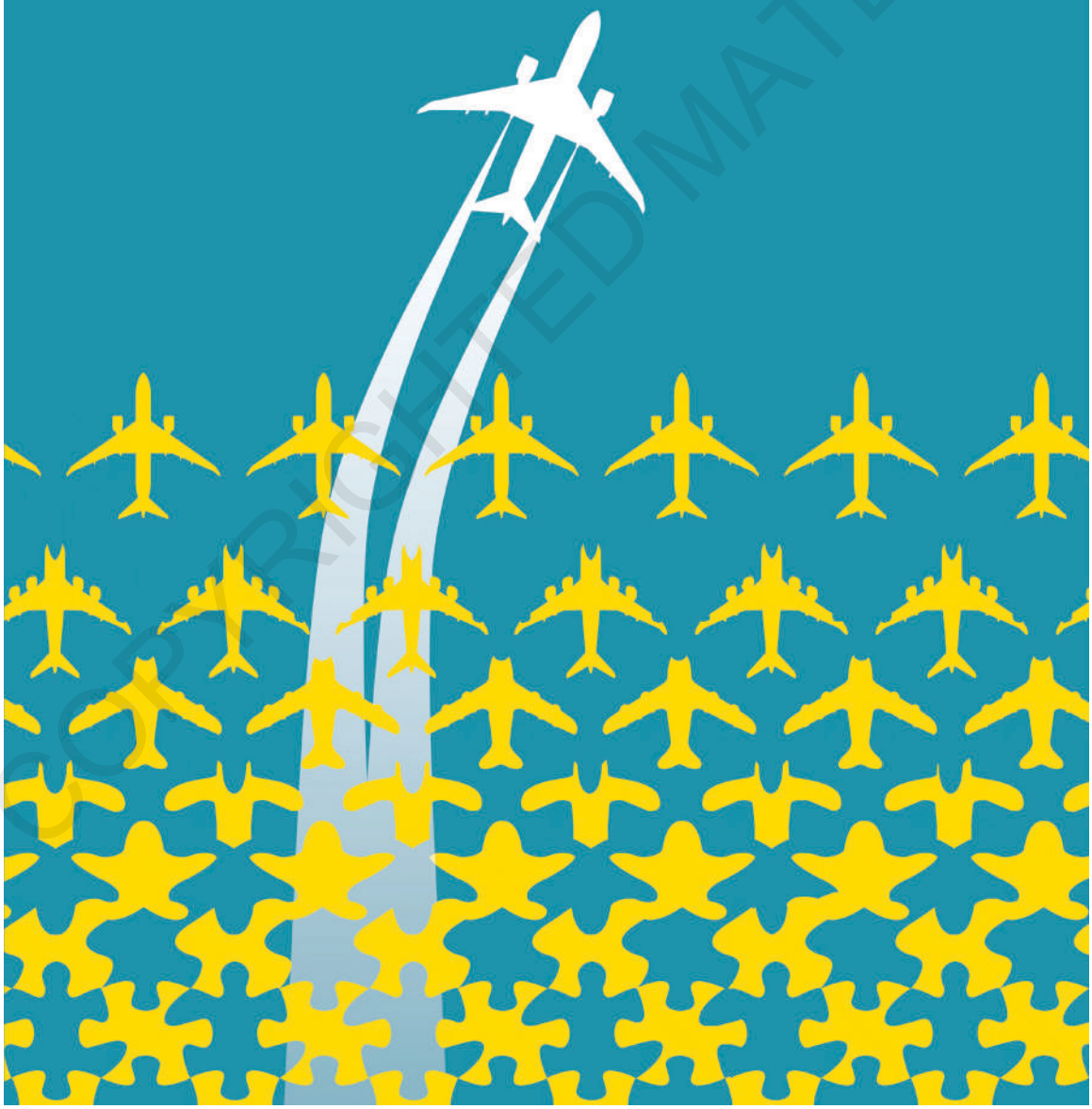
PROJECT TO PRODUCT

MIK KERSTEN

HOW TO SURVIVE AND THRIVE IN THE AGE OF DIGITAL DISRUPTION WITH THE FLOW FRAMEWORK

FOREWORD BY GENE KIM



Praise for

PROJECT TO PRODUCT

With the introduction of the Flow Framework, Mik has provided a missing element to any large-scale Agile transformation. I recommend that anyone involved in complex product delivery read this book and think about how they can apply this thinking to their value stream.

—**Dave West**, CEO Scrum.org and author of *Nexus Framework for Scaling Scrum: Continuously Delivering an Integrated Product with Multiple Scrum Teams*

During our transformation to "100% Agile" BMW Group IT organization, we discovered early on that the former project portfolio approach did not sufficiently support our journey. Therefore, we started with a transition from "project to product." The exchanges with Mik on the topic of product orientation and the Flow Framework was very helpful and a real inspiration for me. The fact that Mik is now sharing his vast knowledge in this book makes me particularly happy. It provides the motivation and the toolset necessary to help create a product portfolio based on a value driven approach. For me it is a must read—and indeed it is also a fun read.

—**Ralf Waltram**, Head of IT Systems Research & Development, BMW Group

Organizing software development as a group of loosely connected projects will never lead to good products. Kersten explains how to tie work products to value streams corresponding to features, defects, security, and (technical) debt . . . . [*Project to Product* is a] major contribution to the theory of management in the age of software.

—**Carliss Y. Baldwin**, William L. White Professor of Business Administration at the Harvard Business School, Emerita, and co-author of *Design Rules, Volume 1: The Power of Modularity*

If you want to get rid of obsolete practices and succeed with the new digital, read this book.

—**Carlota Perez**, author of *Technological Revolutions and Financial Capital: The Dynamics of Bubbles and Golden Ages*

Every now and then, a body of work comes along with such timely precision that you think hallelujah! Mik's book *Project to Produc*t is the perfect antidote for those businesses struggling with digital transformation, broken Agile implementations, and the onslaught of enterprise disruption. In fact, it's a really important component in the world of flow which is at the forefront of business agility. Not only will this framework help your teams to ignite their software delivery cadence but to do it at scale with high quality, reduced costs, and increased value. And more importantly, with happy teams—and with the metrics to prove it.

—**Fin Goulding**, International CIO at Aviva and co-author of *Flow: A Handbook for Change-Makers, Mavericks, Innovation Activists, Leaders: Digital Transformation Simplified,* and *12 Steps to Flow: The New Framework for Business Agility*

I had the pleasure of having an advance copy of *Project to Product* at my company over the summer—and what an eye opener it is. This book is spot on the journey Volvo Car Group is starting up right now. The insight Mik has in our industry and the way his book describes the Age of Software makes this our new "go to" book for our product journey in our digital landscape!

—**Niclas Ericsson**, Senior IT Manager, Volvo Car Corp

Project to Product is going to be one of the most influential reads of 2019 and beyond. One that connects work outcomes to business results. One that provides models to make better business decisions. One that gives technology leaders a framework to enable the change necessary for companies to remain relevant.

—**Dominica DeGrandis**, author of *Making Work Visible: Exposing Time Theft to Optimize Work & Flow*

Many large organizations are still applying a management model from the early 1900s optimized for manual labor to everything they do, including complex, unique, product development. With this book, Mik provides a great articulation of the importance of focussing on the work not the workers, on the value stream network, and lessons learned on what to avoid. Mik, who has many years experience working with hundreds of companies on this topic, shares his wisdom and insights via a Flow Framework, which is immensely valuable for organizations who recognize the need to move to better ways of working.

—**Jonathan Smart**, Head of Ways of Working, Barclays

Project to Product is a very insightful book, and the overall model Mik lays out for the Flow Framework is especially intriguing. Not only does Mik address the complexities of Agile transformation and moving to a product-based development, he also discusses how to get your architecture, process, and metrics integrated in a way to effectively measure value delivery. I got pretty excited about the Flow Framework and look forward to applying it to my own technology transformation activities.

—**Ross Clanton**, Executive Director, Technology Modernization, Verizon

PROJECT TO PRODUCT

MIK KERSTEN

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

First Edition
Printed in the United States of America

22 21 20 19 18 1 2 3 4 5 6

Cover illustration by Rachel Masterson
Figure illustrations by Zhen Wang
Cover and book design by Devon Smith
Author photograph by Janine Coney

Library of Congress Catalog-in-Publication Data
is available upon request.

ISBN: 978-1-942-78839-3
eBook ISBN: 978-1-942-78840-9
Kindle ISBN: 978-1-942-78841-6
Web PDF ISBN: 978-1-942-78842-3

PROJECT TO PRODUCT

MIK KERSTEN

HOW TO SURVIVE AND THRIVE IN THE AGE OF DIGITAL DISRUPTION WITH THE FLOW FRAMEWORK

FOREWORD BY
GENE KIM

IT Revolution
Portland, Oregon

To my mother, who made me who I am,
and my father, who taught me who to be.

CONTENTS

ILLUSTRATIONS

FIGURES

TABLES

FOREWORD

by Gene Kim

The mark of a great book is that it makes obvious what is wrong with the old worldview and replaces it with one that is simultaneously simpler and yet presents a better model of reality. The transition from Copernican to Newtonian physics has been long held as a great example of such a breakthrough. I believe *Project to Product* presents a new way to think that enables a new way of doing.

In today's business landscape, with companies facing the threat of digital disruption, the old ways of planning and executing no longer seem enough to survive. For decades, great minds have been seeking a way to manage technology to achieve business goals—after all, we know there is something very, very wrong with the way we're managing technology, we see the poor outcomes with our own eyes.

Project to Product makes the solid case that in the Age of Software, the methods that served us well for over a century are truly coming to an end: project management, managing technology as a cost center, traditional outsourcing strategies, and relying on software architecture as the primary means to increase developer productivity. And better yet, it provides a wonderful framework to replace it, namely the Flow Framework.

You'll learn what it looked like when an organization spent over a billion dollars on a technology transformation that was doomed to fail from the beginning because it was trying to solve the wrong problem. You will learn how some of the fastest growing companies nearly died by ignoring technical debt that was accumulated in their need to cut corners to ship products quickly, which included Nokia's massive Agile transformation that did nothing to stop its demise.

Dr. Mik Kersten brings the perspective of someone who got his PhD in software engineering only to discover that the massive productivity gains were not to be found there. Instead, those productivity gains can only be reaped when we change how teams across the entire business value stream work together, an epiphany common to so many of us in the DevOps community.

But he also brings the perspective of someone who built the large open-source software community around Mylyn in the Eclipse ecosystem, used by millions of Java developers. As founder and CEO of a software company, he brings a visceral understanding of what it's like to live and die by the ability of business, product, and engineering leadership to work together effectively.

You'll also follow in Dr. Kersten's professional journey and relive his three biggest epiphanies—fans of The Phoenix Project will especially love the lessons learned from the BMW Group Leipzig manufacturing plant, which he rightly calls "a pinnacle of the Age of Mass Production," and the profound lessons that the software industry can learn from it.

Project to Product is an incredible achievement. Dr. Kersten provides a better way to think about how business and technology organizations create value together, and provides the Flow Framework as a way for those leaders to plan and execute together, to innovate for their customers, and to win in the marketplace. To disrupt, instead of being disrupted. The upcoming Deployment Period Age of Software may bring the equivalent of an economic extinction event, so these capabilities are no longer optional for survival.

Every decade, there are a couple of books that genuinely change my worldview. You can tell which books they are, because more than one-third of the pages are bookmarked, indicating something I felt was truly an important a-ha moment or a reminder to myself to study further later. This is one such book.

I hope you find it as rewarding and life-changing as I did.

Gene Kim
Portland, OR
September 4, 2018

INTRODUCTION

The Turning Point

For the majority of our careers, those of us involved with enterprise IT have been dealing with change at a frenzied pace. Technology platforms, software development methodologies, and the vendor landscape have been shifting at a rate that few organizations have been able to match. Those that manage to keep up, such as Amazon, and Alibaba, are further driving change by redefining the technology landscape around their software platforms, causing the rest to fall even further behind.

This daunting and unrelenting pace of change has been seen as a hallmark of the digital disruption. But if we step back and look at the patterns of progress that came before, we begin to see ripples of the great surges of change and development of previous industrial and technological revolutions.

Over the course of three centuries, a pattern emerges. Starting with the Industrial Revolution, every fifty years or so a new technological wave combines with ecosystems of innovation and financing to transform the world economy.[1] Each of these technological waves has redefined the means of production so fundamentally that it triggered an explosion of new businesses followed by the mass extinction of those businesses that thrived in the culmination of the previous surge. Each wave has been triggered by the critical factor of production becoming cheap. New infrastructure is then built while financial capital drives the ecosystem of entrepreneurs and innovators who leverage the new technological systems to disrupt and displace the incumbants of the last age.

Each of these technological revolutions has required existing businesses to master a new means for production, such as steam or the

assembly line. For the digital revolution, the new means of production is software. If your organization has already mastered software delivery at scale, this book is not for you. The goal of this book is to provide everyone else with a new managerial framework that catalyzes the transition to the Age of Software.

Theories that explain the cycles of the last four technological revolutions and the first half of this one are proposed by Carlota Perez in *Technological Revolutions and Financial Capital: The Dynamics of Bubbles and Golden Ages* and by Chris Freeman and Francisco Louçã in *As Time Goes By: From the Industrial Revolutions to the Information Revolution*. Perez expands on the "long wave" or Kondratiev economic model by specifying two distinct periods within each cycle (Figure 0.1). The first half is the *Installation Period*, when a new technology and financial capital combine to create a "Cambrian explosion" of startups, disrupting entire industries of the previous age. At the end of the Installation Period is the *Deployment Period* of technological diffusion, when the production capital of new industrial giants starts taking over. Between these two periods is what Perez termed the *Turning Point*, historically marked by financial crashes and recoveries. This is when businesses either master the new means of production or decline and become relics of the last age.[2]

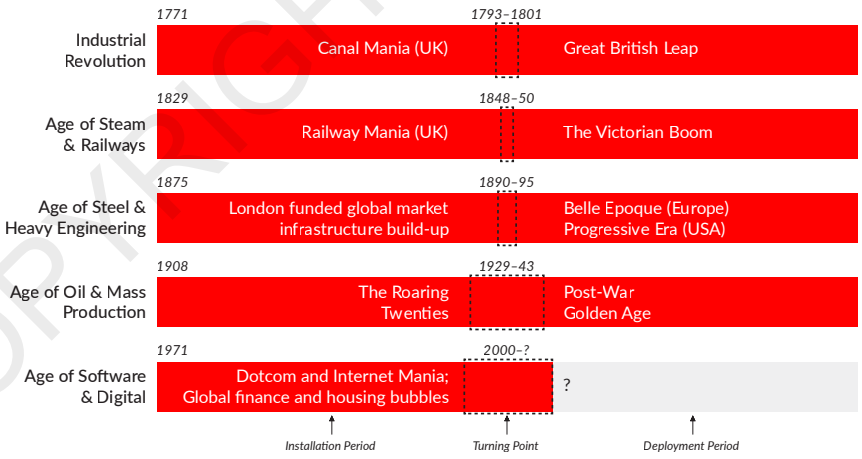


Figure 0.1: Technological Revolutions and the Age of Software.[3]

Fifty years have passed since NATO held the first conference on software engineering in 1968 and the Age of Software officially began. Today, the pace of change feels relentless because we are passing through the Turning Point. At the current rate of disruption and decline, half of S&P 500 companies will be replaced in the next ten years.[4]

These businesses, many of which were founded prior to the Age of Software, are starting to see a growing portion of their spending shift to technology as their market success is increasingly determined by software. However, the productivity of software delivery at enterprise organizations falls woefully behind that of the tech giants, and the digital transformations that should be turning the tide are failing to deliver business results.

The problem is not with our organizations realizing that they need to transform; the problem is that organizations are using managerial frameworks and infrastructure models from past revolutions to manage their businesses in this one. Managerial accounting, organizational hierarchies, and Lean manufacturing were critical to success in previous revolutions. But these managerial frameworks are no longer sufficient to successfully direct and protect a business in the Age of Software.

I had a chance to witness the pitfalls of this trap firsthand. Working with Nokia, I noticed that management was measuring the success of its digital transformation by how many people were trained on Agile software development methodologies and were onboarded onto Agile tools. These activity-based proxy metrics had nothing to do with business outcomes. As I will summarize in Part I, Nokia's transformation efforts failed to address the core platform problems that made it so difficult for the company to adapt to the changing market. In spite of what appeared to be a well-planned transformation, management was not able to realize this until too late. I watched with frustration as Nokia lost the mobile market it had created, in spite of the heroic efforts of my colleagues, who were doing everything they could to save the company.

A few years later, I was invited to speak with IT leaders at a global bank. The bank was six months into its third attempt at a digital transformation, and this time, DevOps tools were added to the mix and expected to save the day. The budget for the transformation was

approximately $1 billion, but shockingly, I realized their transformation plan was even more flawed in its approach than the one at Nokia. Every aspect of the transformation was being project managed to cost reduction alone and not to project overall business outcome with reduced cost as a key metric. As I learned more, I started getting a visceral image that a billion dollars of the world's wealth was going to go up in flames without producing any value. There were still eighteen months left to right the ship, but I knew that with cost alone as the foundation of the transformation, it was too late to alter course. Nokia had left me with an image of a burning mobile platform that destroyed a tremendous amount of wealth and prosperity. I now had a vivid image of the bank's digital transformation lighting fires of waste across its ranks.

That was the day I started this book. There was something so fundamentally wrong with the way business people and technologists worked and communicated that even leaders with the best of intentions could still lead their companies into predictable decline.

How is this possible when we now have five decades of software practice behind us? The Agile and DevOps movements have made great strides in adapting key production techniques from the Age of Mass Production to the technical practice of building software. For example, continuous delivery pipelines allow organizations to leverage the best practices of automated production lines. Agile techniques capture some of the best technical management practices of Lean manufacturing and adapt them to software delivery.

The problem is, with the exception of some tech giants run by former software engineers, these techniques are completely disconnected from the way that the business is managed, budgeted, and planned. Software delivery concepts near and dear to technologists, such as technical debt and story points, are meaningless to most business leaders who manage IT initiatives as projects and measure them by whether they are on time and on budget. Project-oriented management frameworks work well for creating bridges and data centers, but they are woefully inadequate for surviving the Turning Point of the Age of Software.

In this book, we will examine several digital transformation failures that caused organizations to lose their place in the market. We

will then dig further into understanding the current state of enterprise software delivery by looking at a study I conducted with Tasktop, "Mining the Ground Truth of Enterprise Toolchains," that analyzed the Agile and DevOps toolchains of 308 organizations to uncover the causes of this disconnect between business and technology.[5] *Project to Product* will then provide you with a new management framework and infrastructure model, called the Flow Framework, for bridging this gap between business and technology.

