



Technical Coaching

for IT Organizational Transformation

Dave Nicolette

Technical Coaching for IT Organizational Transformation

David Nicolette

This book is for sale at <http://leanpub.com/technical-coaching>

This version was published on 2019-05-24



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2019 David Nicolette

This book is dedicated to my wife Malu, without whose encouragement and support I would most likely do nothing at all.

Contents

About the Author	i
Preface	ii
Notes On the Second Edition	ii
0 Introduction	1
0.1 What's the Problem?	1
0.2 Who's the Audience?	2
0.3 What's the Scope?	2
0.4 Anything new here?	3
0.5 A Note on Agile	4
Part 1: Foundations	6
1 Is Transformation Possible?	7
1.1 A Longstanding Problem	7
1.2 If Organizations Changed Before, They Can Change Again	8
1.3 If We Can Find the Dots, We Can Connect Them	10
1.4 The Missing Element	11
2 Anti-Patterns of Organizational Transformation	13
2.1 Incomplete Solutions	15
2.2 Misalignment of Goals	16
2.3 Lack of Direction	18

CONTENTS

2.4 Separation of Process-related and Technical Im- provements	19
2.5 Misunderstanding the Nature of the Work	21
2.6 Conflating business agility with agile software development	21
2.7 Reintroduction of the Old	22
2.8 Inappropriate Organizational Structures	24
2.9 Big-bang Change Rather Than Incremental Change	25
2.10 Rigid Interpretation of Frameworks	26
2.11 Milestone-driven vs. Directional Approach	27
2.12 The Cult of Scrum	28
2.13 The Soft Stuff Is the Hard Stuff	30
3 Definitions	31
3.1 Capabilities	31
3.2 Leverage	33
Part 2: Strategy	35
4 Connecting Management and Technical Consulting . .	36
4.1 Half a Solution Isn't Better Than None	36
4.2 A Proposed Solution	38
4.3 The Fulcrum Is the Natural Dividing Line	39
4.4 Connecting the Parts	39
5 Begin With the End in Mind	45
6 Target States: Where Do We Need To Be?	51
7 Leverage: Where's the Fulcrum?	57
8 Getting Started	63
8.1 Impact: Structure, Process, Practices	63
8.2 Full-Stack Slices	64
8.3 Up-Front Analysis	65

CONTENTS

8.4 Foundational Activities	67
8.5 Starting Iterative Improvement	70
9 The Cycle of Change	72
10 Monitoring Progress	76
10.1. Anti-Patterns In Monitoring Progress	76
10.2 When Should We Check Progress?	80
10.3 How Should We Check Progress?	81
10.4 Metrics and Managed Change	82
10.5 Cultivating Continuous Improvement	85
11 Metrics	87
11.1 Process-Agnostic Metrics	87
11.2 Directional Metrics	90
11.3 Measuring Capabilities	91
11.6 Measuring Stability of Production Operations . . .	93
11.7 Measuring Software Quality	94
11.8 Measuring Code Health	95
11.9 What Not To Measure	101
12 Structures and Responsibilities	102
12.1 Structures Specific to the Transformation	102
12.2 Consultancy Responsibilities	105
12.3 Client Responsibilities	108
12.4 Joint Responsibilities	112
12.5 Consultant and Coach Responsibilities	114
12.6 Special Consideration for Technical Coaches	114
Part 3 Humanity	118
13 A Chair Is a Resource	119
14 Holding Precious What It Is To Be Human	121
15 Safety	123

CONTENTS

15.1 The Business Value of Safety	123
15.2 Putting People First	124
15.3 Importance of Safety in Times of Change	124
16 Responsibility Over Accountability	126
16.1 Accountability	126
16.2 Responsibility	127
17 Stress	130
17.1 Allow for Stress and Its Effects	131
17.2 Your Helpers Are Human, Too	132
17.3 Hold Course Despite Stress	133
18 Introversion and Collaboration	135
18.1 What is Introversion?	135
18.2 Can't Introverts Just Get Over It?	137
18.3 Making It Work	139
18.4 Introversion and Coaching	139
19 Cognitive Biases	140
19.1 Categorizing Cognitive Biases	141
19.2 Confirmation Bias	141
19.3 Gambler's Fallacy	142
19.4 Sunk Cost Fallacy	143
19.5 Negativity Bias	144
19.6 The Illusion of Explanatory Depth	145
20 Ego Development	147
21 View of Authority	150
22 Profiling	152
22.1 Pigeon-holing	154
22.2 Defamation	155
22.3 Excuses	155
22.4 Monoculture	156
22.5 The Slippery Slope to a Toxic Culture	157

CONTENTS

22.6 Be a Grown-Up	157
22.7 People Can Grow (Resources Can't)	158
23 Organizational Culture	159
23.1 What Is Organizational Culture?	160
23.2 How Is Organizational Culture Used?	161
23.3 A Systems View of Organizational Culture	163
23.4 Stop Worrying About Culture	164
24 Client-Consultant Relations	166
24.1 Factors Resulting in Sensitivity	166
24.2 Earning and Maintaining Trust	171
24.3 Who's the Boss?	173
24.4 Ensure Common Understanding	176
Part 4: Change	178
25 Intentional Change	180
25.1 Current and Target Operating States	180
25.2 Missing Pieces	181
25.3 "How to Change" Doesn't Mean "How to Implement"	181
25.4 Stepwise Improvement vs. Predefined End State	182
26 Initial Changes	185
26.1 Establish Collaborative Workspaces	186
26.2 Form Product-Aligned Teams	186
26.3 Take Baseline Measurements	187
26.4 Choose Initial Experiments	187
27 Workspaces for Collaboration	189
27.1 The Rise of Office Work	190
27.2 The Action Office	191
27.3 The Birth of the Cubicle Farm	193
27.4 The Open-Plan Office	194
27.5 Collaborative Team Work Spaces	196

CONTENTS

27.6 Considerations for Remote Workers	198
27.7 5S and Software Development	202
27.8 Gaining Cooperation from Client Leadership	205
28 Time Management	208
28.1 Outlook-Driven Development	208
28.2 Causes of Time Management Issues	209
28.3 Manager Time vs Maker Time	210
28.4 Managing the Calendar	211
28.5 Protecting Maker Time	215
29 Incrementally Collapsing Functional Silos	216
29.1 Initial Organizational Structure	216
29.2 Starting Point	218
29.3 Organizational Constraints on Silo-Busting	221
29.4 Temporary Scaffolding	223
29.5 First Steps in Silo-Busting	225
29.5 Special Considerations When Decentralizing Responsibility	242
29.6 Team Size	244
29.7 Security	250
29.8 Summary	251
30 Incrementally Improving Estimation	253
30.1 Who Needs to Estimate?	253
30.2 Factors Affecting Estimation	254
30.3 Incremental Improvement	256
31 Incrementally Improving Code Review	260
31.1 The Value of Code Review	260
31.2 From No Code Review to Formal Code Review	260
31.3 Use Checklists	261
31.4 Limit the Time of Formal Code Reviews	261
31.5 Refactor to Increase Review Effectiveness	262
31.6 From Formal Code Review to Pair/Mob Programming	262
31.7 Use Static Code Analysis	263

CONTENTS

32 Incrementally Improving Branching Strategy	264
32.1 Types of Version Control Systems	265
32.2 One or Multiple Version Control Systems	267
32.3 One or Multiple Source Repositories	270
32.4 Branching Strategies	271
32.5 Challenges In Scaling	280
32.6 Typical Starting Points	282
32.7 Step By Step Improvement	282
32.8 Pitfalls and Anti-Patterns	284
33 Improving Infrastructure Support & Operations	286
33.2 Infrastructure Management Options	287
33.2.1 Stress	288
33.3 Leading Practices in Operations	295
33.4 Busting the Final Silos Within Each Technology Stack	306
33.5 Special Considerations for Mainframe Teams	307
Part 5: Coaching	313
34 Coaching Skills	314
34.1 Coaching Competencies	314
34.2 Self-Awareness and Empathy	319
34.3 Course of Least Resistance	321
34.4 Make Things Visible	322
34.5 Manipulation	327
34.6 Acting	333
34.7 Removing Organizational Constraints	334
34.8 Let the Team Stumble	335
34.9 Be the Rock	336
34.10 Conflict Resolution	337
34.11 Principles-Based Adaptation	341
34.12 Having Multiple Ways to Explain Things	346
34.13 Awareness of Context	348
34.14 Knowing When To Quit	349

CONTENTS

35 Shortage of Technical Coaches	351
35.1 Availability of Technical Coaches	352
35.2 A Note on Agile Coaches	353
35.3 Tailored Approaches	353
36 Technical Coaching Approach	356
36.1 Problems With the <i>Status Quo</i>	356
36.2 Addressing the Problems	360
36.3 Team-Level Technical Coaching Model	365
36.4 Scaling the Coaching Model	373
36.5 Developing Internal Technical Coaches	374
36.6 Barriers to Adoption	375
37 Communication Models	377
37.1 Active Listening	378
37.2 Appreciative Inquiry	378
37.3 Clean Language	379
37.4 Crucial Conversations	381
37.5 Emotional Intelligence	381
37.6 Getting To Yes	382
37.7 Mindful Kindness	383
37.8 Powerful Questions	383
37.9 Radical Candor	385
38 Collaboration	387
38.1 Improving Collaboration Improves Flow	387
38.2 Team Cohesiveness	396
39 Internalities	403
39.1 Beginner's Mind	403
39.2 Eye of the Hurricane	405
39.3 Non-Attachment	405
39.4 Nonviolent Communication	407
39.5 The Four Agreements	409

CONTENTS

Part 6 Summary	411
Appendix A IBM Mainframe Considerations	413
A.1 Mainframe UNIX Support	413
A.2 Mainframe ASCII Support	414
A.3 Mainframe as Cloud Host Environment	414
A.4 Legacy Applications and IBM Z Cloud Features . .	415
A.5 Test Automation and Test-Driven Development . .	415
Appendix B Glossary	417
Appendix C References	423
Appendix D Picture Credits	437
Cover Art	437
Figures	437
Index	440

About the Author

Dave Nicolette started work as an application programmer in 1977. Since that time, he has served in many roles in the IT space. In the latest segment of his career he has focused on coaching organizations, teams, and individuals in effective software delivery and support processes and practices.

Preface

After some 42 years in the information technology industry, with the last 35 or so in consulting or contracting, and the last 18 in the Agile and Lean organizational transformation space, I have a few observations to share concerning typical approaches to organizational change and technical coaching.

Some of the observations pertain to consultants and coaches - how they interact with clients and how they effect change. Others pertain to the clients of such consultants - how they engage external helpers, how they track progress, how they manage friction and conflict during the transformation program.

I've had the opportunity to serve in various roles and to engage with clients at levels ranging from the CEO (in small companies) all the way up to direct coaching of technical teams with hands on the keyboard (in companies of all sizes). I've seen a few anti-patterns (recurring failure modes), and I have a few suggestions.

Notes On the Second Edition

The First Edition had a limited amount of information about step-by-step improvements, including incrementally collapsing functional silos, incrementally improving estimation and forecasting practices, and incrementally improving code reviews. There are many more aspects of technical practice and operations to be addressed in an organizational transformation program.

For the Second Edition, I expanded Part 4 with chapters on improving version control and branching strategies (Chapter 32) and operations and production support (Chapter 33). I hope this will make the book slightly more useful.

0 | Introduction

Different isn't always better, but better is always different.

— Bob Marshall

We've seen attempts to improve organizational performance ever since the advent of computers in business. Yet, in the overwhelming majority of cases, nothing much seems to come of these initiatives despite their expense, scope, and length. Some companies have attempted several transformation initiatives over a period of years, only to end up more-or-less where they started.

0.1 | What's the Problem?

As the pace of change, the ubiquity of technology, and the complexity of solutions continue to increase, so too do bugs, security vulnerabilities, usability issues, and delays in delivering value to customers.

Organizations have tried various strategies to improve things, and many have attempted multiple "transformations" or "adoptions" of all kinds of frameworks and methods over a span of years. Despite their best efforts, the problem persists and grows.

With all the help available and the many attempts to improve the situation, why is this still a problem? It seems to me we (as an industry) repeat the same mistakes again and again. Maybe it's time to reconsider our approach to improving organizational performance.

0.2 | Who's the Audience?

This book has one primary audience and three secondary ones.

Primarily, this is for *technical coaches*. A technical coach is a person who provides direct, team- and individual-level coaching, mentoring, and training for the people who perform the hands-on work of creating, maintaining, operating, and supporting software products and services.

Much of the material is also relevant to *management consultants*. A management consultant is a person who provides advice and guidance to business leaders and managers to help them improve the effectiveness of their products and services and their internal operations.

In addition, *technical consultants* will find some of the material relevant to their work. A technical consultant is a person who provides advice and guidance to IT management regarding business decisions and strategies around technical infrastructure, enterprise architecture, product selection, and other matters of similar scope.

Finally, some of the material is relevant to the leadership of organizations that engage consultants and coaches to guide transformation programs. It's aimed at helping them get maximum value from consultants and coaches, and to avoid the common anti-patterns that plague transformation programs.

0.3 | What's the Scope?

One of the recurring failure modes in organizational transformation programs is that the technical side of things is undervalued, underfunded, and under-supported. I attempt to shine a light on several anti-patterns that seem to be at the root of the chronic and recurring failures of transformation programs, and to suggest some practical corrective measures.

The general area where transformation programs fall down covers the explicitly-technical work as well as the “connecting tissue” between IT and business operations. Those are the areas of my expertise and the focus of this book. I don’t attempt to define a strategy for comprehensive, enterprise-wide, business transformation.

However, those transformation efforts cannot succeed without the pieces I discuss in this book. Those pieces are almost always missing or misunderstood.

0.4 | Anything new here?

There’s very little in the book that’s made up out of whole cloth. I draw upon useful ideas and practices that others have defined, and that I’ve seen applied effectively in the field. To the extent anything here is original, it’s the way in which some of the ideas are combined and applied, to avoid the common anti-patterns observed in previous transformations.

The distinguishing characteristics of this approach, as compared with others in the market, are:

- Targeting clearly-defined market-facing competitive business capabilities rather than implementing a standard model or framework
- Measuring progress *directionally* vs. assessing progress at milestones or checkpoints;
- Breaking down dependencies in a particular way to identify the “fulcrum” for getting business leverage from the transformation;
- Measuring outcomes that matter to the client rather than tracking adherence to a set of rules or adoption of specific practices;

- Recognizing that *coaching* skills are distinct from *technical* or *business* skills;
- Recognizing that *making change happen* is a different process from *delivering products and services*;
- Cultivating *internalities* that help consultants and coaches achieve a calm center and focused minds even when working under conditions of stress;
- Focusing strongly on the “respect for humanity” aspect of organizations, to an extent very few others do; and
- Accepting that *stress* will be a reality and that it will lead to friction and conflict, and explicitly managing it to turn conflicts into learning opportunities, rather than pretending the problem doesn’t exist or comes down to personal issues.

These elements of the strategy are intended to address specific anti-patterns in organizational transformation programs that I’ve seen in the field.

0.5 | A Note on Agile

My strategy for IT organizational transformation aims to build the infrastructure, skills, processes, and practices necessary to support the market-facing competitive capabilities the company needs in order to achieve business objectives. It is not specifically an “agile transformation” or “agile adoption” approach.

Why, then, do I mention “agile” as often as I do? As of the time of writing, the market in transformation services is heavily dominated by “agile” this and “agile” that. It’s often hard to describe the differences between one thing and another without mentioning the word. In addition, some of the models and methods that I appreciate have the word “agile” in their names, probably as a way to take advantage of the popularity of the buzzword.

So, you will see the word “agile” here and there in the book, but it doesn’t mean I’m stuck on it. Nor does this disclaimer mean I’m against it. I’m in favor of whatever helps us achieve our goals.

Part 1: Foundations

Wings are like dreams. Before each flight, a bird takes a small jump, a leap of faith, believing that its wings will work. That jump can only be made with rock solid feet.

— J.R. Rim

This approach to organizational change, consulting, and in particular to technical coaching are different enough from conventional thinking that it's advisable to begin with some explanation of concepts, for context.

After so many years of poor outcomes from attempted transformation programs, perhaps the first question to answer is, simply, is organizational transformation possible at all? If not, then we may have been trying to "do the wrong thing righter" all these years, and we need to figure out what the "right thing" is. Otherwise, we need to determine how we've been causing all these poor results, and change whatever we need to change in our own work to improve those results. This question is the topic of *Is Transformation Possible?*

In *Anti-Patterns*, I enumerate several recurring failure modes I've observed in organizational transformation programs over the years, and suggest practical corrective actions to avoid repeating the same mistakes.

1 | Is Transformation Possible?

Everything is what it is because it got that way.
— D'arcy Thompson, 1917 (*Consciousness Explained*)

If we accept the adage that our outcomes result from our actions, then if organizations aren't achieving the outcomes they want, it's because of the actions they take.

The long history of attempted organizational transformations and the sad trail of debris in their wake deny us the luxury to avoid the fundamental question: Is organizational transformation possible?

1.1 | A Longstanding Problem

This is not a new problem. Way back in 1993, a millennium ago, consultants with McKinsey & Company were grappling with the same problem (Dichter *et al*):

Many senior managers today are aggressively trying to transform their companies, seeking radically to improve performance by changing behavior and capabilities throughout the organization. Unfortunately, most leadership groups lack a proven way of thinking about the challenge.

More than 15 years later, we're still in the same spot.

1.2 | If Organizations Changed Before, They Can Change Again

If there's any sense in the quote that opens this chapter, then things "got" the way they are today. The world didn't just wake up to the *status quo* one random morning. If things can "get" one way then they can "get" another way, too.

The question is whether it's practical to "get" things the way we want them by changing our organization or if we have to do it by creating a new organization built from the ground up the way we want it to be. I've seen both approaches achieve limited success. But something is still missing, as evidenced by the fact the problem persists.

McKinsey's approach (at least, *circa* 1993) was to help senior management coordinate multiple improvement initiatives to converge on overarching business goals. It *sounds* reasonable.

I have doubts about the practicality of coordinating multiple management initiatives. By definition, each initiative will have its own goals and priorities. Coordinating them means finding common ground and gaining consensus from the leaders of each initiative.

Each of them is an ambitious senior leader with a personal agenda and a career strategy that may well be based on undermining other senior leaders who are trying to climb the same corporate ladder. And you thought herding *cats* was hard.

I mentioned *doubts*, plural. The second one is that this sort of approach requires a lot of time - years, in fact. The transformation program will take longer to complete than the tenure of the executives who initiated it.

If the program has to be "driven," then it will stop as soon as the driver gets out of the vehicle. Typically, when there's leadership turnover, the new people want to demonstrate that there were good

reasons to bring them aboard. They usually do that by shutting down all the initiatives their predecessors started.

To prevent that failure mode, we need to initiate a program that “runs itself” even after the original leaders have left the building. That means cultivating an organization characterized by continuous improvement, rather than “installing” a former executive’s vision as of several years in the past.

Besides that, there are those who doubt the effectiveness of the very *idea* of transforming an organization by driving change from above. The respected software engineering coach and Agile Manifesto author Ron Jeffries writes (Jeffries):

...any message starting from leadership will inevitably be corrupted by the time it reaches the worker level. This is a fundamental and inevitable result of information theory. We all do our best to avoid this concern, with shorter lines of communication, careful crafting of our messages, and so on, but even with great care, it happens.

The cited McKinsey article mentions *top-down direction setting* and *bottom-up performance improvement* as two (of three) axes of change. I think they were onto something. Yet, in the field, transformation programs tend not to connect those dots.

Apparently, no one has found a way to connect them reliably and repeatably. If they had, then we wouldn’t be experiencing these problems anymore. Either that, or they were never the right dots to begin with.

1.3 | If We Can Find the Dots, We Can Connect Them

It's necessary, but not sufficient, to envision an end state operational model of some kind, however imperfect, like those described in the various popular Agile scaling frameworks and other organizational models on the market.

It's necessary, but not sufficient, to understand key principles of organizational transformation, like those elaborated in Kotter's 8 Steps or the ADKAR model.

It's necessary, but not sufficient, to have an intentional method of introducing change and measuring its effects, like the guidance offered in Lean-based improvement methods.

What's wrong with the available change principles, end state models, and improvement processes? Well, nothing is particularly *wrong* with them, as far as they go.

We can find a good list of principles. We can find definitions of end-state operating models. We can find advice about continuous improvement, Lean principles, and metrics.

Three sets of dots.

No connections.

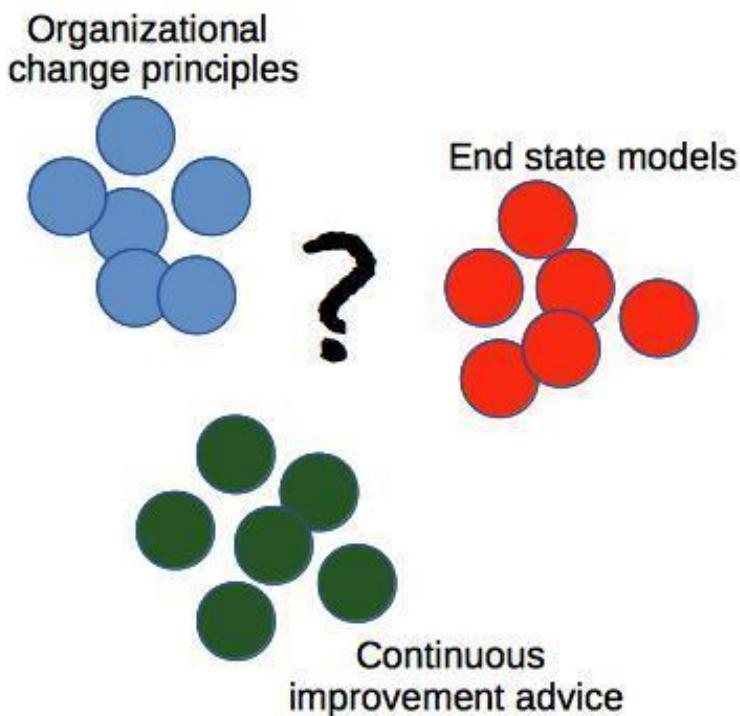


Figure 1.1: Connect the dots

If we *could* find a way to connect the dots, then organizational transformation ought to be possible in principle.

1.4 | The Missing Element

What's missing here? There's no single overarching goal. There's no single "why" for everyone to march toward, each carrying a piece of the solution.

And none of the consultancies offering help today covers all the aspects of successful organizational change. Each one specializes in just one or two. No matter whom they hire, clients will be missing

at least one critical piece of the puzzle.

That is, unless they hire more than one helper, and manage the transformation program themselves so that their own “why” is firmly set as the guiding star. More on that later.

So, my answer to the question is “Yes,” transformation is possible. But you guessed that already, didn’t you? After all, had the answer been “No,” this would be the end of the book.

2 | Anti-Patterns of Organizational Transformation

Progress, far from consisting in change, depends on retentiveness. When change is absolute there remains no being to improve and no direction is set for possible improvement: and when experience is not retained, as among savages, infancy is perpetual. Those who cannot remember the past are condemned to repeat it.

— George Santayana

Every situation is unique in the details, but there are common patterns, or anti-patterns, one can observe in many cases.

In principle, it isn't hard to see how these anti-patterns could be turned around. But it's easier said than done.

This chapter describes several problems with organizational transformation programs that seem to occur again and again. They are:

Incomplete solutions - not all aspects of the problem are addressed, and one of the aspects that is not addressed ends up scuttling the program.

Misalignment of goals - the transformation program has its own goals that are different from the original reason the transformation was undertaken in the first place.

Lack of direction - there is no clarity about the end goal of the transformation program, or there are multiple agendas in play.

Separation of process-related and technical improvements - there is a belief that improvements in process and improvement

in technical infrastructure and practices can be independent of one another.

Conflating business agility with agile software development - the buzzword, “agile,” became popularized in two different contexts at around the same time; many people don’t distinguish between them, and assume that if they address one they are also addressing the other.

