Transform Your Conversations,
Transform Your Culture

"This is an important book. I am glad that Jeffrey and Squirrel have written it, and I look forward to having a stack to "lend" (read: give) out."

—**Alistair Cockburn**,
Co-Author of the *Agile Manifesto*
and Creator of the Heart of Agile

Agile Conversations

DOUGLAS SQUIRREL
and JEFFREY FREDRICK

PRAISE FOR *Agile Conversations*

"I think of this highly practical and immediately applicable book as the lost instruction manual to the most powerful and effective Agile tool we don't even realize we have: Conversations. If you want to be successful with Agile, RTFM: Read This Fabulous Manual."

—**Alberto Savoia**, Google's first Director of Engineering and Author of *The Right It*

"*Agile Conversations* provides much needed guidance on how to have the key discussions that form the foundation for strong and resilient working relationships, as well as a toolbox full of techniques for troubleshooting when conversations go awry. Full of practical, real-world examples, this book is a must read for anyone who has ever been frustrated or puzzled by interactions at work."

—**Elisabeth Hendrickson**, Technology Executive and Author of *Explore It!*

"Many books have been written about improving process and product in companies. I'm so glad this book finally addresses the people aspect: Learn to ask the hard questions, leave your bias at the door, and improve your own communication with this practical guide. If it's hard, do it more often!"

—**Patrick Debois**, Founder of DevOpsDays and Co-Author of *The DevOps Handbook*

"This book provides an engineer's approach to monitoring, troubleshooting, and debugging conversations.... Heuristics such as the Question Fraction are amazing—at the same time simple and memorable and incredibly insightful. Read this book to turn your communication skills into superpowers."

—**Gojko Adžić**, Author, and Partner at Neuri Consulting

"Vital reading for anyone in a leadership role or interested in improving their work culture in general."

—**Andy Skipper**, Chief Coach at CTO Craft

"If you're looking for a practical framework and techniques that will help fix broken team communication and dysfunctional culture, then you should read *Agile Conversations*. Going beyond simple diagnosis, this handbook walks you through the Five Conversations you need to embrace in order to transform a broken culture into one that's healthy and high-performing."

—**Paul Joyce**, Founder and CEO, Geckoboard

"Squirrel and Jeffrey's keen writing and battle-tested techniques make this book a must-read for modern engineering leaders looking to thrive amid the explosion of complexity that we all face."

—**Chris Clearfield**, Co-Author of *Meltdown: What Plane Crashes, Oil Spills, and Dumb Business Decisions Can Teach Us About How to Succeed at Work and at Home*

"This is a very wise and yet readable book. The authors have hit the nail on the head by focusing on better conversations as the way to translate theory into organizational improvement."

—**Rich Koppel**, Co-Founder and CEO of TIM Group

"It's one of the industry's dirty little secrets that most of our 'technology' problems are actually people problems. In *Agile Conversations*, Jeffrey and Squirrel assert that solving these problems is made possible by having better conversations, presenting their advice in a manner which fellow technologists will find reassuringly structured and categorized."

—**Jon Topper**, Founder & CEO, The Scale Factory

"It takes conviction and skill to change company culture. *Agile Conversations* provides the road map to summon your courage and avoid the hazards on your path to success! A masterpiece for any CEO wanting to build a collaborative, cooperative organization!"

—**Brent Delehey**, Turnaround Specialist, CEO

"This book is an extremely helpful and practical guide on how to read between your own lines, and make it safe for others to reveal the fears between theirs."

—**Rebecca Williams**, Software Engineer at QA Chef

Agile Conversations

Transform Your Conversations, Transform Your Culture

DOUGLAS SQUIRREL
and **JEFFREY FREDRICK**

IT Revolution
Independent Publisher Since 2013
Portland, Oregon

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

First Edition
Printed in the United States of America
25 24 23 22 21 20 1 2 3 4 5 6 7 8 9 10

Cover and book design by Devon Smith

Library of Congress Catalog-in-Publication Data

Names: Squirrel, Douglas, author. | Fredrick, Jeffrey, author.
Title: Agile conversations : transform your conservations, transform your
culture / by Douglas Squirrel and Jeffrey Fredrick.
Description: First edition. | Portland, Oregon : IT Revolution, [2020] |
Includes bibliographical references.
Identifiers: LCCN 2019045822 (print) | LCCN 2019045823 (ebook) |
ISBN 9781942788973 (paperback) | ISBN 9781942788669 (epub) |
ISBN 9781942788676 (kindle edition) | ISBN 9781942788683 (pdf)
Subjects: LCSH: Communication in management. | Teams in the workplace. |
Information technology—management.
Classification: LCC HD30.3 .S698 2020 (print) | LCC HD30.3 (ebook) |
DDC 658.4/5—dc23
LC record available at https://lccn.loc.gov/2019045822
LC ebook record available at https://lccn.loc.gov/2019045823

ISBN: 978-1942788973
eBook ISBN: 978-1942788669
Kindle ISBN: 978-1942788676
Web PDF ISBN: 978-1942788683

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

Names have been changed in example conversations and stories for privacy.

AGILE CONVERSATIONS

Contents

Figures and Tables

Introduction

A Word from the Authors

As a leader at your company, you've given the transformation your full support, and the organization has bought in. You've had the consultants in, they've trained the teams, and the process is in place. All that's missing are the promised results. Why aren't things better?

As a contributor—an engineer, product owner, Scrum Master, system admin, tech lead, tester, or any other "doer"—you've had the training, written the tickets, and gone to the meetings. You've bought in and are ready to see improvements. All that's missing are the promised results. Why aren't things better?

After years of study and many missteps, we have come to understand that the key to success is not only *adopting practices* but *having the difficult conversations* that foster the right environment for those practices to work. You, your managers, and your teams are missing the right relationships, built only by having the right conversations. The good news is that you can begin a conversational transformation that builds the foundation for any other improvements you want to make, changing your conversations, improving your relationships, and finally getting results.

We've seen it happen. Between us, we've consulted with over one hundred organizations across a range of subjects and at all levels. It has been our

surprising experience that no matter whether we are talking with a CEO or the most junior developer, a managing director at a multinational bank or an operations engineer at an online retailer, a product owner or a project manager, a designer or a developer, in every case we hear, "Why won't *he* do it better? Why won't *she* change? I can't make them. I'm powerless."

Employees' frustration and despair at their inability to change things exist at all levels and in almost all organizations we've worked with—and we feel empathy, because we get stuck in this pattern ourselves.

So it has been our great joy to offer an alternative: the tremendous power of conversations based on transparency and curiosity.

We have regularly and reliably seen individuals, teams, and whole organizations get unstuck and start seeing improvements faster than they thought possible when they unlocked their conversational superpowers: a children's book publisher that got artists and marketers talking and harnessed creative inspiration for successful sales; an AI startup that involved everyone in setting strategy, seeing big improvements in user satisfaction; a financial-services firm that stabilized its systems through tough discussions of its failings.

Great results follow when you learn that a conversation is about more than just talking; it is a skilled activity. There is more to a conversation than what you can see and hear. In addition to what is said out loud, there is what has been left unsaid—the thoughts and feelings behind our spoken and unspoken words.

As we become more skillful at conversations, we become more aware of *what* we think and feel, and *why* we think and feel the way we do. Therefore, we become better at sharing that information with others. We also become more aware that we don't have telepathy—that we don't actually know what information our conversational partners have—so we get better at asking questions and listening to the answers. These skills are so fundamental, and so neglected, that when we get better at them, our conversations become radically more productive and our culture becomes much more collaborative.

There is no shortage of books that tell you how to *diagnose* cultural problems, offering detailed case studies and stories, diagnostic tests, lots of practices to follow, exhortations to collaborate, and tools to use. But few say anything meaningful about how you actually cure those problems—how to make changes and what to do when you're stuck.