The Flow Framework is a new way of seeing and measuring delivery and aligning all of your IT investments according to value streams that define the set of activities for bringing business value to the market, via software products or software as a service (SaaS). The Flow Framework displaces project-oriented management, cost center budgeting, and organizational charts as the primary methods of measuring software initiatives. These are replaced with flow metrics for connecting technology investment to business results. The Flow Framework allows you to scale the Three Ways of DevOps—flow, feedback, and continual learning (as outlined in *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*[6])—beyond your technology organization and to your entire business.

With each technological revolution, a new kind of infrastructure has been established in order to support the new means of production. Canals, railways, electrical grids, and assembly lines were key infrastructure components that underpinned the technological ecosystems of previous cycles. Many digital transformations have gone wrong by over applying infrastructure concepts of the last revolution to this one. Production and assembly lines are great at reducing variability and reliably producing similar widgets, but software delivery is an inherently variable and creative endeavor that spans a complex network of people, processes, and tools. Unlike manufacturing, in modern software delivery the product development and design process are completely intertwined with the manufacturing process of software releases. Attempting to manage software delivery the way we manage production lines is another instance where frameworks from previous technological revolution are failing us in this one. The Flow Framework points to a new and better way.

What if we could see the flow of business value within our organizations in real time, all the way from strategic initiative to running software, the way the masters of the last age ensured they could see and collect telemetry for every step of the assembly line? Would we see a linear flow or a complex network of dependencies and feedback loops? As the data set of 308 enterprise IT toolchains we will examine in Chapter 8 demonstrates, we see the latter. This flow of business value within and across organizations is the *Value Stream Network*. In the Age of Software, Value Stream Networks are the new infrastructure for innovation. A connected Value Stream Network will allow you to measure, in real time, all software delivery investments and activities, and it will allow you to connect those flow metrics to business outcomes. It will empower your teams to do what they love doing, which is to deliver value for their particular specialty in the value stream.

A developer's primary function and expertise is coding, yet studies summarized in this book have shown that developers spend more than half their time on manual processes due to disconnects in the Value Stream Network. These disconnects are the result of relics that go back two technological revolutions: Taylorism, which resulted in the treatment of workers as cogs in a machine,[7] and the silos that have formed in functionally structured organizations.

Successful businesses in the Age of Mass Production aligned their organizations to the value streams that delivered products to their customers instead of constraining themselves to rigid functional silos that disconnected specialists from each other and from the business. For example, Boeing, a master of the Age of Mass Production, could never have brought the highly innovative 787 Dreamliner to market and scaled its production to meet the growth in demand if the company had been structured like today's enterprise IT organizations. Organizations that manage IT delivery as projects instead of products are using managerial principles from two ages ago and cannot expect those approaches to be adequate for succeeding in this one. Visionary organizations are creating and managing their Value Stream Networks and product portfolios in order to leapfrog their competition in the Age of Software.

The future of software delivery is already here; it's just not evenly distributed yet. Software startups and digital natives have already

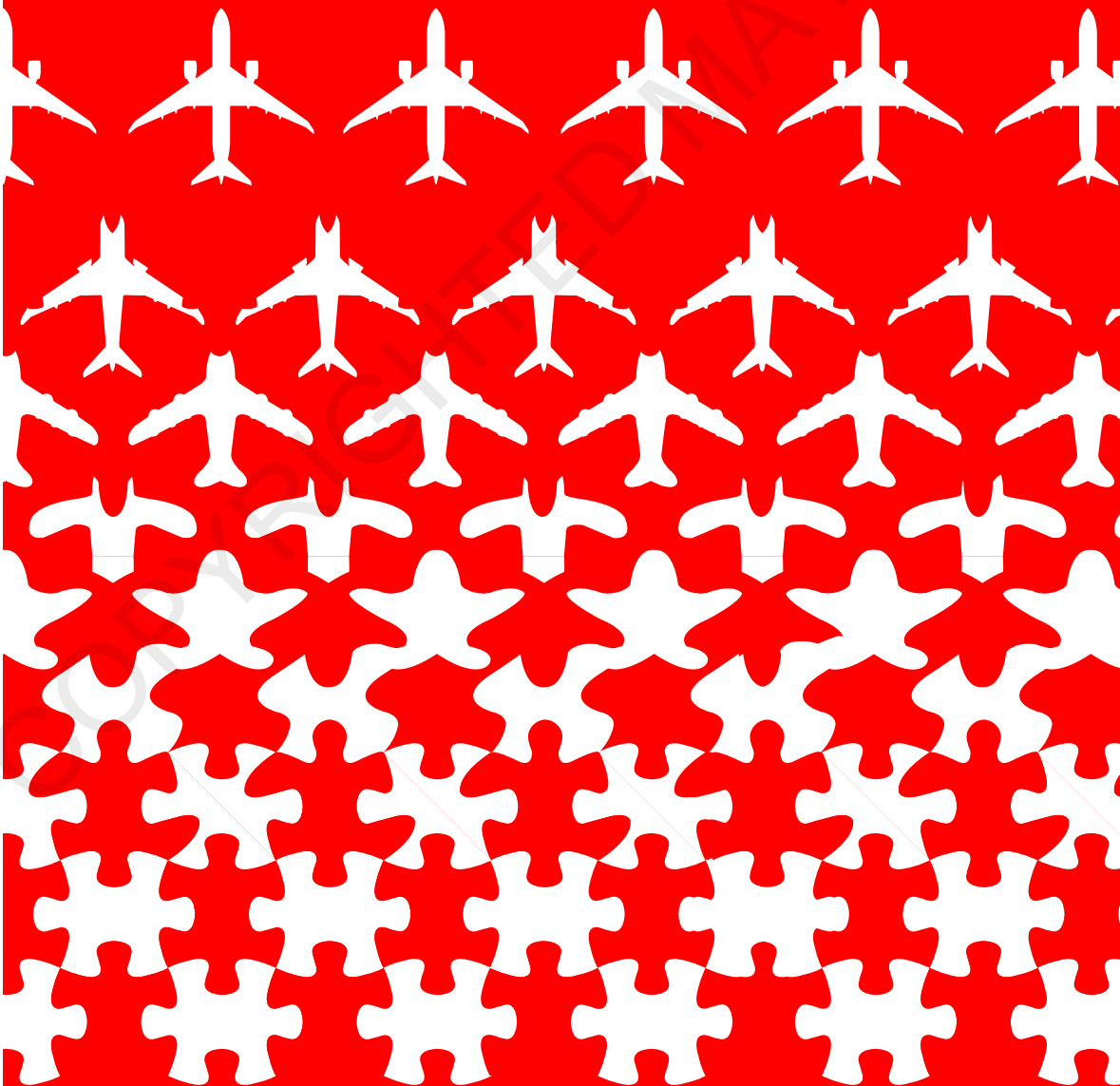
created fully connected Value Stream Networks that are aligned to their product delivery, are focused on flow over siloed specialization, and connect all of their software delivery activities to measurable business results. Their leaders speak the language of developers, often because they were developers, which enables them to effectively direct their software strategies. What does that mean for the fate of every other company? How can we bridge the gap between technology and business leadership to create a common language that allows the rest of the world's organizations to thrive in the Age of Software?

While organizations ponder these questions, the tech giants that have mastered software at scale are expanding into traditional business, such as finance and the automotive industry. The tech giants are mastering traditional businesses more quickly than the world's established companies are mastering software delivery. In doing so, they're amassing a growing portion of the world's wealth and technology infrastructure.

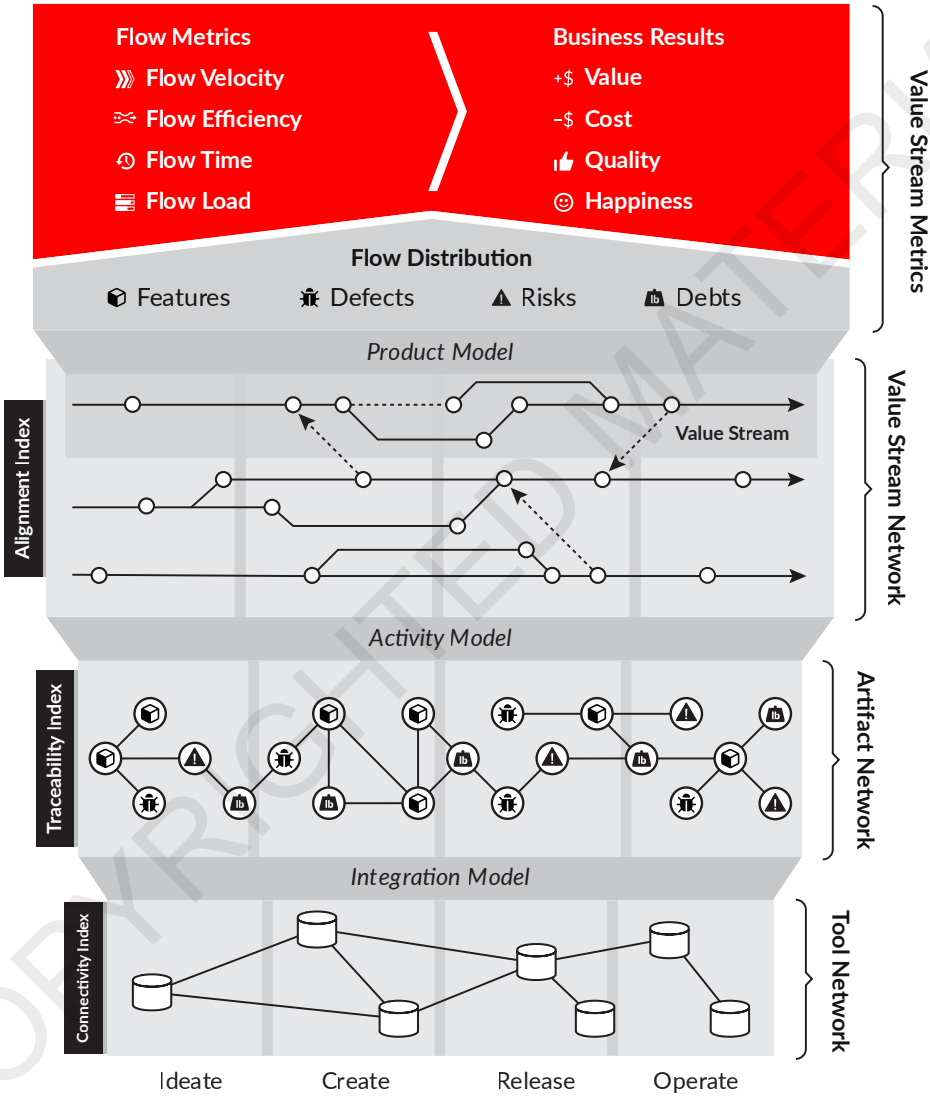
The product offerings they have created are delivering fundamental value to businesses and consumers, and the market pull for that value will only grow. Trying to slow progress or demand is foolhardy; but leaving the economy to a handful of digital monopolies will be problematic for our companies, our staff, and our social systems. If we do not turn this tide—the increasing amount of wealth in the hands of tech giants, and the network effects of technologies making effective government regulation difficult at best—the consequences could be more dire than the mass company extinctions that we witnessed in the four previous ages.

We can create another future. We can make our organizations competitive. We can leverage the lessons of the tech giants and startups and adapt them to the complexity of our existing businesses. We can turn the black box of IT into a transparent network of value streams and manage those the way the digital natives of the Age of Software do. To achieve this, we need to shift our focus from transformation activities to measurable business results. We need a new framework to shift our organizations from project to product, thus securing our place in the digital future.

PART I



Flow Framework™



PART I

THE FLOW FRAMEWORK

In the spring of 2017, Rene Te-Strote invited me to visit the BMW Group plant in Leipzig, Germany. My work with Rene started several years earlier, when we met at an industry conference on application life cycle management. Rene was the E/E-IT-Responsible for the electronic control units in the cars, and the BMW Group was looking for infrastructure tools to integrate and scale its software delivery toolchain. The tools were needed to support the increasing pace of software-component innovation demanded by the new i3 and i8 electric car programs.

Like others at the conference, Rene was looking to bring Agile and DevOps tools and methodologies into an enterprise IT environment. But what fascinated me was the scope of the problem that Rene was trying to solve. He not only needed to connect thousands of internal specialists—including developers, testers, and operations staff—he also needed to integrate dozens of software suppliers into that same toolchain. Each of the suppliers contributed to the over one hundred million lines of code that were run in a modern premium-class car.[1]

All that software, as well as the various internal software-delivery teams, had to be connected. For example, if the BMW Group's continuous integration environment identified a defect in a supplier's software, that defect needed to flow to the supplier's value stream, and then the fix needed to flow back to the BMW Group. Iterating on something as new as the i Series at the rate that the BMW Group was bringing them to market could not happen over transferring spreadsheets and reports.

Throughout our journey to solve this problem, I occasionally joked with Rene that if we succeeded, he ought to take me to the i Series plant and let me drive a car off the production line. As it turned out, Rene took me seriously.

Rene spent the first part of his career working in mass production at the Leipzig plant. The plant is one of the world's ultimate examples of how advanced mass production and manufacturing have become. The resulting visit to the BMW Group Leipzig plant turned out to be one of the most educational and inspirational experiences of my career. What I saw and learned in two full days of walking the plant floor and seeing the ground truth of the most advanced value streams from the Age of Mass Production gave me a new perspective on where we are in the maturity curve of the Age of Software.

In this book, I will walk you through the realizations I had on the factory floor, as they illustrate the errors of our approach to enterprise software delivery. I have, to the best of my memory, attempted to recreate the experience of visiting the Leipzig plant for you in the BMW stories that begin each chapter of this book.

Imagine Rene, who had been thrown from a world where a flawless car leaves the production line every seventy seconds to the world of enterprise IT that we know. The contrast could not have been starker, and I realized at that moment that this massive gap was what Rene wanted to show me. The gap went far beyond what we hear from Agile thought leaders who attempt to teach IT professionals Lean methods, such as those pioneered by the Toyota Production System (TPS). The gap demonstrated how disconnected enterprise IT organizations can be from the means of production. At the Leipzig plant, what amazed me most was the way that the business and manufacturing lines were

seamlessly interconnected, all the way from the production lines to the business needs that were reflected in the architecture of the building complex itself.

Consider today's world of enterprise IT. Businesses measure IT with organizational charts and cost centers. The vast majority of enterprise IT organizations have no formalized notion of value streams or measurement of how business value is delivered. Perhaps most shockingly, they do not even agree on what the units of production are. Agile transformations keep failing to scale, with knee-jerk reactions of "culture" being to blame. Efforts that start out trying to deliver the end-to-end benefits of DevOps get pigeonholed into transformations that only involve "code commit to production"—such a narrow slice of the value stream that the business rarely sees the benefits or takes notice.

The bottom line is that enterprise IT organizations and the businesses in which they live have not yet caught up to the infrastructure and management techniques of the last technological revolution that have been mastered by companies like the BMW Group. Leadership clings to a Taylorist view, established in the Age of Steel, where IT organizations are siloed from the business, functionally specialized, and disconnected from each other. Yet those specialists are expected to deliver more and more as the threats of digital disruption grow. Many IT specialists know that this is a recipe for disaster, but the gap between technical language and business language has not been spanned. The result is that the software delivery efficiency of these companies is abysmal when compared to that of digital startups or the tech giants.

This mind-set has grave implications. If the current trajectory does not change, the incumbent companies that form the backbone of the world economy are at an inherent and significant disadvantage. Does this mean that large and established organizations in every industry are doomed to fail in an age where almost every enterprise is turning into a software company? A disconcerting trend is already visible across multiple markets. For example, at the start of the Age of Software, the average time in the Financial Times Stock Exchange was seventy-five years; today, it is less than twenty years and falling.[2]

The research and data summarized in this book offer a kernel of hope. Organizations can and must change in order to create the software innovation engines that will ensure their competitiveness and survival. To do that, we need to learn from the history of previous technological revolutions instead of assuming that we are in a completely unique moment in time. History may not repeat, but Perez's model suggests that it does have a rhythm.

The differences between manufacturing physical goods, which countless organizations mastered in the last age, and producing digital experiences are vast. The historical context differs as well. Attempts to blindly replicate what worked in the Age of Mass Production for the Age of Software can be catastrophically misleading, as we will see. We need a new way to think about and manage large-scale software delivery. This book proposes that new way.

The most important part of my trip to the Leipzig plant was an uneasy realization that blindly following in the footsteps of manufacturing is fraught with as much peril as not following them at all. Software production is vastly different—as the size of a software system grows, our ability to improve and manage it declines. Even so, we have learned to scale electrical distribution, car production, and other complex manufacturing processes, and we will learn to scale software production too. The problem is that, aside from a handful of exceptions, such as the tech giants, the vast majority of organizations have not yet learned how to effectively scale software delivery systems.

The market demands of large-scale software delivery present problems that most organizations are not equipped to handle. We need a new set of business concepts to understand software delivery at scale and a new kind of framework to manage it so that our businesses have the plasticity to evolve. Part I of this book examines the reach and the urgency of the problem and introduces the framework to overcome it.

In Part I, we will cover:

- *The reasons why your business will be affected by digital disruption, and how your thinking needs to change to survive the next ten years*

- *The three types of disruption and which one applies to your business*
- *An overview of the "Deployment Period" of the Age of Software and why understanding that matters for how you approach your digital transformation*
- *An introduction to the Flow Framework and the concept of software value streams*
- *An overview of the four flow items that define the delivery of business value*

CHAPTER 1

The Age of Software

Each technological revolution has resulted in the disruption ofexisting businesses by those who have mastered the new means of production. For example, in a matter of a few years, Uber demonstrated that a single, well-designed screen that is deployed at internet scale can disrupt an entire industry. An explosion of startups is threatening every aspect of every business as venture capital fuels the disruptions. In parallel, the tech giants continue to grow into new markets. Google and Facebook dominate nearly 90% of global spending on digital advertising,[1] while Amazon is on track to own a majority of the retail business and to use that leverage to expand into adjacent markets.[2] Every business leader needs to figure out when and how this affects them or risk their organization not surviving the next decade.

Each year, the stories and statistics look more dire. In 2017, the CEO of Equifax lost his job due to a security breach. Then, in a congressional hearing, he blamed the problem on a single software developer.[3] No CEO from a company that mastered the Age of Mass Production could blame such a cataclysmic business failure on something so seemingly trivial and manageable in their production system. (We will deconstruct the Equifax scenario further in Chapter 6.)

What's become clear is that no sector of the economy is safe, that the disruptions are accelerating, and that the very talented and highly trained business leaders responsible for the majority of the world's economy do not have the right set of tools and models to properly assess risk and capitalize on opportunity in the Age of Software.

The topic of *digital disruption* is not new and has been well documented. However, the significance of Carlota Perez's work is that

disruption is a predictable outcome for companies that do not adapt to the new means of production. Companies that master the new means of production, even in slower moving parts of the market, will displace those that take more time to adapt. For example, an insurance company that provides a first-rate digital experience will displace the one that does not. And if the insurance sector itself does not move fast enough on the digital front, one of the tech giants could move into that market in search of a new vehicle for growth, turning displacement of specific companies into the disruption of an entire market. The examples in this chapter will highlight that no business is safe—no matter which industry sector or market it is in—because we are heading through the Turning Point of the Age of Software.