Reintroduction of the old - after achieving good results using lightweight processes on a pilot basis, people want to “scale” the approach; instead of continuing to use contemporary thinking to solve the challenges of scaling, they re-introduce Industrial Era methods of operating at scale, and lose the benefits they had gained.

Inappropriate organizational structures - people try to implement lightweight processes without restructuring the organization to align value-producing assets with product lines or value streams, resulting in excessive internal dependencies that introduce waste.

Big-bang change rather than incremental change - people assume they can change the organization to the desired state quickly, as if flipping a switch, and they underestimate the need for gradual change.

Rigid interpretation of frameworks - hoping to find a “canned” solution that can be dropped into place, people shop for a “framework” that promises good results, and implement the framework out-of-the-box without mindful customization.

Milestone-driven approach vs. directional approach - people plan and execute the transformation program as a linear sequence of predefined stages or phases rather than as a series of experiments with course adjustments based on the lessons learned from those experiments.

The cult of Scrum - due to the market dominance of Scrum as of the date of this writing, the majority of clients and transformation consultants assume Scrum must be at the core of all target opera-

tional models, without any analysis to verify the assumption case by case.

The soft stuff is the hard stuff - people underestimate both the importance and the difficulty of managing the human aspects of organizational change, and attempt to implement the change in a “mechanical” way.

This chapter addresses each of those anti-patterns.

2.1 | Incomplete Solutions

Each of the solutions available from organizational change consultants focuses on one or two aspects of the problem while ignoring others.

One dimension concerns defining the end state, understanding key principles of organizational change, and executing on those principles in a practical way. This involves:

- Defining the desired result of the change
- Understanding general principles of organizational change
- Having a practical way to effect the necessary changes

This goes back to the dot-connection problem mentioned in the previous chapter.

Various methods are good at their particular area of focus, but where is the solution that covers all three of the necessary conditions adequately? If it existed, then the problem would have been solved industry-wide by now.

I’ve observed three common failure modes.

First, people may have engaged a consultancy whose specialty is a particular structured solution, like a “scaling framework.” Lacking clear business goals, the framework itself became the goal of the

transformation. Even if they implemented the framework well, there was no connection with the original business problem.

Second, they may have read about organizational change and about robust product delivery processes, and attempted to transform their organization without help. As they hadn't done anything like it before, they weren't aware of the potential challenges and pitfalls, and there was no one around to guide them. They spent a lot of time and money going down various dead-ends and painstakingly finding their way back again.

Third, they may have attempted to effect change virally, starting with technical teams, in hopes the improved performance of the pilot teams would inspire others in the organization to follow suit. This approach is rarely effective, as the inexorable systemic forces surrounding the pilot teams inevitably annihilate them.

So, we see cases where people created a guiding coalition as recommended by Kotter, but couldn't actually get anything to happen on the ground. We see cases where people implement a branded framework, and they run the framework as documented, but there's no effect on the original business problem. We see cases where a few technical teams learn effective practices, but the improvements aren't visible because they are buried under the existing organizational constraints, and those constraints eventually drown the pilot teams.

What's needed is a comprehensive approach that begins with a target end state and applies robust continuous improvement methods guided by clear principles and tracked by appropriate metrics. All the dots together.

2.2 | Misalignment of Goals

Transformation goals are often not aligned with business goals.

Business goals take the general form, “We need the capability to do X so that we can achieve Y.”

For example, we might say “We need the capability to perform experiments in the market so that we can respond to changes in customer demand as well as shape new demand.”

Transformation goals tend to take the general form, “We want to implement our model or framework or method because it’s what we believe in.”

For example, we might say “We want to implement Framework X so that we can see people filling the defined roles and carrying out the prescribed events and activities of Framework X.”

The assumption seems to be that if only people would follow the rules of Framework X, business value would fall into place automatically. Many transformation programs have succeeded in implementing Framework X while completely missing the target with respect to business goals.

Even those consultants who emphasize business outcomes in their presentations tend to assess progress by checking whether clients are following the rules of the process they have recommended.

External help in implementing a framework can be very useful, but it’s up to client leadership to keep the program on target. Understandably, the consultants will be focused on getting their framework up and running, which isn’t the client’s real objective, but only a means to an end.

There are a couple of popular buzz-phrases that are red flags indicating the transformation program may not have a clearly-defined business goal:

- Agile Transformation
- Digital Transformation

In some cases, the idea that “going agile” will solve a multitude of other problems leads people to undertake an “agile transformation.”

Many of these don't yield any business benefit even after years of activity. Often, the reason is no business goals were ever defined. It can be beneficial to "go agile," but only if there's a reason for it.

In some cases, the allure of modernization leads people to chase "digital," without any definition of what it means or analysis of which business capabilities it will support. These programs often don't yield any business benefit, for the same reason: The changes aren't undertaken as a way to achieve a goal, but rather for their own sake. Digital is *shiny*.

The goal of the transformation must be to enable the organization's goals for competitive capabilities and operational capabilities, irrespective of the particular frameworks, models, philosophies, tools, or practices that may be required to accomplish it.

2.3 | Lack of Direction

In all too many cases, the crucial early step of identifying the business capabilities the organization needs to support is glossed over or skipped entirely, as people are keen to get started with improvements. As a result, there is no single "flag in the ground" or "beacon" that everyone can look to for direction, when they become lost in the details and need to remind themselves why they are doing all this.

Sometimes, the particulars of the chosen framework or method become goals in their own right. People focus on learning the roles, responsibilities, buzzwords, documents, metrics, and process steps defined in the framework or method they are implementing.

Sometimes, people in different roles with different perspectives about what "improvement" means will try to improve their own work with no shared context. People in different departments or groups go in different directions.

In either case, people lose focus on the business goals that inspired the transformation in the first place.

The lack of a singular beacon for people to follow has insidious effects. For one, managers who have different personal ambitions connected with the transformation initiative will pursue their own goals, and there are no checks and balances to keep them aligned with any common goal. For another, technical teams may use the transformation as a cover to play with new technologies or to pursue personal career interests that are not necessarily connected with any overarching business goal.

That single focus, whatever it turns out to be in any given situation, must be the “beacon” that lights the way forward for everyone involved in the transformation effort. Otherwise, there will be differing and probably conflicting goals and priorities in play.

2.4 | Separation of Process-related and Technical Improvements

When computers first became useful in business, it made sense to separate the big, noisy, hot machines from the normal office environment. It even made some sense to separate the strange people who liked to work on the machines from the normal people who worked in the office.

Today, computers and software are embedded everywhere, in every thing. Nearly every product or service on the market has programmable technology embedded in it. Nearly every interaction customers have with companies and government agencies is computer-driven.

Understanding customer needs and understanding the technologies that support those needs are no longer separate concerns. They are very closely linked, and in many cases, inseparable.

Most organizational transformation initiatives focus on structure and process to the exclusion of technical matters.

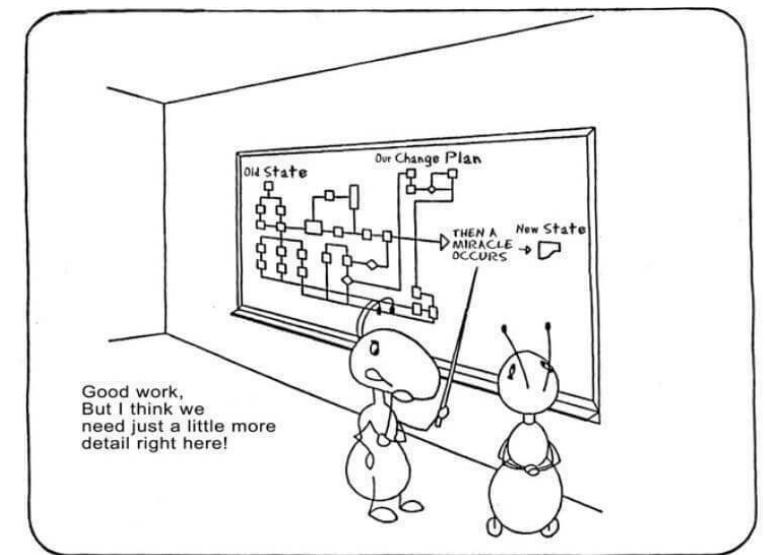


Figure 2.1: A miracle occurs

As we will see in “Leverage: Where’s the Fulcrum?” in this day and age the technical aspects of the transformation are central to success, and all the more so when the subject organization supports the corporate IT function, or when it produces commercial software.

Technical infrastructure and practices have no purpose except to support competitive capabilities for business operations. Similarly, competitive capabilities cannot be supported without close support from technical staff and a technical environment tailored to the need. The two concerns cannot be treated separately.

A key point, and one that I believe is quite different from any mainstream approach to transformation, is that the *business driver* or *reason* for the transformation is not necessarily the same as the *primary focus* for day-to-day improvement efforts. The primary

focus has to be the *fulcrum* for gaining leverage from the IT function to achieve business outcomes.

2.5 | Misunderstanding the Nature of the Work

On one engagement, technical coaches were assigned to one team apiece. In the area where I was working, none of the teams actually developed application software. Mostly they chased down production tickets and handed them off to the appropriate development teams for resolution. Yet, our contract stated we would teach them practices like test-driven development and refactoring to remediate legacy code. This was meaningless for the work they actually performed, and we were deemed unsuccessful.

This kind of thing is a side-effect of misunderstanding the nature of the work: The people who set up the contracts don't properly understand the nature of the technical side of the transformation. To them, it's just a checklist of buzzwords, like "test-driven development" and "legacy code."

2.6 | Conflating business agility with agile software development

The idea of *business agility* and the idea of *agile software development* started to gain popularity around the same time. Possibly for that reason, people often mix the two concepts. They are actually completely different ideas.

Business agility has to do with a company's ability to understand, assess, and respond to a dynamic market effectively. Agile software development is concerned with building and delivering application

software with a minimum of administrative overhead, delay, and defects. Proponents may list additional goals, but that is sufficient to highlight the differences.

Business agility is usually one of the primary reasons organizations seek to cultivate certain competitive capabilities. Agile software development doesn't provide that. Its purpose is to guide effective software development without excessive overhead. Both are important, but they are not the same thing.

The result of conflating the two ideas is that people have attempted to achieve goals pertaining to *business agility* by applying methods associated with *agile software development*. It has resulted in a whole industry around the idea of scaling Agile. This has not produced the expected outcomes, especially in combination with the next factor.

2.7 | Reintroduction of the Old

Agile software development originated in the trenches of application development. Most of the methods associated with it are designed to help a small software development team build applications. They are not designed to manage large-scale IT operations.

To fill the gap, Agile scaling frameworks re-introduce 20th-century structures and processes for portfolio management and governance, rather than re-thinking those functions through an “agile” lens. As a result, the “transformed” organization ends up pretty much the same as its pre-agile incarnation, only with more buzzwords in the air and sticky notes on the walls.

The whole idea of lightweight processes goes out the window when we start to re-introduce high-formality, 20th-century, Industrial Era methods of managing governance and compliance.

Instead, learn what the requirements really are for governance and compliance, and seek ways to meet those requirements using

contemporary thinking and methods.

There is more than one way to solve a problem, and these requirements can almost always be addressed in a lightweight fashion that supports the new way of working in the transformed organization.

A common example in IT organizations is the internal audit group. Work with them to find out what they really need. You may be surprised at how little is required. Cumbersome processes have built up over the years because there was no way to provide the information automatically. Today, it's usually straightforward to produce the necessary information automatically from build and deployment tooling.

Apart from matters of governance and compliance, larger organizations have the challenge of coordinating multiple work streams that have dependencies on one another. Often, a product line is supported by several teams, not just a single team, and “ownership” or full decision-making authority doesn't reside in any one individual, who could serve in the Product Owner (Scrum) or Customer (XP) role in the same manner as a small-scale development operation.

Agile scaling frameworks deal with this by adding administrative layers and process overhead in 20th-century style. A lighter-weight approach, drawing ideas from Lean and Agile, would minimize cross-team dependencies, manage buffers, and limit WIP to achieve the same goals.

It is not *necessary* to re-introduce Industrial Era structures and methods to deal with “scale.” Alternative approaches are well-known, such as Beyond Budgeting, Beta Codex, the idea of federated organizations. Certain companies are often held up as shining examples, like Semco, and Menlo Innovations.

Two examples from the retail clothing industry closely parallel the approaches a company might use to support software products or services in dynamic and stable markets. The Spanish company Zara reserves 85% of capacity for mid-season adjustments. They can respond to short-term changes in market demand quickly,

supplying stores with new designs within a couple of weeks of spotting a consumer trend. The Japanese company Uniqlo takes the opposite approach. Specializing in basic attire only, they focus on quality to maintain market share and schedule production in advance for the whole season to keep costs down (Stevenson).

If this is feasible for physical products, all the way from design to retail distribution in two weeks, then it's certainly feasible for an electronic product, which has no issues such as physical manufacturing, warehousing, or shipping. But outdated, Industrial Era methods won't support it, even if labeled "agile" or "lean."

2.8 | Inappropriate Organizational Structures

A useful rule of thumb from the Lean school of thought is that it's best to align value-producing resources and people with value streams or product lines. A similar idea from the Agile school of thought holds that good results come from cross-functional teams that include all the skills necessary to deliver a product.

Both ideas boil down to minimizing cross-team dependencies. Dependencies are responsible for a large part of the waste in conventional IT organizations. Every request or hand-off between teams introduces delay and increases the risk of misunderstanding and error.

In some transformation programs, teams are left in their original state. Typically, that means functional silos with numerous dependencies and hand-offs.

In other cases, there is an attempt to form cross-functional teams in accordance with Agile team concepts, when the current state doesn't allow for that degree of change in a single step. The result is "games."

Teams still have dependencies and hand-offs, but they try to cover the fact by defining their “user stories” and “definition of done” to align with whatever they are actually able to accomplish.

That isn’t necessarily “wrong” in the early stages of a transformation. It becomes a problem when it isn’t done mindfully and deliberately as one step in a longer process of improvement.

The result *looks* like an agile process on the surface, but in reality teams are delivering no more and no better than before. Because they are hiding information about how they actually work, they make it hard for people to identify opportunities for further process improvement.

Of course, it’s possible to change team structure and composition *incrementally*, shifting toward the goal state little by little over time. That sort of incremental change, done mindfully for the right reasons, is a sound strategy for improvement. Problems occur when people assume they have already transitioned to an “agile” team model, when in fact they’re only pretending.

Current organizational constraints may force us to keep certain service teams, and limit our ability to include literally all necessary skills on every delivery team. In that case, we want to set up the best structure we can at the outset and then incrementally improve it as we continue to evolve the organization.

2.9 | Big-bang Change Rather Than Incremental Change

Exacerbating the previous problem, many people assume whatever changes are recommended for the transformation have to happen all at once. In fact, it’s practical to adjust organizational structure and team composition incrementally as staff acquire new skills and as the necessary process improvements and tooling are implemented.

Many people who adopt a branded framework seem to think it's necessary to "install" the whole ball of wax immediately, and then struggle through the learning curve pain and other issues as best they can.

One of the more popular scaling frameworks at the time of writing is the Scaled Agile Framework, or SAFe. The current version of its guide book is 800 pages long. Imagine the difficulty of implementing something of that size all at once, with new structures and roles and activities and deliverables throughout.

In practice, it's easier and more effective to improve structure, processes, and practices incrementally using a mindful change management cycle, guided by metrics. The chapter, "The Cycle of Change," describes an approach to this. You could apply any sort of PDSA cycle in the same way.

2.10 | Rigid Interpretation of Frameworks

Frameworks for scaling Agile are intended as a starting point for continuous improvement. Clients are expected to tailor the frameworks to their needs and to improve their processes going forward as they gain more skill and capability. Consultants are expected to understand the framework well enough to help clients do this effectively.

Instead, in virtually all cases when a framework is used to guide an organizational transformation, the framework is implemented exactly as documented, with every optional activity and artifact in place. The assumption is that the framework fully defines a permanent end state for the organization, to be followed as a set of hard-and-fast rules.

That assumption is common among clients as well as among certified practitioners of the framework (especially recently-certified

ones).

We need to set aside the assumption that a branded framework is just a set of rigid rules that must be followed by rote whether they add value or not. This is one of the most damaging assumptions people make when they undertake an organizational transformation.

Instead, consider branded frameworks as generic examples of how things *could* be done. Think of them as guidelines rather than rules. Also, think of them as guidelines for a relatively “novice” level of performance, intended to serve as a starting point for ongoing incremental improvement. Ultimately, you want to be able to dispense with the framework altogether without any loss of fidelity or smoothness. They are a starting point, not an end state.

If we were to distill the essential principles from all the Agile scaling frameworks on the market, we would discover they are all based on the same foundations. They differ in details, emphasis, and wording more than they differ in substance. It isn’t necessary or particularly helpful to become enamored of any one of them.

A framework that supports the kinds of changes an organization is seeking can be a valuable tool in the transformation. It’s advisable to think of the framework as one tool among several or many that we are applying to the problem, rather than as “the solution.”

2.11 | Milestone-driven vs. Directional Approach

The nature of an organizational transformation can resemble a sort of journey or expedition. One of the leading consultancies in this space, LeadingAgile, explicitly uses a mountain-climbing expedition metaphor to describe it, complete with basecamps. Others in the Agile community frequently speak of Agile adoption as a “journey.”

And yet, many organizations and the consultants who guide them lay out a linear plan for the transformation. There are fixed dates with predefined milestones. The expectation is that the changes can be put into place in a more-or-less mechanical way. The transformation itself is based on the opposite sort of thinking from the organization's target operating model.

Naturally, things never proceed entirely according to plan. When milestones aren't met in accordance with the original roadmap, it leads to friction between clients and consultants. Sometimes, the whole transformation program is cancelled. That is certainly not helpful.

A directional approach, as opposed to a milestone-driven approach, offers a more realistic way to manage a transformation. With the beacon shining out in the darkness (see section 1.3 above), we focus on making progress in the right *direction*, rather than on meeting predefined success criteria on a fixed schedule.

It's ironic that the majority of organizational change consultants working today emphasize Agile in their marketing material, and yet plan, execute, and track transformation programs using "waterfall" thinking and methods.

2.12 | The Cult of Scrum

All mainstream Agile scaling frameworks begin with the assumption that Scrum will be used at the delivery team level, and Scrum concepts will apply to most or all activities at most or all levels in the organization.

Even consultants who claim their thinking is broader than just Scrum tend to use Scrum terminology and concepts throughout their own models, and in their presentations, talks, whitepapers, books, and websites. Scrum lives deep in the brains of agilists.

The centrality of Scrum in the minds of agilists is an example of one-size-fits-all thinking. The problem isn't that Scrum is "bad," it's that Scrum is the only conceptual frame that most agilists have. It's a hammer in search of nails. And it finds plenty of them.

Scrum is designed to support certain types of work flows quite well. It was developed in the early 1990s based on seminal work published in 1986 to address organizational issues that were prevalent in that era. It serves the purpose well.

But Scrum is not well suited to every type of work flow or every type of activity in an organization. The attempt to force-fit Scrum into every corner leads to excessive overhead and thrashing.

Don't get me wrong about Scrum. It's a well-thought-out and useful framework. It has earned a place of honor in the pantheon of software delivery methods, each of which addressed the problems of its day, and each of which advanced the state of the art: Linear SDLC, V-Model, Spiral, Evo, RUP, Scrum.

Today, a lightweight process based on Lean principles and designed to enable continuous flow is generally regarded as the state of the art. This sort of process need not follow the branded Kanban Method meticulously, but most teams refer to their process as a kanban system.

An advantage is the same system can be used at all levels of the organization and for all types of work flows, making it straightforward to implement on multiple levels to support scaled IT operations consistently and with metrics that roll up.

Another advantage is that a kanban system is easier to learn and apply than previous methods, including Scrum.

In organizations that continue to operate in 1980s fashion, Scrum still has an important role to play. It brings people together into cohesive teams and gives them a framework within which to learn to collaborate. It encourages teams to deliver solution increments frequently, when previously each release might have taken months.

It enables delivery teams to understand the customers their work serves, and to interact with them to some degree (even if indirectly) to explore solutions.

I think of Scrum in much the same way as Forrest Gump's braces. At first, he needed the braces just to stand and walk. When the day arrived for him to *run*, the braces fell apart and came off. They would have restricted him.

We may well determine that Scrum is the best fit for some teams in the organization we're transforming, at least at the outset. It just shouldn't be *assumed* as the long-term solution.

2.13 | The Soft Stuff Is the Hard Stuff

People pay lip service to "human factors" or "soft skills," but in practice this is an aspect of organizational transformation that usually receives insufficient attention, and that is poorly understood.

The anti-pattern in this area is to treat the transformation program as a "mechanical" change and to ignore the effects of stress, uncertainty, doubt, and fear on the part of the people in the organization.

Part 3, *Humanity* is devoted to the human side of organizational transformation, change, and coaching.

3 | Definitions

Most controversies would soon be ended, if those engaged in them would first accurately define their terms, and then adhere to their definitions.

— Tryon Edwards

There is a Glossary where you can find definitions for many terms used in the book. There are a few special terms that I want to describe in a little more detail here for clarity.

I propose approaching organizational transformation holistically by defining specific business goals and keeping everyone aligned to those goals, whether they're working in a technical area or not. That raises the question of what kinds of business goals are relevant as the targets for a transformation program.

I submit that the goal or goals of a transformation program must be *competitive capabilities* for the organization. Let's see what that means, along with a few other terms that will be used throughout the book.

3.1 | Capabilities

A *capability* is the ability to do something effectively, consistently, and sustainably.

The ability to rush work through to production without paying attention to quality is not a *capability*. It's not *effective*.

The ability to accomplish a goal *once*, by running a “death march” or “crunch mode” project and burning people out, or by wrenching parts of the organization out of their normal shape temporarily

by pulling people from here and there, is not a *capability*. It's not *sustainable*.

The ability to “hack” your way through something using random or *ad hoc* methods that no one can reproduce is not a *capability*. It's not *consistent*.

3.1.1 | Operational Capability

An *operational capability* is a capability possessed by an organization to carry out a particular business function as a routine part of its operations. The organization can carry out the function effectively, consistently, and sustainably.

Examples of operational capabilities mentioned in the book include *stable production operations* and *innovate products and services*.

3.1.2 | Delivery Capability

A *delivery capability* is an operational capability that pertains to the delivery of a product or product modification to the market or to production.

Examples of delivery capabilities mentioned in the book include *design for observability, resilience, and replaceability* and *continuous integration*.

3.1.3 | Competitive Capability

A *competitive capability* is a capability of organizations to interact with their market(s) effectively.

Examples of competitive capabilities mentioned in the book include *retain customers, respond to change, and shape the market*.

This is the type of capability that can serve as a meaningful goal or target for a transformation program. One of the key failure modes for organizational transformation programs is that they set an inappropriate goal, such as “implement SAFe,” “improve software quality,” or “reduce time-to-market.” Those things may well be useful, but only if they help the organization achieve one or more competitive capabilities.

3.2 | Leverage

According to Merriam-Webster, *leverage* is “the action of a lever or the mechanical advantage gained by it.” In the context of business and software, *leverage* is a metaphor for a force multiplier to gain business benefit from one or more operational capabilities.

Examples of operational capabilities that provide leverage include *stable production operations*, *dynamic infrastructure management*, and *test automation*.

Stable production operations is a force multiplier that supports competitive capabilities such as *respond to change* and *shape the market*.

3.2.1 | Fulcrum

Merriam-Webster says “[w]hen the word first appeared in English in the middle of the 17th century, ‘fulcrum’ referred to the point on which a lever or similar device (such as the oar of a boat) is supported.”

Building on the metaphor of leverage, the *fulcrum* for achieving the goals of an organizational transformation program is the key dependency that enables the competitive capabilities that are defined as the goals of the program.

While the competitive capabilities are the *business drivers* of the transformation, the *fulcrum* must be the focus of day-to-day transformation work. We identify the fulcrum by analyzing the dependencies of the target competitive capabilities.