For example, Patrick Lencioni's *The Five Dysfunctions of a Team: A Leadership Fable* details the fall and rise of the fictional company DecisionTech. Through this corporate fable, Lencioni develops a theory of dysfunctional hierarchy: Inatten-

tion to Results arises from Avoidance of Accountability, which, in turn, is rooted in a Lack of Commitment, and so on through Fear of Conflict and Absence of Trust.[1] Lencioni's model of dysfunction is helpful, and it inspired four of our Five Conversations (which you'll learn about later in this book). But crucially, Lencioni offers little practical advice for removing dysfunction once you find it.

To build trust, he says, you can do one of five things: share your personal history, discuss your team members' most important strengths and weaknesses, provide feedback, run personality type analyses, or go on a ropes course.[2] While these are likely to make your team more friendly or intimate, Lencioni offers no evidence or argument that they will actually create trust, nor does he provide any alternatives for building trust if they fail.

Lencioni is not alone in hanging the reader out to dry in this way. Business fables, digital transformation guides, Agile manuals—all of them tell you what's wrong with your culture but not how to fix it. As a result, we have seen company after company implement all the right practices but fail to see results because they haven't fixed their cultural glue, the very thing necessary to make those other practices work.

This book and the use of our conversational methods will teach you and your team to not only diagnose your cultural problems but actually cure them. We have seen over and over that holding difficult conversations with an attitude of transparency and curiosity *does* help teams build lasting trust, reduce fear, and make other key improvements; and that it's easy to explain how and why this method works, which is what we will do in this book.

If you have the appetite, you can develop the skills that allow you to embrace the *painful, candid communication* that creates an environment in which teams flourish. At no point will developing these skills be easy. In the words of our friend Mark Coleman, it will demand "difficult, emotional work"[3] from you at every stage. You will have to confront your own dread of painful topics—and more than once, instead of taking on yet another challenging discussion, you will wish you could just bring in a consultant, or polish your burndown chart, or add some monitoring. But we can assure you that there is nothing as rewarding as working in an organization whose members have mastered all of the Five Conversations, and for whom the never-ending quest for excellence is a habit and a joy.

We look forward to you joining us in learning, developing, and implementing the conversational skills that get you there.

Keep talking,
Jeffrey and Squirrel

How This Book Is Organized

We've divided the book into two parts: Part I describes the ideas and theories that underpin the conversational tools that we will introduce in Part II.

Chapter 1 is a bit of software history. If you want to get directly to techniques, you can skip this chapter; but if you are curious about the origins of Agile, Lean, and DevOps, this chapter is for you. We will examine the dramatic changes that have transformed the software industry over the last twenty-five years, recapping what we lived through, including both the progress and stumbles along the way.

In the 1990s the mass manufacturing paradigm provided the intellectual model for the "software factory." Just as factory workers were expected to be interchangeable units, bound to the assembly line, so, too, were software professionals expected to be interchangeable units, following the dictates of the document-driven approach to software development. When that model proved disastrously flawed in practice, it created space for the rise of a host of people-centric methodologies and the waves of transformations that have swept across software organizations, such as Agile, Lean, and DevOps.

Ironically, as commonly implemented as these transformations are, they often miss the people-centric core, and a bureaucratic focus on processes and practices leaves organizations stuck with cultureless rituals. To advance, organizations will need to tap into the unique human power of conversations, overcoming their cognitive biases by learning to have difficult but productive conversations.

Chapter 2 provides the core techniques of our method: the Four Rs provide steps to help us learn from our conversations, and the Two-Column Conversational Analysis gives us both a format we use throughout the book for recording conversations and a method for learning from them. We recommend reading at least the two sections on these techniques and the section "Analyzing a Conversation" before proceeding.

We start the chapter by showing that you already know where you need to go. Following illustrious social scientist Chris Argyris, we'll show you that your "espoused theory" already says that the best decisions require collaboration, transparency, and curiosity from all involved. Unfortunately, your "theory in use," how you actually behave in conversations, is something quite different. We will show you a method called "the Four Rs," a set of techniques that allow anyone and any team to improve their skills in approaching difficult topics in

their conversations, which will help you learn from your conversations and prepare for what comes next.

In Part II, Chapters 3 through 7, we've distilled our experience, learnings, and mistakes into an "instruction manual" for the Five Conversations: crucial discussions of the five key characteristics that *all* high-performing teams share—not just software teams but all human teams.

The Five Conversations are:

1. **The Trust Conversation:** We hold a belief that those we work with, inside and outside the team, share our goals and values.
2. **The Fear Conversation:** We openly discuss problems in our team and its environment and courageously attack those obstacles.
3. **The Why Conversation:** We share a common, explicit purpose that inspires us.
4. **The Commitment Conversation:** We regularly and reliably announce what we will do and when.
5. **The Accountability Conversation:** We radiate our intent to all interested parties and explain publicly how our results stack up against commitments.*

These five conversations address attributes that give teams everything they need to exploit modern, people-centric practices to the hilt. With them it is possible to achieve elite-level delivery speeds, fearlessly adjusting on the fly and committing to show real customers working software that solves their problems. And these are exactly the characteristics missing from the teams we see too often today, where standups are places to hide progress rather than share it, where estimation is a forlorn exercise in futility, where the team's purpose is lost in a sea of tickets, and where frustration is the shared emotion from one end of the organization to the other.

Starting with the Trust Conversation, and continuing with conversations addressing Fear, Why, Commitment, and Accountability, we will show you in

---

* Four of the Five Conversations were inspired by the Five Dysfunctions identified by Patrick Lencioni.[4] The fifth, the Why Conversation, was inspired by Simon Sinek's *Start with Why: How Great Leaders Inspire Everyone to Take Action*.[5] To each we have added our own experiences and approaches, and we are grateful to both authors for their inspiration.

detail and step by step how to improve each of these five key attributes in your team. You'll be able to use these methods whether you are a junior developer or a senior executive, and we'll explain how these improvements will translate directly into improved results from your Agile, Lean, and DevOps practices. We'll illustrate how these methods work in real life, with practical examples of conversations on each topic.

Chapters 3 through 7 are each divided into similar sections:

- A *motivational* section explaining why the chapter's featured conversation is important.
- A *story* section introducing a protagonist experiencing a problematic conversation of this type.
- One or more *preparation* sections that teach you methods introduced in the chapter's conversation.
- An *explanatory* section ("The Conversation") that describes one way to hold the chapter's conversation.
- A section that *continues the story*, where our protagonist learns from the problematic conversation and produces a better result.
- Several *example conversations* that illustrate variations on the chapter's conversation.
- A *case study*, which tells a longer story about how the chapter's conversation helped an organization improve.

But reading to the end of the book is only the beginning. Having learned how to approach each of the key conversations, it will be up to you to practice what you've learned. If you do, we are confident you'll find the effort rewarded many times over. When you transform your conversations, you transform your culture.

The Many Ways to Read This Book

Perl developers have a catchy acronym: "TIMTOWTDI," or "there is more than one way to do it." That's our philosophy, as you'll see throughout this book; so long as you address the Five Conversations in one way or another, we aren't prescriptive about which practices you use. We think answers to questions like how long your iterations should be, whether you need standups, or what color of planning poker cards to use are less important than how you get to those

answers. Similarly, we've tried to write this book so you can use it in multiple ways, depending on your learning style, needs, or mood.

So here are some suggestions for ways to read the book, but TIMTOWTDI, so we don't mind if you come up with your own!