This book is not about business strategies for dealing with disruption. Books such as Geoffrey Moore's *Zone to Win: Organizing to Compete in an Age of Disruption* discuss how to adapt your business strategies to play disruption offense or defense. However, the problem is no longer that companies are unaware of their vulnerability to disruption. In 2017, for the first time, non-tech Fortune 500 companies made more investments in technology companies than technology companies did.[4] This is a sign that company leaders are starting to see the scope of the disruption and are using that to drive their digital transformations.

However, as we will learn in this chapter, the problem is that those transformations are failing. The main reasons for failure, according to *Altimeter*'s report "The 2017 State of Digital Transformation," are a shortage of digital talent and expertise (31.4%), general culture issues (31%), and the treatment of digital transformation as a cost center instead of an investment (31%).[5] None of these are root causes of transformation failure; rather, they are symptoms of a fundamental disconnect between business leaders and technologists that we will review throughout this book. This is not to say that talent is not critical; it is. But a project-oriented approach creates the conditions that push talent out of the organization instead of attracting talent to it.

The bottom line is that even with the best strategies and intentions, software delivery capacity and capabilities are a bottleneck in the digital transformation. Too little happens too slowly, and the business side does not understand why or what to do about it.

This chapter will start by reviewing how digital disruption is affecting the different sectors of the economy. It will address three types of disruption and discuss how software delivery capabilities are critical to navigating each. A summary of past technological revolutions will identify what we need to navigate this one. The chapter will conclude by introducing the "three epiphanies" that led to the Flow Framework—the mechanism for breaking the transformation failure cycle and ensuring that your company can survive the next ten years.

BMW TRIP Toward the Production Line

Entering the BMW Group Leipzig plant in Germany is an awe-inspiring experience. My hosts are Rene Te-Strote and Frank Schäfer. Frank is a plant manager responsible for overall vehicle integration. The enormous Central Building was designed by architect Zaha Hadid, who designed some of the most unique buildings of our time. The unapologetically sci-fi architecture invokes the feeling of walking into the future. The most prominent sight upon entering is an elevated and exposed section of the production line that towers high above eye level (Figure 1.1). Car bodies move across a suspended conveyor and then slowly disappear out of view as they glide over a sea of desks. The production line is visible to anyone who enters the building and to all the staff, and the entire building is designed around it. Every part of the building has some practical aspect related to manufacturing and value delivery. Everything embodies the maturity and scale of one of the masters of the Age of Mass Production.

Gene Kim, a mentor and coauthor of *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win* and *The DevOps Handbook*, once told me that we may only be 2% of the way there in the maturity of DevOps adoption.[6] This statement shocked me, but it also explained so much in terms of the glacial pace with which many traditional businesses move through the Age of Software. It's the slow rate of progress across

the industry that is even more disconcerting than the 2% number itself. I became motivated to see firsthand what the culmination of the Age of Mass Production looked like so that I could extract every ounce of learning from it and apply those concepts to the Age of Software.

Figure 1.1: The BMW Group Leipzig Plant Central Building (with permission of the BMW Group)

A year prior to my visit to the Leipzig plant, the BMW Group celebrated its hundred-year anniversary with the "Next 100 Years" event, which recognized the past century of manufacturing excellence and presented the BMW Group's vision for the future of mobility. The event began with a quote from Alan Kay, of Xerox PARC fame, stating that "the best way to predict the future is to invent it."[7] What struck me most was just how different the next hundred years would be from the last.

The automotive industry is currently at an inflection point, where software-based innovation is starting to overtake the incremental gains in engine performance and other physical aspects of the car experience.

In 2017, the market cap of Tesla overtook Ford. Investors were betting big on the yet-unrealized potential for change embodied by Tesla, given that in 2016 Tesla produced 76 thousand cars versus Ford's 6.7 million and saw revenues of $7 billion versus Ford's $152 billion.[8]

In the Next 100 Years presentation, the BMW Group made it clear they were staying well ahead of the curve, building on their accomplishment of how quickly they brought the electric i3 and i8 cars to market. But that was not the most interesting part of the Next 100 Years vision, which projected a future of intelligent assistants, augmented and autonomous driving, and novel solutions to mobility that reframe the notion of car ownership.[9] The most interesting part of the Next 100 Years vision was that the innovations the BMW Group forecast were all powered by software, as underscored by an announcement from the BMW Group's CEO that said in the future the BMW Group expected more than half its staff to be software developers.[10]

I have witnessed similar inflection points at most of the Fortune 500 companies that I visit, regardless of the market segment they are in. Is every industry going to be disrupted in this way, where more than half the staff in years ahead are going to be IT professionals? Given Gene's "2% of the way there" comment, are all of these enterprise organizations prepared for that shift, from a company organization and management point of view? Did the BMW Group have some fundamental advantage, having mastered the last great technological revolution? What could we learn from the way this plant operated, and could we apply it to the way large-scale software is built?

No Sector Is Safe

Over the past two decades, the first companies exposed to the shift to digital communication and collaboration, such as Kodak and Blockbuster, were some of the first victims of disruption. The difference now is that the entire economy is exposed to disruption.

Consider the four economic sectors as laid out by Zoltan Kenessey. The primary sector involves resource extraction from the planet, the secondary involves processing and manufacturing, the tertiary involves services, and the quaternary involves knowledge work.[11]

The ability to improve discovery, extraction, and logistics through software gives some companies in the primary sector fundamental advantages over those who have not mastered these software-based solutions. While advances in extraction and technologies yield only incremental gains, software and IT systems can drive more transformational discoveries and efficiencies. For example, natural resource and energy companies are increasingly competing with software and data-driven approaches to discovery and extraction. The bottom line is that no business or sector is safe from digital disruption, even though the pace of the disruption will vary across sectors and businesses.

As we move out of the primary sector into the secondary, the shifts resulting from the Age of Software become more dramatic. Mass-manufactured goods, such as cars, have become commodities whose differentiation increasingly comes from a digital experience. Cars are now computers on wheels. Whereas your laptop's Microsoft Windows operating system may have sixty million lines of code,[12] in 2010 cars already contained around one hundred million.[13] For many automotive manufacturers, that makes the unit cost of the software in the car more expensive than the engine (Figure 1.2). This is only the start of the disruption. Autonomous drive systems will multiply the amount of software in the car and electrification of the engine is turning the rest of its core components into software systems as well.

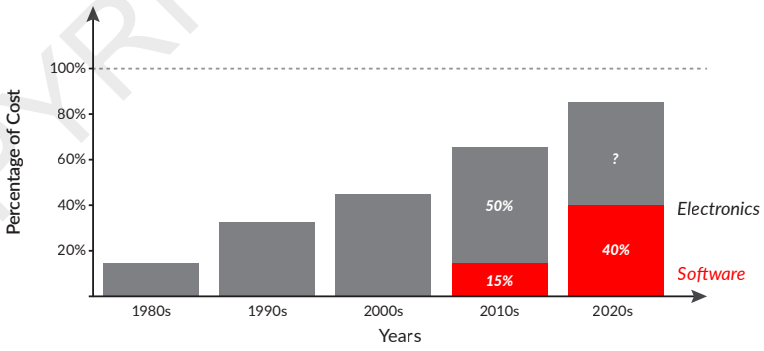


Figure 1.2: Software as Approximate Proportion of Car Cost

Bosch, one of the companies that embody the Age of Mass Production, announced in 2017 that it was hiring 20,000 specialists for its digital transformation, with nearly half of the jobs related to software.[14] Our physical products increasingly rely on a connected experience, and, as we'll learn in Chapter 7, factories, manufacturers, and assembly processes themselves are being transformed by software.

The digital disruptions and displacements we are witnessing in the tertiary sector (services) are spectacular. The software side of this story begins in 1997 with the transformation Netflix brought to the movie rental business. In those days, internet bandwidth was too scarce to deliver digital movies to the home. Inspired by computer scientist Andrew S. Tanenbaum's famous math problem, which asked students to figure out the bandwidth of a station wagon carrying tapes across the US, Reed Hastings (co-founder and CEO of Netflix) determined that software could be applied to the selection and logistical distribution of DVDs.[15] Not long after, venture capitalist Marc Andreessen described FedEx as "a software network that happens to have trucks, planes, and distribution hubs attached" in his seminal essay "Why Software is Eating the World."[16] What Netflix and FedEx realized is that software can yield exponential gains in logistics.

Even so, we are still at the early stages of disruption in retail and logistics. Amazon is now able to combine supply-chain data with both logistics and consumer spending habits, which could disrupt storefronts themselves. Just as Walmart disrupted other retailers by mastering the methods of the Age of Mass Production, Amazon is now doing the same to Walmart and other retailers by disrupting the supply chain and seeing a nearly identical market-share growth profile.[17] What's clear from these trends is that the companies that achieved even a small and early edge on applying software to consumer experiences and logistics have won a seemingly insurmountable edge over the rest of the industry.

Finally, the quaternary sector—composed of knowledge-work industries such as technology, media, education, and government—moves at an even faster rate of digital change by virtue of being the newest and most malleable sector, in terms of how software can affect distribution and infrastructure. For example, we have already seen

multiple waves of collaboration technology within the Age of Software, ranging from email to instant messaging to teleconferencing and digital assistants. Whatever the sector, we now have enough data points from the disruptions to make the trend clear. It's happening in every sector, and the pace appears to be increasing. Companies that have harnessed software innovation are the winners, leaving those who fall behind to decline or get "Blockbustered."

Types of Disruption

While examples of disruptions abound, not all disruptions are equal. In order to understand how the impact of software delivery will affect your business, we need a model of the different kinds of digital disruptions. In *Zone to Win*, Geoffrey Moore provides a model that I will build on in this book. Moore's three types of disruption are the *Infrastructure Model*, *Operating Mode*l, and *Business Model* disruptions.[18] For most companies in the Age of Software, addressing disruptions will require mastering software delivery. However, the type of disruption that your business needs in order to play offense or defense will determine how you shape your IT investments and value streams.

Infrastructure Model disruptions are the easiest to address. They involve changes in how customers access a given product or offering; for example, requiring your products to be marketed via social media but not changing how you sell. The Infrastructure Model disruptions that we are seeing are competition-differentiated digital marketing and communication. Digital services are blending with the social networks that underpin the trust economy.

Operating Model disruptions rely on using software to change the relationship of the consumer with the business. For example, airlines must now provide a first-rate mobile phone experience or risk losing passenger bookings. The Operating Model here changes fundamentally, as agents and call centers play a diminishing role. Competing with startups on this front requires a first-rate digital experience. For example, a consumer bank can expect to be disrupted by new financial services startups that delight their users with better personal-finance management features.

Business Model disruptions involve a more fundamental application of software and technology to a business. This might mean a software and logistics innovation that cuts out a major manual step of consumers getting goods, such as a store visit. Moore states that established enterprises are not capable of disrupting themselves and, as such, must establish an innovation engine that will allow them to catch the next wave of disruption emerging in another market category.[19]

Whatever the examples are for your business, in order to win in the Age of Software you must precisely define which type or types of disruption put your business at risk. Wherever you land, the next step involves a significant investment in software delivery. The success of that initiative and your ability to win your market will be determined by your ability to define, connect, and manage your software value streams.

Unbundling of Every Industry

Consider your next car. Dozens of startups are competing to win a portion of your experience with automotive mobility. This ranges from connected car technology and autonomous driving to management of tire inflation and car repair. Each of these startups is competing with each other and with the incumbent automotive vendors who have traditionally owned all aspects of the car users experience. In Zone Management terms, these are only Infrastructure Model disruptions. Companies like Lyft, Uber, and Car2Go are in the midst of both Operating Model and Business Model disruptions by changing consumers' relationships with cars at the ownership level.

Shifting to finance, banks have been one of the leaders in technology adoption. Their business is fundamentally a knowledge-work business, and any technological advantage they can derive can quickly lead to a market advantage. For this reason, banks have been early adopters of new technologies; for example, many were running open-source software years before other enterprises had even considered using unsupported software produced by a collective of individuals who are not paid for their efforts. For the past decade, banks have been staffing ahead of digital disruption by hiring tens of thousands of

IT workers into their organizations. As an example of scale, Catherine Bessant, the head of Global Technology and Operations at Bank of America, has 95,000 employees and contractors reporting to her.[20]

There are hundreds of startups with venture funding who are targeting key aspects of a bank's business. Each offers a different product or service that exists because it stands to compete with those offered by the incumbents. Given the amount of financing we're seeing in finance technology, a report in 2016 found that over 1,000 fintech (finance technology) companies raised $105 billion in funding that's valued at $867 billion.[21]

A fundamental shift is happening. Even when staffed with tens of thousands of workers, the incumbents create and deliver software at a rate that appears to be leaving large and numerous doors open to disruption. And it's not for lack of discussion or investment to stay ahead of the startups or lack of finding creative ways to play offense or defense to the disruptors. Thanks to their ability to build and iterate on software quickly, and because they are unencumbered by existing customers or legacy systems, startups are innovating how customers interact with their products in services ranging from health insurance to cryptocurrencies.

The examples above illustrate how startups, financed by venture capital, are disrupting entrenched businesses and industries. The other vector of disruption is the tech giants that have mastered software production. Whatever the vector of disruption, before setting a plan of action we must examine where we sit in this technological revolution and what the upcoming wave of change and disruption will look like.

We Are Entering the Deployment Period

Will this pace of disruption continue indefinitely at its current pace? Will the amount of venture capital fueling the growth of startups continue to rise to the point where it is futile for entrenched companies to compete? Will the majority of the economy soon be owned by the tech giants?

Answers to these questions are consequential for our organizations, as they can help guide what and how we invest in our digital strategies in order to survive the Turning Point, the period between the Installa-

tion and Deployment Periods that is marked by a financial crash and recovery. Without understanding these questions, our efforts could be as misguided as investing in teams of horses to compete in the Age of Steam & Railways. And as we will see in the next chapter, that is precisely what the majority of enterprise IT organizations are doing.

Several theories exist to explain technological innovation cycles and their impact on the economy. Kondratiev waves (described as fifty-year cycles of expansion, stagnation, and recession that result from technological innovation and entrepreneurship) and cycles of *Creative Destruction* (the process of industrial mutation that incessantly revolutionizes the economic structure from within, destroying the old one and incessantly creating a new one) are concepts introduced by Joseph Schumpeter in the 1930s in his book *Capitalism, Socialism, and Democracy*.[22] The economist Carlota Perez has expanded on these concepts in her profound book *Technological Revolutions and Financial Capital*.[23]

There is disagreement among economists as to the Kondratiev waves' cause and duration. For example, while some predict that the current wave will be longer, others propose that the waves have been shortening consecutively.[24]

Though the exact length of the current wave cannot yet be determined, Perez's work provides us with a model for distinguishing between the technological systems of the last age from this one, and for using that to better understand how to master the means of production in the Age of Software. (A summary of Perez's work is available in the blog post "The Deployment Age" by Jerry Neumann, a prominent venture capitalist.)[25]

For the purpose of understanding where we are within the Age of Software, the most important aspect of Perez's theory is the concept of the Installation Period and the Deployment Period of new technological systems (Figure 1.3). In the Installation Period, large amounts of financial capital, such as venture capital, are deployed to leverage the new technological system that has formed a critical mass of technology, companies, and access to capital in order to disrupt the age that came before it. This is exactly what we have been witnessing with the Cambrian explosion of startups.

Following the Installation Period is a Deployment Period, where companies that master the means of production earn increasingly larger portions of the economy and the new infrastructure. This creates a period where *production capital*, a good portion of it controlled by the new technology giants of that age, starts displacing startups and financial capital. Production capital is different from financial capital in that it is controlled by company managers who are seeking innovations to make production more efficient by working within established companies (versus the radical and risky innovations with high multiples favored by financial capital). During the Deployment Period, financial capital and startups begin looking for a new home by placing bets on what the next technological revolution could be.

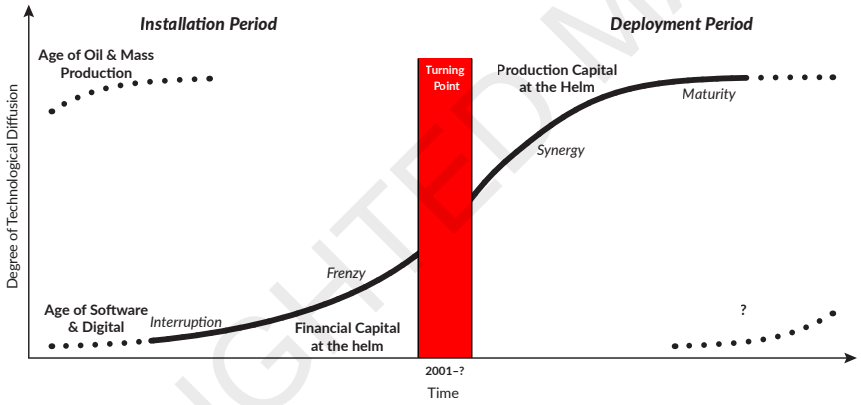


Figure 1.3: From Installation Period to Deployment Period[26]

This pattern has repeated itself four times over, as is visible in Table 1.1. In her book, Perez provides evidence that we are in the midst of the fifth iteration.[27] The Age of Software began around 1970, with the introduction of the microprocessor. In 2002, Perez predicted that around the Turning Point, the point between the Installation Period and Deployment Period, the new masters of production would amass enough wealth and control that governments would start imposing regulations, and we are seeing evidence of that today.[28]

We cannot conclusively determine where exactly we are with respect to the Turning Point or how long it will last. We do not know whether age will be materially different from others. In an interview, Perez told me that this particular Turning Point appears to keep drawing itself out longer and longer.[29] However, by the time we know the precise shape of this age, it will be too late to do anything about it, as those who were early to mastering the new means of production will have displaced those who were too late.

Installation-Deployment	Age	New Technological Systems	New Infrastructure	Triggering Innovations	Managerial Innovations
1771–1829	Industrial Revolution	Water-powered mechanization	Canals, turnpike roads, sailing ships	Arkwright's Cromford Mill (1771)	Factory Systems, entrepreneurship, partnerships
1829–1873	Age of Steam & Railways	Steam-powered mechanization and transport	Railways, telegraph, steam ships	Liverpool-Manchester Railway (1831)	Joint stock companies, subcontracting
1875–1918	Age of Steel and Heavy Engineering	Electrification of equipment and transport	Steel railways, steel ships, global telegraph	Carnegie's steel plant (1875)	Professional management systems, giant firms Taylorism
1908–1974	Age of Oil & Mass Production	Motorization of transport and economy	Radio, motorways, airports	Ford's Highland Park assembly line (1913)	Mass production and consumption, Fordism, Lean
1971–?	Age of Software & Digital	Digitization of the economy	Internet, software, cloud computing	Intel microprocessor (1971)	Networks, platforms, venture capital

Table 1.1: Technological Revolutions[30]

From today's vantage point, we see ongoing examples of financial capital at play, such as the stories of the "unbundlings" of various industries caused by the number of new startups funded by venture capital. However, we are also starting to see the effects of production capital as well.