The fulcrum then becomes the “beacon” that guides everyone’s work in the transformation program, regardless of their role or level in the organization’s formal hierarchy. This helps avoid the anti-pattern of differing and competing priorities.

It’s also more practical than trying to aim directly for the ultimate business goals, as they may be rather far removed from on-the-ground activities. It can be challenging to see any direct relationships between changes in day-to-day practices and a competitive capability, as there are probably many steps between them.

Part 2: Strategy

Can you define “plan” as “a loose sequence of manifestly inadequate observations and conjectures, held together by panic, indecision, and ignorance”? If so, it was a very good plan.

— Jonathan Stroud (*The Ring of Solomon*)

This section outlines my strategy for IT organizational transformation and for connecting the technical transformation work with the management transformation work that will most likely be done by a management consultancy.

The premise is that no consultancy currently in the market has the capability to handle both the management transformation and technical transformation work. Therefore, client organizations will need one helper for the management-level work and another for the technical work. It’s possible even more than two external helpers will be needed, depending on the capabilities and limitations of each consultancy.

With multiple external helpers engaged, it falls to client leadership to ensure everyone stays focused on the goals of the transformation program, and that the various consultancies don’t use the client organization as a sort of battleground among themselves, vying for follow-on business and/or control of the work.

4 | Connecting Management and Technical Consulting

We are like islands in the sea, separate on the surface
but connected in the deep.

— William James

A common failure mode for organizational transformation programs is that client leaders engage a single external consultancy to run the transformation program for their organization. The typical outcome is that either the business side or the technical side of the program falls down.

4.1 | Half a Solution Isn't Better Than None

None of the branded frameworks or management consultancies, as far as I'm aware, provides much guidance regarding improvement in technical delivery capability. Sometimes they will mention it ("we recommend Extreme Programming," or "don't forget to improve your technical practices while you're at it"), but in general there's nothing in any of the frameworks or any consultancy's offering that provides strong guidance on the technical side.

Without robust technical delivery capability, it's all just hand-waving; you can't execute on anything. In fairness to the consultancies, I think this is largely due to the shortage of qualified technical coaches, as described in the chapter, "Shortage of Technical

Coaches.” There’s not much they can do if the coaches literally don’t exist.

Conversely, there are a few small companies that specialize in technical coaching, and they do it well. They don’t offer comprehensive organizational transformation services.

Excellence in technical execution alone won’t yield substantial business value for larger organizations. It must be accompanied by excellence in market analysis, financial management, strategy, and planning.

An implication is that you have to be cautious about management consultancies that claim to be able to bring in an army of qualified technical coaches to support the transformation program. In reality they will only be able to muster a few coaches, and most of them will lack *coaching* skills as described in the chapter, “Coaching Skills,” to complement their technical skills.

You might argue (as will the management consultancies) that it’s just a question of money. They can hire qualified technical coaches if and when they’re needed. There’s a subtle problem with that reasoning. Management consultancies value management skills. They generally consider technical skills to be a commodity.

Whether that’s accurate or not, it’s the general perception in that world. They won’t pay technical coaches at a level that will attract qualified people. Supply and demand operates in this industry just as it does in any other, and the supply of truly *qualified* technical coaches is low.

That means the scope of technical coaching will be limited, and there will be more friction and conflict in that area than in other parts of the transformation program.

Consider a rumpled, stained five-euro note. It’s valid currency. It represents value. You can spend it.

Now consider a note that’s beautifully printed on the front, and blank on the reverse. It’s just a piece of paper. It represents no value.

You can't spend it.

Similarly, a transformation program must include proper consideration of timelines, people, resources, and costs for adequate improvement in both business capabilities and technical capabilities. It must engage people with the appropriate skills in all relevant areas.

Even an imperfect transformation (like a rumpled and stained note) has value. But a perfect implementation of half the solution has no value at all.

4.2 | A Proposed Solution

In many cases, the most practical solution to the problem is to engage two or more external helpers - one at the management level and one that focuses on technical improvement.

Imagine a process like mine for technical improvement (or a better one, if you've got one) plugging into the change method of a management consultancy that's been engaged by the same client. In principle, there should be no conflict, as there's no overlap in the respective mandates of the management consultancy and the technical one.

But don't assume there won't be conflicting agendas. The management consultancy might need to rent out some commodity-quality team coaches it has on the bench, and they're willing to go cheap.

On the other side, the technical consultants might imagine themselves as wanna-be executive consultants, in much the same way as a house cat sees the image of a tiger when it looks into a mirror.

Just remember both are good at what they do, and not so good at what they don't; and for all their knowledge and experience, the last thing they're likely to be aware of is their own limitations.

That means client leadership has to take the reins of the program and keep the external helpers aligned toward the single business

goal and focused on the *fulcrum* - the key dependency on which everything depends.

4.3 | The Fulcrum Is the Natural Dividing Line

Take a look at the dependency tree in the chapter, “Leverage: Where’s the Fulcrum?” Visually, it seems to split apart naturally into a top and bottom half. The business capabilities of interest are shown above the single key dependency, *stable production operations*. Everything below *stable production operations* feeds into it and makes it real.

That means *stable production operations* is the *fulcrum*; the focus of the day-to-day work for everyone, and the “beacon” that lights the way to the business goals.

Planning and tracking change above the key dependency generally falls to the management consultants who are engaged at the executive and management levels of the client organization. Planning and tracking change below the key dependency generally falls to the technical consultants who are handling that side of the transformation program.

That’s a simplification, as the boundaries tend to be soft, but it’s a practical generalization for purposes of this chapter.

4.4 | Connecting the Parts

It’s feasible to “plug” the technical improvement program into whatever change management process is being used for the overall transformation program.

But the *fulcrum* doesn't mark a hard boundary. The concerns of interest to management consultants and those of interest to technical consultants overlap.

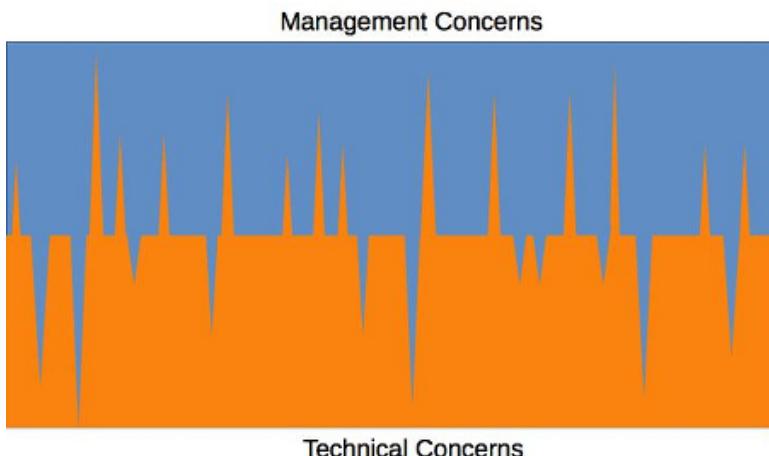


Figure 4.1: Overlapping concerns

The management consultants must know about and influence changes in certain technical areas. The technical consultants must know about and influence changes in certain business operational areas.

You can't engage the two external helpers separately and hold them at arm's length. They must collaborate directly to provide comprehensive support for the transformation program.

4.4.1 | This + A Management Consultancy

Many management consultancies offer sound guidance for business transformation. A technical improvement approach like mine, represented by "This," can interoperate with any management consultancy's general transformation method.

4.4.2 | This + An Agile Scaling Framework

Let's say client management decided to go with an "agile adoption" strategy rather than a general improvement strategy. We can plug this model into that sort of solution, as well. Apart from generic advice to use good practices, frameworks like SAFe don't have much to say about specific technical practices or how they will impact business outcomes. SAFe recommends "Scrum and XP" at the team level, but beyond just saying it the framework doesn't contain any specific guidance.

The lack of a change management piece in frameworks like SAFe leaves us free to apply the Cycle of Change for continuous improvement for everything that feeds into the key dependency. A fairly smooth continuous improvement process is possible on the technical side. This does not conflict with the framework implementation.

4.4.3 | This + A Lean Process Improvement Method

Client leadership might approach the transformation using Lean principles and a continuous improvement model such as Kanban Method or Kaizen. Both of those draw heavily from TPS, Theory of Constraints, and Lean Thinking, so we know they are based on demonstrated practical methods.

These methods don't include any specific advice regarding technical practices that are relevant to IT operations, as they aren't specific to IT organizations. To create a complete transformation package, we can plug our technical improvement model into a Lean-based process improvement model.

The Cycle of Change is a PDSA implementation, and all the thinking behind our approach to technical improvement is based on Lean principles, so there's no friction. In fact, this is the easiest marriage

of the technical improvement approach with an organizational change management system.

4.4.4 | This Alone

Most commercial frameworks and consultancies tend to generate a large quantity of documents, charts and graphs, reports, and other materials. There tends to be a heavy meeting schedule that's fairly intrusive on client management. We can take a simpler approach without losing control of the program.

Unless the transformation goal is to revamp the entire enterprise from top to bottom in profound ways, it isn't necessary to engage a second "helper" to handle the transformation above the key dependency. It's feasible to follow a lightweight change management approach to keep the transformation program on track, when the focus is on IT support for a short list of competitive capabilities, as the majority of changes will be within the jurisdiction of the IT group.

4.4.4.1 | Set Up the Guiding Coalition

Identify the people who should be on the Guiding Coalition and see that they can be engaged with proper focus and without distractions from business as usual. See the chapter, "Structures and Responsibilities," for more information.

4.4.4.2 | Set Up the Consulting Team

Form the Consulting Team and establish their charter and operating standards. See the chapter, "Structures and Responsibilities," for more information.

4.4.4.3 | Preliminary Investigation

Clarify what business outcomes are to be achieved, how progress will be measured, and what dependencies exist for achieving the goals.

- Identify market-facing products and services that fall into the four quadrants in the Process Alignment Model
- Decide how market-differentiating, mission-critical products and services should be supported
- Assess current-state capabilities for supporting market-differentiating, mission-critical products and services to identify gaps between current and future state capabilities
- Map out dependencies to identify the key dependency that will be the focus of improvement efforts (the “fulcrum” to get leverage from IT)
- Determine the metrics to be used to track progress and detect issues; focus on the target competitive capabilities and the key dependency; try not to rely on indirect measures; use metrics that aren’t dependent on how work is done, but only look at outcomes
- Analyze interactions between applications and systems and data flow through the technical infrastructure to identify coupling between systems that needs to be loosened to isolate the systems that provide market differentiation
- (If at scale) Recruit internal candidates to become technical coaches and plan their training.

See the chapters, “Target States: Where Do We Need to Be?” and “Leverage: Where’s the Fulcrum?” for more information.

4.4.4.4 | Align Teams with Products

Organize teams to align with the market-differentiating and mission-critical products and services, to the extent other existing organizational constraints will permit.

Initially, this may not result in the optimal team structure. There could be practical limitations that preclude setting up end-to-end teams that can support systems from concept to operations support. We want to compress functional silos to the extent practical given current constraints, and plan to compress further as we advance.

See the material in Part 4 for more information.

4.4.4.5 | Training on the Cycle of Change

Introduce staff to the Cycle of Change and how to apply it at different scopes and time frames for different kinds of improvements.

See the chapter, “The Cycle of Change,” for more information.

4.4.4.6 | Take Baseline Measurements

Measure current state performance with respect to the target competitive capabilities and the key dependency to establish baseline performance metrics for future comparison.

See the chapter, “Metrics,” for more information.

4.4.4.7 | Start Rolling

Begin to apply the Cycle of Change under guidance from coaches. Track progress against the selected metrics.

5 | Begin With the End in Mind

In the end, we begin.

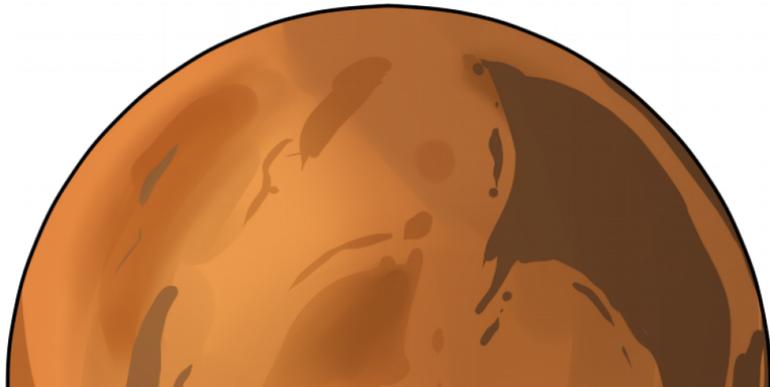
— A. D. Posey

How do we start? We choose a destination. There's little value in instituting an improvement program if we have no idea where we want to go.

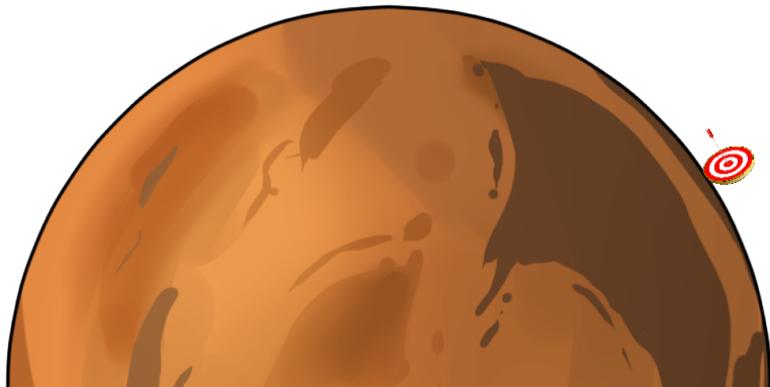
Granted, it isn't realistic to believe we can predefine every detail of our goal state in advance. We won't even be able to visualize what is possible until we've started to take steps in one direction or another. Our imagination is necessarily limited by the horizon we can see from wherever we are at the moment. Our target is subject to change as we progress and learn.

One of the challenges in leading an organizational change initiative is the difficulty of visualizing possible futures. Most of us can't see very far beyond the current state, try as we might.

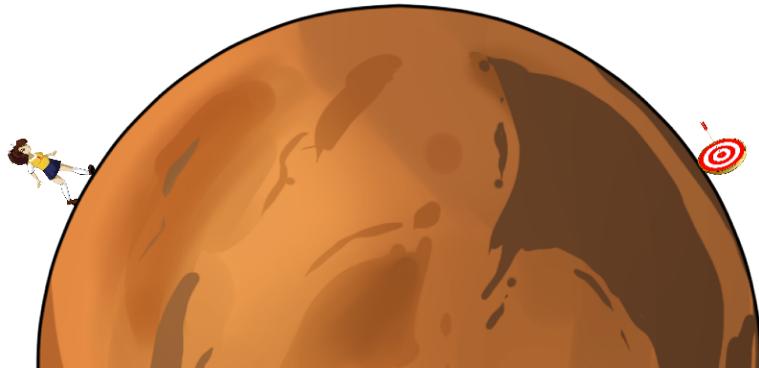
Let's say we're embarking on a journey across a planet, like this:



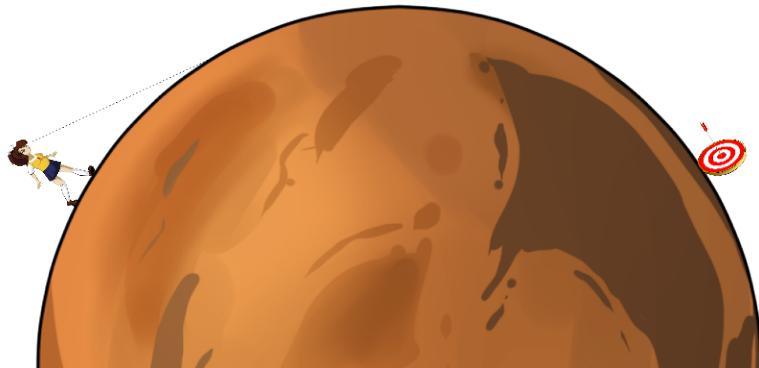
We've identified a desired goal state based on market-facing operational objectives. Let's say the goal state lies here on the planet's surface:



And here we are, ready to embark on our journey:



Ah, but there's a problem. From where we're standing, we can't see the target. We can't imagine how our organization could possibly operate in that manner. Our line of sight goes no farther than the horizon we can see.



Okay, so we define interim goals, starting with a target we *can* see from our current position:



But there's more. We also have our existing organizational structure and formal processes in place, as well as our deep experience working in the *status quo* operational environment. Our imagination may be limited by these factors.

We might be tempted to throw up our hands and say, "It's impossible!" We can't see the future through the walls of our castle. And it's such a *beautiful* castle, we're loathe to tear it down or poke holes in the walls.



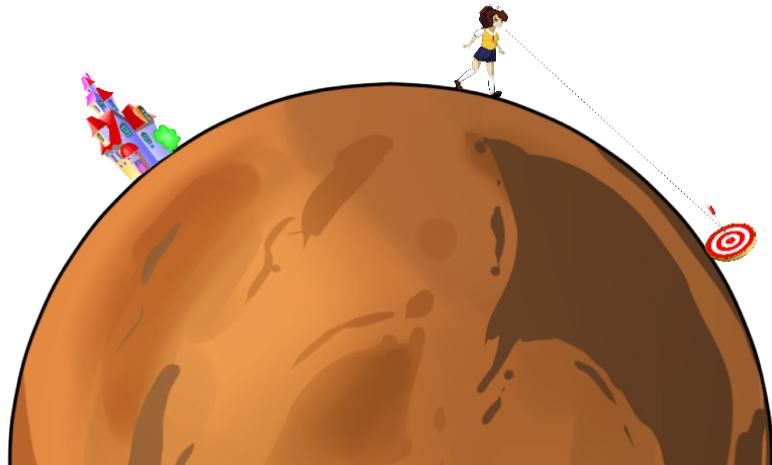
So, the first order of business is to clean house. Then we'll be in a position to see where we're going, at least as far as our current line of sight allows.

But that's easier said than done. We've never visited the territory where the target lies. We might not even believe it's all that has been described to us. How can we push through our conceptual barriers and move forward into the unknown with any degree of confidence?

That's where knowledgeable helpers can add value. They've "been there and done that." They know what it looks like, and they know how to get there from here. They help by charting out a series of interim targets; a roadmap. The first target is within our line of sight. We can imagine going there. It will be easier if someone walks along with us, but at least we can visualize it.



Step by step, interim goal by interim goal, we finally reach our destination.



Or, conversely, we see a new destination that's better; a destination that wasn't visible to us until we made a certain amount of progress.

6 | Target States: Where Do We Need To Be?

Too often my target is whatever happens to have wandered in front of me instead of being something that I have intentionally put myself in front of.

— Craig D. Lounsbrough

Our information technology (IT) organization supports numerous different business activities. Some of these are revenue-generating and some are overhead.

The revenue-generating activities interact with markets in different ways, calling for different kinds of IT support. Some are market-differentiating and some are necessary for parity with competitors. Some are dead-center of our corporate mission, and others are peripheral.

To know where we need to be, we have to align our systems and operations with the business activities they support. This is consistent with an idea from Lean Thinking, to align value-producing resources and activities with each value stream. For us, it amounts to structuring our IT organization to align with business operations.

The overhead activities may be necessary or unnecessary, and if we're like most IT organizations, we might not be exactly sure which are which. We have to figure that out, and determine how best to eliminate the unnecessary overhead activities and minimize the cost of the necessary ones.

It's likely we will end up with more than one target operating model. We may have two sets of market-differentiating products

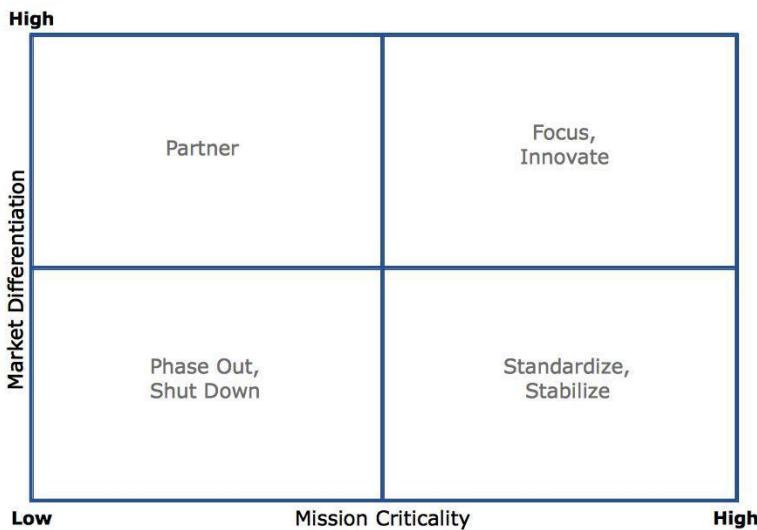
or services, one of which calls for highly dynamic interaction with the market featuring frequent changes and customer feedback, and one of which calls for stable operations with changes released on a predictable cadence.

There may even be subsets of the same system that support, respectively, dynamic market interaction and stable product delivery. Teasing them apart can help us understand what changes are needed to support the transformation goal without unnecessary additional effort.

Our internal IT processes must be able to handle both cases smoothly. We can't afford to be too slow to respond to the market in the first case, and we don't want to overwhelm our customers with overly frequent change in the second case.

One tool that can help us understand how best to support each product or service is Niel Nicklaisen's *Purpose Alignment Model*. In this chapter we'll show how to use the model to assess IT support in a hypothetical bank.

The model looks like this:

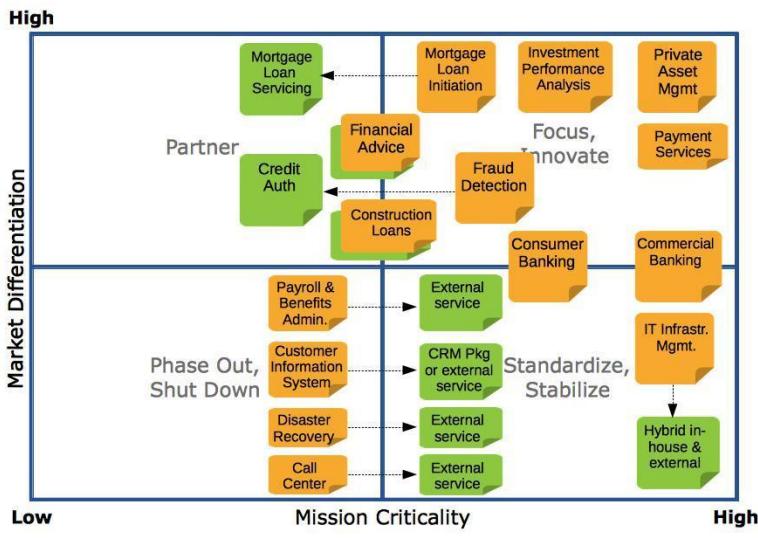


Niel Nickolaisen: Purpose Alignment Model

Figure 6.1: Purpose Alignment Model

Products that are market-differentiating go into the upper right-hand quadrant, which we label “Focus, Innovate.” Nickolaisen labeled it “Differentiating.” Products that are mission-critical but not market-differentiating go into the lower right-hand quadrant, which we label “Standardize, Stabilize.” Nickolaisen labeled it “Parity.”

Here’s how we might apply the model for a hypothetical bank:



Niel Nickolaisen: Purpose Alignment Model

Figure 6.2: Sample analysis using PAM

In the example, we've determined that our services in the areas of Private Asset Management, Payment Services, and Mortgage Loan Initiation differentiate us from the competition; but Mortgage Loan Servicing doesn't provide a good revenue stream, so we're happy to engage a partner to handle that. Also, the quality of our Investment Performance Analysis system and Fraud Detection system, which are not visible to external customers, help distinguish several of our services in the market.

We've also decided that we don't want to focus on Financial Advice as such, but we need to retain control of that offering so the advice given is consistent with our principles and our other services. Similarly, we can use a partner to take on most of the work for Construction Loans, but we aren't ready to relinquish control. We have a market leadership position and we want to maintain it.