Linear. If you like to understand every idea the first time you encounter it, this is the method for you: start at page one and keep reading until you run out of pages. We've tried to define and illustrate each new idea or technique before using it, avoiding forward references wherever possible. So if you master Test-Driven Development for People in the Trust Conversation, you won't have trouble when it crops up two chapters later in the Why Conversation. And within each chapter, you'll find the kind of logical progression you like: first the reason for having that chapter's conversation, then techniques to use it, followed by the conversation itself, and finally practical examples. Reinforce your learning by using the Four Rs to work through the sample conversations at the end of each chapter and your own examples. Involve one or more friends in your step-by-step learning if you can.

Technical. "Don't confuse me with stories; get to the methods I can use." If this is you, then start in the preparation section of each chapter, where we explain techniques you can begin practicing immediately to improve your conversations and, therefore, your team's performance. Keep reading through the explanation of the main conversation, which brings the techniques together into a whole, until you get to the example conversations, which illustrate the conversation in action, and from which you can lift phrases and approaches. Having chosen this path, we recommend you proceed at the rate of no more than one method per week, and spend each week deliberately practicing the methods in your everyday conversations. At the end of each day, make a count of how often you were able to apply each method, and choose one conversation to analyze using the Four Rs. Outwardly slow and steady, this reflective practice will quickly build skills if you stick with it.

Social. As we describe in the Conclusion, other people who are also interested in learning these skills can be a tremendous aid in learning the material. The cognitive biases that make these conversations

difficult also make it more difficult to spot our own mistakes. Other people will have no such difficulty! If you are fortunate enough to have such a learning group, we would recommend that, as a group, you follow a similar path as the Technical approach on the previous page. Cover no more than one chapter per week, keep and share your count of how often you could apply the methods of that chapter, and then discuss and analyze one of your conversations in a group session. Role playing and reversing roles with others will help you gain confidence in your performance; the practice of giving feedback to others will help you spot opportunities for improvement in your own conversations.

Whatever approach you take to reading the book, it is worth emphasizing that understanding is simply not enough; you build the skills by practicing. There is no other way.

Part I

Chapter 1

Escaping the Software Factory

According to Michael Gale, author of *The Digital Helix: Transforming Your Organization's DNA to Thrive in the Digital Age*, 84% of digital transformations fail.[1] Trying to understand why the other 16% succeed, he found that success required a "fundamental shift in how people had to think about how they interact, how they collaborate and work, and if you don't spend time changing people's behaviors, you don't spend time changing culture and how people make decisions, all of this falls flat."[2]

Our way to make this fundamental shift is to turn to the most human of abilities: the conversation. Humans have a uniquely powerful and flexible language. To get the most out of it, we need to learn the skill of conversation and how to overcome our innate biases, which work against collaboration and connection. When we change our conversations, we change our culture.

To understand the change we need, it helps to understand the culture we are coming from. As we describe in this chapter, we are still in the process of emerging from the mass-manufacturing paradigm of the software factory. This document-only model represents an attempt at communication without conversation. The failure of the software factory model has created space for the rise of new, people-centric models like Agile, Lean, and DevOps. But well-meaning attempts to adopt these new models fail when the focus is on processes and methods, reinventing some of the same mistakes of

the software factory on a smaller scale, creating "feature factories," as John Cutler calls them.[3]

Laboring in the Software Factory

We both began our careers in the 1990s in medium-size software companies,* wrangling C code on huge, desk-bound PCs alongside dozens or hundreds of our colleagues doing the same thing. We were small parts embedded in a much larger system. And no wonder, since the systems we were part of were an expression of the twentieth-century philosophy known as Taylorism.

A machinist and mechanical engineer by trade, Frederick Winslow Taylor led a professional crusade against waste and inefficiency, becoming one of the first management consultants in the process. At the heart of the waste, in Taylor's mind, was the wide variation in how work was performed from one worker to the next. It would be much better, he reasoned, for everyone to be taught the one right way, and then for workers to follow that way without deviation; any other approach must necessarily be less efficient. And who determined the one right way? Professional managers and consultants such as Taylor himself.

"Scientific management," the philosophy of which Taylor was the most vocal proponent, says that it is the job of managers to devise and understand the best way of working and then enforce unswerving standardization. With his influential 1911 book *The Principles of Scientific Management*, Taylor provided the intellectual underpinning for mass manufacturing based on the assembly line, with low-skilled workers doing the same simple tasks again and again under the watchful eye of management.[4]

Taylor's view of the world created a very distinctive and dehumanizing workplace culture. The factory was envisioned as one giant machine. Managers were the mechanical engineers, designing how all the pieces should work and checking that they were operating correctly. Workers were merely replaceable parts: they operated within specified tolerances, or they were defective and to be discarded. Communication was top-down and limited to commands and corrections. Conversations were not required. Collaboration

* Jeffrey at Borland, Squirrel at Tenfold. Both have long since been swallowed by bigger fish.

was not required. Thought was not required beyond doing the job you had been told to do.

Taylorism in the Cubicle

The software industry we joined in the nineties had transplanted Taylor from the factory into the cubicle. Consultants and salesmen promised managers ease and efficiency in the new tools, new processes, and new methodologies they were peddling. "Having trouble with software development? Bugs and delays got you down? Never fear! We have the best practices written down and ready to go." Management could buy the system off the shelf and tell developers to follow along. And as the work flowed through every defined checkpoint and process gate, you could be confident that you would be on time and on budget. Or so was the promise.

Lacking a mechanical assembly line, the software industry created one from paper. Managers adopted or invented logical models describing the single, right way to work and codified them in documents with step-by-step instructions and flowcharts. This documentation-driven development was designed to stamp out any possibility for error by specifying how each part of the final program would function and what each software worker would do to make it happen. There was the marketing requirements document, the product specification, the architecture documentation, the implementation specification, the test plans, and more—no human activity was left unaccounted for. Thick manuals prescribed in painstaking detail the attributes of every data structure, which language constructs could be used, and even the format of comments. Meticulous software designs arrived on designers' desks, with every database column specified, every validation defined, and each screen carefully illustrated down to the last pixel.

But there *was* a logic to this. Everyone knew it was more expensive to fix defects after the software was shipped to clients. In fact, the earlier we found the bugs, the cheaper they were to fix. It was less effort to fix the designers' flowchart than to fix the code, and less effort still to update the specification document than to change the flowchart. The inexorable logic dictated that we should spend the time up front to get everything right, and then mechanically implement to save ourselves time and money downstream. It was all very sensible, all very rational.

Unfortunately, it didn't work very well.

The Software Crisis

Just how poorly this system worked was documented by the Standish Group in their infamous 1994 *CHAOS Report* on the shocking level of failure in software projects. Unlike the failures of bridges or airplanes or nuclear plants, the authors noted that "in the computer industry . . . failures are covered up, ignored, and/or rationalized." So they set out to identify "the scope of project failures," "the major factors that cause software projects to fail," and "the key ingredients that can reduce project failures." The report concluded that 31% of software projects in the United States were cancelled, costing American software companies $81 billion. Only 16% of projects were completed on time and on budget.[5] The report laid bare the failures of Taylorist methods and the software crisis they had created.

With the crisis widely acknowledged, there was no shortage of people searching for answers. One school of thought was captured in the Capability Maturity Model (CMM) from the Software Engineering Institute at Carnegie Mellon University. Created to help the US Department of Defense assess software contractors, CMM doubled down on the importance of documentation and process. Adherents of this approach implemented greater supervision, more checks, and additional specifications in their quest for predictability. "A software development process which is under statistical control will . . . produce the desired results within the anticipated limits of cost, schedule and quality," they asserted.[6]

Others from across the software industry, including software practitioners on the ground and those working with them, found inspiration and ideas in other places. Being on the front line of projects, they had the scars showing that the idealized, mechanical approach, no matter how reasonable it sounded, didn't explain why software projects succeeded or failed. Rather than working from first principles, they observed what worked in practice. In trying to understand their experiences, they found the answer couldn't be captured in reams of documentation. It wasn't a tool you could buy, and it wasn't in the mechanical application of process. It was *people*!