Consider Jawbone, a nimble digital native funded by top-tier venture capitalists with no lack of access to capital. Jawbone created multiple category-defining products, ranging from Bluetooth headsets to wireless speakers to wearable fitness trackers. In total, Jawbone raised $930 million of financial capital between 2006 and 2016 from top-tier firms, but ultimately it ended up in an asset sale, making it the second-costliest venture-capital-backed startup of all time.[31]

Even with the combination of innovation and great products, Jawbone lost out to production-capital companies like Apple. The smartwatch innovator Pebble closed its doors in December 2016 for similar reasons.[32] In addition to this growing graveyard of consumer hardware startups, it is becoming increasingly difficult to launch a new social media company that gets to scale before Facebook either acquires or destroys it.[33] These growing effects of production capital are signals of our passing through the Turning Point.

While we do not know how long this Turning Point will last, if it follows Perez's model of a five-decade cycle and we know we have been seeing the signs of the Installation Period since the 1970s, then we can assume that with each year we are getting closer to the Deployment Period. Once we reach the Deployment Period, companies that have not adapted to the new means of production will decline in relevance and market share. In the next decade, a significant number will lose their place in the market. We have seen other companies try and fail to scale their IT, then Agile, and now DevOps transformations in a meaningful time frame. For many of these companies, this is the last call if they want to have a fighting chance at surviving the latest technological age, let alone the next.

Three Epiphanies

My career has been dedicated to understanding and improving how large-scale software is built. I spent nearly two decades working on new programming languages and software development tools, and have had a chance to work with some of the best technologists in the world. But I have come to realize that, due to where we are in the Turning Point, technology improvements are no longer the bottleneck.

Technology improvements will be relevant but incremental, yielding productivity gains of less than ten percent to organizations via new programming languages, tools, frameworks, and runtimes.

In contrast, the disconnect between the business and IT is massive, as are the disconnects within IT organizations. The common approach to enterprise architecture is wrong, as it tends to focus on the needs of the technologists and not on the flow of business value. Contrast this with the BMW Group's Leipzig plant, where the entire plant is designed around the changing needs of the business, from the extensibility of the buildings to the modularity of the production lines themselves.

For me, the realization that technologists' pursuits were bringing diminishing returns did not come as a single eureka moment. Rather, I had separate realizations, each of which caused me to make a major pivot in my career. Each of these "epiphanies" involved a collection of experiences that reframed my view of software delivery and kept me awake through the night as I slowly digested how many of my previous assumptions were flawed.

The first epiphany came from my first job as a developer working on a new programming language. During that time, I realized the problem we were solving spanned well beyond the source code. The second epiphany came from a culmination of hundreds of meetings with enterprise IT leaders that made it clear to me that the approach to managing software delivery and transformations was fundamentally broken. The third epiphany came during my visit to the BMW plant and revealed that the entire model that we have for scaling software delivery is wrong. (I will expand on these epiphanies in Part III.) Each epiphany is connected by our trying—and failing—to apply concepts from previous technological revolutions to this one. To summarize, my three epiphanies were:

- **Epiphany 1:** Productivity declines and waste increases as software scales due to disconnects between the architecture and the value stream.
- **Epiphany 2:** Disconnected software value streams are the bottleneck to software productivity at scale. These value stream disconnects are caused by the misapplication of the project management model.

- **Epiphany 3:** Software value streams are not linear manufacturing processes but complex collaboration networks that need to be aligned to products.

The first epiphany—that software productivity declines and waste increases when developers are disconnected from the value stream—came as the result of a personal crisis. While on the research staff at Xerox PARC, I was an open-source software developer and consistently worked seventy to eighty hours per week. Most of that time was spent coding, plus regularly sleeping under my office desk to complete the cliché. The number of hours at the mouse and keyboard resulted in a seemingly insurmountable case of repetitive strain injury (RSI). It grew progressively worse, along with the heroics and coding required to get release after release out, and my boss repeatedly cautioned me that he'd seen several PARC careers end in this way. With the staff nurse offering little help beyond advising caution and providing ibuprofen, I realized that every single mouse click counted.

This led me to do PhD research by joining Gail Murphy and the Software Practices Lab that she created at the University of British Columbia. As mouse clicks became my limiting factor, I started tracking the events for each click by instrumenting my operating system, and I came to realize that the majority of my RSI-causing activity was not producing value; it was just clicking between windows and applications to find and refind the information I needed to get work done.

I then expanded my research to six professional developers working at IBM, and I extended the monitoring and added an experimental developer interface for aligning coding activity around the value stream. The results were surprising to both Gail and I, so we decided to extend the study to "the wild" by recruiting ninety-nine professional developers working within their organizations and having them send before-and-after traces of all of their development activity. (The full findings are detailed in Chapter 7 and were published at the International Symposium on Foundations of Software Engineering.)[34]

The conclusion was clear: as the size of our software systems grew, so did the distance between the architecture and the effort it took to add one of the hundreds of features being requested by our end users.

The number of collaboration and tracking systems we used grew as well, causing yet more waste and duplicate entry. These findings were the inspiration for Gail and I to found Tasktop, a software company dedicated to better understanding this problem.

Several years later, while getting an overview of a large financial institution's toolchain, I had the second epiphany. This problem of thrashing was not unique to developers; it was a key source of waste for any professional involved in software delivery, from business analysts to designers, testers, and operations and support staff. The more software delivery specialists involved, the more disconnects formed between them and the more time was spent on thrashing, duplicate data entry, or the endless status updates and reports.

The challenges I was personally facing from my declining productivity and increased thrashing were being mirrored, at scale, across thousands of IT staff. The more staff, the more tools, and the more software scale and complexity, the worse this problem became. For example, after conducting an internal study on one bank's software delivery practices, we determined that, on average, every developer and test practitioner was wasting a minimum of twenty minutes per day on duplicate data entry between two different Agile and issue-tracking tools. In some cases, that grew to two hours per day, and the overhead for first-line managers was even higher. When we dug deeper into how developers spent their time, we found that only 34% of a developer's active working time at the keyboard went to reading and writing code.[35] Yet this is what developers are paid to do and what they love to do. This was a deep and systemic problem.

As Gail and I started working more with enterprise IT organizations, we realized just how different this world was from the much simpler and more developer-centric world of open source, startups, and tech companies, but we lacked empirical data on enterprise IT delivery. At the BMW Group plant, I was simply able to look down at the line to see the flow of work. Unfortunately, no data was available on how work flows across the tools that form a value stream across enterprise IT organizations. But we now had a broad enterprise IT customer base, including close to half of the Fortune 100, and realized that we had a very unique data set, as the majority of those organizations had shared

with us all the tools involved in their value stream and the artifacts that flow across those tools. We collected and analyzed 308 Agile, Application Lifecycle Management (ALM), and DevOps toolchains from these organizations. We started calling these *tool networks* once we saw how the tools were interconnected. (See Chapter 8 for more.) In the process, I personally met with the IT leaders of over two hundred of those organizations to better understand what we were seeing in the data.

With those 308 value stream diagrams in mind, while walking over ten kilometers (about six miles) of the Leipzig plant production line, I felt the kernel of the third epiphany form. The entire model for how we think about a software value stream is wrong. It is not a pipeline or a linear manufacturing process that resembles an automotive production line; it is a complex collaboration network that needs to be connected and aligned to the internal and external products created by an IT organization, and to business objectives.

This is what the data was telling us, yet this approach is completely at odds with the project- and cost-oriented mentality with which enterprise organizations are managing IT investment. The ground truth (that is, the truth learned through direct observation) of these enterprise tool networks is telling us that all the specialists in IT are already starting to work in this new way by adopting Agile teams and DevOps automation, but these specialists lack the infrastructure and business buy-in to do so effectively.

On the flip side, the business is further losing the ability to see or manage the work that the technologists are doing. Leadership seems to be using managerial tools and frameworks from one or two technological ages ago, while the technologists are feeling the pressure to produce software at a rate and feedback cycle that can never be met with these antiquated approaches. The gap between the business and technologists is widening through transformation initiatives that were supposed to narrow it. We need to find a better way.

Conclusion

An amazing effect of the BMW Group Leipzig plant is that it places visitors and employees at the juxtaposition of the last phase of the

Deployment Period of the Age of Mass Production and the Installation Period of the Age of Software. You can watch the culmination of advanced manufacturing and factory automation producing cars that are increasingly powered by software. In contrast, the world's top enterprise IT organizations are more like the nearly three hundred car manufacturers that were trying to master production in Detroit in the 1900s but went extinct while the likes of Ford pulled ahead.[36]

In this chapter, we saw the scale of the change that is happening due to the maturation of the Age of Software. The biggest problem is that, at a managerial level, established businesses are using the managerial and production methods of previous ages, and thus, are failing in this one. In the next chapter, we will dive into the evidence collected from enterprise IT organizations that shows the symptoms of this problem, and we'll highlight the cause of organizations' failure to transform. After that, we'll explore the solution.

CHAPTER 2

From Project to Product

Transformation may be the most overused term in IT. However, when looked at through the historical lens of technological revolutions, the overuse is less surprising, as it is rooted in an existential problem for companies faced with the need to embrace change in order to survive the Turning Point.

In this age, survival is dependent on an organization's ability to deliver software products and digital experience. Underscoring the scale and urgency, IDC, a market research firm, has estimated digital transformation as an $18.5 trillion opportunity by 2020, which represents 25% of the global GDP.[1] Those that succeed will reap the rewards and displace those that do not. Many large organizations have already kicked off their transformation initiatives; others are noticing their software investment creeping up on them, with CFOs often the first to realize how much of next year's budget and headcount is related to IT.

In the Age of Mass Production, IT was a separate silo that played a supporting function, enabling the productivity of other means of production, such as facilitating communication or sales force automation. IT will continue to play a key supporting function in that regard; for example, the kinds of factory automation envisioned by Industry 4.0 will yield significant productivity results in mass production through the "cyber physical systems" proposed in Industry 4.0 initiatives.[2] However, these are extensions of the last Installation Period, and they are less disruptive than the change in markets and business models that are now happening. But in the Age of Software, digital technology has become the core of the organization and cannot be compartmentalized to an isolated department.

So, how do our organizations and managerial techniques need to adapt? In this chapter, we will examine two transformations that failed even though they were formed with the best of intentions for surviving the Age of Software. The first was an Agile transformation failure that contributed to Nokia losing the mobile market. The second involves a large financial institution we'll call LargeBank that spent $1 billion on an Agile and DevOps transformation without delivering any measurable increase in the delivery of business value. Both of these failures share a common thread of how paradigms that worked in previous ages can fail us in this one.

Next, we'll discuss the project management paradigm and why it creates a chasm between the business and IT. We'll look at the creation of the Boeing 787 Dreamliner and contemplate how it exemplifies product thinking. Then we'll review why moving from project-oriented management to product-oriented management is critical for connecting software delivery to the business, and how shifting our managerial perspective from project to product paves the path for success in the Age of Software.

Before we dive into the Flow Framework and its inception, it's important to look at how operations at the BMW Group Leipzig plant illustrate the success of a different way of measuring flow and of defining software value streams along product lines.

BMW TRIP Discovering the Plant Architecture

The Central Building opens into an enormous open space. To the left is the exposed production line, with car bodies moving steadily past as large, orange robotic arms swivel and dip to put the cars together piece by piece. Combined with the futuristic architecture, the entire space has the feeling of entering a building where the next version of the starship *Enterprise* could be manufactured.

"Is it possible that a certain kind of architecture can positively influence teamwork and productivity within a plant?" the plant brochure asks. "The Central Building of the BMW Group

Leipzig plant, designed by the famous architect Zaha Hadid, is the implementation of this idea. This unique building is the center of communication, and it connects all production areas."

All of IT sits to the right of the exposed production line.

"The plant CIO's desk is right over there." Rene points to a sea of several hundred desks and dual-monitor workstations to the right of the exposed production line.

It hadn't occurred to me that the plant would have its own CIO with a sizeable IT infrastructure and staff, but given the scale of operations that is visible, there must be countless internal applications managing everything from supply chains to final assembly.

Each of the people is wearing a blue vest, blue jacket, or entirely blue jumpsuit. Some of the blue apparel hangs on chairs and desks. Rene hands me a vest with my name embroidered on it, which gives me the feeling of belonging in the building the moment I put it on.

"These vests are antistatic," Frank says. "You must wear them at all times around the production line. We will also attach special static dischargers to your shoes." He hands me discharge stickers from his vest pocket.

"All plant staff wear these, including the IT staff, the CIO, and the CEO," says Rene.

Most of the startups I have visited have branded clothing to communicate their identity and culture, with T-shirts and hoodies being the most common. But there is something more to the form and function of these vests.

We watch the 1- and 2-Series cars moving along the production line. Rene explains, "In 2017, we produced 980 cars each day, with a new car being completed every seventy seconds. Everything that you are about to see ensures we can achieve these production rates and flow. Later in the day, we will also see one of the newest innovations of the plant, the production of the i3 and i8."

I recall from the brochure that the plant was a two-billion-euro investment representing the "peak of production, automation, and sustainability."[3] That cost is in the ballpark of a semiconductor

fabrication plant. But whereas modern chip "fabs" are built around creating the same cutting-edge processor over and over, something different is happening here.

Each vehicle is made specifically to a customer's order. This exemplifies the idea of "just in time." Just-in-time approaches delay processing and other work until the last moment the work can be done economically, thus optimizing workflow and resource use.

"Not only does the plant implement just-in-time inventory," Rene continues, "the cars are manufactured just-in-sequence."

"The cars come off the production line in the same sequence that customer orders are placed. Each car is tailor-made to the customer's specifications and preferences," Frank adds.

"The cars remain in the same order on their entire journey on the production line?" I am having trouble grasping how they could implement this.

"Interesting question," Frank continues. "No, there is one part of the production line where the car bodies need to be taken out of sequence. Then they need to be temporarily inventoried, and after that, put back into sequence. It is a very complex process and is the bottleneck of our plant."

"So, where is the bottleneck?" I ask, in what must have registered with Frank as a combination of overeagerness and naiveté.

"This entire building is designed around the bottleneck. But before we look at that, let's go to the Assembly Building," Frank says.

As Frank leads us along the exposed production line, I take out my phone and open up maps in satellite mode. I see dozens of large and interconnected buildings (Figure 2.1). The arrangement of buildings looks strikingly similar to the computer motherboards that I used back in the days when I would assemble my own PCs—so much so that I do a double take and stop walking for a moment. The Central Building looks like a CPU and its interconnects.

"Ah, yes, there we are," Frank says as he uses the satellite map to explain the plant's layout. "Here you see the Central Building,

and you can see that we are nearing the Assembly Building. The structure of the Assembly Building is very interesting," Frank continues. "We call this the 'five fingers' structure."

The massive building did indeed have the shape of rectangular fingers and a hand.

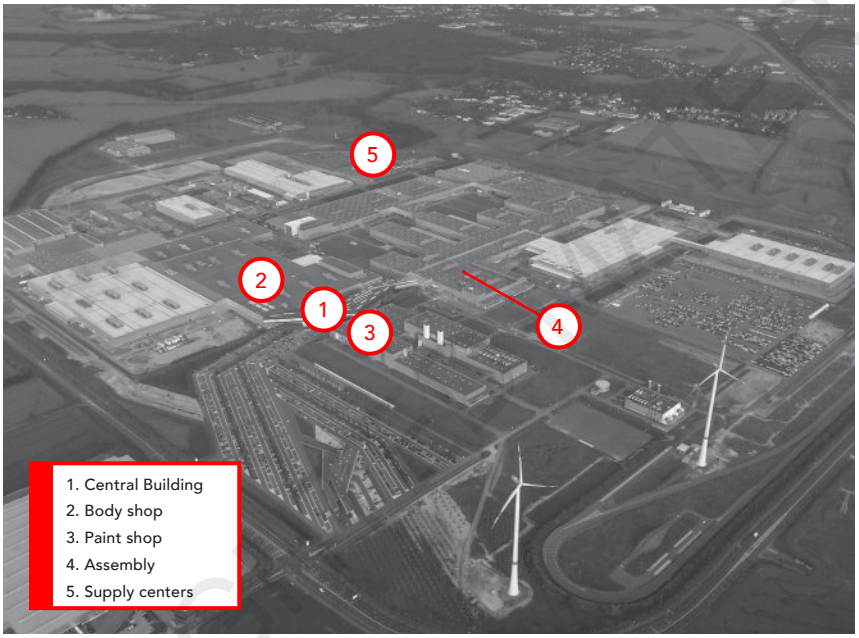


Figure 2.1: The BMW Group Leipzig Plant (with permission of the BMW Group)

"In software architecture, you have extensibility," says Rene. "Maybe it is hard to see, but this plant is also architected for extensibility along its main production lines."

"Yes," adds Frank. "When additional production steps are added to the line, we are able to extend the length of the 'fingers.' The buildings have been extended over time as we have expanded and added more automation and more production steps. You see that the 'fingers' are different lengths."

Frank then points at a building that stands out from the rest. It is white in color but also attached to the "hand."

"The 'hand' houses the 1- and 2-Series production lines," Frank says. "That building is a newer one. It is where we make the i3 and i8 electric cars."

I want to get a better look at this building and instinctively clicked the "3-D" button on the map. It goes into a "flyover" mode and starts navigating around the building.

"Look there," says Frank. "You see those trucks?"

He points at large trucks that are connected right to the "fingers" of the assembly building.

"As Rene mentioned earlier, the plant functions with just-in-time inventory. Any stockpiled inventory would be waste. So, the parts are delivered 'just in time,' right to the part of the assembly line where they are used. The BMW Group has around 12,000 suppliers worldwide, so this is quite an operation," says Frank. "Let's skip the bottleneck for now, as we have to go to that area at lunch anyway. Let me take you right to the 1- and 2-Series production line."

We walk into the Assembly Building and onto a catwalk suspended three stories in the air and looking along the length of the "hand." It is an extraordinarily vast space, and processing the scale takes a few moments. The scene is so visually complex that it is difficult to grasp. But this complexity is nothing like the chaos and clamor of Times Square on a busy summer day. Instead, perfect order and coordination of hundreds of machines and moving parts were orchestrated at what seemed a mind-boggling scale.

This massive mechanical ballet produces some of the most complex objects made by humanity. Over 12,000 suppliers, over 30,000 parts in each car, and immense functional specialization along the line, producing a new car every seventy seconds in the sequence that customer orders are made.

"By the way, Mik, you cannot take out your phone again," Frank says in a friendly but unmistakably serious way.

Agile Transformation Failure at Nokia

The idea of drawing on automotive manufacturing lessons and applying them to software is not new. Countless books on Agile methods draw on Lean manufacturing and the Toyota Production System in particular. While I was already familiar with that literature when I visited the BMW Group plant, the difference between what I thought I understood about advanced manufacturing and what I learned at the plant was enormous.