Our Consumer Banking and Commercial Banking offerings are differentiating, but also rather stable. We want to pay attention to

them lest we lose market share, so we plan to standardize the parts of those services that make sense to standardize while continuing to nurture and innovate the parts that distinguish us in the market.

We've decided our internal IT infrastructure management activities belong in the Parity or "Standardize, Stabilize" quadrant. The way we manage IT infrastructure is not a differentiator for us. A high-level assessment of our infrastructure reveals nearly all of it falls into one of two categories: (1) Microservices development and support, and (2) mainframe development and support.

Most of the infrastructure management around microservices can be handled by an external supplier on a commodity basis, but we think it's best if we control certain systems ourselves. We decide on a hybrid solution.

Outsourcing mainframe support is very expensive; we decide to keep that in-house. Other one-off systems, such as web and mobile applications for consumer banking and B2B data exchange interfaces, will also be supported in-house. These don't account for a large proportion of our infrastructure support work.

That leaves back office systems. Nickolaisen calls this the Who Cares? quadrant. We label it "Phase Out, Shut Down."

Payroll and Benefits Administration, Disaster Recovery, and Call Center support are very different from one another. What they have in common for our bank is that they are neither mission-critical nor market-differentiating. We need not devote our expensive internal technical staff to supporting these things.

All of them can be outsourced, as there are companies for whom each of these is a mission-critical and market-differentiating service. We can take advantage of their focus on these activities, just as our own customers take advantage of our expertise in our differentiating services.

The other "Phase Out, Shut Down" system is the customer information system. Like many large, long-established banks, we

have multiple home-grown customer information data stores and applications spanning multiple legacy platforms. Keeping all the data up to date and in sync is a significant operating cost, and it grows more difficult with each passing year as the technologies age.

We decide to bring a Customer Relationship Management system into play. We defer the decision of whether to host it in-house or outsource it.

The Purpose Alignment Model is a fairly simple tool to help us sort through our systems and determine where they fit in the larger scheme of the bank's business operations. We could have used any other tool or method that serve a similar purpose. There's no magic in any particular tool.

7 | Leverage: Where's the Fulcrum?

There is no need to use force. Instead, create a path of least resistance, and gravity will do the rest.

— Michael Dunlap

To obtain leverage from the IT area to support the target business operating models, we need to understand which factors in our complex environment have the greatest impact on our ability to support business needs. Those must become the focus of the transformation initiative.

My approach is to map out the dependencies necessary to support the desired *competitive capabilities* of the organization. Competitive capabilities are supported by *operational capabilities* and *delivery capabilities*.

The goal of the transformation is to establish sustainable operational models that support the desired *competitive capabilities* (and not to “implement” a framework). I call this *outcome-oriented* transformation.

Let me clarify: By *outcome* I mean a *competitive capability*. I’m not talking about the kind of “outcome measurement” that’s proven to be problematic (Bolton, Chris).

Let’s see what is necessary to enable and support the two operational models we discovered when we applied the Purpose Alignment Model for our hypothetical bank: “Differentiating, Dynamic Market” and “Differentiating, Stable Market.”

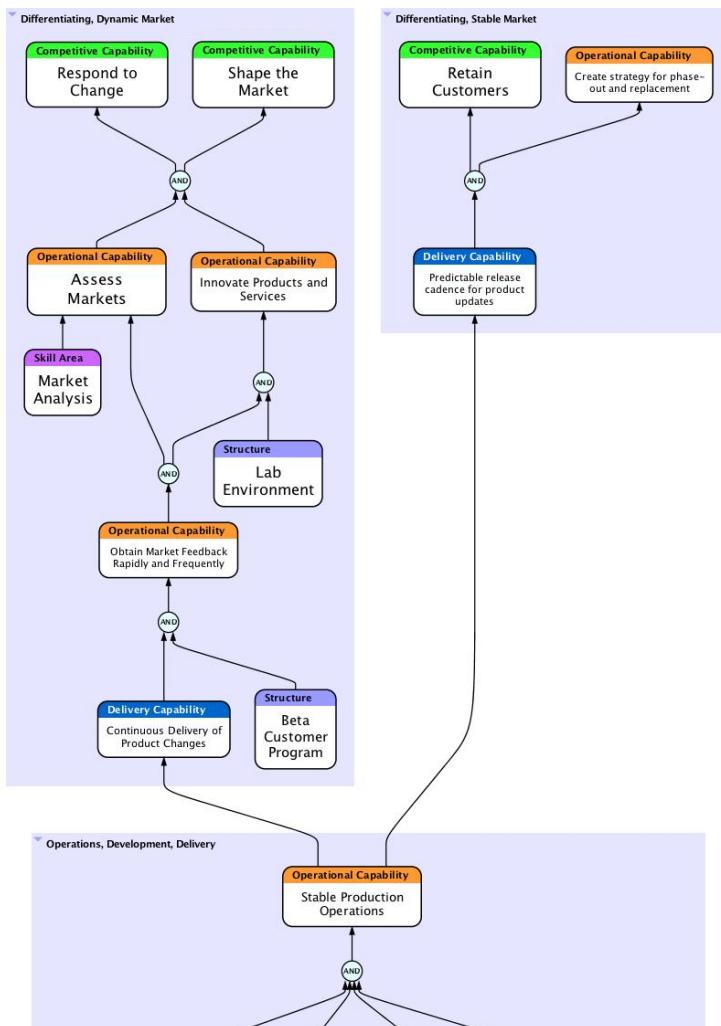


Figure 7.1: Key dependency of business goals

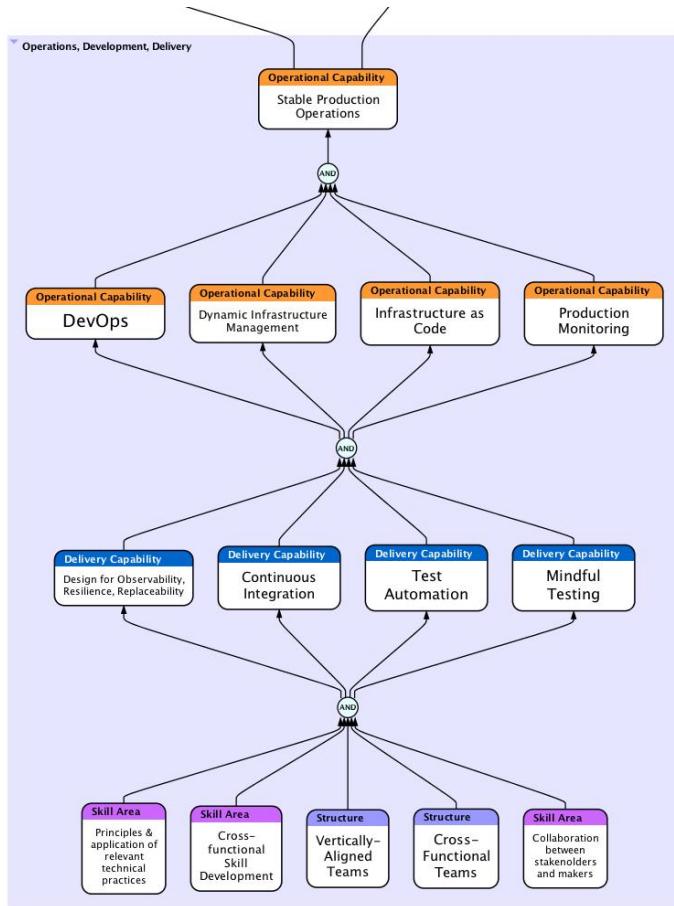


Figure 7.2: Dependencies of stable production operations

Let's take a moment to clarify terms.

A *competitive capability* (green in the diagram) is a market-facing organizational capability. It's the kind of thing that can be a meaningful business goal for the transformation. The need for a competitive capability may be the business driver or purpose of the transformation.

An *operational capability* (orange in the diagram) is an internal

organizational capability. In general, operational capabilities support competitive capabilities. In my view, an operational capability cannot be the business driver or reason for the transformation. It is a supporting capability. A competitive capability may *depend* on one or more operational capabilities.

A *delivery capability* (blue in the diagram) is an operational capability that pertains to the organization's ability to deliver products (typically, software systems) to a production state. I call that out separately from other operational capabilities because delivery tends to be a direct responsibility of the IT area. Therefore, improvements in delivery capabilities will be addressed differently from other operational capabilities.

Now to the analysis.

At a glance, we can see that one element is foundational to all competitive capabilities. That's pretty obvious just from the shape of the diagram even if we don't read the labels. The same element is dependent on a large number of specific organizational structural considerations, processes, and technical practices covering all aspects of IT support.

If we seek leverage from our IT area to support business goals, the fulcrum must be that single element: *stable production operations*.

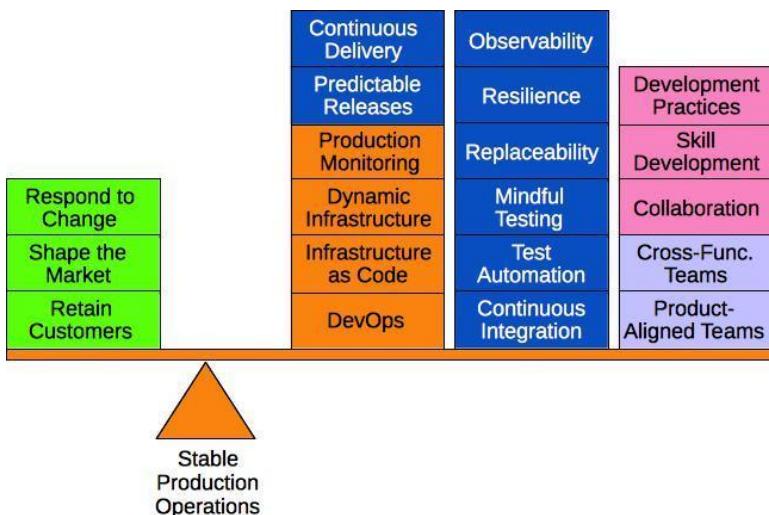


Figure 7.3: Leverage

Note there is a great deal more weight on one end of the lever than on the other. The implication is the core or heart or primary focus of the transformation effort lies within the technical sphere: Ensuring our production operations are stable and reliable, and building high confidence that any changes can be released to production safely and automatically.

High-level business concerns determine *where* the fulcrum is located in our organization. Those considerations *drive* everything, and yet they may or may not fall out of our analysis as the necessary *focus* of the transformation.

Confusing the *driver* of the transformation with the *focus* of the work on the ground may be a cause of transformation failures in many cases. With this approach, we avoid that error, as we focus on the systemic element that has the greatest impact on competitive capabilities.

In this case, *stable production operations* must be the focus with respect to the IT organization specifically, as it is the enabler for both operational models.

I'm not debating the relative importance of "business value" versus "technical excellence." I'm only going where the dependency analysis leads. If something else had turned out to be the key dependency, then that would be the focus of transformation work.

Reliable production operations enable any competitive capability we might choose to pursue. Thus, business stakeholders must lend their support to creating and sustaining stable production operations.

Similarly, all improvements in technical practices must be aimed at achieving stable production operations. Technical improvements are not undertaken for their own sake, or to comply with a theoretical model of "best practices."

One problem with previous transformation programs, and particularly Agile adoptions, has been the focus on specific technical practices. Is the team doing [insert favorite practice here]? If so, then the transformation is a success; otherwise, it isn't.

But what's the value in doing [insert favorite practice here] if every release is a fire drill, and we don't achieve desired competitive capabilities because we can't rely on operations? It's possible some teams will need to use [insert favorite practice here], but if so it must be for the purpose of maintaining the stability of production operations, and not because the technical coach personally likes [insert favorite practice here].

Transformation initiatives that focus on anything other than the "fulcrum" are likely to experience challenges above and beyond those inherent in the complexity of the task. In our view, the inherent complexity of an organizational transformation provides sufficient challenges to keep the work interesting. We need not add ancillary challenges unnecessarily.

Given the centrality of production operations, if we get into the habit of asking ourselves what the impact on production stability is likely to be of each change we propose, at every level, both technical and non-technical, we probably won't go too far wrong very often.

8 | Getting Started

Don't try to figure out the whole race. Just figure out where to put your foot for the starting line. Just start.

— Jeff Olson (*The Slight Edge*)

Even if we intend to guide the transformation *via* a series of iterative changes based on the scientific method, there is considerable work to do initially to get the work off to a proper start.

8.1 | Impact: Structure, Process, Practices

It's important to understand what *kinds* of changes are likely to result in the greatest improvement for the least cost.

Improving *organizational structure* has a larger impact than any other category of changes. Bad structure will prevent good processes from functioning well. Hobbled processes will cancel out the effects of good practices.

The second most impactful changes are improvements to *process*. Improvements to *practices* are also necessary and important, but their impact may not be felt until the larger issues of structure and process have been corrected.

Think of a water pipe. In principle, one can calculate the flow of water based on the diameter of the pipe. But if the pipe takes many twists and turns, the structure can interfere with flow.

In our organization, we want to straighten the pipe so water can flow freely. That means putting people together who have all

the skills necessary to support a product, and giving them full responsibility for that single product, and nothing else, from end to end (including production support).

With a suitable structure in place, we need to ensure our *processes* don't impede the flow of work.

If, for long-forgotten reasons, the inside of the pipe is fitted with baffles, flow is impeded. Only a trickle emerges from the end of the pipe. Running the pump harder only causes pressure to build up in the pipe, possibly leading to a backup into the pump or a burst tube.

Similarly, if our processes include needless overhead that blocks work, then flow will not be smooth and delivery will be delayed. Often, such delays also lead to defects, rework, and other issues. Telling people to work faster is like running the pump harder; it only causes work to back up and clog the pipeline, and people to stress out.

If the water pipe is old and the inside is encrusted with corrosion, flow is impeded.

Similarly, if we don't apply good practices in our work because of old habits that never caused much of a problem before, we'll introduce unnecessary defects, rework, delay, and other issues to our work flow. If we didn't notice this effect before, it was probably because of the twists and turns in the pipe or the baffles built into it.

8.2 | Full-Stack Slices

A common mistake made by conventional Agile consultants is to ignore the back end environment in larger organizations. Agile methods are applied to the front end only - mobile, web, mid-tier servers, mid-tier databases, external APIs, and the client side of

internal APIs. Everything behind the internal API layer is treated as a black box.

Why? Agile consultants will tell you it's because the systems that live on legacy platforms in the back end need not operate at the same level of responsiveness as the newer systems that live on modern platforms.

I will tell you it's because very few Agile consultants have any expertise with legacy back end technologies.

The determining factor for whether a system needs to operate with a high level of responsiveness is neither the platform it lives on nor the consultants' experience with legacy technologies, but rather the market-facing competitive capability it supports.

If we find a system supports such a capability, and parts of that system live on legacy back end platforms, then our mandate for transformation includes those systems. No one promised it would be easy.

8.3 | Up-Front Analysis

The initial work on the transformation program has been described in previous chapters. The objectives were to identify the business drivers of interest, the single key dependency (or very short list of dependencies) for those business drivers, and the systems that support market-differentiating and mission-critical business operations.

8.3.1 | Identify Competitive Capabilities

Identify the *competitive capabilities* the organization seeks to achieve by undertaking this transformation program. This is covered in “Target States: Where Do We Need To Be?”

The objective is to gain clarity regarding which business operations provide market differentiation and are within the company's core mission, and which belong in other categories defined in the Purpose Alignment Model.

8.3.2 | Identify Value Streams

Identify the products, product lines, and/or value streams for which IT systems provide support, and that fall into the categories leadership decides will best be handled by internal people and resources. This is informed by the analysis described in the chapter, "Target States: Where Do We Need To Be?"

The objective is to arrive at a preliminary list of applications and systems that are to be supported in house, those for which we should seek a partner, those that can be outsourced or treated as commodity services, and those that can be sunsetted.

A key outcome: This analysis tells us how product-aligned cross-functional teams should be organized, and where there may be dependencies we can't break initially.

8.3.3 | Identify Focus for Transformation Work

Identify the key dependency that will be the focus of day-to-day work and the beacon to guide transformation activities. This is covered in "Leverage: Where's the Fulcrum?"

The objective is to understand where the focus of day-to-day transformation work should be, in order to ensure we strengthen the key dependencies for the target competitive capabilities.

In our example, this is *Stable Production Operations*.

8.3.4 | Identify Key Metrics

Based on the initial analysis, we work in collaboration with client leadership to select the metrics that will be used to quantify progress in the transformation program.

8.3.5 | Take Baseline Measurements

We observe the current system in operation for a period of time and take measurements of its effectiveness using the metrics that were selected to track progress in the transformation program.

When feasible, it's best to measure at least one complete delivery cycle from concept to cash. In larger organizations whose current operations require long timelines, it may not be feasible to do this. In those cases, we want to make a reasonable judgment about how long to observe and measure, and how to extrapolate the end-to-end performance of the system to provide a meaningful baseline.

8.4 | Foundational Activities

To set the stage for starting the iterative improvement process, we need to understand dependencies, data flows, inter-system boundaries and interfaces, and potential technical roadblocks deeply enough to set a direction and define initial improvement experiments.

8.4.1 | Identify Dependencies Supporting the Focal Point

Determine the underlying dependencies that support the focal point of the transformation.

In our example, the focal point is *Stable Production Operations*, and the dependencies are the technical infrastructure, delivery tooling and procedures, staff knowledge and skills, and technical practices that may have an effect on the stability of production operations.

In fact, this includes every technical activity of every sort that the IT organization handles.

8.4.2 | Identify Connections Between Systems

Determine the connections and dependencies across the various IT systems and applications currently in place. We are looking for the “seams” where we can tease apart systems that have become entangled over many years of operation and maintenance, so that we can separate the systems that require different treatment going forward based on the “target states” analysis.

Careful analysis can help us separate subsets or components of applications that may be handled differently going forward.

For example, in our hypothetical bank, we determined that Mortgage Loan Initiation is a market-differentiating and mission-critical function, while Mortgage Loan Servicing is a market-differentiating but non-mission-critical function. The existing systems are probably tightly coupled, and we know we have work to do to separate them.

There may be opportunities for the same sort of separation that are less obvious based on the initial analysis. In many older organizations, solutions have been layered on top of solutions over multiple generations of technology. Some remnants of earlier solutions may still play a part in the systems operating today. We can simplify the environment, and thereby reduce risk and cost, by eliminating the older remnants and folding any necessary functionality into the current solutions. Development work will probably be necessary to accomplish this. We can discover some of these opportunities

incrementally over time, but we may be able to get a strong start by up-front analysis at this stage.

Not to be underestimated: Clear understanding how data flows through the entire collection of systems and applications. Typically, no one in the organization knows this fully, and most data flows are not documented anywhere. This is often a challenging aspect of IT organizational transformation.

There's often an obvious clue that this is the case: When client staff are reluctant to change or delete a data structure; remove or alter a data element from a database schema, file layout, or data feed; or modify the meaning of coded values that may be accessed by downstream systems (but no one can name which systems), it indicates people are not really sure about how data flows through the organization.

8.4.3 | Identify Gaps

This gap analysis must be extensive horizontally, but need not go into great depth. It includes structures, processes, and practices applied on “the business side” that pertain to the “connecting tissue” between business operations and IT support.

It also includes all sorts of gaps within the IT area, including infrastructure, tooling, review and approval procedures, technical practices, automation, and production system monitoring.

This gap analysis will inform the choice of initial experiments and the scope of those experiments. In general, the priority will be on the systems that support the particular market-differentiating and mission-critical business operations that, in turn, support the competitive capabilities of interest to client leadership.

8.5 | Starting Iterative Improvement

Informed by the results of the initial analyses, we select the first round of experiments using the Cycle of Change. Experiments may be undertaken at multiple levels of abstraction concurrently. Some will be of large scope and require weeks or months to complete. Others will be of small scope and require hours or days to complete.

It's advisable to choose a cadence for running experiments, collecting and assessing feedback from the experiments, and working with client leadership to understand the implications of the results and to choose next steps in each cycle.

So many variations are possible regarding details that it would be pointless to try and list every permutation in a book of this kind. Here are a few examples to illustrate the concept.

- Replace the annual budget cycle with a quarterly process of assessing market conditions and re-evaluating business priorities, and allocating funds to various initiatives for the next quarter.

Scope: Large. Timeframe: Nine to twelve months.

- For an application that needs to interact with its market dynamically, supported by a continuous delivery process, treat all expenditures as expense. The system is live as of the first iterative release and development is complete only when the system is retired; therefore, it is difficult to know which parts of the investment could be capitalized, or even whether that is a legal option in this scenario.

Scope: Large. Timeframe: Six to Nine months.

- For development and support teams aligned with Product A, rotate developers into production support so they can learn how to manage production operations.

Scope: Medium. Timeframe: A few weeks to a few months.

- To support automation of end-to-end functional checking for Application X, develop a system for generating engineered

test data.

Scope: Medium. Timeframe: A few weeks to a few months.

- For development team B, establish hard stops for commits controlled by static code analysis settings to encourage specific improvements in code base health.

Scope: Small. Timeframe: A few days to a few weeks.

- For server resources associated with Application C, convert manual provisioning to scripted provisioning under version control.

Scope: Medium. Timeframe: A few days to a few weeks.

- For development team D, strive for 100% test-driven code changes and measure the differences in code quality, cycle time, and developer stress.

Scope: Medium. Timeframe: A few days to a few weeks.

9 | The Cycle of Change

Repetition, repetition, repetition; equals Results.
— Jeanette Coron

A failure mode in organizational transformation programs that involve IT is that the consultants insist on extremely radical changes in organizational structure, roles and responsibilities, process flow, work item definition, and metrics as a *first step*. It's often more than organizations can handle, and the program stumbles.

Another failure mode is the opposite - the consultants say “start where you are” without making any foundational changes, and attempt to effect incremental improvements. The weight of existing organizational constraints is too great to enable any of the changes to take hold.

To avoid the risks of these extremes, I propose an approach to organizational transformation that involves several initial changes, based on what has been observed to be effective in industry, followed by an ongoing series of “experiments” guided by a plan-do-study-act (PDSA) cycle.

Any PDSA model may be used. In this book, I use one that I created by extending an earlier and simpler model by consultant Corey Ladas. I call it the *cycle of change*, and it looks like this:

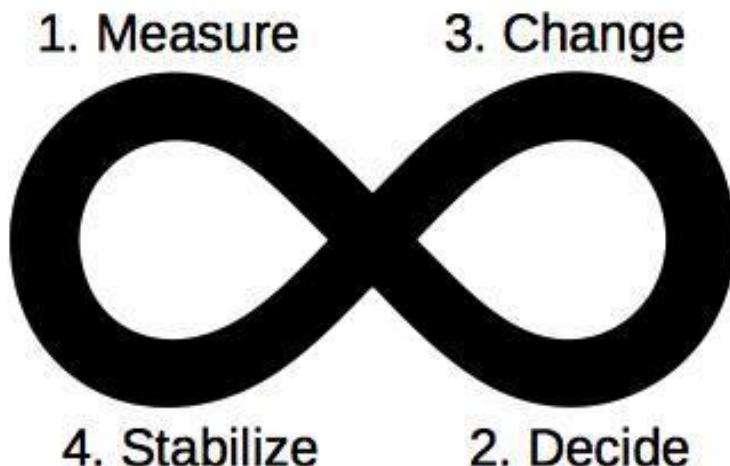


Figure 3.1: Cycle of Change

We begin with measurements of whatever factors are relevant to the thing we aim to improve. That way, we understand the “before” state of the system. We then decide on a change that we think will result in improvement. We make the change.

Normally, there’s a period of stabilization after change is introduced to any system. We wait for that to occur so that we won’t get a false reading when we measure the “after” state.

When we measure the same factors again, we can tell whether the change we made had any effect on the system’s performance. If the change moves us closer to the goal, we keep it and continue the cycle. Otherwise, we decide whether we want to try and adjust things to make the change effective, or reverse the change and try something different.