Dr. Alistair Cockburn, a keen and eloquent observer of software practices in the wild, captured the insight in his paper "Characterizing People as Non-Linear, First-Order Components in Software Development." The title tells the story. In stark contrast with the CMM, Cockburn said he, "now consider[s] process factors to be second-order issues."[7] He found that it was largely people

who make projects succeed or fail, and suggested we focus our improvement efforts on harnessing the unique attributes of people.[8] For example:

1. People are communicating beings, doing best face-to-face, in person, with real-time question[s] and answer[s].
2. People have trouble acting consistently over time.
3. People are highly variable, varying from day to day and place to place.
4. People generally want to be good citizens—are good at looking around, taking initiative, and doing "whatever is needed" to get the project to work.[9]

This view of people is antithetical to the Taylorist view of people as mechanical, interchangeable parts. Expecting them to act as such ignores human nature and is doomed to failure. The empirical finding was that the culture of how people relate to one another and how they communicate on projects was important. Practitioners could see that we should be designing approaches and projects around people, not processes. We needed to be having the right conversations, building the right culture, if we wanted to improve our chances of success.

Smash the Machines—Maybe?

The idea that people are the central concern of software methods sparked an extended transformation that has reshaped the building of software since the turn of the century. Lean manufacturing disrupted and transformed the previously dominant Taylorist mass-manufacturing paradigm, scoring tremendous gains in productivity and quality by changing the culture of the factory. Rather than viewing workers as replaceable parts, Lean manufacturing relies on "both an extremely skilled and a highly motivated workforce," one that anticipates problems and devises solutions.[10]

Agile software development, Lean software, and DevOps have similarly disrupted and transformed the software factory. Each of these approaches targeted different elements of the factory, but they all started with a break from the dehumanizing, mass-manufacturing approach. They changed the culture by breaking down divisions of labor and introducing collaboration in the place of rigid process.

As we're about to illustrate in the following sections, the early proponents in each movement implicitly espoused two fundamental values, *transparency* and *curiosity*, which led them to advocate methods that developed some or all of our five key attributes of successful software teams: high trust, low fear, understanding why, making commitments, and being accountable. And these values and attributes were all about human connections, information flow, eliminating barriers, and collaboration—everything the software factory wasn't.

The initial wins from each successive movement were amazing: early adopters reported dramatic improvements in time to market, reduced bug rates, and higher team morale. The Lean Startup advocates, for example, boasted about "doing the impossible [releasing to production] fifty times a day."[11] Unsurprisingly, many others, including both of us, climbed on the bandwagon and tried the new methods to see if we could get the same results.

The problem—and the reason for this book—is that during the explosion of Agile development, and then of Lean software and DevOps, later adopters missed the importance of human interaction. Leaders thought they could behave the same as they always had—could keep their factory mind-set—and that dictating change to others would be enough. As a result, they focused on the easily monitored, more superficial process changes: standups, work-in-progress limits, tool selection.

Without the human element and without the right conversations, these changes were singularly ineffective. As a result, across hundreds of organizations between us, we've repeatedly seen disillusioned executives and frustrated teams declaring that Agile development (or Lean software or DevOps) *just doesn't work*.

To summarize, they took a mechanistic view of joining the human-centric transformation, missed or denied the importance of relationships, and then wondered why nobody was cooperating and nothing was getting done.

This book, by contrast, is all about getting back to the basic human interactions you'll need to succeed. Let's start by reviewing the history of each movement, which will prepare us to look at the simple technique you need to master to get people back into your process: the conversation.

Agile: People-Driven Development

By the end of the 1990s, the rebellion against the software factory had produced a Cambrian explosion of alternative approaches to software. Bucking the

dominant paradigm of "documentation driven, heavyweight software development processes,"[12] the members of the new movements advocated heretical practices like just-in-time design, frequent delivery of working software, and involvement of real customers in software production. Most extreme was a cultural change that called for a radical reduction in planning activities in favor of collaborative interactions between individuals doing the work. To those used to the dominant practices of the software factory, these leaders seemed like a crazed mob bent on creating mayhem. And yet there were stories of amazing results—elevated morale, rapid delivery, high quality—in the teams brave enough to try these new methods.

Then, seventeen of the most visible proponents of the "lightweight software" movement gathered at the ski resort in Snowbird, Utah, in February 2001. As documented by James Highsmith, they were a diverse group, including the founders and advocates of Extreme Programming (XP), SCRUM, DSDM, Adaptive Software Development (ASD), Crystal, Feature-Driven Development (FDD), Pragmatic Programming, and others.[13] The question was, could they find common ground?

As it turns out, they did, and the result was, as Martin Fowler said, a "call to arms," a "rallying cry":[14] the Agile Manifesto, which continues to be widely used nearly two decades later.

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.[15]

The group followed the manifesto with an often-overlooked set of twelve accompanying principles. This set of principles remains a useful touchpoint for organizations who aim for agility.

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity—the art of maximizing the amount of work not done—is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.[16]

Together with the manifesto, these principles reflect the people-centric nature of the new methodologies. Calling out a few of the principles, we can see that common practices provide a framework for transparency and curiosity in an Agile team:

- *Business people and developers must work together daily throughout the project:* Commonly embodied as a short daily meeting—and called a

standup regardless of the actual posture of the participants—this is an opportunity for developers to transparently share both progress and obstacles, and to ask for support from others on the team as needed.

- *The best architectures, requirements, and designs emerge from self-organizing teams*: Agile teams are expected to collaborate openly, discussing the trade-offs across alternatives, with each member transparently sharing their professional judgement and being curious about the judgement of others. This can be witnessed in practices such as the Planning Game, where estimates are publicly shared (transparency) and the differences are used to spark conversations (curiosity) to uncover what influences differences in opinion.
- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly:* One of the signature Agile practices is the retrospective, an opportunity for team members to discuss their experiences, both as individuals and as a team. There is a wide range of retrospective activities, such as those captured in the book *Agile Retrospectives: Making Good Teams Great*, and they all rely on the ability and willingness of team members to transparently share their experiences, and the curiosity of the team to learn about the experiences of others on the team.

Perhaps the most dramatic change introduced by Agile development was not how people on the team related to one another, but how Agile practitioners embraced customer collaboration, as reflected in the first two principles: *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software"* and *"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."* Together, these two principles enshrine transparency and curiosity as the core protocols of Agile development between the team and the customer. By delivering software frequently, the Agile team provides the customer a transparent view of their progress, and the team is curious about what the customer would value next, even if it means upsetting their plans.

Agile development has never been just one thing, one set of practices, and certainly not the Taylorist vision of one best way to do things imposed everywhere. What it provides is a set of values and guideposts to build a resilient

organization that can adapt to the tumultuous world in which we live.* At the core, the Agile approach demands a culture that can support collaboration and learning—a culture that has created conversations between people who had no reason to talk to each other in the software factory. The success of Agile has opened the doors to radical thinking far beyond the software teams in which it incubated.

Why, then, do we now see, over and over again, "feature factories" operating in the same spirit as those software factories of the nineties while claiming to be Agile? In fact, only the visible artifacts and processes have changed; the mind-set and the conversations, and therefore the culture, are the same. Instead of mounds of documentation and two-year project plans, these teams are fed a steady stream of features to implement without any connection to the upstream customer needs or the downstream business impact. The software assembly line has been replaced by a sweatshop, with piecework assigned to each worker. The labels are different—requirements may come in the form of stories or acceptance criteria instead of a monolithic document; there may be a burndown chart in place of a Gantt chart—but the outcome is the same: disconnected development, barriers to collaboration, endless handoffs, painfully slow progress, and often wrong software at the end of it all.