My journey using Agile methods for day-to-day software delivery started in 1999, in the relatively small scale of a single team working on an open-source project and using Kent Beck's Extreme Programming (XP) methodology. Ten years later, at the Agile 2009 conference, I presented what I learned from adapting Agile methods to an open-source project that I was leading. It was my first time at the conference, and the most interesting thread I noticed was that of scaling Agile.

Numerous consultants were using Nokia as proof that Agile development methodologies scaled to large enterprises. The "Nokia Test" was cited frequently.[4] It was a simple method of determining whether an organization was following Scrum. The test was developed by Nokia Siemens Networks, and it further cemented Nokia as the namesake and poster child of scaling Agile.

I saw the potential for scaling Agile and was thrilled when my company got the opportunity to start working with Symbian, the mobile operating system (OS) Nokia had acquired in 2008. My first meeting with a CIO was with Symbian's, later in 2009. That led to Nokia becoming Tasktop's first enterprise customer, when we supported a project on connecting Agile tools to developer workflow. Nokia and Symbian had some tremendous visionaries internally, in addition to hiring the best external contractors and thought leaders to help guide their transformation.

The problem was that the entire effort was set up for failure in spite of the leadership's best intentions and the organization's willingness to transform. A lot of energy fueled the transformation. Everyone was saying and appeared to be doing the right things, and the various consultants and vendors were indicating that they were on track.

The "Nokia Test" offered a series of questions on whether development was done iteratively and whether it followed the principles of Scrum, allowing a mechanism for testing each team to determine the state of how Agile they were. I was genuinely impressed by the sheer scale of Nokia's commitment to their Agile transformation and the degree to which the company and the teams that we worked with tracked activities to the impressive Agile model that they had created. It was clear that the executives had realized how much Agile could benefit the company in terms of their ability to adapt to the rapidly changing market.

However, as I worked with more development teams, the writing on the wall became evident. What struck me was the degree to which the activities and adherence to the model were being measured without a clear sense of the outcomes surfacing through those activities. Given that we were providing open-source tools to Nokia's developers, we started interacting more and more with the developers and noticed this disconnect became even more prominent as we worked our way toward the leaves of the organizational chart—the development teams.

In order to figure out how we could better connect the delivery layer and the planning layer, I realized it was time to get a sense of the ground truth. I asked my main contact whether I could interview some engineers across various teams to get a better sense of what was going on. The results were eye opening.

The developers I spoke to had no issue with any of the Agile practices and were mildly favorable of them; they had much bigger problems. They had major issues downstream from them because of the long build/test/deploy loop that was partly due to Nokia's otherwise formidable software security processes. They had even more significant issues with the architecture of the Symbian OS, which was making many of the changes the business wanted to bring to market difficult or overly time consuming to implement. The Symbian OS was not structured for the kind of extensibility that was needed; for example, it could not support the installation of third-party applications or what we now call an "app store."

Finally, while the developers were positive on Scrum overall, their daily work was disconnected from the higher-level planning that was

being done with a whole different set of tools. The enterprise-level Agile tool that was selected was not being used by the developers, who preferred simpler developer-centric tools. Instead, they would document the work completed for the release at the end of the iteration (or "sprint"), after the work was done, as user stories (a description of a software feature from an end-user perspective). The tool, which had all the features of a modern best-of-breed Agile tool, effectively became a documentation tool, not the mechanism for flow and feedback that had been intended.

After conducting those interviews, I realized that the transformation was in trouble. In hindsight, this was nothing like what I saw at the Leipzig plant, where every production metric relevant to the business was understood, well defined, visible, and automated. In addition, at the plant, the business side intimately understood car production. In contrast, at Nokia, the tie-in between business outcomes and software production metrics was either not explicit or nonexistent.

In every way that it was being measured, the transformation was on track—all of the right activities were happening, right down to the adoption of the Agile tool. But the developers were suffering from major friction, both in what it took to build code and in what it took to deploy it. Even more consequential was how difficult adding features had become due to the size and architecture of the Symbian OS.

If the transformation had been measured according to outcomes or results instead of activities, the picture would have looked much different. The fundamental bottlenecks that the developers were encountering would have surfaced. The investment needed in Nokia's core platform, the Symbian OS, could have been made in a way that would allow it to compete with new, software-savvy entrants to the market, like Apple. But that crucial feedback was not making it back to the business because of the way development was disconnected from the business. And the downstream disconnects and inefficiencies in building and deploying the software meant that any progress in improving this would run at too slow a pace.

At a business level and as a market leader, Nokia was well aware that it needed to move and adjust quickly in the rapidly evolving mobile ecosystem. This was the reason to roll out Agile in the first

place: to more quickly adapt to that marketplace and the growing role of software within it. Though the proxy metrics could deem the Nokia Agile transformation a success, the lack of actual business results of that transformation contributed to the business' failure and inability to shift from elegant handsets and buttons to a software and screen-centric mobile experience.

This is not to say that Nokia made no strategic missteps on the hardware front. For example, Nokia was slow to move to the capacitive touchscreens that Apple innovated with the launch of the iPhone.[5] But Nokia's strengths were on the hardware front; and in the end, they lost to two vendors with hardened software expertise when Apple's iOS and Google's Android OS took over as the mobile platforms of choice.

Nokia had an engine and infrastructure for innovating on the hardware front that was a pinnacle of the Age of Mass Production, but they did not have an effective engine and infrastructure for the Age of Software and did not have the management metrics or practices in place to realize that until it was too late.

If we step back and imagine Nokia's end-to-end value stream, the Agile transformation was a local optimization of the value stream. In other words, while a tremendous amount of investment went into the transformation, the bottleneck to delivering an operating system capable of supporting a mobile ecosystem was not the Agile teams.

Was it downstream of the Agile teams, in a lack of continuous integration and delivery capability? Was it in the architecture itself, which could not support the kinds of feature and product delivery that were needed? Or was it upstream of development and closer to the business, which was so disconnected from delivery and the architectural investment needed, such as technical debt reduction, that they did not realize Agile planning would fail to drive any of the desired results?

My interviews hinted at these issues, and I got the sense that there was no business-level understanding of what the real bottleneck was, as the gulf between what IT and developers knew and what the business assumed was so vast. That, in turn, led to the Agile transformation—implemented as a local optimization of the end-to-end value stream—yielding little result and not addressing the bottleneck.

Even if the teams had attained a theoretical ideal of agility, would Nokia have been able to adapt more quickly without upstream changes to how the business was measuring delivery? Or adapt downstream changes in how the software was deployed? Or the architecture changes that were slowing developers down in the first place?

In my opinion, that narrow-minded and activity-oriented view of Agile was the root cause of Nokia's failed digital transformation. The failed transformation made fast iteration and learning from the market impossible, as the lead times for delivering new features, such as an app store and an elegant home screen, were far too slow. This hindered the business's ability to learn and adapt, and that inability to adapt was a key factor in Nokia's downfall.

Lesson One: *To avoid the pitfalls of local optimization, focus on the end-to-end value stream.*

In the context of a software value stream, the concept of "end-to-end" includes the entire process of value delivery to the customer. It encompasses functions ranging from business strategy and ideation all the way to instrumentation of usage to determine which values were most adopted by the customer base. It is this end-to-end process that we need to understand and find bottlenecks in before considering the optimization of any particular segment of the process, such as feature design or deployment.

Contrast the approach that Nokia took with the BMW Group story earlier in this chapter. The entire Leipzig plant is designed to make the value stream visible, and the buildings are architected around the bottleneck. The architecture of the buildings is extensible to support the evolution of production technologies and changes in market conditions. Nokia had this level of maturity for its devices, but in spite of mastering mass production, it was not able to make the pivot into applying these lessons to software delivery.

Next, we will further analyze the reasons why this disconnect between the business and IT created an environment in which the business was set up for failure in its efforts to undergo a digital transformation.

DevOps to the Rescue?

It is tempting to blame the failure of Nokia's software transformation on Agile or Scrum. But this argument is just as flawed as claiming that Nokia was a success case for Scrum in 2009. Nokia's problems were not with Agile or Scrum; many organizations have had significant success adopting the exact methods that Nokia adopted with Scrum. No matter how effective Agile or Scrum could have been for Nokia, the organization's problems lay beyond the boundaries of Agile development teams.

Eliyahu M. Goldratt's theory of constraints and its applicability to Agile software development are discussed in Kent Beck's book *Extreme Programming Explained*.[6] Goldratt famously explained how investments made in areas other than the bottleneck are futile.[7] This was the futility of Nokia's Agile transformation. Nokia could have had the most supportive leadership and culture on the planet, transformed twice as fast, achieved twice the agility, and invested twice as much into the Agile transformation and still have seen no change in the slope of their decline due to the fact that the effort was not being applied at the bottleneck. What's worse, the outcomes of the effort were not being measured.

Could adoption of DevOps practices such as continuous delivery have turned the tide at Nokia? Possibly. Some of these practices were already in place, like automated testing. In my interviews, the subjects reported major inefficiencies that would have been addressed by the other key practices summarized in *The DevOps Handbook*—the automation of the entire deployment pipeline and support for small batch sizes.[8] In my experience, those practices are critical to an effective value stream, and if they are not adopted, it is only a matter of time before they become the bottleneck.

However, it would be incorrect to assume that applying DevOps practices to their delivery pipeline would have altered the curve of Nokia's decline. For example, if there was a managerial or cultural misalignment between the business and development, that could have been the bottleneck, and there were signs of that. Or if the architecture was as tangled as some of the engineers were concerned it was, that could have been the bottleneck. In hindsight, the most shocking aspect

of this was that nobody could see the whole value stream, so nobody knew. Yet massive bets and investments on the transformation were being placed at the leadership level.

Had Nokia adopted the Three Ways of DevOps (Figure 2.2), they would have at least started on the path to identifying the bottleneck. By focusing on "flow" and "feedback" from Dev to Ops, Nokia might have seen indications of very long lead times for deployment. And "continual learning," if elevated beyond just development leaders, might have caused company leadership to start asking the right questions about organizational structure or the software architecture.

Or not. Nokia could have taken a very tactical approach to DevOps transformation, focused on continuous integration and application release automation alone, and not noticed the architectural or organizational bottlenecks. At a managerial level, they were missing the infrastructure and visibility that would allow them to see what was going on in their value stream. If treated like they treated Agile, DevOps would have been relegated to a technical practice rather than being elevated to the business, and would not have altered the outcome.

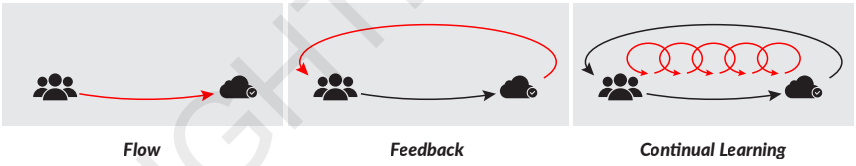


Figure 2.2 : The Three Ways of DevOps

Failing to Transform: The Story of LargeBank

The Nokia failure happened at a time when both scaled Agile and DevOps practices were less broadly understood. The story of this next transformation started with similarly noble goals but also resulted in a failure to deliver business results, even though it happened much more recently. This story, as well as the need to better understand why these failures happen instead of giving up on the organizations that seem to

be struggling with the transition, drove me to study why the principles of DevOps and Agile appear to break down at scale.

I vividly recall sitting on a Boeing Dreamliner in June 2016 en route from Europe to my home and office in Vancouver, Canada, and reflecting on a particular meeting I had had with IT leadership at a bank. I was seated near the wings and admiring their organic-looking beauty; their flexible carbon-composite materials allowed them to swoop upward, reducing drag. This was long before my visit to the BMW Group plant, but it is the first recollection I have of a profound mental struggle to understand how, across the industry, we can be so good at making airplanes and cars while only a small portion of companies have truly mastered making software at scale. After staring at the wing's subtle shifts for an hour, I realized that my mind was stuck, as if on a Zen koan that I could not unfold.

This bank, which we'll refer to as LargeBank and of which I have redacted all identifying details, was undergoing the most massive IT transformation I had ever encountered. LargeBank is one of the top twenty-five financial institutions on the planet, and the project itself was an incredibly impressive set of Gantt charts that fit together like puzzle pieces along a precisely defined two-year time frame and touched on all parts of a multi-billion-dollar IT organization. Notably, this same story has unfolded in similar ways at other large institutions.

It was my third visit to LargeBank, and I had gone approximately every two years. Each time, the discussions were part of a large digital transformation initiative that the bank was undertaking, and this was attempt number three in a process that never changed. Many tool vendors, consultants, and other experts would be brought in. Since my company's business involves integrating various Agile and DevOps tools, we would be walked in detail through every tool and process that was involved in the transformation. Then, two years down the road, we would hear that the transformation had failed to deliver results, and the VP or SVP responsible would be fired. When the next transformation started, I would meet some of the new leadership and listen to the new approach. And thus, we would start again.

LargeBank was now six months into its third transformation. This one was at a larger scale than the previous attempts, as it encompassed

all of IT. The time frame was again two years. The budget was in the ballpark of $1 billion US. All the right transformational, Agile, and DevOps terms were being summarized at the meeting, and the internal presentations looked polished. However, having witnessed Nokia and then many other transformations that went sideways, and seeing the same pattern here yet again, I began to see a vision of $1 billion of the world's wealth being wasted without delivering a measurable improvement to value delivery.

To anyone with a Lean mind-set, that is a profoundly disturbing image, one that we want to do whatever we can to stop. That image got me mapping out the ideas for a framework that would make it possible for the business to understand what was going on and what was going wrong, and hopefully not repeat the same mistakes a fourth time. It was also that disturbing vision that got me to start writing this book, beginning with an article titled "How to Guarantee Failure in Your Agile DevOps Transformation" that I wrote immediately after the meeting.[9]

That third transformation effort has now concluded. The executives leading it were predictably removed once it looked like it was off track, as were the other IT and toolchain leaders who were involved. Again, I have had the opportunity to interview and learn from those who lasted long enough to witness how things after the transformation were worse for its key stakeholders than before the transformation started.

While I was sure at the outset that the transformation would fail to deliver a productivity increase, I was still shocked at hearing that it made things worse. I had thought that, with DevOps as a central component this third time around, there would be at least some successes, just not the promised business results. But in this case, from a value-delivery and talent-retention point of view, things got worse.

Wasting $1 billion of shareholder money or customer value should feel reprehensible, but a key premise of this book is that the leadership of the business and of IT would not have deliberately allowed that to happen. There is something fundamentally broken about the decision-making framework or organizational visibility that enables a business to get into this state over, and over, and over again.

A Disconnect between the Business and IT

To understand the lessons from LargeBank's transformation failure, we need a clearer picture of the business environment that enabled it. LargeBank is a successful financial-services institution, one forward-thinking enough to allocate a multi-billion-dollar budget to IT. It's an organization with a portfolio of thousands of applications and a desire to differentiate and compete on its digital assets, as evidenced by the organizational and budgetary commitment that it had to the transformation. In other words, from a strategic point of view, this organization positioned itself around the Age of Software much more than Nokia did. The CEO endorsed the transformation, but what was happening below the surface?

At LargeBank, IT is run as a cost center under the CFO. The business outcome that was being measured and managed was how much cost could be cut as a result of the transformation. At the outset, I did not see this as a fundamental problem, as so many of the organizations I work with are in a similar situation. However, after interviewing people involved with the transformation and hearing them discuss it, many of the unintended consequences appeared to point back to this as the source of the problem. For example, managing the cost alone meant that IT could run the transformation without closely involving the business stakeholders. If the goal is cost, who knows better than IT how to reduce infrastructure spending, staff costs, and other overheads?

On the business side, there were digital initiatives underway in parallel, with goals to design and create new digital experiences for mobile and web. However, these were divorced from the IT transformation—akin to building a great dashboard for a car without having a car or even a line of sight to a car that could support all of those fantastic new features. For example, other than proving a proof of concept in one part of the application portfolio for just one country or region. Due to this compartmentalization, it was impossible to know whether these mobile experiences could ever run on every type of system that the bank had across the globe.

The transformation was once again a local optimization of LargeBank's value streams. On the IT side, it was focused on just the IT parts

but not on the "value" parts; that is, those parts that needed to deliver value to the customers and to the business. On the digital side, it was ignoring the IT parts that would make the digital vision a reality, such as an architecture and delivery pipeline that could support the new user experiences envisioned. And the measurement of the transformation's success was cost reduction and adherence to the transformation project timelines rather than delivery of more business value at lower cost. In that sense, the outcome was predictable. Cost reduction would be achieved but at a significant reduction of the actual delivery capacity. This is a recipe for fumbling future survival through the Turning Point, as it leaves an open door for startups and tech giants to move in at the time of their choosing.

Lesson Two: *If you manage a transformation according to cost alone, you will reduce productivity.*

Falling into the Cost Center Trap

LargeBank's billion-dollar transformation project was functionally composed of countless subprojects, all of which were running to this two-year time frame. By virtue of being managed as projects, their goal was to be on time, on budget, and to deliver the business goal of cost reduction at the finish line. If every puzzle piece of this massive project fit perfectly with every other, and all were delivered on time and on budget, would that mean success? From an activity and project-oriented view, the answer is yes. But what about the business results? How were those measured at each step? How were bottlenecks identified across the tens of thousands of people, hundreds of processes, and dozens of tools? The bottom line is that they weren't.

Here, we find the root of the disconnect. When IT is treated as a cost center, the transformation takes on the same mentality. The focus becomes the successful reduction of cost at the end of the project time frame. Yet at the executive level, the very business case for the transformation would have touted the benefits of Agile and DevOps, such as faster time to market, more competitive product offerings, and more efficient delivery. However, those outcomes are not what gets

measured by an organization that manages according to cost alone. While it's true that cost reduction can be a critical component of a transformation, that's not the issue. The issue is that a cost-centric framework did not deliver increased velocity, productivity, or efficiency, and instead resulted in the business getting a lot less for less instead of more for less.

Given the sophistication of LargeBank's transformation, more than cost metrics alone were used. The typical Agile transformation metrics, such as the number of teams following the Agile model, were also used, as were DevOps metrics, such as the number of deploys per day. But these are metrics of activities, not results. An IT team could be deploying a hundred times per day, but if their work intake is not connected to the needs of the business, the results will not materialize for the business. Once again, the proxy variables of "number of people trained on the Agile process" or "deploys per day" will only be meaningful if training or deployment are the bottleneck. But when the business is disconnected from IT, the Agile teams and DevOps pipeline never get the opportunity to become the bottleneck.

The problem is not the use of proxy metrics themselves; the problem is that we are relying on proxy metrics for decision making rather than finding the metrics that directly correspond to business outcomes. Consider Jeff Bezos's statement from his 2017 letter to shareholders in which he spoke, among other things, about resisting proxies:

> Resist Proxies
> As companies get larger and more complex, there's a tendency to manage to proxies. This comes in many shapes and sizes, and it's dangerous, subtle, and very Day 2.
>
> A common example is process as proxy. Good process serves you so you can serve customers. But if you're not watchful, the process can become the thing. This can happen very easily in large organizations. The process becomes the proxy for the result you want. You stop looking at outcomes and just make sure you're doing the process right.[10]

In other parts of our business, we have outcome-based metrics, like revenue, daily active users, and Net Promoter Scores (NPS). The problem is that organizations do not have an agreed-upon set of metrics for measuring and tracking work in IT, and as such, settle for these proxies. And the wrong set of metrics is coming from measuring not the flow of value delivered but the "successful" execution of IT projects. In the next chapter, we will dive into the identification of a new outcome-based way of tracking value streams. But first, we need to further examine the origins and issues of the project-centric mentality as it applies to production.