The approach is based on the idea that an organization of humans is a *complex adaptive system*, not (merely) a *complicated* system. In that sort of system, it isn’t possible to predict the results of making a change, as cause-and-effect doesn’t operate linearly. Instead, we use an approach called *sense and respond* (Diana). We probe the

system and then respond to the feedback it provides.

The cycle of change is a fractal pattern; that is, it applies at all scales in the organization. Large-scale changes usually require more time to settle in and demonstrate themselves than small-scale changes.

For example, an executive may choose to implement some form of incremental funding and to discontinue the annual budget cycle. The cycle of change in that case might take a year to observe and measure. The scope of the change is large. The impact of the change is large.

In contrast, a technical team may decide to change from a multi-branch code commit strategy to a trunk-based strategy. The effects of this change will become apparent in no more than a couple of weeks. The scope of the change is moderate. The impact of the change is moderate. The impact will increase with repetition by other teams.

Finally, an individual developer may decide to try incrementally refactoring a piece of legacy code. The effects of this change will become apparent (to the developer) in no more than a few minutes. The scope of the change is small. The impact of the change is small. The impact will increase with repetition by other developers and other teams.

The cycle of change is structurally similar to the *scientific method*. There are slight variations in descriptions of the scientific method, but the following steps are pretty common. In parentheses I've added notes relating the steps in the scientific method to the cycle of Change.

- Observation (measure)
- Question (assess the measurement)
- Hypothesis (decide on a change)
- Experiment (implement the change, allow system to stabilize)
- Results (measure again)
- Conclusion (assess the measurement)

From this perspective, the cycle of change appears to be a framework for experimentation. Many of the changes we might attempt in the course of a transformation are in the nature of experiments or trials.

For example, we might wish to adopt a proactive approach to monitoring production operations, rather than the *status quo* reactive approach of waiting for support tickets to be entered. We try out a particular tool to support this. We can measure the stability of production operations and the duration of any outages before and after the change to decide whether to keep the new practices and the tool to support them.

However, there are certain organizational structures, processes, and practices that are strongly supported by experience and observation of other organizations, and we may recommend these directly without any experimentation. For example, the advisability of aligning value-producing resources with value streams is not really in doubt. Restructuring teams in this way is not an experiment, but a correction.

10 | Monitoring Progress

Every line is the perfect length if you don't measure it.
— Marty Rubin

My observation is that in the majority of cases, there are challenges in how progress is measured and tracked. This often leads to canceled transformation programs.

10.1. | Anti-Patterns In Monitoring Progress

There are two general anti-patterns in monitoring progress in transformation programs. Either nothing meaningful is tracked, or progress is assessed in large chunks, although actual progress occurs incrementally and gradually. Borrowing an idea from Lean, I think of the latter as large batches of change.

10.1.1 | No Meaningful Monitoring

I've been surprised many times by large-scale transformation initiatives in which neither the client nor the external helpers guiding the transformation had any real idea how well things were progressing. In many cases, the transformation eventually petered out and the consultants were released, but it was unclear whether any genuine improvement had been achieved.

The pattern reminds me of a child running outside to play after a rainstorm. The child finds a puddle and stirs up the mud from the bottom with a stick. After a few minutes of watching the mud swirl

around in fascination, the child grows bored and runs off to climb a tree or play a game. The mud settles back down to the bottom of the puddle where it started.

Many large organizations have gone through this process multiple times over a period of years. Successive attempts don't build on the improvements made in previous ones, as results were not quantified and no institutional "memory" of the improvements was created. Each transformation program is a new child with a new stick in hand, watching the same mud swirling in the same puddle.

Perhaps there is turnover in client leadership, and when the new leaders review the initiatives in flight to see how money is being spent, the consultants have no way to quantify the value of what they've been doing.

Perhaps existing leadership becomes disillusioned with the consultants due to the usual stress of change and the common issues around communication and misunderstanding, and they don't see any evidence of improvement.

Perhaps the helpers in the transformation are predisposed to distrust metrics, and prefer instead to ask people how they feel about the way things are going. As soon as there's a change in leadership or something goes badly, there can be bad feelings with no objective information to fall back on.

In any case, it seems sensible to select some quantitative measures to use as indicators of progress, and as warning signals of problems.

10.1.2 | Milestones With Assessments

Progress with organizational change occurs gradually and, one hopes, fairly steadily. Yet, most transformation programs are divided into stages or phases with discrete milestones. Different consultancies use different words, but the pattern is the same. Progress is assessed against the goals of each milestone in a pass/fail fashion.

The result usually comes down to a checklist of items that are judged by consultants or coaches; either the client organization achieved the milestone or not.

There are three fundamental problems with this approach.

First, operational excellence is not a binary question. There is a range of performance, and no organization performs identically at all times or under all circumstances.

A pass/fail model doesn't align with reality. A directional model that allows for variation and ranges of performance would be more realistic.

Second, the exact meaning of "pass" is rarely well-defined. An item may be checked off as a "pass" or "complete" when in reality the organization has barely scratched the surface of satisfying that item.

Often, two consultants or coaches who observe very similar outcomes by different client teams will reach different conclusions about whether each team has completed the milestone. The result is inconsistent assessments.

Just as often, the organization achieves success with the item but the specific criteria that were defined by the consultants weren't quite met. The organization has achieved the milestone by any meaningful measure, but doesn't get credit for it because there are items on the consultants' initial checklist that ultimately prove to be irrelevant.

Third, no one achieves improvement in large "batches" of changes. There tends to be a rush toward the end of each phase or stage to collect information about the criteria for passing the milestone, and to check the boxes on various assessment checklists.

Just as many clients implement a published framework as-is, even when the creators of the framework remind them that they are expected to tailor it to their needs, many consultants likewise take the sample templates and checklists provided by their employers

exactly as-is, rather than selecting and customizing specific items for each client. The result is that the documents and guidelines used to assess progress aren't consistent with the particular changes that were made during the milestone period.

The assessment exercise turns into a meaningless administrative formality, done in a rush to satisfy the expectation that the consultancy will submit a progress report. Making the result even less meaningful, progress against milestones may be reported in the form of a "traffic light" metaphor - green, yellow, or red - or another conceptually-equivalent format.

This anti-pattern is similar to the situation in public schools in the United States in the late 20th and early 21st centuries. Great emphasis was placed on standardized tests.

The intent was positive - to ensure all students received equal opportunity for education, and to ensure school systems provided a solid grounding for young adults to function in the society and economy.

But due to the way schools and districts were assessed, the program resulted in "teaching to the test" rather than teaching generally and then allowing the test to demonstrate student progress.

Similarly, checklist-style assessments at the end of each milestone tend to result in people learning how to pass the assessments, without necessarily truly understanding the substance of the recommended changes.

Consultants and coaches may be complicit in gaming the system, as their own performance reviews by the consultancies they work for may hinge on the "success" of their clients, and "success" is defined (merely) as passing the assessment for each milestone.

10.2 | When Should We Check Progress?

The Cycle of Change model suggests the frequency with which we should check progress. For cycles of large scope and long duration, we can keep an eye on key metrics throughout the cycle. For cycles of small scope and short duration, we can use the length of the cycle as the cadence for taking measurements.

As we want to avoid large “batches” of changes, and we also want to avoid slipping into a “waterfall” style of managing the transformation program, we want to collect observations to generate trend lines as smoothly as possible, rather than doing “assessments” at months-long intervals.

I like to call this *directional* measurement. If the trends suggest the organization is moving in the right *direction* (toward transformation goals), then all is well. Otherwise, some form of root cause analysis is called for.

As for the *rate* of change, that is going to be whatever it needs to be based on organizational constraints, leadership support, staff readiness and openness, the size of the initial gap between current state and target state performance, and any blockers or challenges that weren’t apparent in the beginning and that are exposed through the ongoing transformation work.

By tracking progress incrementally, we avoid unpleasant surprises about milestone objectives that can’t be met “on schedule.” We can make course adjustments accordingly. This is more constructive and more realistic than repeated customer disappointments and hasty explanations to try and prevent the contract from being cancelled.

10.3 | How Should We Check Progress?

The transformation involves

- Establishment of new or improved competitive capabilities
- Improvement in all structures, processes, practices, and tooling on which the target capabilities depend
- Cultivation of an environment characterized by mindful improvement guided by the scientific method.

To track progress in these areas, we need

- Well-selected metrics (direct measurement of the target competitive capabilities as well as their key dependencies)
- A consistent, repeatable approach to introducing change to the system at all levels (the Cycle of Change, in the present model)
- Attention to stress-related behaviors and deliberate conversion of conflict into learning opportunities (the joint responsibility of client leadership and the consultants guiding the program)
- Attention to the health of the relationship between client personnel and those helping to guide the transformation program.

“Progress” means movement in the direction of target business capabilities. Therefore, we should check progress primarily by comparing the current state with the goal state at fairly frequent intervals. The trend lines that emerge will indicate whether we’re moving in the right direction. If we aren’t, then we’ll know we need to determine the reasons why and adjust our experiments accordingly.

An advantage of incremental improvement with close-to-continuous monitoring is that we know we will achieve at least *some* improvement even if the program is canceled before the end goal has been reached. With stages and milestones, the typical pattern is that when the program is canceled, everything reverts to the *status quo ante*, leaving the organization with no benefit at all in exchange for the expenditure they've made.

10.4 | Metrics and Managed Change

Often, people tend to measure factors *other* than the explicit goals of the transformation. They may check to see that people are following the “rules” for a branded framework. They may check to see that technical teams are using (or they are convincingly *pretending* to use) the recommended technical practices. For some reason, they don’t directly measure the *things that matter*.

This approach reminds me of legislation that was proposed in Germany some years ago to introduce low speed limits on the Autobahn. The purpose was to reduce carbon emissions from vehicles. The proposal didn’t get very far. Germans like to drive.

It occurred to me at the time that if the government wanted to reduce carbon emissions, they could set limits on carbon emissions. Then the automotive industry and individual inventors would look for ways to build fast cars with low emissions.

If you’re goal is A, then why not measure A? What’s the use of measuring B, C, D, and E?

10.4.1 | Determine What To Measure

In the approach presented in this book, everything begins by identifying one or more *competitive capabilities* the organization needs but does not presently have. We can judge whether we’re

making progress by comparing the current state at any moment with the desired end state. If we're improving, the gap between the two will shrink as the transformation program progresses.

In "Target States: Where Do We Need To Be?" we presented a sample analysis for a hypothetical bank. We determined there were three *competitive capabilities* the bank needed - two that pertain to responsive services that interact with their markets dynamically ("Respond to Change" and "Shape the Market"), and one that pertains to stable services that interact with their markets on a defined release cadence ("Retain Customers").

Those capabilities are the *things that matter* to the client. If we measure anything at all, we should at least measure the things that matter to the client.

Referring to the dependency diagram in "Leverage: Where's the Fulcrum?", we see the key dependency, and therefore the primary focus of day-to-day transformation work, is *stable production operations*. It seems sensible, then, to choose measures of stability in production operations, and track improvement in those measures.

It's an indirect indicator of progress, but with so many dependencies in play and so many different categories of improvement, we need to track a few key indirect indicators to help us understand where the challenges lie in meeting the transformation goals.

Feeding into *stable production operations* we see quite a few different dependencies of a technical nature. Each element in that diagram is just the tip of an iceberg. There may be a large number of new skills to learn, new tools to install, new procedures to institute, and new team structures to put into place.

Depending on the scope and complexity of the transformation program, we may well want to track specific measures within each of these functional areas.

Rather than specifying a set of "favorite" metrics, I want to suggest that the specific metrics may vary from situation to situation,

but the general approach of focusing on the end goals and key dependencies will help us steer the program and drive toward the business goals.

10.4.2 | Start With a Baseline

Recall the Cycle of Change:

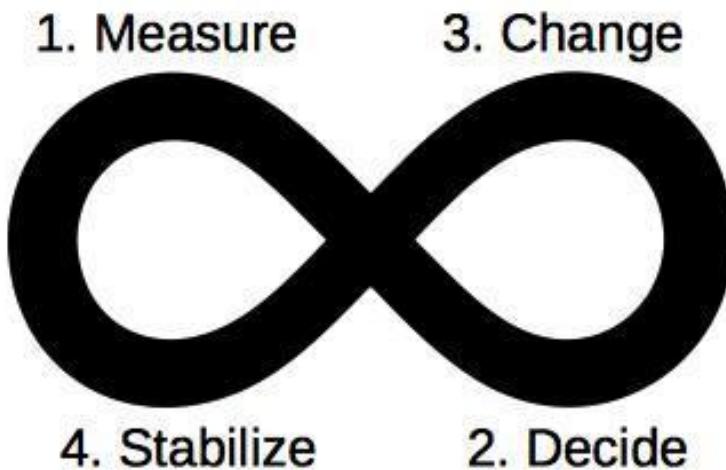


Figure 10.1: The Cycle of Change

The cycle begins with *measure*. Before we even begin making changes, we need to understand the organization's current state with respect to the transformation goals.

We gain that understanding by measuring current state performance against the goal state measures. That must include direct measurement of the *competitive capabilities*, and it probably will include additional indirect measures as well.

Every cycle at all levels in the organization will be guided by measurements. The *decide* step is based on an objective to improve one or more measures.

After the *change* is made and the new (modified) system *stabilizes* (that is, when people overcome the natural “change curve” to we can observe the behavior of the system with the change in place), we measure again.

Measurements occur at all levels and scopes of work throughout the transformation program. As lessons are learned, the specific metrics we track may change, but on the whole the metrics selected at the outset will be the ones that tell us whether we’re making progress.

10.5 | Cultivating Continuous Improvement

Transforming the work environment from one of rigid plans, blame, and finger-pointing into one of flexible plans, responsiveness to change, collaboration, and learning would be labeled “culture change” by most consultants.

I consider it more in the nature of changes in habitual behaviors, professional expectations, and standard procedures. Deeper cultural change, in the true sense, may or may not follow with time. For the immediate purpose of establishing the target competitive capabilities, it isn’t necessary for the organizational culture to change on a deep level.

However, to establish the desired habits, expectations, and standards it will be necessary to discover and shepherd every instance of friction and conflict among staff members, among consultants/coaches, and between staff members and consultants/coaches.

This is a formal responsibility of the Guiding Coalition, the Consulting Team, and every individual involved in the transformation program at a leadership level or in a consulting or coaching role.

We can use both quantitative and qualitative measures to track progress in this area. Quantitative measures may include counting

the number of incidents per unit of time.

Qualitative measures may include judgments about how severe each incident is, whether it caused damage to the transformation program or to relationships among people involved, and whether we were able to turn it into a positive learning event.

11 | Metrics

All conflict in the world is essentially about our differences in measurement.

— Joseph Rain, (*The Unfinished Book About Who We Are*)

In my book, *Software Development Metrics*, I suggest that there are a couple of main purposes for metrics in a software development and delivery environment: Steering work in progress, and tracking improvement.

Both categories of metrics have a role to play in monitoring progress in a transformation program.

11.1 | Process-Agnostic Metrics

We're changing the organization. That means today will be different from yesterday, and tomorrow will be different from today. We'll be changing our structure, process, and practices. Therefore, we can't use metrics that depend on *how* we work to track improvement in *how* we work. We need metrics that track our *effectiveness*, and that yield the same information even when we change *how* we work.

11.1.1 | The Problem With Velocity

An obvious example of a process-dependent metric is *velocity*, commonly used with Agile software delivery processes. *Velocity* depends on two things: (1) iterations of the same length, and (2) delivery of a solution increment to production in each iteration.

In many cases, people don't deliver a solution increment to production in each iteration, for one reason or another. Maybe they don't really understand what it means. Maybe they haven't worked out a way to do it just yet. Maybe organizational constraints preclude delivery of small increments to production on a short cadence. Maybe the inherent complexity of the product is such that no single, small team could possibly deliver a fully-functional solution increment alone. Maybe the nature of the product doesn't lend itself to incremental delivery. Many reasons are possible.

When that's the case, then whatever people say their velocity is, it isn't. Unless the two defining characteristics of time-boxed iterative delivery are met, there is no "velocity." There may be some sort of count of some sort of thing, but it isn't *velocity*.

Let's be more positive about it and say that teams *are* delivering solution increments on a steady cadence based on time-boxed iterations. Great. Their velocity indicates their delivery capacity. You could say they have "improved" when their velocity improves.

But what happens on the day when the teams out-grow the need for the time-boxes, and shift into a continuous-flow delivery mode? With no iterations, what does "velocity" mean? It means nothing at all. If we've been using velocity to measure team delivery performance, we now have no consistent metric to indicate further improvement.

Even an interim step between time-boxed iterations and continuous flow could "break" velocity as a metric. Consider a team that uses User Stories, and bases short-term planning on Story Points. Their velocity is expressed in terms of Story Points.

When the day comes that they've learned to slice the work into reasonably same-sized pieces, all the User Stories have the same relative size. Most teams that achieve this use a size of 1 point for all User Stories.

They're still using time-boxed iterations, so velocity as such still makes sense, but now they're counting User Stories instead of

Story Points. They don't have an apples-to-apples comparison to demonstrate or quantify their improvement.

11.1.2 | Lean Metrics

Basic metrics associated with the Lean school of thought are process-agnostic. That is, they measure outcomes regardless of *how* those outcomes were attained. If we measure performance using these metrics, we can change our structure, process, and practices as much as we want without "breaking" the metrics.

The most commonly-used Lean metrics are:

- Throughput - the number of "value units" delivered per unit of time
- Lead Time - the total time it takes to deliver value to a customer (including wait times)
- Cycle Time - the mean time it takes to complete a single work item
- Cycle Efficiency - the percentage of Lead Time in which value is added to the product (also called Process Cycle Efficiency)

Improvements in the effectiveness of delivery can be indicated by any of these metrics. They don't depend on *how* the work is done, so as we improve our processes and methods the improvement is reflected in the numbers.

11.1.3 | Caveats for Lean Metrics

I like to use Lean metrics in transformations, as they don't depend on *how* work is done; they just measure *results*. So we can change how we work without breaking our metrics. That's perfect for a transformation, because we're intentionally changing how we work, and we need a way to measure the results.

But it's good to keep context in mind. We're borrowing Lean metrics from the manufacturing sector, but our work is in the nature of product development, not manufacturing. We might want to increase cycle efficiency by reducing variation, if we're producing No. 6 machine screws on an assembly line. But with software products, it's the variation that provides value to customers, and enables market differentiation.

That doesn't mean we shouldn't care about cycle efficiency at all; consider the degree of improvement in software delivery if we raised cycle efficiency just to 5%. That's a 2.5x to 10x improvement over the *status quo ante*. I just want to suggest it's wise to avoid going to extremes.

11.2 | Directional Metrics

I mentioned in "Monitoring Progress," that it's problematic to treat the transformation program as a series of phases or stages with predefined milestones. Instead, we want to carry out the transformation through a series of experiments using the Cycle of Change (or some other PDSA mechanism).

It isn't meaningful to say the organization has achieved a list of criteria in a binary, pass/fail way as of a given milestone date. It's more meaningful to say the organization is moving in the right direction, toward the stated organizational goals.

Directional metrics amount to a series of observations of *performance* metrics visualized as trend lines. We want to see the trend lines leading toward the goal state.

It turns out that this approach works well for measuring improvement at all levels of abstraction, from improvement in the organization's target competitive capabilities all the way down to improvement in the "health" of application code bases.

With directional metrics you never pass or fail a milestone. You're moving in the right direction or you aren't. That's all. It's quite intentional that there's no point where you say "good enough."

There may be a point where you don't need the consultants anymore, but by that time you should have cultivated the habit of continuous improvement in the organization. That habit means you're never "finished" improving. (Having that habit is, in fact, one of the main indicators that you really *don't* need the consultants anymore.)

As you advance, you'll be able to see opportunities for further improvement that you may not have been able to imagine previously. The metaphor is the walk around the curved surface of a planet, as presented in "Begin With the End in Mind."

11.3 | Measuring Capabilities

For purposes of this model, I've distinguished among three categories of capabilities: competitive, operational, and delivery. A competitive capability is market-facing; an operational capability supports one or more competitive capabilities. A delivery capability is an operational capability that lies within the purview of the IT function of the organization.

These differences are meaningful for some purposes. For example, to qualify as a meaningful business goal, a capability must be market-facing. Otherwise, it's a supporting capability only.

When it comes to measuring progress, we can use the same mechanism for all three. What we need are:

- a clear definition of what it means to have the capability; and
- metrics that pertain to that definition.

Let's consider one of the competitive capabilities we identified for our hypothetical bank: *Respond To Change*. This is a capability the organization needs in order to support its products and services that interact dynamically with the market.

We might define a certain level of performance as the target; say, we want to be able to respond effectively to an unexpected change in customer demand or an unexpected security exploit within one week.

That becomes the target for the capability. Frequently throughout the transformation program we measure the time it takes us to respond to unexpected market changes. If we are moving in the right direction, the gap between current performance and target performance will narrow. Should we go off course, the gap will widen. The more frequently we measure, the quicker we'll detect any problems.

Now let's consider one of the operational capabilities that supports the competitive capability, *Respond To Change*. In our sample analysis, we see *Innovate Products and Services* is a supporting operational capability.

We can follow the same pattern to track progress in improving this capability. First, we define what it means to be able to innovate products and services in a way that effectively supports our capability to *Respond To Change*. Then, we repeatedly compare current performance with target performance.

Delivery capabilities can be monitored in the same way. From our sample analysis we see that the operational capability *Innovate Products and Services* depends on the delivery capability *Continuous Delivery of Product Changes*.

We begin by defining what we mean by "continuous delivery" in the context of our organization and business needs. For discussion purposes, let's say we decide it means we can activate new software features in production four times per day, without creating

regressions, security exposures, or any other issues that affect the stability of production operations or the experiences of our users.

We then track progress by comparing current state delivery performance with that target.

11.6 | Measuring Stability of Production Operations

Our dependency analysis determined that *stable production operations* is the key dependency for all the competitive capabilities of interest. I consider that result to be general; it will apply in nearly all cases.

We need a practical definition of “stability” for production operations. That may be a fairly complicated matter. “The” production operations in a large enterprise is usually *multiple* production operations, including some managed in-house, some managed externally, and some residing on legacy platforms that weren’t designed to support contemporary methods of monitoring and operating systems.

A general sense of a definition will probably include:

- placing modified application software or changing application configuration settings will not result in a customer-visible outage or a “crash” or “hang” of any component of production operations
- emergent behaviors of production operations (considered as a complex adaptive system) will not cause customer-visible impact to accuracy, responsiveness, availability, usability, accessibility, or security of the system - this becomes more of a factor as people move toward numerous small, interdependent services as opposed to monolithic solution architectures

- updating selected components or libraries in production operations will not cause any instability, outages, security vulnerabilities, or other issues

Once you get into the details in your own situation, you will probably think of more factors to include in the definition of *Stable Production Operations*. Whatever the list turns out to be, we then need to identify appropriate metrics to track our progress in improving the situation. Ultimately it comes down to comparing current performance with target performance on a frequent basis.

11.7 | Measuring Software Quality

People have struggled for decades with the question of how to measure software quality. Often, people resign themselves to counting reported production defects and leaving it at that.

There is good news. When we measure outcomes rather than underlying details, it becomes easier to define “quality” for software. It boils down to the impact of software changes on production operations.

If application code is causing instability in production, then quality is low. Otherwise, quality is good enough.

Here you might protest that production operations could be stable (that is, up and running), and yet users are unhappy with the systems they depend on. In that case, we need to adjust our definition of “stable production operations.” Anything that makes the environment unsuitable to any stakeholder for any reason is a form of “instability” for our purposes.

In principle, if people start to use robust methods of software design and testing throughout the development cycle, “defects” will only be a significant factor early in the transformation. Counting them will soon become irrelevant as people start to use better methods

and practices to develop, test, package, and deploy application software.

Similarly, if configuration and provisioning of servers or VMs or containers is causing instability in production operations, then when people begin to generate those components from version controlled specifications, subject to the same engineering principles as application code, then we should see improvement in the stability of production operations.