Lean Software: Empower the Team

Stealing from Toyota

In 2003, just a couple of years after the Snowbird meeting gave rise to Agile software development through the Agile Manifesto, two deeply experienced programmers and software team leaders (who happened to be married to each other) brought ideas from Lean Manufacturing into Agile circles with their book *Lean Software Development: An Agile Toolkit*. Mary and Tom Poppendieck delved heavily into the Toyota Production System's just-in-time, waste-reducing manufacturing methods to find insights, and translated them for the software world.

* In fact, Alistair Cockburn and a number of his colleagues have begun coaching companies on adopting a methodology-agnostic approach that provides exactly these simple guidelines, called "Heart of Agile."

The Poppendiecks distilled the essence of Lean Software into a set of challenging principles:[17]

1. Eliminate Waste
2. Amplify Learning
3. Decide As Late As Possible
4. Deliver As Fast As Possible
5. Empower the Team
6. Build Integrity In
7. See the Whole

The Poppendiecks emphasized themes of optimization everywhere, rapid learning through frequent delivery, and system tuning and thinking. None of these themes are compatible with the software factory, and many build on the Agile principles that were already chipping away at the status quo when the Poppendiecks wrote their first book in 2003.

Soon, Lean thinking and practices began to spread. Lean software teams:

- *Draw value-stream maps*, just like their manufacturing colleagues, to locate inefficiencies (also known as *muda*, a Japanese word for waste).
- *Look for and eliminate bottlenecks in their processes*, from initial feature invention all the way to customer delivery and adoption.
- *Aim to limit work in progress*, so a QA team, for example, will accept only ten new features to test rather than building up a huge backlog of work it can't process.
- *Emphasize "pulling" features* from one process step to the next rather than "pushing" them; so a programmer might pause her coding if there is no one available to review her work, rather than continuing to build up more "inventory" of unreviewed, unreleasable code.

As more companies embraced Lean software principles, more Lean manufacturing ideas showed their utility, including the Theory of Constraints for managing bottlenecks and the Kanban method, which did away with even the lightweight timeboxing of "sprints" in favor of a direct application of pulling work to regulate flow. Always pushing to "see the whole," Lean software expanded to encompass operations in organizations of all sizes, from startups (as in Eric Ries's *The Lean Startup: How Today's Entrepreneurs Use Continuous*

Innovation to Create Radically Successful Businesses) to multinationals (as in *Lean Enterprise: How High Performance Organizations Innovate at Scale*).

Empowerment Is the Key

At first glance, unlike the Agile Manifesto, there is little explicitly in the Poppendiecks' principles or Lean practices that suggests we need to worry very much about messy, inconvenient people and their difficulties in communication. Looking at the principles, we see that all but the second (Amplify Learning) and fifth (Empower the Team) are about processes and efficiencies. It's easy to conclude that all we need is technical analysis of value stream maps and cold, calculated elimination of waste—and many adopters of Lean methods did exactly this, to their cost.

But as Mary and Tom Poppendieck say in *Lean Software Development*, "The foundation of human respect is to provide an environment where capable workers actively participate in running and improving their work areas and are able to fully use their capabilities."[18] Now that certainly sounds more like Cockburn's "non-linear, first-order components," i.e., people! And if we continue to look deeper, we see more connections to the fundamental, people-oriented values of transparency and curiosity:

- To eliminate waste, incorporate integrity, and work with the whole system, we will need to be transparent about where we have erred in creating inefficiencies and about how the system works (and doesn't work!) as a whole.
- To amplify learning and deliver quickly, we will have to be curious about what options we have to achieve our goals and perhaps even experiment with multiple options simultaneously (a classic Lean strategy for quick learning).
- And to empower the team, we will need to be transparent with them about what we are trying to achieve and where they can contribute, and we will need to encourage their curiosity about all aspects of our customers and our business.

In fact, the successful Lean software teams we have seen rely heavily on vigorous, candid, continuous communication, with direct customer feedback, information radiators (like build status indicators and big, visible business-

relevant charts), and even Toyota-style Andon lights (personal red/green/amber indicators used by team members to announce widely that they are stuck or blocked). These tools and practices, grounded in transparency and curiosity, are used to spur productive conversations about continuous improvement.

The problem is that, as with Agile development, too many organizations adopt the practices without the underlying spirit. We have seen organizations with the Lean Six-Sigma Green Belt certifications pinned to cubicle walls but lacking the culture of collaboration and continuous improvement. In our experience, this is a symptom of executives who thought transformation was something that could be purchased, and then wonder why the value stream maps and pull systems lie discarded and unused.

DevOps: Operators Are People Too

Sysadmins Stand Up

By 2009, the time was ripe for the humanistic software movement to broaden, and none knew it better than Patrick Debois, a frustrated consultant and project manager in Belgium. He was annoyed because on project after project, he saw how the deep divide between the responsibilities of developers and system administrators was holding back progress. Though many development teams were undoing the negative legacy of the software factory, they continued the same old practices when interacting with the operators who deployed and ran their code—minimal communication, low trust, and avoidance of difficult conversations. And just as they had within development teams, those behaviors slowed progress and kept everyone from delivering the right working software. It was clear that Agile development needed to move downstream: "The operations team needs to be agile, and it needs to be integrated into the project."[19]

Patrick began trying to find others interested in what he called "Agile System Administration." There were few takers at first—at one conference, only two people showed up to a session on the new topic.[20] But there were others thinking the same way, notably John Allspaw and Paul Hammond, heads of operations and engineering respectively at Flickr. Their presentation "10+ Deploys Per Day: Dev and Ops Cooperation at Flickr," which called passionately for collaboration and trust between developers and operators, rapidly spread through Agile circles.[21] Patrick watched a livestream of their talk with increasing excitement and, soon thereafter, launched the first "DevOpsDays"

conference in Ghent. The DevOps movement, armed with powerful values and technical tools, had been born.

Respect, Trust, and No Blame

There is no DevOps czar, and there was never a DevOps equivalent of Snowbird, so there is no definitive list of DevOps principles. But the seminal Flickr presentation is one of the clearest statements of goals for DevOps that we are aware of, and it's useful as a litmus test to gauge how teams that claim to be DevOps focused truly behave. The principles from the second half of that presentation are below (lightly edited to fit nicely into a list):

DevOps Principles

1. Respect
    a. Don't stereotype
    b. Respect others' expertise, opinions, and responsibilities
    c. Don't just say "No"
    d. Don't hide things
2. Trust
    a. Trust that everyone is doing their best for the business
    b. Share runbooks and escalation plans
    c. Provide knobs and levers
3. Healthy attitude about failure
    a. Failure will happen
    b. Develop your ability to respond to failure
    c. Hold fire drills
4. Blame avoidance
    a. No finger-pointing
    b. Devs: remember that broken code wakes people up
    c. Ops: provide constructive feedback on painful areas[22]

Notice how explicit Allspaw and Hammond are about the crucial elements of trust, respect, and collaboration; this is clearly a movement for people, not machines.

That's not to say that there aren't key DevOps technical and team practices. These include:

- *Cross-functional team*: Developers and operators work together in a single team rather than in separate groups with a handoff of completed code from one to the other.
- *Cattle, not pets*: For DevOps-focused teams, servers are not special snowflakes with individual identities and custom configurations but undifferentiated, identical, fungible machines that can be replaced at a moment's notice.
- *Infrastructure as code*: Instead of manually configuring servers, system admins write programs (in special-purpose languages provided by tools like Puppet, Chef, or Kubernetes) to set up and test their machines.
- *Automated deployment*: Once the server is running, system admins and developers write more code together to make deployment to that server a one-click operation. That deployment can be triggered by continuous integration tools, further tightening the link to developers and their work.
- *Sharing metrics*: A team operating with a DevOps mind-set will have engineers and system admins alike looking at system uptime, error rates, user logins, and many more indicators of operational health, and addressing any indicated problems together.