From Puzzles to Planes

How is the approach at LargeBank so different from that of the BMW Group Leipzig plant? Is car production simpler somehow? Does it lend itself more easily to end-to-end measurement? How was the BMW Group able to transform how it builds cars so quickly, creating the i3 and i8 production lines without ever having mass produced electric cars or carbon-fiber bodies before? The BMW Group is just one example of the level of maturity, measurement, and adaptability that have been mastered in the Age of Mass Production.

Consider another highly complex artifact that epitomizes the Age of Mass Production. The Boeing 787 Dreamliner contains 2.3 million parts built in 5,400 factories.[11] Across all of its value streams, Boeing manages the production of 783 million parts across the hundreds of aircraft that it delivers each year.[12] It needs its products to stay relevant in the market for decades and bets the company on each new product introduction.

How is it that the BMW Group and Boeing can both manage existing production lines, transform their business to support new ones, and continually adapt to changes in technology, competition, and the market? The bottom line is that they are not stuck in the gridlocked puzzle pieces of project management. Instead, they have mastered a product-centric view of delivering value to their market.

My first exposure to this was a story told to me by Gail Murphy, then my professor in a third-year software engineering course, about

the production of the Boeing 787's predecessor, the 777. The 777 was Boeing's first "fly-by-wire" plane. In other words, the software had to work, as it was purely software that was controlling the flaps and rudder and preventing the plane from falling out of the sky. Gail recounted that, due to the criticality of the software, Boeing decided to put all the heads of software engineering on the test flight. During the test flight, the plane started shaking, and the software engineers were able to implement a midflight fix via the turbulence control software.[13] I have yet to find a better example of an organization putting software leaders' skin in the game of high-stakes product development.

The depth of Boeing's understanding of the business implications of production and long-lived value streams is underscored by an event that unfolded during the production of their next plane, the 787 Dreamliner. The 787 Dreamliner project was Boeing's most ambitious to date and even more software-intensive than the 777. The Dreamliner was the first commercial plane to run on an electrical platform for everything ranging from cabin heating to wing ice protection, in comparison to previous commercial airlines that used much less efficient engine air-bleed systems.[14] In addition, Boeing decided to dramatically restructure its supply chain in order to lower the production costs of the plane—all while shifting to carbon-fiber wing and body parts. These and other complexities resulted in further delays.

In 2008, while following the program, I read of yet another delay, but the reason for this one seemed much more interesting. In this article, the general manager of the Dreamliner was quoted as saying: "It's not that the brakes do not work; it's the traceability of the software."[15]

This was fascinating to me—and not only because I was happy to read that the new plane would ship with functioning brakes. It was also intriguing to me because at the exact same time I was working on features in the Eclipse Mylyn open-source project to automate linking software requirements and defects to the lines of source code that had changed while developers worked on those items. For me, as well as my open-source colleagues at the time, the need to manually enter IDs to ensure traceability was tedious and error prone, and it was quite easy to automate since the Mylyn developer tool always knew what item the developer was working on.

Due to the hundreds of contributors on the Mylyn project, I required that every change to every line of code have traceability back to its originating feature or defect; otherwise, every time new work came in that was related to that code, we would have to manually search for why that code was there in the first place. That was far too tedious to do on such a resource-constrained project with millions of end users constantly submitting defects and requests, so I added a new feature to automate it. But why on earth would Boeing care about traceability to the extent of further risking the delivery time frames of the plane? Surely Boeing had a model that indicated an immense cost or risk for further delay. They could not possibly have the resource constraints that we were experiencing, so there must have been something more fundamental about their need for traceability.

In researching this further, I learned something even more fascinating. During a visit to Boeing, I remember learning that they design planes for approximately three decades of active production followed by another three decades of maintenance.[16] In other words, they are thinking ahead six decades in terms of support costs for both the hardware and the software. The brake software had been outsourced to General Electric, who, in turn, outsourced it to Hydro-Aire.[17] Hydro-Aire then delivered the working brake software using Subversion SCM, and provided General Electric and Boeing with both the source code and the binaries.[18] The software worked, passing the tests and meeting the requirements.[19] However, the source code did not have any traceability links to those requirements.[20] Adding traceability links after the fact is difficult and error prone. Given a sixty-year maintenance window and the overhead of compliance certifications, at a business level, Boeing knew that the most economical decision was to rewrite the brake software.

In spite of the Dreamliner's complexity and the scale of the transformation required to produce it, Boeing created an amazing product that has seen tremendous success in its market. What is it that Boeing understands about product development that so many IT organizations do not? How does Boeing think and plan beyond the typical one- or two-year project time frame to enable it to make such fundamental business decisions based on technical details? How do we get our organizations out of the fixed puzzle-piece mentality of enterprise

IT projects and into the excellence of production that we see in Boeing's and BMW Group's plants and organizations? How do we shift from project to product?

> ***Lesson Three:*** *Engineering/IT and the business must be connected.*

Toward Product Development Flow

If you've ever worked for a software startup, a tech giant, or a modern software vendor, you might be wondering what the fuss is all about. Of course, there is more to software products than minimizing costs; there are revenues, profits, active and delighted users, and all of those other metrics that populate objectives-and-key-results (OKR) systems. Tracking software delivery to business outcomes and treating it as a profit center is one of the main reasons why tech companies are faring so much better than their enterprise IT counterparts, who are stuck in what Mary Poppendieck calls the "cost center trap."[21]

Is this cost center approach endemic to large enterprise organizations? Consider how a large and very cost-conscious company like Boeing is managing the production of its Dreamliner. Costs are, of course, a key factor. But the success of Boeing depends not only on cost reduction but on the adoption and profitability of each plane for the span of its life cycle in the market—this is why Boeing innately takes a long view on the traceability of its software. Boeing knows that life cycle profitability will be affected if the software cannot be economically maintained, or if current or future regulation changes cannot be easily addressed in the software.

What Boeing demonstrates across its operations is that it treats its plane development as a profit center. The business sets goals, metrics, culture, and processes in a completely different direction than what we saw at LargeBank. I do not believe there is any less cost consciousness at Boeing, which continually works at reducing the production and supply-chain cost of every aircraft. But by doing so with an eye to revenue and profitability, an entirely different set of decisions can be

made, like investing in the modularity of its value streams to modernize legacy offerings. A very visible example of that is the decision to modernize the 747, which first flew in 1969, with 787-style wings and engines to create the 747-8.[22]

As another example, consider what I witnessed at the BMW Group Leipzig plant. The scale of the 1- and 2-Series cars was impressive due to the massive automation and the seventy-second takt time (the rate at which a product step needs to be completed to meet customer demand). But what impressed me even more was the very different way that the i3 and i8 lines were set up, as I will recount in an upcoming story.

The market adoption and profitability of the BMW Group's electric cars would be hard to determine in a changing market, so the BMW Group created a production architecture to support learning from the market before investing further in automation of the lines. The profitability and product/market fit drove the architecture of the value stream, not vice versa. Again, it was a complete one-eighty from the approach at LargeBank, where IT was transforming for IT's sake. The production infrastructure, architecture, and managerial approach could not be more starkly different. That was the point that hit home when I visited both LargeBank and the BMW Group on the same trip, and was flying home with my head pressed against the window of a Boeing Dreamliner.

To those who have studied the concepts of Donald Reinertsen's *Product Development Flow*, none of this will be news. Reinertsen makes a very clear and compelling case for throwing out proxy variables and measuring for a singular economic objective: life cycle profits.[23] However, depending on where in the company or product maturity curve your focus is, this objective might change.

In *Zone to Win*, Moore provides a model with four distinct investment zones (Figure 2.3).[24] The *Productivity Zone* is focused on making the bottom line and includes systems such as HR and marketing. The *Performance Zone* is about the top line and includes the main drivers of revenue. The *Incubation Zone* is where new products and businesses are developed before just one or the other is moved into the *Transformation Zone* and used to play disruption offence or defense. In defining a value metric for different product lines, the zone and

its goals must be identified. For example, in the Incubation Zone, the business objective might be monthly active users, before transitioning to a Transformation Zone where the focus is more on revenue than on profit (Figure 2.3).

The mistakes many organizations are making in their digital transformations are in using the metrics from the Productivity Zone—costs and the bottom line—for measuring their entire IT and software delivery capabilities. Prior to the Age of Software, all of IT could be relegated to the Productivity Zone; but the whole purpose of a digital transformation is to allow the organization to launch and manage products in the other zones, which is what will determine their future relevance in the market.

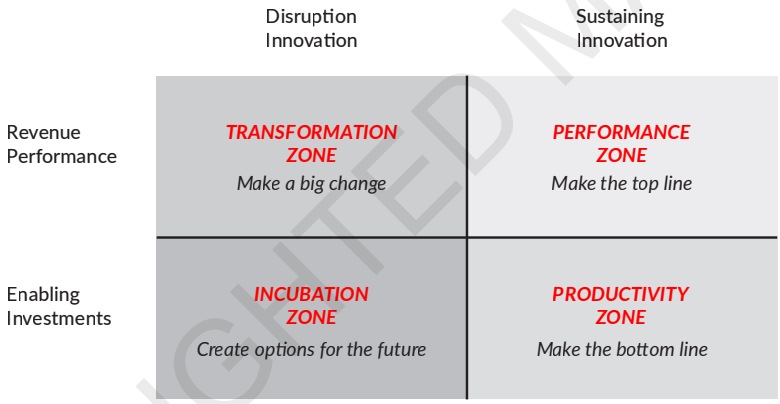


Figure 2.3: Zone Management[25]

Projects versus Products

Project management is a practice that has enabled some of the world's most visible and impressive accomplishments. It has been iconified by the Gantt chart, created by Henry Gantt in 1917 and subsequently used to build the Hoover Dam, the largest concrete construction of its time. This was the tail end of the Deployment Period of the Age of Steel, during which the practices of Taylorism were adopted in order

to improve and scale labor efficiency. Those practices provided a way of creating standard work processes and best practices, as well as the specialization and division of labor at scale.

While it may not have been Taylor's intent, when put to practice by others, Taylorism assumed that people could be treated as interchangeable resources that could be assigned and reassigned to projects. Such treatment of workers as machines is not only dehumanizing but also shortsighted, as later demonstrated by Henry Ford, who realized the importance of decentralized decision making and autonomy.

These problems are so fundamental that the Age of Mass Production was, in part, catalyzed by the methods applied by Ford. Fordism put significantly more emphasis on the actual worker, their training, and their economic well-being.[26] The companies that excelled in the Age of Mass Production built on Fordism and extended it with approaches that further connected production to the business, such as Toyota's innovation of the Andon cord.[27]

The effectiveness of this is exactly what the walk through the BMW Group Leipzig plant identified. What that journey led me to conclude is that many enterprise IT organizations are still managing to the project-oriented world of Taylorism from the Age of Steel. This disconnect is what is causing the massive communication gap between business leaders and technologists.

Software delivery is, by its nature, creative work. Software specialists are skilled at automating repetitive processes when given the chance, leaving only the complex work and decision making that humans continue to excel at. Applying management frameworks from a hundred years ago to organizations that need to compete on digital assets is futile. To make this more concrete, Table 2.1 contrasts a project-oriented approach with a product-oriented one.

Budgeting

One of the most important aspects of structuring IT and software investments is budgeting, as budgeting has such an influence on organizational behavior. The budgeting of projects assumes a high degree of market and resource certainty, as it creates a fixed end goal and

measures success to being on time and on budget. It also creates an incentive for stakeholders to ask for as large a budget as possible, since the budget has to factor in any uncertainty to the project time frames. In addition, going back for more budget requires significant effort or the creation of a new project.

	Project-Oriented Management	Product-Oriented Management
Budgeting	Funding of milestones, pre-defined at project scoping. New budget requires creation of a new project.	Funding of product value streams based on business results. New budget allocation based on demand. Incentive to deliver incremental results.
Time Frames	Term of the project (e.g., one year). Defined end date. Not focused on the maintenance/health after the project ends.	Life cycle of the product (multiple years), includes ongoing health/maintenance activities through end of life.
Success	Cost center approach. Measured to being on time and on budget. Capitalization of development results in large projects. Business incentivised to ask for everything they might need up front.	Profit center approach. Measured in business objectives and outcomes met (e.g., revenue). Focus on incremental value delivery and regular checkpoint.
Risk	Delivery risks, such as product/market fit, is maximized by forcing all learning, specification, and strategic decision making to occur up front.	Risk is spread across the time frame and iterations of the project. This creates option value, such as terminating the project if delivery assumptions were incorrect or pivoting if strategic opportunities arise.
Teams	Bring people to the work: allocated up front, people often span multiple projects, frequent churn and re-assignment.	Bring work to the people: stable, incrementally adjusted, cross-functional teams assigned to one value stream.
Prioritization	PPM and project plan driven. Focus on requirements delivery. Projects drive waterfall orientation.	Roadmap and hypothesis testing driven. Focus on feature and business value delivery. Products drive Agile orientation.
Visibility	IT is a black box. PMOs create complex mapping and obscurity.	Direct mapping to business outcomes, enabling transparency.

Table 2.1: Project-Oriented Management vs. Product-Oriented Management

This is where the mismatch becomes immediately apparent. DevOps and Agile are all about creating a feedback loop to address the inherent uncertainty of software delivery and then providing the feedback loop to the business to adjust accordingly. The more certainty there is, the more optimal a long-term allocation of resources gets created by a project plan. However, due to the degree of complexity in

software delivery and the inherent rate of change in the market due to the Turning Point, baking all that uncertainty into project plans not only creates tremendous waste but gives the business visibility into the wrong things, providing a view into activities and proxy metrics over visibility into the incremental delivery of business results.

In an Incubation or Transformation Zone initiative, there could be more uncertainty than certainty. This results in project plans that incentivize delaying releases and customer testing in order to implement everything that was needed at the start of the project. And this multiplies the product/market-fit (PMF) risk by removing the possibility of iterative learning from the market and pivoting accordingly.

In contrast, product-oriented management focuses on measuring the results of each unit of investment that brings value to the business. Those units are products; they deliver value to a customer, and as such, the measurement must be based on those business outcomes. Funding of new value streams is based on a business case for that product, as is ongoing investment in those value streams.

The approach need not be disruptive to the annual planning cycle. For example, at Tasktop we create annual budgets for the product and engineering departments that are signed off by our board; but every quarter we review the allocation of those budgets to the products' value streams (e.g., staffing up a promising new Incubation Zone offering once it has customer validation).

More aggressive Lean Budget approaches have also been proposed, to more quickly respond to cost overruns or revenue opportunities for a particular value stream. Whether an annual or a more frequent budgeting cycle is used, what's important is that products, not projects, are the unit of investment.

Time Frames

One of the biggest problems that project-oriented management faces stems from the consideration of time frames. A project has a defined time frame after which resources ramp down. This makes perfect sense when building a skyscraper, as there is a definitive and well-understood end to the project: after the skyscraper is erected, the project moves into a maintenance period. However, products, be they software or

hardware, have a life cycle, not a definitive end. Products can be end-of-lifed; for example, Google has killed dozens of products, including Google Reader and Google Wave. As long as a product is available, defect fixes and new features are requested, as the ecosystem around software products is constantly evolving.

Applying the project-oriented mentality of creating then launching a software product, and then assuming it can be reduced to a fraction of its investment when in maintenance mode, has many unintended consequences. For example, one enterprise organization that I worked with did a survey to assess the state of project management across thousands of IT staff. They found that the number of projects that an engineer was assigned to over the course of a year ranged from six to twelve on average, depending on where they worked in the organization.

I personally went through this in the early days of Tasktop, allocating people and teams to multiple open-source services projects. I noticed a dramatic productivity reduction when an engineer was assigned to more than a single value stream. This staffing antipattern comes from the annual allocation of people to projects and the assumptions that the projects will not require much work during maintenance, leaving people with the impression that it will only take a small slice of any given person's time. The reality is that if the product is used, the need to do fixes is regular, and as I witnessed, the thrashing becomes a major drag on both happiness and productivity.

Some organizations address the post-project maintenance of software by outsourcing it to organizations such as the global systems integrator (GSI). This additionally reduces the apparent cost of maintaining the software, potentially removing it as a liability on the balance sheet. This outsourcing appears to work in theory, but it can cripple flow and feedback loops, as organizational boundaries need to be crossed. In addition, it further disconnects that software from the business. If the software was core to the business, this is debilitating in terms of continuing to deliver business results, as the need for change and updates is constant in software.

The false notion of a software project's end is also wrong from an economic point of view. Some of the perceived economic benefits will

be removed with the implementation of IFRS (International Financial Reporting Standards) revenue recognition rules in the US, which may also remove the balance sheet bias toward outsourcing. But in product-oriented management, the focus needs to be on life cycle costs and profitability, as exemplified by the Boeing example earlier.

It gets worse. With this "project-end fallacy," key aspects of the economics of software delivery are not visible to the organization. For example, one of the core concepts that we will review in the next chapter is technical debt. The accumulation of technical debt that results from normal software development creates problems that are well documented. And if this debt is not reduced regularly, the software becomes prohibitively difficult and expensive to add features to or to fix.

This was a key part of the failure that we saw in Nokia's story, where technical debt contributed to the loss of the mobile market that it dominated. In project-oriented management, there is no incentive to reduce technical debt; its effects do not materialize until after the project ends. This results in application portfolios that are dead ends for the companies that created them and in the constant accumulation of more legacy systems and code.

Success

At a leadership level, the success metrics that we place on our organizations and on our teams will determine behavior. Project-oriented management tends to take a cost center approach, which is still common for enterprise IT. As we witnessed with the story of LargeBank, expecting an increase in the delivery of business value from a cost center is pointless.

Project-oriented management also brings with it some side effects that go against the principles of DevOps. For example, the capitalization of software development results in an incentive to create large projects. Stakeholders are incentivized to ask up front for everything that might be needed during the course of the project, which goes directly against Lean thinking and continual learning. The companies who are navigating the Turning Point measure software investment in terms of business outcomes, like internal adoption or revenue

generation. This results in a fast-learning managerial culture of incremental value delivery and regular checkpoints.

As Figure 2.4 depicts, product orientation enables an alignment of the organization to business outcomes, not functional silos.