Our measurement of production operations stability actually provides an indication of code quality, as well. If code quality were low, production operations would be unstable. This would be evident through functional and non-functional problems with applications that were moved to production status.

The *outcome* of interest is *stable production operations*, and not (necessarily) academically-perfect code or eliminating every bug. Is a bug that doesn't affect any user, never triggers an alert by our monitoring tools, and never results in a production support ticket really a bug? Does it matter? What matters for supporting our competitive capabilities is the stability of production operations. So, that's what we measure.

11.8 | Measuring Code Health

Although we are mainly interested in measuring directly the capabilities and the key dependency of those capabilities, there are so many small improvements we might make in the area of software development and delivery that it's sensible to measure some of them, as well.

The notion of *code health* is that a “healthy” code base will yield a number of benefits, including simplicity, stability, reliability, understandability, and low risk of change. All these characteristics contribute to the goal of *stable production operations*.

11.8.1 | Static Code Analysis

Static code analysis tools provide ample information to track improvements in the “health” of a code base. Such tools often define a large number of rules by default, and we cull out the ones that aren’t important to us.

Some rules are relatively insignificant in most situations; for instance, detecting blank spaces at the end of lines in source files. That’s the sort of thing you can configure your text editor to remove automatically, if it’s important to you.

Other rules provide good indicators of code health. For instance, SonarQube offers a metric called *Cognitive Complexity* that indicates how difficult it is for a human to understand the source code (Campbell).

Quite a few other measures, many of which may be more familiar, can be used to assess code health. SonarQube in particular defines four categories of rules grouped into three domains: Maintainability, Reliability, and Security. Hundreds of specific rules are available. Other tools have their own schemes.

11.8.2 | Directional Measurement of Code Health

It’s straightforward to include static code analysis in the continuous integration build for each application. It provides early warning of many common types of problems in code.

For purposes of a transformation program, there are two additional factors to consider for tracking improvement in code health:

- tracking changes in code health over time (directionality); and
- maintaining psychological and political safety for developers (no finger-pointing).

Static code analysis tools are usually used to detect point-in-time problems that were just introduced in the current build, so developers can fix them immediately. We can use the same data points to track code health *directionally* over time.

Stable production operations have as a dependency *healthy* or “clean” application code, provisioning scripts, database scripts, etc. We can extract raw data from static code analysis databases to construct trend lines that will indicate whether we’re moving toward or away from a desirable level of code health.

Patrick Welsh, a respected technical consultant in the midwestern United States, came up with a way to express code health metrics that doesn’t identify individual developers.

The method is to use the static code analysis tool to detect questionable lines of code, and to report the source units where those lines are located. We count the lines of source code in those units, and report the percentage of source lines in the code base that “live” in source units that triggered static analysis rules.

We might see, for instance, that 9% of the source for Application A lives in *modules* that exceeded our threshold for *Cognitive Complexity*. That gives us a point-in-time observation.

Going forward, we repeat that measurement frequently, and develop a trend line of observations of *Cognitive Complexity* (or any other metric we deem appropriate). If the trend is positive, we’re on course. Otherwise, we have some root cause analysis to do.

Because the information is aggregated by source units, no individual developer is ever put into the spotlight.

11.8.3 | Custom Code May Be Needed

I don’t know about other static code analysis tools, but SonarQube does not output exactly the raw data we need to present code health metrics in this way. It does expose an API we can use to

find data we need, such as the starting and ending line numbers of methods, modules, and classes. We have to write code to generate the directional code health metrics in the form Welsh devised. This is not terribly burdensome, but as of the time of writing there were no off-the-shelf tools to extract the data we need.

The custom code we write has to “know” a little about the source languages in question so that we can report code health in terms of meaningful source code units. We might refer to “modules” and “classes” for Ruby applications, to “methods” and “classes” for Java, “paragraphs” and “programs” for COBOL, and “modules” and (possibly) “functions” for F#, and .h files and .cpp files for C++.

Besides that, in order to calculate the number of source lines in each source unit, the custom code has to know how to recognize the beginning and end of a source unit in the programming language at hand. For Java, the code could look for a method declaration and count lines until it found the corresponding closing curly brace. For Ruby, it could look for a method declaration and the corresponding ‘end’ statement. For Python code, it could use the indentation in the code to detect the end of a source unit. For a COBOL paragraph, it could look for a paragraph declaration and the line preceding the next paragraph declaration, or the end of the source file.

11.8.4 | Keep It Simple

There’s little value in getting carried away with charts and graphs, but a visual representation of code health can give interested parties a sense of how things are going at a glance. Here’s how a point-in-time measurement of certain static code analysis rules might be charted as a bar graph.

This is point-in-time, not a trend, so it doesn’t say anything about progress with improving code health. This represents the “health” of a whole codebase, based on four analysis rules.

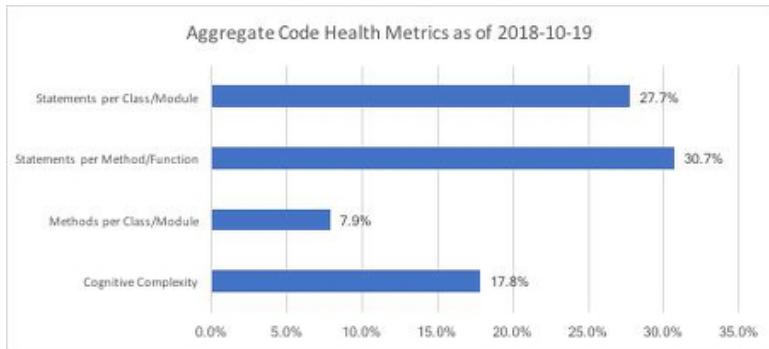


Figure 11.1: Aggregate code health, point in time

From this, we see that across the whole codebase, 27.7% of the code lives in methods that contain code exceeding the threshold we set for “Statements per Class/Module” in SonarQube, and the results for three other rules, as well.

In case it isn’t already clear, the 27.7% doesn’t represent the percentage of source lines that violate the analysis rule; it represents the percentage of source lines that live in source units that contain *some* code that violates the analysis rule. That’s a mouthful. I’m not sure how to say it in a simpler way.

We can just plot the results for a single application, too:

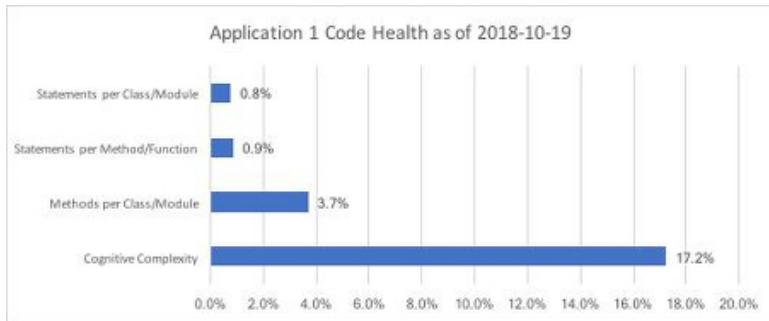


Figure 11.2: Application 1 code health

From this we can see that as of 2018-10-19 Application 1 looks

pretty good, except that about 17% of the code lives in methods that contain code exceeding the threshold for “Cognitive Complexity” that we set in SonarQube.

To get *directional* metrics indicating progress with code health, we plot a trend line based on multiple point-in-time observations like these. It could look something like this:

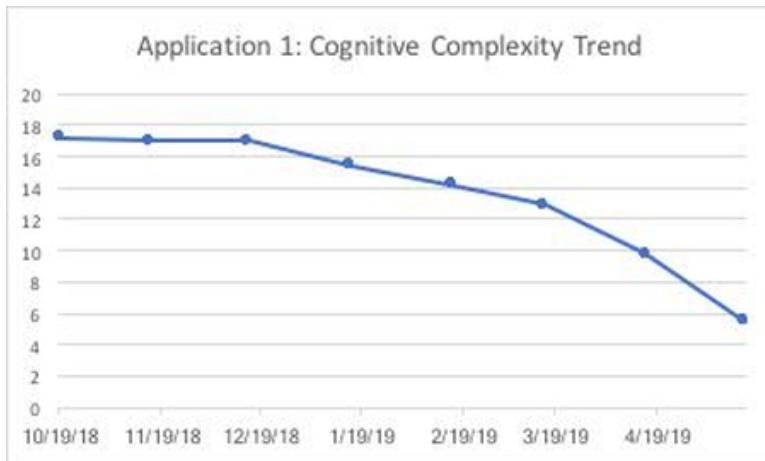


Figure 11.3: Cognitive Complexity trend, Application 1

Here we see the change in Cognitive Complexity for Application 1 over the course of several months. The development team is gradually improving the quality of the codebase.

This way of measurement doesn’t show exactly *how* the team is accomplishing the improvement, and doesn’t point any fingers at individual team members. In reality, it doesn’t matter *how* they’re doing it or *who* is doing *what* within the team. We’re only measuring what matters: The fact the quality of the code base is improving over time.

11.9 | What Not To Measure

I've mentioned that in many transformation programs, people either measure nothing at all or they measure "activity" rather than outcomes. Measuring activity doesn't give you any information about how the transformation is progressing.

Following the rules of a method or framework is fine, if you've determined objectively that the method or framework is helpful in your situation, but it isn't the *goal* of the transformation, and it isn't the *outcome* of your day-to-day operations.

It may be useful to ensure a selected method or framework is being used correctly, but be careful not to assume that's the same thing as "success" in the transformation as such.

Similarly, it isn't meaningful to track technical practices at a detailed level. People can get distracted by this and lose sight of more-important things. It doesn't matter if your team uses pair programming 45% of the time, or has 65% line coverage with unit tests.

Those are neither business outcomes nor prerequisites to stable production operations. Stay focused on the goal; the "beacon" on the hill that lights the way forward. It's likely that sound technical practices will fall into place naturally, as there's no other way to achieve the real goals.

12 | Structures and Responsibilities

The price of greatness is responsibility.

— Winston Churchill

A successful organizational transformation requires a partnership between the leadership of the organization and the external helpers who are engaged to guide the changes. Direct collaboration within a clearly-defined structure, with specific responsibilities on both sides, helps ensure everyone remains engaged and on track to meet the goals of the transformation.

12.1 | Structures Specific to the Transformation

These are temporary organizational structures whose purpose is to guide the transformation as such. They are not part of any product delivery or support operations.

12.1.1 | Guiding Coalition

In keeping with one of the 8 Steps defined by Kotter for effective organizational change, this model calls for a Guiding Coalition to be established with the mandate to steer and monitor the transformation program.

This book describes an approach to organizational transformation that focuses on technical operations to help support the competitive

capabilities the client organization requires to meet its business goals. To that end, the model focuses on operations within the IT area and on the “connecting tissue” between IT and business operations. It is not a comprehensive enterprise transformation strategy that covers other aspects of business operations (such as mergers and acquisitions, taxes, etc.).

To meet this objective, membership in the coalition must include:

- Client leaders who have influence across internal decision-making and operational areas pertinent to gaining business leverage from IT
- Client leaders who can obtain internal funding for expenditures pertinent to the transformation, including those that are discovered after the initial plans have been made (e.g. hardware, software licenses or subscriptions, personnel relocations, facilities changes, etc.)
- Client technical leaders who have the formal power and staff respect necessary to drive progress within the IT area
- Consulting leaders who are responsible for providing guidance to the client to move the transformation forward; at a minimum, the “engagement manager” or equivalent role, and probably including others, such as, for example, a senior technical architecture consultant.

The Guiding Coalition is created at the outset of the transformation program. Its first task is to create its own charter and rules of operation, including the procedures for changing its membership, its charter, and its rules of operation.

12.1.2 | Consulting Team

The consultants, trainers, coaches, and any client personnel who are to be mentored in leadership roles can be most effective if they operate as a cohesive team.

The members of the Guiding Coalition who work for the consultancy guiding the transformation are also members of the Consulting Team and participate in its events.

The benefits of forming a team around this work include:

- Consistent message horizontally through the organization. It's important that client staff hear a consistent message from all their guides throughout the stressful period of change.
- Consistent message vertically through organizational layers. It's important that the transformation program be consistent and coordinated vertically through all levels of management. With management-level consultants and technical consultants working together, it's easier to maintain this consistency and to take corrective action when the messages goes out of sync or are interpreted differently by people with different backgrounds.
- Wide field of view. Opportunities, challenges, and needs can appear in any quarter. When the guides are working together, the entire team can learn of anything that arises in any part of the organization affected by the transformation, and take effective action.
- Mutual assistance. No single individual has all the knowledge and experience necessary to support all the changes that may be necessary. With a cohesive team, individuals with different backgrounds can step in to support any need that arises.
- Mutual support. As noted elsewhere in the book, everyone involved in the transformation will feel stress, experience conflicts, and have moments when they feel ineffective. The team members can support each other to defuse stress, resolve conflicts, and provide mutual encouragement.

12.2 | Consultancy Responsibilities

Organizational transformation consultancies need to take ownership of certain responsibilities in order to provide effective guidance to client organizations.

12.2.1 | Hire and Train Qualified People

The first responsibility of organizational transformation consultancies is to ensure they have qualified people on board, and they provide direction and training as needed to see that the consultants and coaches have *all* the requisite skill sets for the work.

The ability to show people how to perform specific tasks within a narrowly-defined functional area is not sufficient. *Bona fide* coaching skills are also required.

Because of the shortage of qualified technical coaches, consultancies interested in operating in the transformation space may have to consider investing in appropriate training for their own personnel. It is not the client's responsibility to train the consultants.

12.2.2 | Keep Consultants and Coaches Aligned With Goals

While a transformation program is in progress, it falls to the consultancy guiding the program to keep consultants and coaches focused on agreed goals and objectives. This includes both the overarching transformation goal and the interim goals defined for each Cycle of Change.

When blocked by organizational constraints, well-intentioned consultants and coaches often look for any and all forms of improvement they can promote, circumventing the organizational blockers.

On the ground, this feels like “doing the right thing” or “doing whatever is possible under the circumstances.” But when the next Guiding Coalition Workshop comes up (see “Monitoring Progress”), client leadership only sees that the objectives that were agreed in the previous workshop have not been met.

A common impact on consultancies is that they agree to return or waive a portion of the consulting fees for the period of time in question. If this happens more than very occasionally, it can have a crippling effect on the consultancy’s business, not to mention spoiling the client relationship.

A common impact on the consultants and coaches is that they feel their own employer has blind-sided them with criticism after they did the best work anyone could be expected to do within the prevailing constraints.

But there is rarely any progress when this occurs. Whatever organizational constraints prevented the consultants and coaches from achieving the interim goals remain in place. No one has taken any action to remove the blockers. For practical purposes, it is as if no work was done at all.

Another potential downside is that the opportunistic improvements the coaches were able to promote are not targeted to the transformation goal. *Something* about the organization may indeed be better, for *some* definition of “better,” but no real progress toward the transformation goal was made.

12.2.3 | Coordinate and Monitor the Transformation Program

The burden of coordinating the work and monitoring progress falls mainly on the consultancies. However, the *direction* or *leadership* of the program is the responsibility of client leadership.

Principal consultants participate in Guiding Coalition meetings to

share information about progress and issues, to adjust expectations as necessary in light of lessons learned along the way, to assess the results of experiments, and to plan each new round of experiments that involve business areas.

They participate in Consulting Team meetings to keep the transformation moving forward and to assess results and plan new rounds of experiments within the technical area.

They must ensure client leaders are available for consultation in between the scheduled meetings when necessary.

They must collaborate with client leaders to identify appropriate metrics and then ensure their consultants measure results regularly, so that any progress or setbacks are objectively quantified.

That final point is especially important in view of the high level of emotion connected with any change initiative, even in the best cases. If there is a *perceived* problem, but the objective measurements everyone agreed to suggest things are on course, then things are on course. The consultancy must drive agreement with client leadership at the outset of the program that the metrics will officially resolve any such issues that may arise.

12.2.4 | Handle Stress-Related Issues Among Consultants

The consultants and coaches are subject to the effects of stress just as client staff are.

In addition to collaborating with client leadership to recover smoothly from any incidents, the consultancy has the responsibility to work with the consultants or coaches involved in incidents to help them learn and progress from the experience.

It is not appropriate to punish or fire consultants or coaches who occasionally exhibit a human stress response. Only when there is chronic unprofessional behavior or an incident that violates

professional standards of conduct (such as sexual harassment) should punitive measures be considered.

12.2.5 | Play Their Part in Joint Responsibilities

Some responsibilities are jointly owned by the consultancy and the client. The consultancy is responsible for participating in activities pursuant to those responsibilities, and for fulfilling their part in them.

12.3 | Client Responsibilities

Client responsibilities for the success of the transformation program encompass anything and everything pertinent to achieving the goals that the consultancy cannot address. Generally, this means anything internal to the organization.

Client leadership must also direct or lead the transformation program holistically. Otherwise the consultants involved may begin to pursue their own agendas. See “Client-Consultant Relations,” for more on this topic.

12.3.1 | Align Staff With the Transformation Program

Client leadership has the responsibility to ensure their own staff are informed about the goals, structure, expectations, and current state of the transformation program at all times.

12.3.2 | Manage Staff Expectations

Everyone in the client organization will be nervous about what the future holds for them. Client leadership owns the responsibility to help staff members understand their professional options going forward, and how those will be affected by the changes.

It is possible client leadership themselves may not fully understand the implications of the transformation for staff members in various roles. Their guides can advise them, but the responsibility still resides with client leadership.

12.3.3 | Provide Safe Space for Change

Client leadership must ensure there is psychological and political safety for their staff members to express concerns, raise issues, ask questions, conduct experiments, and consume learning-curve time to work toward the transformation goals.

They must also ensure that others in the organization who have a stake in IT operations understand that an effort is underway to make changes for the better, and there may be a slow-down in delivery rate in the short-term as the organization makes the necessary adjustments.

12.3.4 | Provide Supplemental Training as Appropriate

From time to time a skills gap may be identified among client staff. Client leadership is responsible for providing any supplemental training necessary to close that gap, whether directly through their transformation guides or through an external training provider.

12.3.5 | Provide Appropriate Physical Work Spaces

Certain categories of work call for a collaborative, team-centric approach. Team-based collaborative work is best carried out in physical work spaces designed for the purpose.

Note that these do *not* include (a) individual cubicles or offices, or (b) an open-plan office design.

When it is necessary to rearrange furniture or obtain new furniture to establish collaborative work spaces, client leadership owns the responsibility for getting it done.

See “Collaborative Work Spaces,” and DeMarco and Lister in *References* for more information.

12.3.6 | Obtain Necessary Resources

From time to time the need for additional software licenses or subscriptions, hardware or cloud resources, or other resources may be identified in the course of the transformation program. Client leadership own the responsibility for obtaining any such resources necessary to achieve the transformation goals.

12.3.7 | Be Available and Participate In Joint Activities

Client leadership must not be “too busy” to participate in the scheduled Guiding Coalition Workshops, or cause excessive waiting time when the consultants need to speak with them about an issue. The transformation must be treated as an appropriately high priority among leaders’ other duties.

12.3.8 | Manage Back-Channel Chatter

As staff members experience stress from change or experience friction with consultants and coaches, it's common for them to go outside of official communication channels to voice their frustration. In many transformation programs, this results in "back-channel chatter" that undermines or completely scuttles the transformation program.

Client leadership is responsible for understanding the source of any such chatter, and distinguishing between harmless expressions of personal frustration and *bona fide* issues that may affect the transformation goals.

This sort of chatter is mostly noise, but may also contain some signal. The signal, if present, may be significant as it does not appear to be something that could be handled through normal channels.

12.3.9 | Handle Stress-Related Issues Among Staff

Client staff will be under continuous stress throughout the transformation program, even if they understand the outcome of the program will be positive.

In addition to collaborating with the transformation guides to recover smoothly from any incidents, client leadership has the responsibility to work with their own staff members involved in incidents to help them learn and progress from the experience.

It is not appropriate to punish or fire people who occasionally exhibit a human stress response. Only when there is chronic unprofessional behavior or an incident that violates professional standards of conduct (such as sexual harassment) should punitive measures be considered.

12.3.10 | Play Their Part in Joint Responsibilities

Some responsibilities are jointly owned by the consultancy and the client. Client leadership is responsible for participating in activities pursuant to those responsibilities, and for fulfilling their part in them.

12.4 | Joint Responsibilities

These are the general areas of responsibility that are jointly owned by client leadership and their transformation guides.

12.4.1 | Guiding Coalition Workshops

On a regular cadence to be decided by the Guiding Coalition, a series of workshops will be conducted throughout the transformation program for purposes of

- Assessing the results of the latest experiments (Cycle of Change)
- Determining whether a change in direction or objectives is appropriate
- Deciding on next steps; typically, the next round of experiments (Cycle of Change)
- Incorporating any lessons learned since the last workshop into the general operating procedures for the transformation itself (double-loop learning)
- Agreeing on what should be communicated to client staff to keep them apprised of the progress of the transformation program
- Making any changes to its own membership, charter, or rules of operation

12.4.2 | Communicating With Client Staff

Any announcements or presentations concerning the transformation program are to be given jointly by client leadership and the consultancy guiding them in the transformation.

There is a positive purpose in this: To create and maintain the impression of a “unified front” so that staff will feel confident in what is happening.

There is also what may be called a “defensive” purpose: Staff who don’t want the transformation to move forward will know they cannot drive a wedge between their own leadership and their guides.

12.4.3 | Turning Conflicts Into Learning Opportunities

Friction between client staff and consultants/coaches is inevitable. It’s likely to be the norm rather than the exception throughout the transformation program.

The question is not how to *avoid* conflict, and still less how to *punish* people in conflict, but rather how to turn conflict into opportunity.

The responsibility for doing so belongs jointly to the consultancy guiding the transformation and client leadership. A consistent approach and a unified voice are crucial to prevent minor issues from ballooning into show-stoppers.

Unity and consistency will also demonstrate to staff that conflict is nothing to be feared, and that it often leads to the discovery of additional opportunities for improvement or more-effective ways to achieve transformation goals.

12.5 | Consultant and Coach Responsibilities

Those who carry out transformation work at ground level have responsibilities to themselves, to their colleagues, and to client personnel.

- Internalize and apply a high standard of professionalism.
- Be worthy of the trust and respect of client personnel, colleagues, and yourself.
- Understand and mindfully practice the full range of skills necessary in your work, including *coaching, mentoring, teaching, and facilitating* as defined in the Coaching Competency Model, as well as any technical skills relevant to the kinds of consulting, teaching, and coaching you perform.
- Participate openly on the Consulting Team by showing up, being engaged, giving and receiving feedback, offering help, asking for help, and being open to self-improvement through your collaboration with others.

12.6 | Special Consideration for Technical Coaches

Technical coaches have an additional responsibility beyond those of other consultants. It has to do with their awareness of the context in which their services have been engaged.

Many, if not most technical coaches work as independent contractors. They are advanced practitioners of technical skills in one or more areas of specialization, and they are highly confident in their ability to solve problems and help other people learn technical skills.

When they are engaged independently at the team level, technical coaches use their own personal methods of coaching and they emphasize whichever technical practices they personally prefer. They interact with client personnel in their own individual way.

They tend to think of themselves as consultants and as experts whose personal preferences represent the true state-of-the-art for coaching, mentoring, training, and application of technical skills. As a group, they might be described as “highly self-confident.”

All of that is perfectly fine, in my opinion. But technical coaches may also be engaged in other contexts besides individual contracts at the team level:

- as a team-level coach working with a team of other team-level coaches to provide technical consulting, coaching, and/or training to a client;
- as a team-level coach providing the technical side of a broader organizational transformation program (the main subject of this book);
- as a team-level coach providing technical guidance pursuant to an “adoption” or “implementation” of a defined framework; or
- as a management-level consultant providing high-level technical guidance regarding matters such as enterprise architecture, recommendations for tools, and legacy remediation at the architectural level.