No More BOFH

In the 1980s, Simon Travaglia invented the ultimate sysadmin caricature for the online publication *The Register*.[23] The Bastard Operator from Hell (or BOFH) despised developers and users alike and made it his purpose to increase their misery without limit. Travaglia went overboard for comic effect, of course, but he touched on the deep suspicion and mistrust present between developers and system administrators in traditional organizations, with impenetrable separation between the teams.

Thus, it's no surprise that the DevOps principles and practices above are so clear in their demand for collaboration above all else: share your runbooks, display your metrics, discuss your failures (transparency). And respect the other "side" by avoiding finger-pointing and by finding out what effects your actions have (curiosity). By focusing on shared concerns and bringing developers and system administrators into a conversation with each other rather than about each other, DevOps moved away from the assembly-line mentality, with

people-oriented values and attributes underpinning the DevOps movement—at least as it was originally conceived.

Bewilderingly, among some enterprises, there is a recent trend of anointing a special team that is separate from development and operations: the "DevOps" team. The whole point of DevOps is to create unity and collaboration among different specialties, not more silos. We even see job ads for "DevOps engineers," who apparently are a special breed different from normal engineers and system admins. What happened? We believe this is the result of a buzzword-bingo approach to management. Rather than cultivating "individuals and interactions," we have organizations hoping to avoid rethinking how to operate and instead get by with a reconfiguration of the software factory. And the surprising thing is that many have achieved that dubious goal.

Detour into the Feature Factory

The success of Agile development, Lean software, and DevOps in transforming the landscape of software is undeniable. Ideas that seemed extreme are considered normal now. Completing a feature in a day or an epic in a week is no longer astonishing, even in the largest corporations. As described by Eric Minick, program director at IBM,

> Looking at history, the most striking thing to me is that delivery has actually gotten better. Just look at release cadences. Teams were content with an annual release cycle. As agile hit corporations, they were then proud to get to quarterly. If you're at quarterly now, that's slower than average. Monthly is more normal now. Almost every big enterprise has some cloud-native team releasing daily or better. Today's cadence is an order of magnitude or two better than 15–20 years ago. Not bad.[24]

While a lot has changed in the schedules and the scope of our projects, there are times when we feel a sense of déjà vu. Large organizations are often stuck with a few Agile or Lean or DevOps processes uncomfortably coexisting with the old methods in a weird combination of practices—a "water-Scrum-fall" chimera.[25] And we've met many small organizations and startups where the practices from Lean, Agile, and DevOps are on display, yet the designers and developers and operations people describe themselves as working in a "feature factory," with all the same micromanagement and autonomy-destroying

practices as before. It's as if the giant software factory has been reassembled using smaller pieces with different names. There have been real benefits, but this isn't the "enthusiasm, close collaborative teamwork, superb customer connection, and conscious design thinking" that inspired Richard Sheridan to write a book with the radical title *Joy, Inc*.[26] What happened?

One part of the answer comes from Niels Pflaeging, who tackled the question in his article "Why We Cannot Learn a Damn Thing from Toyota, or Semco." Pflaeging ponders why so many well-known examples of practices that work at pioneering organizations generate so little change. His insight is that what holds back transformations is the lack of "that magic ingredient . . . our image of human nature, the way we think about people around us, and what drives them."[27]*

Organizations have embraced the process and tools created by the Agile transformation, yet the Taylorist factory mind-set remains. There's a lot less documentation to write, fewer specifications to read, and hardly any mandated signoffs, but these practices have merely been replaced with endless planning meetings and many pages of tickets in a project management tool—practices that still offer the Taylorist promise of giving management the insight and control they demand, because the role of managers is still to ensure that the right things get done.

What about the people doing the work? Remember those nonlinear, first-order components of software development? They remain first order and they remain nonlinear. Cynthia Kurtz's and David Snowden's Cynefin framework (see the sidebar on page 20) gives us language to discuss them.[28] The feature factory wants to put humans in the lower-right, "obvious" quadrant: "If we have everyone in the planning meeting and the standup and the retrospectives, then they will collaborate." This cargo-cult approach to collaboration doesn't work with humans, whose nature is squarely in the upper-left, "complex" quadrant. Deliberately cultivating the dynamics of an effective organization takes a lot more work and a lot more skill than just putting a group of people cheek by jowl and calling them a team.

If we understand that the individuals and teams in our organizations are complex systems in and of themselves, then what should we do? According to the Cynefin framework, the appropriate way to navigate in complex scenarios,

* See "Preparing: Theory X and Theory Y" in Chapter 7 for more on this key ingredient.

where there are no guaranteed right answers, is to "probe-sense-respond."[29] So, how do we probe-sense-respond with humans? That's called a conversation—and it's the way out of the factory.

Sidebar: The Cynefin Framework

Cynefin is a *sense-making* framework whose goal is to allow shared understandings to emerge and to improve decision-making (see Figure 1.1).

Figure 1.1: The Cynefin Framework

While there are a rich set of activities and applications in the Cynefin community, the first lesson of the framework is that appropriate behavior depends on which domain you find yourself in.

- When the situation is Obvious (causes and their effects are well understood), tools such as flowcharts are useful, because there are a limited number of possibilities, and the current state determines the next right step.
- When the situation is Complicated (causes and effects are known but only to specialists), with intricate needs such as troubleshooting unexpected behavior in a sophisticated machine, you'll need

to develop the relevant knowledge, either through homegrown analysis or by bringing in an expert.

- When the situation is Complex (causes and effects are only understood retrospectively), with unpredictable parts such as evolving team dynamics over the course of a project, past experience in other contexts (other teams) is not a sufficient guide for what to do next; instead, you need to experiment and develop multiple perspectives to understand the patterns that exist before deciding how to respond.
- And in a Chaotic situation (with no link between causes and effects), such as an outage in a distributed system, it is appropriate to act first in an attempt to bring the situation into a state where the normal relationships between cause and effect again apply.

As a body of theory, Cynefin is relevant to software and humans several times over. The software systems we build are, at least, Complicated and often have (unplanned) Complex emergent behaviors. The teams building the software system are complex systems themselves. And Cynefin further recognizes that humans are complex in and of themselves, not bound by simple rules. The framework gives us a good language for describing why the mass-manufacturing approach to software, with simple jobs performed by supposedly interchangeable workers, had such disastrous results.

Chapter 2

Improving Your Conversations

This chapter will prepare you for the key relationship-building conversations you need in order to escape the feature factory: the Trust Conversation, the Fear Conversation, the Why Conversation, the Commitment Conversation, and the Accountability Conversation. Before you tackle the specific conversations, however, it is important to learn how to analyze and build your conversational skills generally. You will learn about why conversations are humanity's unique superpower and how to harness this power effectively through study and practice.

This chapter will also describe the core challenge to improving our conversations, which is that our behavior doesn't match our beliefs, and we are unaware of the gap. To combat this problem, we will provide a process to help you become aware: the Four Rs. We will show you how to Record your conversations, how to Reflect on them to find problems, how to Revise them to produce better alternatives, and how to Role Play to gain fluency. Finally, we will provide some sample conversations that will allow you to see the process in action.

Once you have the foundation of the Four Rs, you'll be ready for Part II of the book, where you will learn how to have each of the specific conversations.