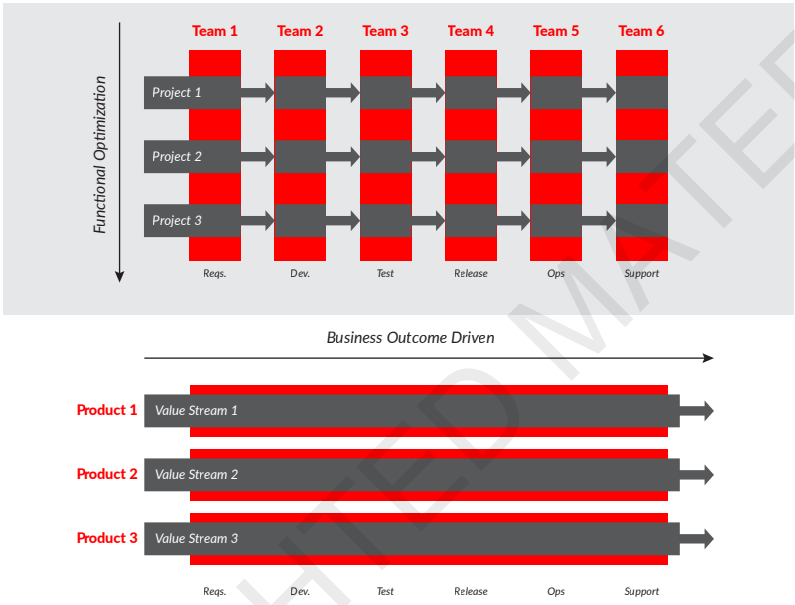


Figure 2.4: Functional Optimization vs. Business Outcomes

Risk

Project-oriented management is designed to identify and create contingencies for all risks that could be incurred during the project. However, that requires up-front knowledge of all the risks, which works in some domains but not in the highly uncertain and changing world of software delivery.

The Cynefin framework provides a taxonomy of decision-making contexts, including obvious, complicated, complex, and chaotic.[28] Due to the rate of change in technology stacks and in the market, software initiatives tend to fall into the complex or chaotic zone. As a result,

project-oriented management, which is optimized for the obvious and the complicated contexts, ends up padding projects for any contingencies that may come up during the project's time frame. This results in overly conservative time frames and inflated budgets. But not even those can stave off product market risk, where any up-front planning is less effective than regular hypothesis testing and learning.

In contrast, Lean Startup and approaches such as minimum viable products (MVPs), are a key part of the product-oriented mind-set. In addition to reducing risk, the incremental product-oriented approach creates option value by allowing the business to pivot at regular checkpoints. This is not without an overhead, as the more frequent reviews and checkpoints require expensive managerial bandwidth. But due to the complexity of software and the rate of change in the market, this overhead is best spread out across a product life cycle rather than incurred at project inception and again at failure.

Teams

With project-oriented management, resources are allocated to projects. This follows the Taylorist philosophy that people are fungible and expendable. That assumption breaks down completely in software delivery, which is one of the most complex disciplines of knowledge work.

Modern software value streams are built on millions or tens of millions of lines of code. At Tasktop, the most complex part of our codebase takes a senior and highly experienced developer six months to ramp up to full productivity. Consider the productivity impact of allocating people to new projects every twelve months. Unfortunately, this is not far from the norm, as most enterprise IT organizations do not model or measure developer productivity, engagement, or ramp-up time; and with a Taylorist mind-set, are unaware of the overheads.

Overlaid on top of the costs to the individual IT specialist's productivity and happiness are the costs to the team. Teams working on complex problems go through what psychologist Bruce Tuckman coined the forming, storming, norming, and performing life cycle.[29] Reallocating people disrupts that cycle; the more people are moved, the higher the productivity cost for the team.

The project-oriented management approach of "bringing people to work" is not suited for complex knowledge work, like software delivery. High-performing software organizations have already learned that "bringing work to people" is more effective. Long-lived teams allow for expertise (both individual and team) and relationships to build over time, improving both velocity and morale. This enables other benefits as well, such as problems being solved at the lowest level of the organization instead of having the nonscalable, constant escalations to management that result from changes to plan.

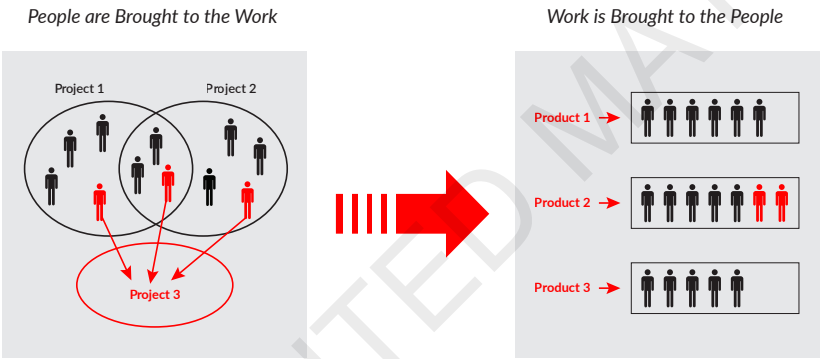


Figure 2.5: Bringing the People to the Work vs. Work to the People

In large-scale software, the optimal allocation is a one-to-one allocation between teams and value streams in order to maximize team and expertise building. The adoption of *feature teams* is one example of this kind of allocation, though value streams for larger products will often consist of multiple feature teams.

Prioritization

In project-oriented management, the project plan drives priorities. Changes to the plan are expensive in terms of management overhead, communication, and coordination; and as such, adjustments to the plan tend to be minimized. In software delivery terms, this tends to

drive a "waterfall" software delivery model, as that is the model that naturally aligns to a cascading project plan. While this is suitable to projects with a high degree of predictability, it is counterproductive for software delivery. Product-oriented management sets priorities based on product road maps of features and constant hypothesis testing. At an organizational level, this means applying the feedback and continual learning principles of DevOps at all levels of the organization, right up to senior management.

Visibility

Last and perhaps most important is the problem of visibility. In the Nokia and LargeBank examples we reviewed earlier in this chapter, the common theme is the disconnect between the business and IT. What is the source of the disconnect? Given that the leadership and business representatives have such a broad purview, how is it possible that there is a lack of visibility into IT? In a time when we have ubiquitous access to big data and analytics tools, how can IT feel like a black box at so many organizations?

The problem is not due to data access; the problem results from a mismatch of the data models with which IT works and with which the business is operating. IT and software delivery professionals already work and think in product-oriented mind-sets and methodologies. That is what the business has tasked them to do—to deliver value through software offerings. However, if the business side is still thinking and managing in terms of projects, constant mapping and remapping needs to be done between the more iterative nature of software delivery and the more static nature of techniques, such as project and portfolio management and earned-value management.

The end result is the "watermelon" phenomenon.[30] When engineering leads are asked by project managers if they are on track, the answer is yes, because the question is ambiguous. Once the release plays itself out and business goals are not met, it is clear that the projects were not on track. The projects were "green" on the outside and "red" on the inside (like a watermelon). But the problem is not with the projects; instead, it is with the management paradigm that was never designed to handle the complexity and dynamics of software delivery.

Conclusion

Marked by a project-over-product mentality, an emphasis on cost over profit, and adherence to time frames over delivery of business value, this disconnect between the business and IT is at the core of IT and digital transformation failures. This is what our organizations need to learn from the last Deployment Period in order to have a shot at competing against the upstarts of the current one and set the foundation to thrive in the next decade.

The challenge is that, at enterprise IT scale, we have not had the management framework nor the infrastructure needed to manage large-scale software product delivery. As you will discover in the next chapter, the Flow Framework provides a new approach to managing software delivery to business results instead of technical activities.

The goal of the Flow Framework is to provide the missing layer between business-driven digital transformations and the technical transformations that underpin them. If the digital transformation is focused on processes and activities instead of business results, it is unlikely to succeed; and the business is unlikely to realize that until it is too late. The concepts of organizational charts and Scrum teams will continue to be as relevant as ever. But for an organization wanting to become a software innovator, these are secondary to product-oriented value streams. The Flow Framework ensures that your transformation is grounded in connecting, measuring, and managing your Value Stream Network, which is the critical layer needed to succeed with software delivery at scale.

Neither the shift to a product-oriented management nor the Flow Framework are sufficient to assure success in the Age of Software. Organizations need a managerial culture and understanding of a rapidly changing market in order to bring software-based products to market and adapt. But before we consider how to deliver more value, we must first define how we measure business value in software delivery. That is where the Flow Framework comes in.

CHAPTER 3

Introducing the Flow Framework

What we have discovered so far is that enterprise organizations are attempting to use managerial mechanisms from previous ages to direct software delivery in this one. IT and software delivery costs have been growing for decades, yet our organizations do not have adequate visibility or understanding of what is now one of the largest costs of doing business. Meanwhile, the tech giants and digital startups have already mastered the managerial frameworks necessary to succeed in the Age of Software. So have many of the technologists working within enterprises, and these technologists are pushing hard on their organizations to deploy the DevOps and Agile practices that they know are critical to transformation.

The problem is that the principles of modern software-delivery approaches are not translating to the business. For example, enterprises are still managing IT as a set of projects or a cost center, rather than taking the product-oriented mentality that defined the winners of the Age of Mass Production.

We need our businesses to adapt to this product-oriented mind-set and to do so in a way that supports the vast differences between producing physical widgets and infinitely malleable software components. We need a new framework, one that elevates the best practices of Agile and Lean frameworks to the business. We need to define business outcome–oriented metrics instead of relying on activity-oriented proxy metrics.

In this chapter, I introduce the Flow Framework as a new approach for connecting the business to technology. The Flow Framework is not intended to help you spot market shifts or strategize offerings that will

disintermediate disruptors; it is intended to provide you with a layer that bridges the gap between business strategy and technology delivery. The Flow Framework opens up the black box of IT so you can create an organization-wide feedback loop, accelerating the flow of business value to customers and the organizational learning to adapt as the market continues through the second half of the Age of Software.

To support the deployment of the Flow Framework, this chapter introduces Value Stream Networks as the key infrastructure concept needed to bring about the same kind of automation and visibility for software delivery that we see in manufacturing. The chapter concludes with an overview of the Flow Framework and a definition of the four flow items that are at its core. But before we learn more, let's revisit the plant.

BMW TRIP Walking the Lines

Looking down at the 1- and 2-Series production process in action, it is hard not to marvel at the choreographed actions of the robots and blue-vested production-line workers. We walk about a mile around the Assembly Building, slowing down to examine the various workstations. Some are fully automated, with large robots welding, assembling, and gluing. Others have workers performing intricate assembly steps. Frank stops us at a particularly intricate part of the line, pointing out some wiring harnesses and describing how each of these will form the electrical nervous system of the car.

"Every single harness is different," Frank says. "Each car is made to order, which means countless combinations of options for electronic components. Because of this, each wiring harness is assembled specifically for that car prior to arrival at the line."

Frank describes just how complex harness fitting on the production line is. He explains that if something goes wrong with the installation and it does not finish in the takt time of seventy seconds, a cord is pulled; then, assistance can come from the next workstation to complete the job. The production line is structured

to ensure this highly complex job is completed reliably, without having to remove cars from the line for rework.

Frank explains how complicated it would be to pull a car off the line at this point, because there are still miles of production line downstream. Every single workstation would need to compensate for the resequencing of the flow of parts. As such, many additional steps and processes are in place to avoid removing a car from the flow.

I'm struck by the parallel between this and what happens when a software team breaks a build due to working out of sync with the latest code, and just how expensive that scenario is for a software organization. Here, everything is synchronized to ensure continuous flow, including the slack needed if a worker cannot complete the harness install in time.

Further along the line are the "knuckles"—the parts of the building where the line takes a ninety-degree turn to the left, proceeds into the "finger" building (which looks like an endless corridor of additional assembly steps), then returns to where we are standing before proceeding to the next finger.

"Sunroof installation is so complex that we never want to move this workstation," Frank says. "The robots are bolted right into the floor, unlike the other stations, where they can be moved. Now you see why the building has this architecture. We can extend the fingers with new manufacturing steps, but the knuckles themselves are the fixed points in the production line."

The entire physical architecture of the plant has been optimized around the current and potential future flow along the line. The five knuckles are the most complex parts of the value stream, which is why the buildings are built around them—to maximize flow and future extensibility within these key constraints of the line's value stream architecture.

Why could we not think with this kind of high-level clarity about constraints and dependencies in software delivery? Why is it that we architect around technology boundaries and not around value stream flow?

Why We Need a New Framework

The transformation challenges outlined in Part I are fundamental. Doing something about them is not required, and many companies will end up staying the course in their ineffective approach to managing software delivery. Today, definitive data exists to determine how quickly the next disruptions will happen, or which approach or framework is most effective at addressing them.

By the time data is available to analysts and researchers it will be too late. The winners and losers of the Age of Software will have gained enough market share that those applying management techniques of previous ages will find it difficult or impossible to catch up without regulation or other forms of government intervention. We see signs of this already: whenever Amazon's share price goes up, the share price of retailers like Target, Walmart, and Nordstrom's goes down;[1] and vice versa. This does not represent typical market dynamics. We are seeing a zero-sum game that will keep playing itself out industry by industry as we continue to head through the Turning Point.

Numerous methodologies and frameworks exist for transforming, modernizing, and reengineering every aspect of your business. Some, like the Scaled Agile Framework (SAFe), are focused on enterprise software delivery. Recent advances in DevOps practices address bottlenecks in how software is built and released. Other frameworks, like Moore's Zone Management, address transformation from a business reengineering point of view.

Such practices and frameworks are as relevant as ever, and the Flow Framework assumes that the best-suited practices for your business are already underway. The role of the Flow Framework is to ensure that the business-level frameworks and transformation initiatives are connected to the technical ones. It is the isolation of these initiatives that is causing so many transformations to stall or to fail.

To achieve the Three Ways of DevOps—flow, feedback, and continuous learning—we need to scale the ways of DevOps beyond IT to the business. We need a new framework to plan, monitor, and ensure the success of today's software-centric digital transformations. This new framework cannot be separate from the business; it must be

connected directly to the measurement of business objectives and key results. It also cannot ignore the idiosyncrasies of software development or assume that software can be managed like manufacturing. And it cannot be overly focused on one aspect of software delivery, be that development, operations, or customer success.

The new framework must encapsulate the management of large-scale software delivery in a similar way to how value stream mapping, enterprise-request processing, and supply-chain management provided the managerial building blocks needed to master manufacturing. This is the role of the Flow Framework.

At the Leipzig plant, all staff know who the customer is. All staff can see the activity of the company's value streams along the production lines. All staff know what business the customer pulls from those value streams: cars that deliver on BMW's mantra of "Sheer Driving Pleasure." And all staff know what the plant's bottleneck is. Contrast that to today's enterprise IT organizations, where not just the staff but the leadership have problems answering the questions most fundamental to production:

- Who is the customer?
- What value is the customer pulling?
- What are the value streams?
- Where is the bottleneck?

For example, at LargeBank, the delivery efforts were not structured into products supported by value streams, so there was no clear or consistent way of identifying the customer for each part of the project portfolio. For many internal applications and components, the customer was not specified; and in many cases, delivery was more closely aligned to legacy software architecture than to internal or external customer pull. Extracting the value streams from project-oriented management was impossible due to all of the overlap between the projects and a lack of alignment between the projects and the software architecture. And due to all the disparate systems and focus on local optimization and tracking of activities instead of results, it was impossible for anyone to reliably know where the bottleneck was.

The Flow Framework provides a simple path to answering these questions. There are key staff within your organization who already know the answers, but their efforts and vision need to be connected to an organizational strategy and approach. Most important, it provides you with a way of connecting your Value Stream Network, measuring the flow of business value and correlating that to your strategy and business outcomes. The Flow Framework will allow you to:

- See the end-to-end flow of business value in real time
- Instantly spot bottlenecks and use them to prioritize investment
- Test hypotheses based on real-time data from every value stream
- Rearchitect your organization around maximizing flow

A digital organization that competes without a connected and visible Value Stream Network will become akin to a manufacturer trying to compete in the last age without an electrical network. These organizations will learn that managing IT without flow metrics or something equivalent is like managing a cloud infrastructure without a mechanism for measuring the cost of electricity and computer power.

Focus on End-to-End Results

Organizational charts and enterprise architecture are the best representations of value creation that we have; but they are failing us, and we know it. Software investment and staffing decisions are made anecdotally, using static and stale pieces of data and activity-based proxies for business value rather than metrics that directly correlate technology investments with business results.

At the BMW Group Leipzig plant, the flow of value was clear and visible. Units of value—the cars—flowed along assembly to final production. The velocity of delivery and the quality and completeness of each unit could be inspected individually and in aggregate. In software organizations, we do not have the benefit of tangible and visible objects flowing through a production line.

But what if we did? What if we could take a real-time, animated MRI of the software organization? What would we see flowing from the business to the customer? What patterns would we see in the flow? Could we spot the bottlenecks where flow is impeded? These are the questions the Flow Framework answers.

The Flow Framework provides a system for the end-to-end measurement of the results of software delivery. The focus is on the measurement of the ground truth of software delivery—the actual work being done—and connecting the work to results, such as revenue generation. The focus is entirely on result-oriented business metrics, like revenue and cost, not proxy metrics for value creation, like lines of code created or deploys per day.

This is not to say that proxy metrics of that sort are not important. For example, if lack of continuous delivery automation is the bottleneck for a value stream, then measuring deploys per day becomes a critical metric. However, the Flow Framework is focused on the end-to-end metrics used to identify bottlenecks wherever they lie in the value stream. In addition, the Flow Framework avoids measures of activity in favor of results. There are no metrics of "how Agile" a team or organization is; there's just a focus on how much business value flows. If Agile development is a bottleneck, then measuring a proxy such as the number of people trained on Scrum can result in an increase to the flow of business value.

But the role of the Flow Framework is not to specify how to achieve agility; that is the role of Agile frameworks and training programs. The role of the Flow Framework is to help you track, manage, and improve your investments in automation and agility.

Lean Thinking Required

While the Flow Framework does not require the implementation of a specific Agile framework or working model, nor any specific approach to DevOps or customer success, it does require a commitment to the Lean concepts that are the foundation of those approaches. At the highest level, the purpose of the Flow Framework is to provide an actionable way of implementing the concepts of Lean thinking for large-scale

software delivery. Those concepts are defined by James P. Womack and Daniel T. Jones in their *Lean Thinking* book as follows:

> . . . lean thinking can be summarized in five principles: precisely specify value by specific product, identify the value stream for each product, make value flow without interruptions, let the customer pull value from the producer, and pursue perfection.[2]

The Flow Framework requires a business-level commitment to product and value stream thinking, and the principles of flow and customer pull that underpin Lean thinking. As we move through this book, we will identify these five principles with an emphasis on how they relate to managing software delivery. To do that, we must first precisely define how these concepts of flow, pull, and value streams translate from the Age of Mass Production to the Age of Software.

What Is a Value Stream?

One of the first principles in Lean thinking is to "identify the value stream for each product."[3] We will go into detail on how we do that in Chapter 9; for now, consider it to include every person, process, activity, and tool related to delivering that software product.