The general industry problem I’m addressing in this section is the tendency for technical coaches to behave *in all contexts* the same way as they do when they are engaged individually at the team level.

The issue is that when they are *not* operating strictly as independent individuals, they have a professional obligation to function seamlessly with the others who are involved in the engagement. They may have to “back off” from their personal preferences.

When operating as a member of a coaching team that has been engaged by a mid-level IT manager to help a group of teams, and *not* by executive leadership to transform the whole organization, a technical coach has the responsibility to reach a consensus with the other coaches regarding terminology, approach, and recommendations they will use with the client. They also must limit their activities to the scope within which they have been engaged, and refrain from trying to behave as if they were management consultants.

When operating as part of a broader organizational transformation program that's mainly driven by management consultants, the technical coach must use terminology, methods, metrics, tools, and practices that are consistent with the management consultant's model.

When different consultants say different things or express disagreement with one another, client personnel can become worried about the quality of the advice they are receiving. Given they are already under stress due to the transformation itself, this sort of friction can lead to loss of work, and broken relationships between the technical coach and the management consultancy.

When supporting an “adoption” or “implementation” program, technical coaches must support the chosen framework or model when working with client personnel, even if they personally don’t especially like that particular framework or model. Making disparaging comments about the framework will give mixed messages to client personnel and cause friction.

When engaged to provide management-level advice about technical strategy, architecture, and tooling, the technical coach has to refrain from going to a hands-on level, and must remember to use language, attire, and manners appropriate to the level of client management with whom they must interact.

I think of this as context-awareness. Consultants in non-technical roles may not have to think about this as explicitly as technical

coaches, as they generally interact with clients in the same way regardless of context. Technical coaches don't have that luxury.

Part 3 | Humanity

Man is the only creature who refuses to be what he is.

— Albert Camus

It seems to me that in nearly all organizational transformation programs, not to mention quite a bit of day-to-day management under normal conditions, the fundamental humanity of the people involved is either ignored, denied, suppressed, or punished. As all organizations comprise humans, it might be more effective to acknowledge, seek to understand, accept, and work with people as they are.

This section covers several aspects of the subject, including:

- a chair is a “resource;” a human is not
- Toyota’s “respect for humanity” - what does it mean?
- psychological and political safety
- responsibility over accountability
- the effects of stress
- introversion and extraversion
- cognitive biases
- ego development
- attitudes toward formal authority
- the perils of personality profiling
- organizational culture
- consultant-client relations.

13 | A Chair Is a Resource

The primary purpose of human resources is to protect the company at the expense of the employees.

— Steven Magee

As obvious as it may appear, it seems to be news to many that people are not resources. In my 2015 book on *Software Development Metrics*, I observe:

[Treating humans as resources] may be the mother of all management anti-patterns. Management science has treated human beings as interchangeable machine parts at least since the time of Frederick Taylor’s “scientific management” in the early twentieth century, and possibly much longer than that. Even today, many managers loosely refer to workers as “resources” without realizing the implications of the word.

A *resource* is an asset whose performance can be calculated and predicted with a high degree of accuracy and precision. For example, as I write this, I’m sitting in a chair. Should the chair break, I can sit in another chair. The new chair will immediately function equally as well as the old one did before it broke. The chair requires no training before it can carry out its function. It has no mood swings and never gets tired, sick, or hungry. It doesn’t take vacations or need to pick up its ottoman from furniture daycare. The chair doesn’t worry about other chairs from the same furniture factory that may be going through a rough patch. The new chair doesn’t have a different personal style of chairness than the

old chair. It doesn't interact differently with the other chairs in the room than the old chair did. It's easy to calculate the number of chairs necessary to seat 10, 100, or 1,000 people. The chair is a *resource*.

I observe that most of the problems companies are experiencing, and that inspire them to seek outside help for correction and improvement, stem from treating organizations as machines and humans as their parts. Similarly, I observe that most of the challenges and poor outcomes from organizational transformation initiatives stem from the very same mentality.

Transformation is difficult because the people involved feel stress and worry about how the changes will affect them. Yet, nearly all transformations are carried out in a mechanical fashion - milestones and timelines are defined, and people are expected to undertake a series of defined steps to achieve them, as if programmed like computers.

14 | Holding Precious What It Is To Be Human

All sentient beings should have at least one right—the right not to be treated as property

— Gary L. Francione

The Toyota Production System (TPS) is quite well-known in the Western world. It has two pillars: Respect for People, and Continuous Improvement. Of the two, Continuous Improvement gets by far the most attention in Western organizations and improvement programs.

Many Westerners assume they understand what “respect for people” means - it’s about being outwardly polite, maintaining a false smile, not raising one’s voice, avoiding naughty words, saying “please” and “thank you,” etc.

As this is so obvious, people don’t bother to study it. They focus instead on the more mechanical aspects of TPS.

In the TPS philosophy, Respect For People has two components:

- Respect: We respect others, make every effort to understand each other, take responsibility and do our best to build mutual trust.
- Teamwork: We stimulate personal and professional growth, share the opportunities of development and maximize individual and team performance.

Respect for People is usually translated as Respect for Humanity. In a 2008 article on the Gemba Academy site, Lean expert and Japanese-English translator Jon Miller clarifies:

The phrase 人間尊重 is not rare within the CSR (corporate social responsibility) statements of major Japanese corporations. The word 人間 means “human”, “humans” or “people” and 尊重 can be translated as “respect.” But the phrase used at Toyota is a bit different. It is 人間性尊重. The observant reader or student of Asian languages may recognize the extra character making “human” or “people” into “humanity” or “humanness.”

Humanness is an interesting English coinage, as it seems to imply *the defining characteristics of humans*, as opposed to *humanity*, which suggests, simply, *the members of the set, Human*.

Miller further clarifies:

So our current understanding of “respect for people” must be broader than simply respecting the rights of every person [...] To be wordy, the literal meaning of Toyota’s phrase 人間性尊重 is “holding precious what it is to be human” [...] “respect for people” in my view is pithy but does not convey the full weight of these words in the original language.

Putting the phrase 人間性尊重 into Google Translate (as of April 15, 2019), the result is “Respect for Humanity.” So, the full weight of the words isn’t apparent unless you look a little deeper. It’s worth a look.

The value for an organizational transformation lies in understanding what the defining characteristics of humans are and taking steps to create the conditions in which humans can thrive. It is the diametric opposite of treating people as “resources.”

15 | Safety

In one study investigating employee experiences with speaking up, 85% of respondents reported at least one occasion when they felt unable to raise a concern with their bosses, even though they believed the issue was important.

— Amy C. Edmondson

In any learning organization, people need *safety*. In context, that means psychological and political safety to speak out about issues, to try experiments, and to make mistakes.

15.1 | The Business Value of Safety

In any organization that wants to be responsive and adaptable, safety is a prerequisite. The “mechanical” reason is that adaptability and responsiveness require decision-making authority to be pushed “down and out” throughout the organization. (Hope, Bognes)

A hierarchical decision-making structure in which every decision must be made (or at least vetted) by a “higher authority” can’t quickly adapt to the unexpected, change course to take advantage of emergent opportunities, or innovate to shape new demand.

In most organizations, anyone who makes a mistake or who attempts to apply a new idea that isn’t immediately successful will suffer consequences that may range from being marginalized and denied future career opportunities to just being fired outright. If the transformation goals include dynamic interaction with the market, then a work environment that values the humanity of the people working there is essential.

15.2 | Putting People First

Few consultancies that offer organizational transformation services place a high value on this sort of safety. As I write this, I'm aware of only two. Neither of them has the resources or capability to guide a comprehensive organizational transformation program. Each has its particular area of specialization within which they are excellent, but neither covers all the bases for organizational transformation. (It may be worth mentioning that I don't think *any* consultancy covers all the bases. Each covers different bases.)

I want to avoid mentioning specific companies in this book except when it helps illustrate a point, because companies change over time and what was true at the time of writing may not be true at the time of reading. I'll make an exception in this case and mention Industrial Logic, a California-based consultancy that operates in the "agile" space.

This is one of the very few consultancies that places humans at the center of everything, and they have quite a unique approach. A slogan of theirs is "Make people awesome," and it refers to customers, client teams, and their own consultants. Their consultants have the job title, *anzeneer*. It's a coinage based on the Japanese word 安全 (*anzen*) and the English word *engineer*. They are engineers of safety (Kerievsky).

15.3 | Importance of Safety in Times of Change

I've mentioned several times that stress is a natural effect of change, even when everyone agrees that the changes are necessary and will be beneficial. Under normal operating conditions, most people in most organizations already feel they aren't "safe" in the

sense we're discussing here. When we add the stress induced by a transformation program, they feel even more exposed than usual.

A good organizational transformation program must recognize, acknowledge, and manage this aspect of change. Purely "mechanical" changes in organizational structure, processes, tooling, and practices will have limited impact if the people are fearful of speaking up, trying things, or making mistakes.

With the approach suggested in this book, based on repetitions of the Cycle of Change, experimentation is key to the entire program. It's in the nature of experiments to reveal *better questions* rather than *final answers*. Wrong turns, blind alleys, and errors are normal, expected, and a natural part of the improvement process. Safety is fundamental.

16 | Responsibility Over Accountability

To say you have no choice is to relieve yourself of responsibility.

— Patrick Ness (*Monsters of Men*)

The definition of Respect for People in the Toyota Production System includes the phrase, “we...take responsibility.” What does it mean to take responsibility?

16.1 | Accountability

In the “agile” community, it’s common to hear that team members must *hold each other accountable*. People have had extended, circular debates about the words *accountability* and *responsibility*. Some say they are equivalent. Others say *accountability* just means “the ability to give an accounting of something,” in the sense of being able to explain how or why something occurred. In that sense, it’s a pretty neutral word.

But many people have had negative work-related experiences in which the word *accountability* meant they were to be blamed and punished for outcomes over which they may have had very little influence. The word carries emotional baggage.

When we add the word *hold*, then the emotional baggage grows heavier. *Hold accountable* means that someone else is going to blame and punish you, against your will and without recourse, for outcomes over which you may have had very little control.

That isn't what the dictionary says, of course. But the dictionary doesn't have the final word on words. The impact of words on people is often much more important than the formal definitions.

Strangely, many consultants who claim to be human-focused in their work adamantly refuse to consider the emotional weight of words, and insist on using words that people interpret negatively. Then they have to try and recover from the damage. In my view, this wastes time and introduces unnecessary barriers to a successful transformation.

16.2 | Responsibility

Responsibility has a more positive, meaningful, and actionable definition than *accountability*. An interesting aspect of *responsibility* is that it completely encompasses the standard notion of *accountability* and more besides, without the connotation of blame and punishment.

I accept the definition of responsibility offered by Christopher Avery in his *Responsibility Process*.

The basic idea is that whenever something happens that isn't immediately positive, people naturally go through an internal process to reconcile themselves with the outcome. This process may take split seconds or a lifetime, depending on the situation and on the person.

This internal process also kicks in when people feel stressed about change. No harm has come to them *yet*, but they worry that it will.

In any case, here's the process:

The first impulse is *denial*. It's ignoring the issue, or acting as if you don't see it. This isn't a "wrong" or "bad" thing (unless you get stuck at that stage), it's just a natural impulse.

The next step is to *lay blame*. You identify someone else or something else that is to blame for the problem. It could be anyone or anything that is not you. Like *denial*, this is a normal part of the process and not a problem (unless you get stuck there).

The next step is to *justify*. It means thinking of reasons or excuses for the problem. For change agents, this is an important stage of the process. Most of the mental barriers people erect to avoid change manifest as justification for the *status quo*, or at least rationalization for why the recommended change “can’t work here.” But it’s also a normal part of the process and not a problem (unless you get stuck there).

Beyond justification comes *shame*. This is taking the blame onto yourself, but not in a particularly constructive way. On his site, Avery defines this in brief as “Laying blame onto oneself (often felt as guilt)”.

According to an article by Dr. Joseph Burgo in *Psychology Today*, “The Difference Between Guilt and Shame,” the words *shame* and *guilt* aren’t quite the same. He writes:

Guilt and shame sometimes go hand in hand; the same action may give rise to feelings of both shame and guilt, where the former reflects how we feel about ourselves and the latter involves an awareness that our actions have injured someone else. In other words, shame relates to self, guilt to others.

At this point there is a branch in Avery’s model. The person may choose to bail out of the situation. Avery calls it *quit*. Applying Burgo’s explanation here, a person might quit if they become saddled with shame, and can’t find a way forward. Shame can be paralyzing, so if a person can’t advance from that stage they may have to quit in order to continue functioning at all.

But if they *can* find a way forward, it might be to move to *guilt*;

acknowledging and accepting that they have harmed someone else, rather than fixating on their own feeling of *shame*.

The next stage in Avery's model is *obligation*. Here, the person does what is necessary to try and recover from the mistake, but only because it is what is expected of them; or possibly what they *assume* is expected of them. It's going through the motions, and quite possibly doing "the right thing," but not driven by a genuine sense of *responsibility*.

The final stage is *responsibility*, wherein the person fully owns the action and the consequences and takes appropriate steps to remediate any damage done to others and to set things right.

The relevance to a successful organizational transformation is profound. For that reason, the Responsibility Process is woven into my consulting approach at all levels. I apply all four skill areas defined in the Coaching Competency Model - training, coaching, mentoring, and facilitating - to help people understand and apply the Responsibility Process in their work, and I try to remain conscious of it in my own interactions with client personnel and colleagues. In my view, it should be part of basic training for all coaches.

17 | Stress

To achieve great things, two things are needed: a plan and not quite enough time.

— Leonard Bernstein

Change is stressful, and there are limits to the amount of change a person can absorb as well as the rate of change a person can sustain. Depending on the current and future operational states, there may be significant restructuring of the organization, processes, decision-making, roles and responsibilities, metrics, performance review, technical practices, tooling, applications, and technical infrastructure. Nearly everything in people's work environment is subject to change.

There will be high stress throughout this process. There will be setbacks, emotional outbursts, and many times when people regret having started a transformation. There will be conflicts between coaches and client staff. There will be disagreements among coaches regarding approach or methods. Even when things appear to be progressing smoothly, it's likely that problems are lurking just below the surface. We must set expectations accordingly and be prepared to steer through these issues when they arise.

In a large transformation, we're asking perhaps *thousands* of people to stride boldly from their comfort zones into the unknown. It isn't easy for them. Even if they believe the target operational models will be beneficial, it will be difficult for them to go from here to there.

17.1 | Allow for Stress and Its Effects

My observation is that practically no organizational transformation consultants confront the issue of stress directly and with eyes open. Transformation processes tend to be mechanical, and to assume human factors will fall into place on their own, somehow.

Instead, consultants talk about “resistance to change” and how to overcome it. In my view, there is no such thing as “resistance to change.” People are not resistant to things that benefit them. If you won the lottery, would you decline the prize?

Change in itself produces stress, regardless of whether the outcome of the change is positive or negative. Even if people understand on a conceptual level that the changes are for the better, they will feel stress. It’s a natural human response.

Stress can lead to the *fight or flight* response. As we work through the normal challenges and difficulties of an organizational transformation, stress will reach a level in some people that causes them to balk or rebel at the organizational changes. They may express this by lashing out at the recommended changes as such, at the expected rate of change, at the consultants, their colleagues, or their leadership. They may do so openly or covertly.

People will be alert to anything that confirms their doubts about the transformation. Even under normal conditions, people have a tendency to remember negatives and forget positives.

They will latch onto anything that goes wrong, or any hint that an experiment or a suggestion has not resulted in a flawless outcome. Consultants, coaches, and leaders in the client organization must be aware of this tendency and mindfully manage it. Staff members can’t be expected to have any training in this area, or to be conscious of it.

17.2 | Your Helpers Are Human, Too

The consultants and coaches are just as human as the client teams, and from time to time the stress gets to them, too. They may mis-speak or seem rude, harsh, or impatient. All parties involved in the transformation are human, and it's best to accept that rather than trying to circumvent or suppress it. It's a question of setting realistic expectations and taking appropriate actions to keep the initiative on course when conflicts occur.

It's fair to say that a high level of awareness of this issue, and some degree of skill in self-management, are expected on the part of consultants, coaches, and client leadership and senior management. It's unlikely the majority of staff members will have been trained in this area, and we must set expectations accordingly. Yet, even those whose roles call for skill in managing their own stress are human, and will make mistakes from time to time.

There's an underlying assumption (or *hope*) that the transformation can be carried out without any friction, misunderstandings, disappointments, or elevated emotions on anyone's part. When any sort of conflict occurs, the standard response is to swap out one or more of the consultants or coaches.

The hope is that a coach who is the "right fit" can be found. But there is no "right fit" that will prevent all conflicts and misunderstandings. It is simply unrealistic. With 3 people or 10,000 people affected by the organizational changes, there is no way to say anything at all that won't trigger a stress response in at least one of them.

Of course, we aren't suggesting that people excuse offensive language or socially-unacceptable behavior. If we take Norm Kerth's "Retrospective Prime Directive" as a baseline expectation for behavior...

Regardless of what we discover, we understand and

truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.

...then we can ask everyone to grant a little slack to everyone else.

Swapping out consultants and coaches inhibits the transformation effort. It removes people who have learned the organizational context and culture, and replaces them with people who have to learn those things from scratch. Eventually, they will experience conflicts, as well. Swapping them out is wasted motion at best, and counterproductive at worst.

It's better to accept the reality of stress, resolve conflicts as they occur, and use the incidents as learning opportunities for everyone. Both consultants and client personnel can learn to improve their ability to cope with stress and their ability to resolve conflicts. That's more effective than continually starting from scratch with new people.

17.3 | Hold Course Despite Stress

We must cultivate the self-discipline and maturity to hold our course even through stormy weather and high seas. After all, if organizational transformation were easy, people would "just do it," and they wouldn't need any external helpers.

Guiding client personnel in process-related change is different from guiding technical staff in technical change. The process changes aren't inherently complicated. If anything, contemporary methods are simpler than traditional methods, and thus easier to learn and apply. The barriers tend to be caused by fear that things will go out of control and the organization will fail to deliver. This can usually be mitigated without slowing the transformation drastically.

On the other hand, many of the technical changes are inherently difficult to learn and master. People may need to learn techniques to produce code that doesn't introduce high risk in production, such as Design by Contract or Test-Driven Development, and to include considerations in their designs that they haven't in the past, such as observability, resiliency, or feature toggles. People may need to learn effective testing methods. There may be quite a bit of automation to put in place, and new tools to learn. There may be missing infrastructure. The level of stress introduced to technical staff will be high. We must expect progress to be slower for technical changes.

I find a practical way to mitigate the stress of too much change too quickly is to limit the number of new things technical teams must learn at any given time, and introduce new practices in the context of real work to the extent possible. I take lessons from some of the world's leading technical coaches here. The approach is described in "Coaching Approach."

We must allow time for people to internalize unfamiliar practices and experience the outcomes from those practices, as well as to reflect on the differences in the before and after. Otherwise, once the consultants leave the premises, the organization will revert to the *status quo ante*.

18 | Introversion and Collaboration

In an extroverted society, the difference between an introvert and an extrovert is that an introvert is often unconsciously deemed guilty until proven innocent.

— Criss Jami (*Venus in Arms*)

The concept of *introversion* and *extraversion* gets a lot of play these days. It seems to me people tend to read too much into those labels.

A popular way to avoid working collaboratively is to say, “I’m an introvert.” Often, this is stated flatly, with a tone of finality, as if the word “introvert” completely explains why it is impossible for the individual to collaborate with others.

Obviously, an introvert has to work alone all the time. They are hard-wired that way. It’s innate. It’s an immutable trait. There are no variations or nuances. End of argument. Now I’m going to turn my back on you and put my headphones back on. Go away.

18.1 | What is Introversion?

Introversion is a preference, not a hard-and-fast personal trait. It’s easy to find definitions and descriptions online. Most of them are pretty similar. Here’s an excerpt from the site Personality Max that’s representative of most definitions I’ve found.

Extraversion is characterized by a preference to focus on the world outside the self. Extraverts are energized by social gatherings, parties and group activities.

Introversion is characterized by a preference to focus on the inside world. Introverts are energized by spending time alone or with a small group.

That site has more to say about it, and so do other sources. The information is generally consistent with those two statements.

As an introvert myself, I can confirm the fact that spending a lot of time in a large group feels tiring and, at times, nerve-wracking. That's especially true in an open-ended social setting where there are no explicit boundaries on what people might say.

But here's the thing: A software team is

- not a large group
- not having a party

Collaboration on software-related work takes place in a well-defined context. That's helpful for introverts to cope with the need to interact with other people. The work has several characteristics that mitigate the effects of introversion, including

- conversation topics are bounded; introverts need not fear personal topics will be broached (talking about things that are within the defined context of the work doesn't trigger stress)
- collaborative working sessions are time-limited; introverts need not fear there is no escape from "the group;" they know when it will end, and when they can go away and recharge
- based on the type of work at hand, some tasks are best done by the group, some by subsets of the group, and some by individuals working solo; intense interpersonal interaction is not continuous, and a good deal of interaction is in a small group (as few as two people)
- introversion is not constant. The same individual may exhibit stronger or weaker preferences with respect to introversion and extraversion in different circumstances, or even in similar circumstances when their mood or frame of mind is different.

Context matters, too. You probably know or have seen speakers at conferences and user group meetings who seemed very animated and energetic, and who interacted a lot with participants in their sessions. You might be surprised at how many such speakers are introverts. There's a big difference between making a prepared presentation or facilitating a session about a well-understood topic and having to interact in an open-ended and unbounded way with a bunch of strangers. Those are two very different experiences for an introvert.

The bounded context and time limits of collaborative work methods are quite acceptable for most introverts most of the time. Those things provide a sort of safety zone for introverts. It is absolutely not the same experience as being dropped into the middle of a party where 60 or 70 strangers are asking personal questions (or *might* ask personal questions).

18.2 | Can't Introverts Just Get Over It?

Personal questions that seem normal to extraverts feel like an interrogation to an introvert. I'm talking about questions like "What did you do this weekend?" or "Do you have any children?"

Extraverts often find it hard to accept, when this is explained to them. It doesn't intuitively make sense to them. But it's real.

In a work context, introverts don't have to worry about this. People are going to ask things like, "Do you think we should extract a method here?" or "How can we remove the blocker on Story #345?" Pretty safe stuff.

So-called "icebreakers" can be horrifying experiences for introverts. I recall a time when we were facilitating a training event for a client. There were 5 or 6 of us in the room along with, maybe, 150 or so

client people. To kick off the event, one of my colleagues asked us to stand and take turns telling the group something “interesting” about ourselves.

Had I known he was going to do that, I would have arranged to be outside the room at the time. As my colleagues took their turns, I began to tremble and sweat. When my turn came, I muttered something I can’t remember and sat down as quickly as I could.

If there were other introverts in the group, they were not listening to the interesting stories. They were just waiting for the exercise to end. The sharing of interesting stories didn’t have the intended effect.

Extraverts have no idea this is a real “thing.” There seems to be no way to get them to understand. They say something like, “Oh, that’s nonsense! It’ll be fun! You’ll see!”

Team-building activities have a similar effect on introverts. I remember a time when an enthusiastic new team member threatened to make us attend a baseball game together so that we could “bond” and become “like a family.”

Introverts or not, everyone doesn’t enjoy the same activities. Forcing the issue may well cause the team-building exercise to backfire.

We can build good working relationships within our team quite well enough in the context of work. Nothing builds cohesiveness and trust on a team more effectively than delivering results together.

Extracurricular activities don’t really have the effect extraverts may believe they do. I’ve seen cases when people had a good time together outside of work, and yet team members wanted to get rid of a person who wasn’t contributing well on the job.

18.3 | Making It Work

On a deep level, introverts can't "just get over it" in the way extraverts think we should. And yet, we must adapt. We have to live and function in the world as it is.

We have an obligation to learn to function on a team, both to ourselves as professionals and to our families and anyone else who depends on us. Just saying "I'm an introvert" as an excuse to avoid collaboration doesn't cut it.