Conversations: Humanity's Secret Weapon

Our Special Power

In his book *Sapiens: A Brief History of Humankind*, Yuval Noah Harari explores what has allowed humans to become the dominant species on the planet. His answer is that we have a special kind of communication, unique among animals.[1]

Many animals can communicate the idea "Run away from the lion!" through barks, chirps, or movement. Building on top of that, the development of human and animal communication seems to have been driven by the need to share information about others of the same species—the need to gossip. Gossiping allowed us, as social animals, to understand each other and have established reputations; and this, in turn, allowed us to collaborate in larger groups and to develop more sophisticated collaboration. In fact, understanding other humans, developing a "theory of mind," is so important that philosopher Daniel Dennett, in *From Bacteria to Bach and Back: The Evolution of Minds*, makes the case that our own consciousness arose as a byproduct of understanding the minds of others.[2]

Though our ability to gossip surpasses that of other species, Harari says that what is really unique about human language is our ability to discuss nonexistent things.[3] With this special power, we are able to create and believe shared fictions. These fictions allow us to collaborate at tremendous scales and across groups of people who have never met. In this way, a community's belief in a crocodile-headed god can create flood control works on the Nile, as described by Harari in another of his books, *Homo Deus: A Brief History of Tomorrow*.[4] And a shared belief in continuous improvement can allow us to create a learning environment and a performance-oriented culture rather than a power-oriented or rule-oriented culture, as described in *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* by Nicole Forsgren, Jez Humble, and Gene Kim.[5]

Why Our Power Is Flawed

Conversation makes collaboration possible but not inevitable. We don't live in a world of universal acceptance, peace, and understanding. Earnest and well-intentioned people can disagree, and even come to view another person as an enemy, as "the other." Along with our amazing powers of conversation, we also

come equipped with pre-existing, built-in flaws—our so-called cognitive biases (see Table 2.1 for a sampling of our favorite biases, and Daniel Kahneman, *Thinking, Fast and Slow*, for many more). These are biases that seem to be built into the functioning of our brains. And these cognitive biases inhibit the sort of collaboration our language makes possible.

Name	Distortion
Egocentric bias	Give self undue credit for positive outcomes
False consensus effect	Believe that personal views are commonly held
Gambler's fallacy	Believe that a random event is influenced by previous outcomes
Illusion of control	Overestimate control over external events
Loss aversion	Value keeping a possession over gaining something of greater value
Naïve realism	Believe personal view of reality is accurate and without bias
Negativity bias	Recall unpleasant events more readily than positive ones
Normalcy bias	Refuse to plan for a novel catastrophe
Outcome bias	Judge decisions by their results instead of by the quality of the decision-making process

Table 2.1: A Sampling of Cognitive Biases

Our cognitive biases pose a threat to any adoption of Agile, Lean, or DevOps methods because they can seriously damage collaboration, relationships, and team productivity.

In the previous sections, we described how transparency and curiosity are woven into the fabric of people-centric practices, but these are undermined by a host of cognitive biases. An example is the false-consensus effect, where we believe our own views to be commonly held. This bias makes us less likely to either share our reasoning or to ask about the reasoning of others. What's the point when we believe we already agree? Naïve realism, the belief that we see reality as it is, without bias, is yet more corrosive to team dynamics in that we see any disagreement as a sign that the other party is uninformed, irrational, lazy, biased, or perhaps all of those! Under the influence of these and other cognitive biases, Agile, Lean, and DevOps practices can fail to deliver the promised benefits.

Learning from Conversations

Conversations as an Investigative Tool

Social scientist Chris Argyris studied organizational behavior, particularly in businesses, in a long and illustrious academic career at the business schools of Yale and Harvard. His areas of research included individual and organizational learning, and effective interventions that "promote learning at the level of norms and values."[6] The humble conversation was the central tool Argyris used for investigating group effectiveness and for improving organizational performance. What Argyris found was that conversations, together with the unexpressed thoughts of the participants in those conversations, revealed everything he needed to know about the "theories of action" of the people and organizations he studied.

Argyris and collaborator Donald Schön use the term "theory of action" to describe the logic—the "master program"—behind our actions.[7] According to Argyris and Schön, we all have outcomes we want to achieve, and we use our theory of action to choose which steps to take. If my theory of action has a focus on learning, then I will take actions that generate information, like sharing everything I know that is relevant to the situation and asking others about what they know. If my theory of action is centered on getting my own way, then I will only share information that supports my position, and I won't ask questions to which I don't know the answer.

In general we don't explicitly think about our theories of action; however, as with the two examples we just provided, we can understand them after the fact by examining our choice of action. One of the findings of Argyris and Schön is that there is often a gap between what we say we would do in a situation (espoused theory) and what we actually do (theory-in-use).[8]

Defensive versus Productive Reasoning: What We Do and What We Say We Do

Before reading on, consider this question: If you had an important choice to make as a group, how would you recommend the group go about making the decision?

When we ask this question of our audiences, we get remarkably consistent answers. The typical response is something like, "I'd have everyone share all the information they have, explain their ideas and reasoning, and then see if we can agree on the best way to proceed."

If your answer sounded like this, congratulations! You have espoused what Argyris and his colleagues call the Model II Theory of Action,[9] or "productive reasoning."[10] You claim to value transparency, sharing your reasoning and information. You also claim to value curiosity, hearing everyone's thoughts to learn their reasoning and what information they have that you don't. Finally, you claim to value collaboration and jointly designing how to proceed. While you might have used different words, these are commonly understood and accepted practices to increase learning and make better decisions. In fact, you likely *do* behave this way in nonthreatening situations, where nothing important is at stake. Unfortunately, if you are like the more than 10,000 people that Argyris studied across all ages and cultures[11] (and those we've met!), *your behavior won't match your words* when the topic is something important—like introducing a company strategy or leading a cultural transformation.

Argyris and colleagues found that although almost everyone claims to adopt the approaches and behaviors of productive reasoning, things change when the situation is potentially threatening or embarrassing. In those cases, what people *actually* do closely matches a very different theory-in-use that Argyris terms the Model I Theory of Action, or "defensive reasoning."[12]

We contrast these two theories of action in Table 2.2. When using a defensive reasoning mind-set, people act to remove the threat or potential embarrassment. To do so, they tend to act unilaterally and without sharing

their reasoning, they think in terms of winning and losing, they avoid expressing negative feelings, and they attempt to be seen as acting rationally.

	Model I	Model II
Governing Values	Define and achieve the goal Win; do not lose Suppress negative feelings Be rational	Valid information Free and informed choice Internal commitment
Strategies	Act unilaterally Own the task Protect self Unilaterally protect others	Share control Design tasks jointly Test theories publicly
Useful When…	Data is easily observed Situation is well understood	Data is conflicting or hidden Situation is complex

Based on Argyris, Putnam, and McLain Smith[13]

Table 2.2: Model I and Model II Theories of Action Compared

This gap between our espoused theory and our theory-in-use gets at the heart of a paradox of team productivity. In theory, we value diverse teams because we understand that diversity can be a strength. A diversity of experiences, a diversity of knowledge, and even a diversity of modes of thought—in theory, these all make a team stronger, because every new element gives the team more information and more ideas, and therefore, more options to make better choices.

What we should be seeking from our diversity is *productive conflict*, through which we harness our differences to create new ideas and better options. In practice, we tend to see differences of opinion as threatening and potentially embarrassing, so we react defensively. Our defensive reasoning leads us to suppress the diversity we claimed to value and to avoid the productive exchange of ideas that we claimed to seek!

What does this defensive reasoning look like in practice? We will illustrate many flavors of defensive reasoning with examples throughout the book, but to paraphrase Tolstoy's *Anna Karenina*, each productive conversation is alike, and each defensive conversation is defensive in its own way. That said, there are common elements; and defensive reasoning in conversations will tend to feature hidden motives, undiscussable issues, and reacting to, rather than relating to, what is said—all characteristics that inhibit learning and corrode relationships.

Transforming Conversations

So, why do people choose these counterproductive, defensive behaviors rather than the behaviors we all agree would produce better results? The answer is that we don't consciously choose. In everyday activities, this gap between the theories we espouse and the theories we use is invisible to us. We effortlessly produce the defensive behavior through years of practice—so effortlessly, in fact, that we aren't aware of what we are doing, no matter how counterproductive it is for us or how much it contradicts our espoused theory of productive reasoning. Even worse, we are so unaware of our defensive reasoning that we will deny we are acting defensively if someone else tries to bring it to our attention.