> ***Value Stream:*** *The end-to-end set of activities performed to deliver value to a customer through a product or service.*

Each product needs to be well defined as a packaging of software features that a customer uses, either directly or embedded as part of another physical or digital offering. That means the customer needs to be well defined too, but the customer need not be defined strictly as an external user. For example, an internal business user of a billing system is also a customer, meaning that the billing system can and should have its own value stream. Some organizations may have a team that produces an internal platform or API (application programming interface) that is only consumed by other developers within the organization. In that case, the customers are the consumers of that API.

Each product has a customer who consumes the software produced by that product's value stream.

Value streams are composed of all the activities, stakeholders, processes, and tools required to deliver business value to the customer. While this may sound obvious, my second epiphany was all about the fact that instead of creating abstractions around end-to-end value streams, organizations keep creating them within functional silos. For instance, if support teams or business stakeholders are excluded from the process, the result is no longer a value stream but a segment of the value stream. As such, Agile teams are segments of the value stream, as are DevOps teams. Even cross-functional feature teams rarely constitute the full value stream at a large organization. For example, as they tend to exclude the support team.

This is not to say that value stream segments are not important, only that managing and measuring them is not the topic of this book. The practices we have around the various segments are mature when compared to how organizations are approaching end-to-end value streams. For example, many enterprise IT organizations are using robust combinations of requirements management, project and portfolio management, enterprise Agile, continuous delivery and DevOps, ITIL, and customer success. Each has multiple frameworks, tools, and metrics that are continuing to evolve. The Flow Framework states that we need a new practice for managing end-to-end value streams in a similar way to how value stream mapping gave the Age of Mass Production the boost it needed to master large-scale delivery of physical products.

From Mapping to Architecture

As manufacturing matured throughout the Age of Mass Production, best practices formed to handle the complexity and management of the end-to-end process. A key practice in manufacturing plant operations is value stream mapping, as summarized by Mike Rother and John Shook in *Learning to See*.[4] This practice provides a visual notation and set of metrics for the management of production flows and the identification of waste and bottlenecks in production systems. An example of a value stream map can be found in Figure 3.1, where

we can see how production is mapped out to support a customer pulling widgets through a manufacturing flow. We need a similar way of understanding, architecting, and optimizing the flow of business value in large-scale software delivery.

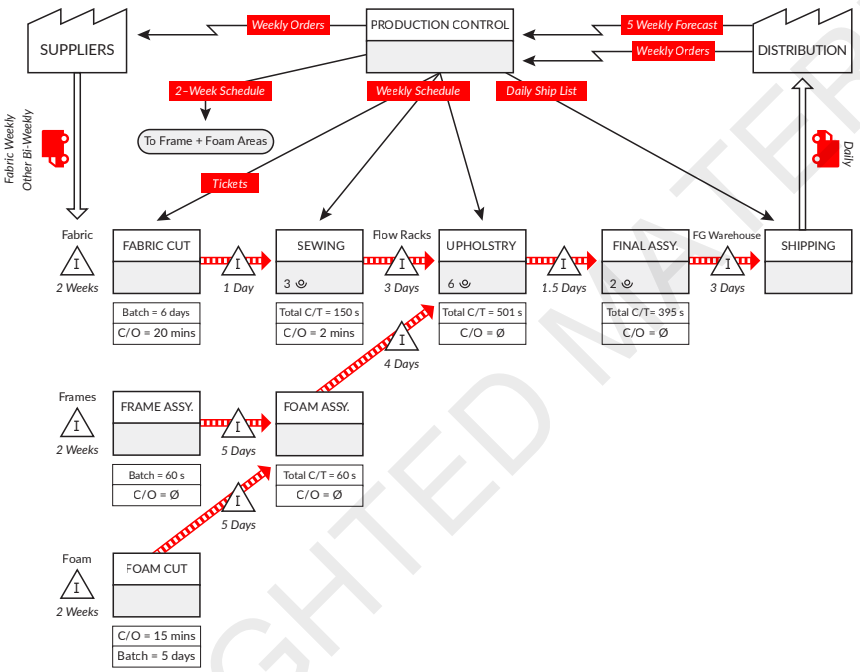


Figure 3.1: Manufacturing Value Stream Map

Finding Flow in Software Delivery

The Flow Framework started with my attempt to visualize manufacturing like production flow for software delivery. The core premise of the Flow Framework is that we need to measure the end-to-end flow of business value. If we measure a subset of the flow, such as the time it takes for developers to complete an Agile "user story" or the time it takes to deploy software, we can only optimize a segment of the value stream. The goal of the Flow Framework is an end-to-end view that

we can correlate to business results. As such, the top level of the Flow Framework only focuses on how end-to-end flow items and metrics are correlated to business outcomes. The definition of flow is similar to what we know from manufacturing, but it is specific to what flows through a software delivery process.

> ***Software Flow:*** *The activities involved in producing business value along a software value stream.*

The Flow Framework focuses entirely on the end-to-end value stream flow and its correlation to business results (Figure 3.2). The measurement is done via the ground truth of software delivery that is observed by the flow of artifacts through the Value Stream Network (as detailed in Part III). Agile and DevOps metrics and telemetry lie a layer down below the Flow Framework. For example, if an Agile team is constantly struggling with meeting its release goals, the SAFe or Scrum frameworks can provide metrics and guidelines for better prioritization and planning. In contrast, the Flow Framework is focused on the end-to-end metrics used to identify where those bottlenecks might lie in the first place—if they are upstream or downstream of development, for instance.

In addition, the Flow Framework avoids measures of activity in favor of tracking flow metrics and correlating them to results. There are no metrics of "how Agile" or "how DevOps" an organization is, just a focus on how much business value flows through each value stream and what results it produces. If responsiveness to the market is a key need, the Flow Framework can highlight flow and feedback cycles that are too slow for a particular value, implying that more Agile practices may be needed.

The role of the Flow Framework is to help you determine the outcomes of your investments in Agile and DevOps practices, and to supply you with the metrics needed to improve those practices. In summary, the goal is to provide you with a means of scaling flow, feedback, and continual learning to work not just for Dev and Ops but for the end-to-end business process of software delivery.

At the top level, the Flow Framework provides two things. First, the Value Stream Metrics allow you to track each value stream within the organization so that you have a way of correlating production

metrics to business outcomes. Second, the Value Stream Network layer provides the infrastructure needed to measure the results delivered by each product.

At the highest level, the Flow Framework is a mechanism for aligning all delivery activities in your organization around your software products, tracking the business results of those activities in order to create a results-driven feedback loop.

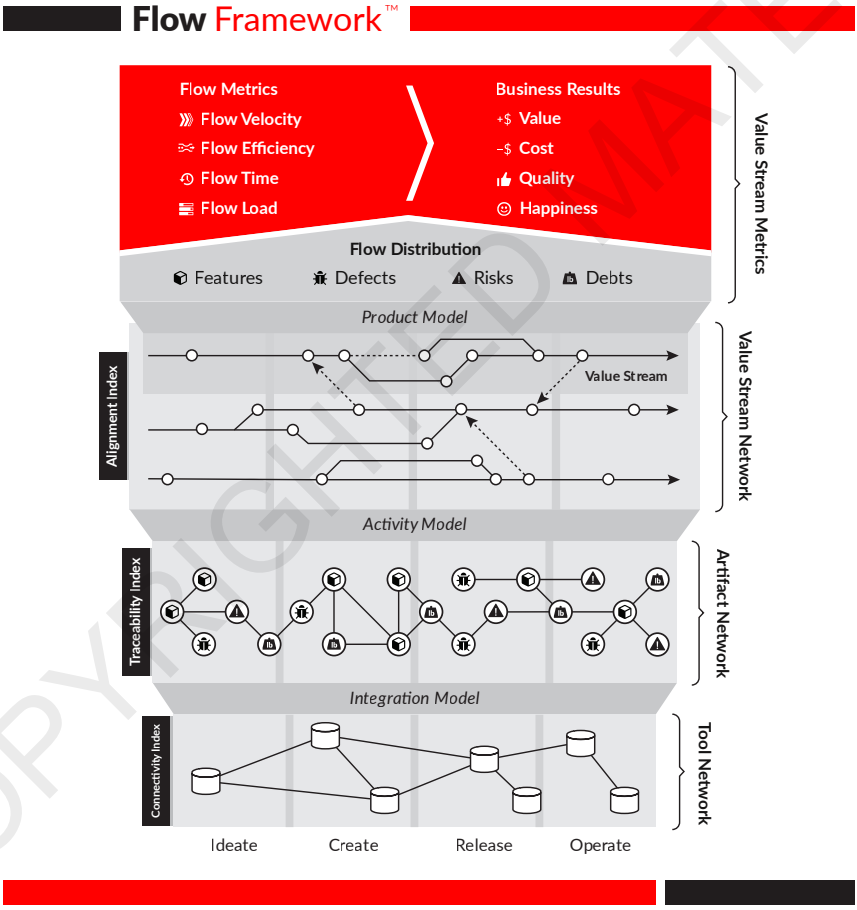


Figure 3.2: The Flow Framework

To do that, we must switch to first principles and define the customer, what they are pulling, and how this pull can be implemented as value stream flow. Once one or more value streams are defined, we need to focus on making value(s) flow smoothly across those value streams. But before we do that, we must define the units that flow along a software value stream.

The Flow Framework is designed to work at the largest of organizational scales and to support stringent regulatory requirements where needed (discussed in Part III). This means that even the most traditional, complex, or safety-critical organizations can apply the concepts to drive software innovation at the right pace for their business. In order to do this, we first need to understand the four main flow items that make up the framework.

The Four Flow Items

Every time that I have asked a senior- or executive-level IT leader where their bottleneck is, I have received either a blank stare or a vague answer. But when set in the right context, just the thought process of exploring this question makes a serious issue apparent. The vast majority of enterprise IT organizations do not have a well-defined productivity measure for what flows in their software production process.

It is impossible for the business to have a shared understanding of a bottleneck without having a shared understanding of productivity. Contrast that to the automotive industry, where the number of cars produced is a very clear productivity measure of an automotive value stream. Worse yet, it's not just those organizations that are scrambling to align around metrics that matter; it's the software industry as a whole.

There is no clear consensus from academia or from industry thought leaders on what constitutes software development productivity. Organizations know it when they see it—perhaps through products that drive market adoption and revenue faster than others. But correlating development activities to those results has been more of an opaque art than a disciplined activity. To define productivity in a value stream, we must first define what flows.

To do that, we need to go back to the first principles of Lean thinking summarized earlier in this chapter. Lean thinking starts not with the product but with the value the customer pulls. If we think back to the early days of software, with companies stamping out installer disks packaged in shrink-wrapped boxes, we could try to draw an analogy to car production and define the widgets produced as those boxes. But that analogy was weak then and is rendered irrelevant in this time of continuous delivery and the cloud. If customers are not pulling releases, what value does the customer pull?

To pull value, the customer must be able to see that value and be willing to exchange some economic unit for it. For an internal product, this could be adoption (e.g., having different business units adopt a common authentication system). For an external product, the unit can be revenue; or in the case of a product with indirect or ad-based monetization, such as a social media tool, it might be time engaged with the product. For a government or not-for-profit organization, it can be the adoption rate of a newly-launched digital offering.

Using any of these scenarios, consider the last time you derived new value from a product or went back to using a product that you had previously not used. What triggered that exchange of value in terms of spending your time or your money? Chances are that it was a new feature that met your usage needs and perhaps delighted you in some way. Or it was the fix for a defect that was preventing you from using a product that you had otherwise valued. And here lies the key to defining what flows in a software value stream: if what we are pulling is new features and defect fixes, those are the flow items of a software value stream.

Flow Item: *A unit of business value pulled by a stakeholder through a product's value stream.*

If these are the flow items, that means we could characterize work across all the people and teams within a value stream as applying to one of these items—and we can. Given full visibility into every process and tool in the organization, you could identify exactly how many designers, developers, managers, testers, and help-desk professionals were

involved with creating, deploying, and supporting a particular feature. The same goes for a defect. But is this the only work that is being done within the value streams?

The "Mining the Ground Truth of Enterprise Toolchains" analysis of 308 enterprise IT tool networks (the study I noted in Chapter 1) identified two other kinds of work that are invisible to the user but are pulled through the value stream by a different kind of stakeholder.[5] First, there is work on risks. This includes the various kinds of security, regulation, and compliance that must be defined by business analysts, scheduled onto development backlogs, implemented, tested, deployed, and maintained. In other words, this is work that competes for priority over features and defects, and as such, is one of the primary flow items. This type of work is not pulled by the customer, as regulatory- or compliance-risk work is usually not visible to the customer until it is too late (e.g., a security incident that leads to a number of security defects being fixed and security features being added). It is instead pulled internally by the organization; for example, by the Chief Risk Officer and their team.

The final and fourth kind of work is technical debt reduction, which describes the need to perform work on the software and infrastructure codebase that, if not done, will result in the reduced ability to modify or maintain that code in the future. For example, a focus on feature delivery can result in a large accumulation of technical debt. If work is not done to reduce that technical debt, then it could impede future ability to deliver features; for example, by making the software architecture too tangled to innovate on. Table 3.1 summarizes the four flow items.

While the concepts of risk and technical debt are not new in the Flow Framework, the focus on the measurement of each flow item results in a very different set of conclusions as to how they should be managed. In using the Flow Framework, the only technical debt work that should be prioritized is work that increases future flows through the value stream. Tech debt should never be done for the sake of software architecture alone, like using it to improve the separation of architectural layers. This means that the flow of each of the flow items should shape the software architecture and not the other way around, which is counter to the way many enterprise architectures have been evolving.

Flow Items	Delivers	Pulled By	Description	Example artifacts
Features	New business value	Customers	New value added to drive a business result; visible to the customer	Epic, user story, requirement
Defects	Quality	Customers	Fixes for quality problems that affect the customer experience	Bug, problem, incident, change
Risks	Security, governance, compliance	Security and risk officers	Work to address security, privacy, and compliance exposures	Vulnerability, regulatory requirement
Debts	Removal of impediments to future delivery	Architects	Improvement of the software architecture and operational architecture	API addition, refactoring, infrastructure automation

Table 3.1: Flow Items

By focusing first on flow, the other aspects of architecture such as infrastructure cost and information security can be planned for in relation to their business relevance. For instance, investing in architecture to reduce cost before an Incubation Zone product is validated could be a waste when compared to rearchitecting around cost reduction once the product has demonstrated viability and is ready for the Transformation Zone.

The need to focus on flow is similar for products in the Performance Zone. Case in point: At the 2017 DevOps Enterprise Summit, John Allspaw presented a case for treating production software incidents as unplanned investments in a system's architecture.[6] This is precisely the approach that the Flow Framework is intended to measure and support.

Rather than focusing on the software architecture to support any contingency, the focus should be on predicting the future flow of incidents through the product's value stream and on optimizing the architecture for that. This means architecting for resiliency that minimizes the likelihood of those incidents and creating a software architecture, infrastructure architecture, and value stream architecture that can quickly respond to the remaining unforeseen incidents. The result is analogous to what the BMW Group did with the "fingers"

structure of the buildings: they predicted how the architecture needs to be adaptable to future flows rather than building in all of the support for those flows up front.

The four flow items follow the MECE principle of being mutually exclusive and collectively exhaustive. In other words, all work that flows in a software value stream is characterized by one—and only one—of the flow items. This means that activities such as prioritization of the various flow items are a zero-sum game, as we will explore in Chapter 4.

Other characterizations of software work items exist, such as Philippe Kruchten and colleagues' decomposition of work into a quadrant of positive/negative and visible/invisible (e.g., features are positive and visible, whereas architecture improvements are positive and invisible).[7] These characterizations can be useful for planning development work. Similarly, ITIL defines the important differences between these problems, incidents, and changes that can be useful for characterizing IT service-desk work.[8] However, these taxonomies are a layer down from the flow items and more useful for characterizing the artifact types being worked on in the delivery of the flow items.

Since the flow items are designed for tracking the most generic characterization of work in a way that is most meaningful to business stakeholders and customers, other taxonomies may crosscut flow items. For example, in SAFe taxonomy, which provides detailed definitions of the many kinds of work items in software delivery, the term for architectural work is *enablers*.[9] This kind of architectural-enabler work can be done to reduce debts, to support the addition of a new feature, to fix a defect, or to address a risk by providing the infrastructure needed to support compliance. This means that architecture work items could fall under any of the flow items. The story is similar for performance improvements, as performance work can be done in support of feature work, such as scaling to a new market, or as part of defect fixes, if the existing user base is experiencing a related set of performance problems.

While the layer below the flow items is critical, the Flow Framework's primary focus is on connecting technology and architecture to the business with the minimal number of concepts that executives and technologists can agree on and understand. As such, each of the units

or work items being done by all specialists in the value stream needs to map into one of the four flow items.

Finally, you'll see there is no separate flow item representing improvements to the Value Stream Network to improve flow. In the shift from project to product, the Value Stream Network itself needs to be treated as a product, with its own stable delivery team, and not as a project with a defined end. The majority of Value Stream Network improvements, be they connecting different stakeholders or creating dashboards for flow metrics, will fall on this team. In the cases where teams on a particular value stream need to make changes to their work process—for example, to remove waste by switching from manual compliance checks to an automated security tool—that work falls under the debt flow items for that team.

Conclusion

A large gap exists between what technologists have learned about effective software delivery and how businesses approach software projects. While DevOps and Agile principles have made a significant impact on how technologists work, they have been overly technology centric and have not been adopted broadly by business stakeholders. To bridge the gap, we need a new kind of framework that spans the language of the business with the language of technology and enables the transition from project to product. We need that framework to scale the three ways of DevOps—flow, feedback, and continuous learning—to the entire business. This is the goal of the Flow Framework.

Conclusion to Part I

In Part I, we learned about the five technological revolutions and about how success in the Age of Software is dependent on an organization's ability to shift from project to product. Carlota Perez's work describes how each age is separated into an Installation Period followed by a Turning Point and then a Deployment Period. We are approximately fifty years into the Age of Software and, according to Perez, still somewhere in the midst of the Turning Point.

Organizations that master the software-based means of production and their digital transformations have a chance to both survive and to thrive through this Turning Point. Those that continue to apply managerial paradigms of past ages are likely to decline or die. Tech giants have already mastered this new means of production, and digital startups have been born into this new way of working, but the majority of the world's organizations have not. This is not for want of trying, but the combination of scale, complexity, legacy, and dated managerial paradigms is making that transition impossible to achieve in a time frame that ensures survival. We need a new approach.

Making the transition to thrive in the Age of Software requires a switch from project to product for managing software delivery. Chapter 2 summarized the pitfalls of project-oriented management, while Chapter 3 introduced the Flow Framework as a remedy. The four flow items—features, defects, risks, and debts—provide the simplest and most generic way to unlock the black box of IT and software delivery.

The challenge now is for the business to learn to see what's in that box once opened. Technologists already see it. They are able to track the value delivered on the software products they work on and have had a decade of mastering Agile practices to understand, prioritize, and communicate with each other. The problem is we have not equipped all stakeholders with a common language that bridges the gap between the business and technology. In Part II, the language of Value Stream Metrics is introduced to do just that.