The good news is that Argyris found that reflecting on conversations allowed participants to become aware of and then change their behavior.[14] Through regular effort and practice, you can learn the behaviors of transparency and curiosity that will promote joint design and learning; the sharing of knowledge across organizational boundaries; and the sharing of and resolution of difficult, previously taboo issues. The bad news is that this takes substantial effort, and worse, this effort involves difficult emotional work.

The difficulty comes because it requires recognizing that your behaviors are contributing to the problem. Are you willing to consider that you might be contributing to unproductive meetings and defensive relationships? This is not a price everyone is willing to pay. Finally, even if you are willing to be humble and put in the effort, developing these new skills takes time. Argyris and colleagues describe overcoming our routine behaviors as taking about as much practice "as to play a not-so-decent game of tennis."[15] If this seems daunting, it may help to remember that you get the opportunity to practice every day as

you work to solve real problems in your organization. We can give you the skills to practice if you have the drive to improve.

Later in this chapter, we will show you how to practice productive reasoning by learning from the conversations you are having today. In addition to providing the core technique for learning from conversations (the Four Rs), we will provide examples for you to practice your analysis. Through the remainder of the book, you'll be using this same Four Rs approach again and again to learn the specifics of the Trust, Fear, Why, Commitment, and Accountability Conversations. These five conversations address the common pitfalls that prevent us from using the productive reasoning we espouse. These pitfalls are:

1. We won't be transparent and curious when we lack **Trust**.
2. We will, consciously or not, act defensively when we have unspoken **Fear**.
3. We will be unable to generate productive conflict when we lack a shared **Why**.
4. We will avoid definite **Commitments** as long as the situation feels threatening or embarrassing.
5. We will fail to learn from our experiences if we are unwilling to be **Accountable**.

It is only after we have overcome each of these challenges that we can really have the productive learning conversations required for a high-performing organization.

Sidebar: Types of Conversations

From cover to cover, this book is about conversations, so it is worth taking a moment to explain the range of conversation types where this material is applicable.

The first image that comes to mind when we say "conversation" is probably a face-to-face encounter with two or more people in the same room. However, most of us have a number of other types of communication channels we use regularly. Email is ubiquitous. Chat systems such as Slack, Microsoft Teams, and IRC (internet relay chat) are being adopted right and left. Dis-

tributed meetings with video are increasingly common, a big step up from the voice-only conference call.

We believe the material in this book is useful across all these conversation modes, though it is worth considering the trade-offs inherent in the different options. Figure 2.1 provides a useful visual reference for these trade-offs based on a model by Alistair Cockburn.[16]

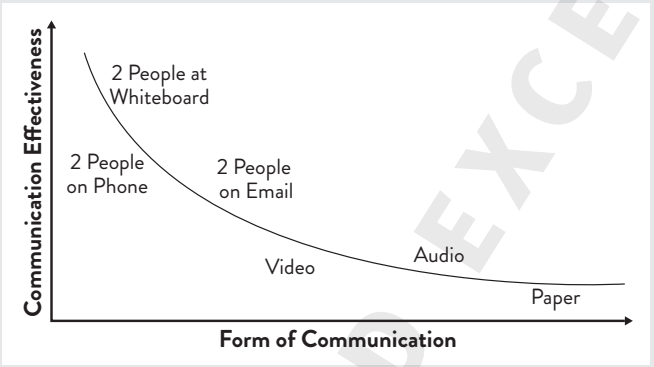

Figure 2.1: Effectiveness of Different Modes of Communication

As Cockburn says, "the most effective communication is person-to-person, face-to-face, as with two people at the whiteboard."[17] This scenario is the most effective, because, among other attributes, it offers the maximum possible nonverbal information to the two participants, and the fastest response rate. However, these same attributes can make it more difficult when you are learning to have difficult conversations and one or both parties are feeling strong emotions. A red face is additional information, but it can also be intimidating and distracting.

As an opportunity for learning, asynchronous channels can offer some benefits. One is that you'll likely have a better record of what each party actually said, which can be a big help in performing a later conversational analysis.* Even better, an asynchronous channel allows us to make multiple drafts before responding. As an example, we've applied the Four Rs to draft emails, allowing

* Recording video of your face-to-face conversations at the whiteboard for later review is a great practice that we'd recommend, though it's underused in most teams we've worked with.

us to use a technique we are learning and to incorporate the insights into the email we finally sent.

Ultimately the skill you are after is the ability to apply these techniques face to face and in real time; making good use of the learning opportunities of asynchronous communication can help you gain that ability.

The Four Rs

Experiences give us the *opportunity* to learn, but most people don't take the time to actually learn from them. We apply the Four Rs—Record, Reflect, Revise, and Role Play—as our preferred way of learning from conversations. (As you can see in Figure 2.2, there are two additional Rs that can sneak in along the way: Repeat and Role Reversal.)

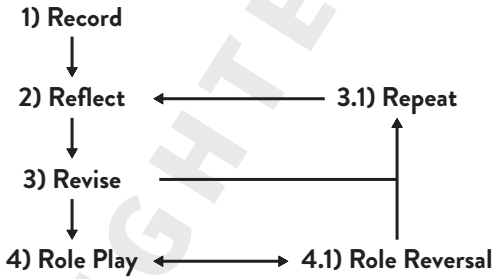

Figure 2.2: The Four Rs

To start using the Four Rs, you will need to Record a conversation in writing. In the next section, we describe our preferred method for this, the Two-Column Conversational Analysis. It may be tempting to avoid pen and paper and just think about the conversation "in your mind's eye," or talk about it with a friend. *Do not do this!* The act of writing down the words on paper is an inherent part of the process because it forces your brain to think about the conversation at arm's length, as if it were happening to someone

else. This distance is vital to gaining insights through reflection and revision, as we'll see later.*

After you have Recorded the conversation, it's time to Reflect on it, paying attention to the tool or technique you are trying to use at the time. For each of the Five Conversations, we will suggest particular tools. With time and practice, you will move between them as the conversation requires, but to start with, we recommend you use one tool or technique at a time. We will give you guidelines on how to score your conversation using the tool, and that reflection will lead you to possible improvements.

Having scored the conversation, Revise your conversation to try and produce a better result. How will you know if you've improved? Repeat: take your revised dialogue and Reflect again. Did you improve your score from the first time around? You may be surprised to find that your revised version in fact shows no better scores than your original! Don't be discouraged—this is very common, especially as you are learning a new skill. It might take you half a dozen or more attempts to produce a revision that checks all the boxes of the technique.

Having created the dialogue for an alternative conversation, there's still an important step remaining: Role Play. Find a friend who is willing to help, and try saying your dialogue aloud, with your friend taking the part of your conversation partner. How does it feel to say the words out loud? Often what seems okay in writing feels unnatural coming out of your mouth. Perhaps the words need to change, or perhaps you just need practice talking in a different way.

Another good check on your progress is the other hidden R: Role Reversal. Trade places in your dialogue and have your friend say your words. How does it feel to be in the other person's shoes and to hear your revised language? Frequently, hearing your own words will give you clues as to how you can further tune the dialogue to feel more natural while keeping in place the skills you are trying to practice.

Following all of the Four Rs for a single conversation will offer the most learning from that single experience. Following them for a series of conversations will dramatically increase the volume and pace of your learning overall, and should quickly give you and your team substantial practical gains.

---

* An extreme version of this distancing was practiced by our friend and teacher Benjamin Mitchell, who used an audio recorder to capture his conversations. He tells us that when he first listened to himself on tape, when he noticed mistakes he was making, he would shout at the recorder, "Benjamin! Don't do that!"