Similarly, extraverts need to learn that everyone isn't the same. Their exuberant nature can be off-putting, even *frightening*, to some people.

Team members can craft a collaborative working style that works for everyone. To do so, people first must become aware of the differences and accept that they are real.

18.4 | Introversion and Coaching

For coaches, there's an additional dimension to adaptation. An introvert's style of coaching may work with some people and in some circumstances, while other people or other circumstances may call for an extrovert's style. Coaches aren't "just themselves" when they're trying to influence others to change assumptions and habits. We have to be able to "present" in whatever way is called-for by the situation and the people we're working with. That goes for both introverts and extraverts.

19 | Cognitive Biases

Most misunderstandings in the world could be avoided if people would simply take the time to ask, “What else could this mean?”

— Shannon L. Alder

Cognitive biases affect everyone’s worldview, responses to others, and choices. Even people who have spent decades studying cognitive biases and who actively try to mitigate the effects of bias on their own thinking remain subject to it. Therefore, I consider it a fundamental aspect of human nature.

An organizational transformation program is a high-stress situation. When people are under stress, they tend to revert to their most basic, instinctive behaviors and thought patterns. That means they will be even more vulnerable to their own biases than they are under normal circumstances.

As the transformation program proceeds we will dissect the organization and examine all its component parts and interlocking processes. This has an effect similar to what Alan Cooper describes as *cognitive friction*: “[T]he resistance encountered by a human intellect when it engages with a complex system of rules.” (Cooper)

The implication is that in addition to the normal stress of change, a transformation program introduces two additional human challenges:

- expanding people’s awareness from their previous worldview to think about their organization as a complex adaptive system; and
- being open to new ideas that run contrary to their established habits and beliefs.

19.1 | Categorizing Cognitive Biases

There's quite a long list of cognitive biases. A common way of parsing them is to categorize them as follows:

- Too much information
- Not enough meaning
- Need to act fast
- What should we remember?

If we accept this taxonomy, it seems to reflect a desire to summarize input and reach a conclusion quickly. This may be a result of evolution; our distant ancestors probably needed this trait in order to survive (my speculation).

19.2 | Confirmation Bias

An article in *The New Yorker* from 2017 (Kolbert) summarizes a couple of seminal psychological experiments from the 1970s that deal with cognitive bias, a subject which at the time had not been as thoroughly explored as it is today. It highlights one in particular that is relevant to consulting and coaching: Confirmation Bias.

Confirmation bias is the tendency to accept information consistent with our existing beliefs, and to reject information contrary to those beliefs. The bias operates so automatically that we tend *not even to notice* information contrary to our beliefs.

There's an old saying that we find what we're looking for. People tend to look for confirmation of whatever they already believe. They find it.

The relevance to organizational transformation consulting and coaching is clear: People who by nature tend to reject information

contrary to their existing beliefs, and who are now operating under abnormal stress, will have difficulty accepting the recommendations of consultants and coaches to change the way they work and the way they think about work.

19.3 | Gambler's Fallacy

The Logically Fallacious website describes the Gambler's Fallacy this way:

Reasoning that, in a situation that is pure random chance, the outcome can be affected by previous outcomes.

The canonical example is flipping a coin. When you see heads come up six times in a row, you might expect it to come up heads a seventh time. But the coin has no memory of previous flips. The chances are 50-50 every time.

This can relate to coaching, when team members have had previous experiences with a method or technique or tool that you want to suggest. You'll hear responses like:

- We used that on a previous job, and it was awful.
- We've tried that before, and it didn't work.

Gambler's Fallacy can manifest from *indirect* information, too. You'll hear comments like:

- I've heard bad things about that.
- I read a study that said that doesn't work well.

As a coach you could respond along these lines:

- This is a different situation; the factors that led to problems before aren't in place;
- There's strong management support for it this time;
- The people involved learned from the experience and won't repeat the same mistakes;
- This time you have guidance from coaches to help you.

There is one response that's guaranteed to backfire: "You did it wrong." People don't want to hear that. Think of something else to say...even if they really *did* do it wrong.

19.4 | Sunk Cost Fallacy

Logically Fallacious describes the Sunk Cost Fallacy this way:

Reasoning that further investment is warranted on the fact that the resources already invested will be lost otherwise, not taking into consideration the overall losses involved in the further investment.

The Sunk Cost Fallacy is the common thread in many decisions in IT organizations. The general pattern is that a certain amount of time, effort, and money have been invested in going down the wrong path, or a "right path" whose time has passed, and people are loathe to cut their losses because they worry about having wasted their investment.

- We'll re-write or remediate this application rather than starting fresh, as we've invested so much into it already.
- We've invested so much in [ITIL | Prince2 | SAFe | *name-your-poison*] that we really ought to try and tweak the bits that aren't working, rather than change our approach entirely.

- Let's try and identify near-duplicate Gherkin statements in our collection of 300,000 and consolidate them, rather than leaving the mess alone and writing new statements when new needs arise, as people have already spent so much time creating them.
- Our cloud provider has never kept our applications available per the contract, but after all this time they surely must be on the verge of solving their problems, so we won't move to a different provider.

The cure for Sunk Cost is called “cut your losses,” but this is often a hard sell for consultants.

19.5 | Negativity Bias

Life Coach Tim Brownson identifies *Negativity Bias* as one of four key biases coaches should understand. He writes (Brownson):

As Human Beings we are hardwired to place more emphasis on negative events that happen in our life than we do on positive events with a similar level of standing because they make a greater mental impact.

As client personnel are under stress from the transformation program and worried about what might go badly, they're attuned to anything that seems wrong or questionable or off-target or even just *imperfect*.

They'll be alert to any real or perceived flaws or weaknesses in any team structure, process, technique, or tool we might recommend, as well as to any inconsistencies, poorly-chosen words, or other irregularities in what we say.

It's necessary to recognize this for what it is, and to defuse situations rather than getting into debates or arguments.

19.6 | The Illusion of Explanatory Depth

I don't know whether this is officially on the list of cognitive biases as such, but it's relevant to our work.

Psychologists Leonid Rozenblit and Frank Keil reported the results of two-phase studies that suggested “[m]ost people feel they understand the world with far greater detail, coherence, and depth than they really do.” (Waytz)

Waytz writes:

In a first phase, they asked participants to rate how well they understood artifacts such as a sewing machine, crossbow, or cell phone. In a second phase, they asked participants to write a detailed explanation of how each artifact works, and afterwards asked them re-rate how well they understand each one. Study after study showed that ratings of self-knowledge dropped dramatically from phase one to phase two, after participants were faced with their inability to explain how the artifact in question operates.

As change agents, we frequently encounter responses along the lines of “That sounds fine in theory, but it will never work here,” or “We already have a way of doing X that’s working fine.”

The temptation is to explore the reasons why “it” will never work here, or exactly how X is “working fine.” This approach is ill-advised in most cases, as it will only trigger people’s defenses. It isn’t necessary to “explore” current processes very deeply before we reach the end of people’s true understanding of those processes. If we make people feel stupid or poorly-informed about their own processes, they will shut down and we will get nowhere with our efforts to effect change.

Those who think new ideas “won’t work here” probably think so because those ideas *aren’t* working here. It may be because the ideas haven’t been *attempted* here before, but pointing that out won’t help people get around their biases.

People who think X is “working fine” probably think so because they aren’t aware of any other ways of achieving the outcomes X provides. We have to show them alternatives in the context of their everyday work, little by little as the transformation progresses. We have to be sensitive to the way they are responding to our advice, and adjust accordingly.

20 | Ego Development

“Not everything is about you,” Clary said furiously.

“Possibly,” Jace said, “but you do have to admit that the majority of things are.”

— Cassandra Clare (*City of Glass*)

Quite often, consultants and coaches are puzzled by the way in which client personnel respond to them. The same words, approach, and presentation may elicit very different reactions from different individuals in the client organization. An understanding of how the ego develops may help us understand and relate to clients more effectively.

Ego is a personal area of development, so it applies to individuals in the client organization rather than broadly as a “cultural” phenomenon. It can be helpful to understand this with respect to individual coachees.

Jane Loevinger arrived at a model of human ego development that posits nine stages or levels (Loevinger). A tenth level was suggested later by researchers in Germany (Binder).

As consultants and coaches, we need not apply this model in the same way or with the same depth as a clinical psychologist might do. It’s sufficient to use it in an informal way to help us gain empathy for our clients, and to choose a path for guiding them toward the organization’s goals for the transformation program.

In a nutshell, the levels are:

- Pre-social (E1) – this is the level of ego development in an infant; it does not occur in mature adults
- Impulsive (E2) – acts on impulse, no sense of responsibility

- Self-protective (E3) – self-centered, all problems are caused by others
- Conformist (E4) – follows rules, has binary “right/wrong” worldview
- Self-aware (E5) – interest in interpersonal relations, ability to see alternatives
- Conscientious (E6) – nascent sense of personal responsibility with recognition of guilt as opposed to blame
- Individualistic (E7) – respect for self and others, tolerance of differences
- Autonomous (E8) – ability to synthesize and integrate ideas, desire for self-fulfillment
- Integrated (E9) – ability to reconcile internal conflicts, pursuit of self-actualization, acceptance of natural limitations
- Ich-Entwicklung (E10) – focus of life is self-development and transcendence; probably not useful in a work environment.

Levels E1 through E9 are part of the basic model and E10 is an extension of the model. An adult will not exhibit E1 ego development. Adults who exhibit E2 ego development would most likely be diagnosed with some sort of sociopathic or psychopathic disorder. Thus, these levels are not relevant to organizational transformation programs.

E3 is common among managers in industrial-era organizations, and E4 is common among non-management personnel in such organizations.

E5 through E7 are quite common in professional organizations such as those in which we coach teams and individuals in most of the industrialized world. My observation is that the Nordic countries are at a higher level of social development than the rest of the world, so we find more E7 and E8 individuals there.

People who exhibit E9 ego development are rare. Those at E10 are unlikely to regard conventional employment as a meaningful

use of their time, and so we tend not to encounter them in IT organizations.

Bear in mind this is a loose, amateurish, and subjective use of the model on my part. It's only a shorthand way for me to remember how each individual is likely to respond to interactions.

To put it into context with respect to Avery's Responsibility Process, I would say that a person below the E6 level would be unable to apply Avery's model. A person at the E5 level would probably be able to understand it on a theoretical level but would have difficulty in applying it.

21 | View of Authority

Any fool can make a rule
And any fool will mind it.
— Henry David Thoreau (*Journal #14*)

Individuals' relationship with formal authority and management structures can affect their response to any recommended changes consultants might make to advance the goals of the transformation program. One model for understanding this is called Spiral Dynamics (Spiral Dynamics).

This is a branded, commercial model supported by an organization that provides training, certification, and consulting services. My intent here is not to go into it in that degree of depth, but rather to use it loosely as a way to gain a general idea of how people relate to formal authority.

Here's how I think this model maps to personalities commonly encountered in coaching.

- Beige – Archaic-instinctive (unlikely to occur in a professional setting)
- Purple – Animistic-tribalistic (unlikely to occur in a professional setting)
- Red – Egocentric-exploitive (industrial era non-management)
- Blue – Absolutistic-obedience (industrial era management)
- Orange – Multiplistic-achieivist (post-industrial self-centered)
- Green – Relativistic-personalistic (post-industrial self-centered, advancing)
- Yellow – Systemic-integrative (natural affinity for agile and lean thinking)

- Turquoise – Holistic (rare and likely to be suppressed in a large organization)

I want to be very clear that this is not an attempt to apply Spiral Dynamics properly. It's just a shorthand way to remember how a given individual is likely to respond to various styles of interaction.

Using this model loosely and subjectively, we can say that people at the Beige and Purple levels are unlikely to be able to hold a steady job, so we won't encounter them in coaching work.

Red and Blue are common in traditional (industrial-era) organizations, where their personal traits function as an effective survival mechanism. These individuals will not necessarily be able to adapt to contemporary organizational patterns. As coaches, this can give us guidance in "picking our battles."

Orange and Green are the most common among people who work in traditional organizations and who are keen to be among the first to try something new, with an eye toward improving their organization. Yellow is a very good fit for an agile culture, but unfortunately by the time you get to the Yellow level in this model, you've almost run out of people.

A person at the Turquoise level would not be happy in most conventional organizations, although they might fit in well in a synergistic culture, as defined by Bob Marshall. As a lean/agile coach, you're unlikely to encounter anyone like this in the organizations that require your assistance.

22 | Profiling

As someone who has spent three decades in media, I can tell you the technology around profiling is advancing way faster than our ability to digest its implications, and I urge you to continue asking a lot of questions and not take simple solutions at face value.

— Ken Goldstein

In a well-intentioned effort to deal with the human challenges of change, some consultants (and client leaders) place excessive faith in pseudo-scientific models of human behavior, like the DISC Profile, Personality Colors, Personalysis, Myers-Briggs Type Indicators, and many more. These models are presented in a highly polished form, and are accompanied by well-marketed programs of testing and assessment by “certified” practitioners.

All this gives an aura of objectivity and credibility to models that are really no better than a horoscope, a fortune cookie, or bone-reading (Yronwode).

The risk is that you or your transformation guides will inappropriately label individuals as incapable of collaboration or unable to learn new skills, based on their answers to a few questions that were presented under abnormal conditions such as a job interview or a team assessment.

National Public Radio’s *Hidden Brain* program ran an episode about how these tests are used and how they affect people. It was broadcast on April 19, 2019, and was entitled “What Can Personality Tests Tell Us About Who We Are?”

For the episode, reporters visited a Harry Potter convention and interviewed people regarding the Hogwarts houses into which they had been “sorted.” They generally found it an enjoyable exercise.

In other interviews in different contexts (university and work), some people liked being given a “mold” into which they could shape themselves; they felt a sense of self-knowledge and belonging. It didn’t matter whether the test results were objectively accurate. The fact they had a “place” was the important thing to them.

One person said she felt the Myers-Briggs test had been beneficial to her when she was at university. After a couple of decades of life, she worried that her experiences had changed her. She took the test again, taking care to give answers that would assure the same result as before, so that she would feel she was still the same person she had always been.

That’s a basic problem with these tests, in my experience. You can make the result anything you want it to be by answering accordingly.

Others interviewed on the show had been emotionally and financially damaged by being pigeon-holed based on their test results. That is the risk of taking this sort of thing too far, for our purposes here.

Companies are using tests like these for insidious purposes. An individual’s professional options may be limited for no reason other than management believes their personality type is unsuited to certain types of work. In many cases, companies use these tests to screen job candidates.

In some cases, companies are using test results to fire people who are already employed there, not for lack of performance but because their personality test results don’t conform with management’s preferences. The negative impact on people can include personal distress rising to a level that requires therapy, as well as professional marginalization and loss of income.

And yet, *none* of it is objectively valid or supported by science. Even if there *were* some level of clinical evidence that some of the tests were useful for specific diagnostic or therapeutic purposes

under controlled conditions, Human Resources departments or line managers are certainly not qualified to use them.

The practice of using personality tests in this way is growing, as of the time of writing. More and more people are reporting personal damage and work-related damage resulting from management's reliance on these tests. It will not surprise me to see litigation in this area, possibly followed by legislation to limit the use of such tests or to provide protection to workers.

22.1 | Pigeon-holing

At best, the time spent in conducting and analyzing the tests amounts to useless thrashing; at worst, you may deny your organization the full potential of employees who are pigeon-holed as biologically or temperamentally incapable of personal growth or situational adaptation.

I have seen cases in which mistakes were never discussed with people. Instead, they were steered away from situations their managers believed they couldn't handle well. The intent was positive, but the result was destructive - people were marginalized and denied opportunities to learn, improve, and practice their craft.

When asked why, the managers replied along the lines of, "There's nothing they can do to change. They're hard-wired that way, according to our tests."

That represents a fundamental misunderstanding of human nature. These managers literally believe human beings are incapable of learning and growth. This false belief can have negative effects reaching as far as litigation and psychological or emotional damage.

It is perhaps the ultimate expression of humans-as-resources. People are merely "things" who are prisoners of their personality types. It's a compelling view for managers who find it challenging to deal with the complexities of real humans.

22.2 | Defamation

You may impede a person's career growth for no objective or legally-supportable reason. In extreme cases, the person so harmed may have legal recourse against you.

Harming a person's professional reputation is a form of *defamation*, which is actionable in most countries and, in Finland (and maybe other places), is a criminal offense that carries jail time. In countries where defamation is a civil offense, it's possible for the person who is harmed to collect damages for "future loss of earnings," based on estimates of what they could have earned had they not been denied the opportunity to work or to pursue career growth opportunities in their chosen field.

Given that people have different personalities, conflict is unavoidable and solvable, and personality tests are not based on real science, it seems a rather risky proposition. Why go there?

The real solution is to accept the fact that people are different (and they're *supposed* to be different), learn to resolve conflict, help people learn from mistakes, and set expectations for professional behavior.

See the entries for FindLaw, LegalMatch, and Kelly Warner Law in *References* for more information.

22.3 | Excuses

The downside effects of these tests don't usually rise to the level of legal action, but they can reduce the effectiveness of the transformation program nonetheless.

The test results offer excuses to staff members who would rather not play along with the transformation. You'll get excuses like, "I'm an introvert; introverts can't collaborate."

You can't counter the excuses using the same rationale you used to pigeon-hole people. You painted yourself into a corner by relying on the tests in the first place.

22.4 | Monoculture

Yet another risk is that you will create a monoculture. This outcome isn't a "given," but in practice most people tend to try and use the pseudo-scientific personality tests as a way to filter out the "wrong" people and look for the "ideal" people. The organization may lose the potential benefits of diversity.

People with different work experiences, cognitive styles, problem-solving approaches, communication styles, formal education, age cohorts, sexes, ethnicities, nationalities, neurological characteristics, cultural backgrounds, religions, family situations - even different personal hobbies and interests - tend to form robust and adaptable teams, able to address a wide range of problems both planned and unplanned.

In contrast, a monoculture is characterized by "groupthink," and can be stymied by the simplest of problems, if that problem doesn't happen to align with the group's worldview and assumptions.

You may recall an episode of the television series, *Star Trek*, entitled "The Immunity Syndrome," in which a ship crewed entirely by Vulcans was destroyed and all hands lost. Despite their high intelligence and mental self-discipline, the Vulcans were unable to "think outside the box" to deal with the threat because they were conditioned by their education and socialization to think in the same way. The *Enterprise*, with its crew of people from many different backgrounds, found a solution. It's common to see the same phenomenon in our own universe, as well.

22.5 | The Slippery Slope to a Toxic Culture

An insidious effect of monoculture is that over time it can devolve into a *toxic* culture. The emphasis on thinking, acting, and speaking alike leads to an environment in which people can't express concerns about work-related issues or anything else for fear they will run afoul of "acceptable" cultural norms.

In extreme cases, questioning authority for any reason at all, or disagreeing with colleagues who have learned to play the system to their advantage, can be career suicide.

My advice: Don't fall for pseudo-science. It's a waste of time and money, and can be counterproductive (or worse).

22.6 | Be a Grown-Up

Set the expectation that people must learn how to operate in a collaborative working environment. It's true this will present unique challenges to different individuals, depending on their neurology, personality types, life circumstances, general tendencies, and personal agendas. Some people will find it harder than others.

But that's just *life*, no different from operating in the world beyond the company's walls. The fact people have different personality types doesn't absolve anyone of the professional responsibility to learn how to function in the real world, and how to interact effectively with people who are different from themselves.

We don't have to be trained psychologists. We just have to do our best to empathize and relate to others, while giving them some slack to be themselves without complaining about every little thing they say or do that isn't "perfect" from our point of view.

Was John blunt or rude when he criticized your design idea? Maybe he was having a bad day, or maybe that's just the way he talks. You also have bad days, and your own way of talking. Get over it and move on.

Was Mary disrespecting you when she failed to return your "Good morning?" Maybe she was preoccupied with a problem she was working on at the time, or just didn't hear you. Get over it and move on.

I call it "being a grown-up."

22.7 | People Can Grow (Resources Can't)

A person who tends to dominate others can learn to hold back. A person who tends to avoid conflict can learn to be more assertive. A person who is uneasy about close collaboration can learn to cope. A person who blurts out an inappropriate remark while under stress can learn self-awareness and control.

One of those precious characteristics of humanness is that humans are not prisoners of their base natures or their initial wiring diagrams. They have the capacity to become whomever and whatever they choose.

Let them.

Your organization will be the better for it, no matter the technical results of your formal transformation program.

23 | Organizational Culture

Culture does not change because we desire to change it.
Culture changes when the organization is transformed;
the culture reflects the realities of people working together every day.

— Frances Hesselbein (*The Key to Cultural Transformation*)

Many whose work involves helping organizations change the way they operate believe the best place to start is with the organization's *culture*. In my view, there are some challenges with this approach.

First, there doesn't seem to be clarity about what "organizational culture" means. It's hard to base a change initiative on something that isn't clearly defined.

Second, there doesn't seem to be agreement on whether an organization's culture can be intentionally created (for instance, through a mission statement or declared values), or it emerges organically (as suggested by the similarity between the words *organically* and *organization*).

Third, people use the term *organizational culture* to try and create impressions around their brand, to control behaviors, or to build personal power.

"Culture" is used as a way to create a sort of mystique about certain companies. You've seen and heard people talk about Google's culture or Amazon's culture, as a way to distinguish those companies from others. Yet, there really isn't any magic.

“Culture” isn’t consistent throughout an organization. People who work in one part of a company experience a different culture from those who work in other parts of the company. Is it meaningful to speak of “the” corporate culture?

“Culture” is sometimes used as a passive-aggressive weapon. If you violate “the culture,” you’re a Bad Person. When you push back on someone’s ideas, they may counter with something like: “Our culture is to disagree and then commit to the group’s consensus. By continuing to debate, you’re in conflict with the cultural values you agreed to when you were hired.”

If we aren’t sure what organizational culture is, and we can’t effectively create an organizational culture by design, and people are already abusing the term for various purposes, then how can we *begin* a transformation program by changing an organization’s culture? It doesn’t appear to be firm footing.

23.1 | What Is Organizational Culture?

People talk about organizational culture without defining it, as if the meaning were so obvious that no definition is necessary. They may be right: Each person may have an intuitive understanding of the concept that, from their point of view, seems too obvious to require clarification.

Michael Watkins attempted to find a definition in 2013 by facilitating an online discussion. He summarized the results of the discussion in an article in Harvard Business Review (Watkins).

Participants in the discussion suggested quite a few interpretations of the concept of organizational culture. It’s how organizations do things; or it’s how organizations define themselves; or, it’s the values and rituals of the organization; or, it’s the organization’s

immune system; or, it's a civilization in microcosm; or, it's a subset of the culture of the surrounding society.

An organization's culture can be explicitly defined; or, it emerges from the activities people carry out within the organization; or, it adapts to change like a living organism.

Okay, which is it, then? Or is it something different from any of those suggestions? It seems there isn't a clear and widely-shared understanding of the meaning of organizational culture. It means different things to different people.

23.2 | How Is Organizational Culture Used?

On a practical level, the ways "organizational culture" is used and the effects on people are probably more meaningful than any abstract definition.

In their book, *Nine Lies About Work*, Marcus Buckingham and Ashley Goodall examine the ways in which organizations use the concept of "culture." They cite three main uses for the concept:

First...it tells you who you are at work. If you're at Patagonia, you'd rather be surfing. [...] If you're at Goldman Sachs, then never mind the surfing - you'd rather be winning. [...] It means something to say that you work for Deloitte, or for Apple, or for Chick-fil-A - and this meaning says something about you, something that locates you and differentiates you, that defines your tribe.

Second...culture [is] how we choose to define success.
When Tesla's stock was on the rise in the early part

of 2017, it was [...] because Elon Musk had created a culture of cool, a place where you couldn't even see the cutting edge because it was so far behind you.

Third...culture is now a watchword for where we want our company to go [...] job description of senior corporate leaders has become to create a specific sort of culture, a culture of “performance,” perhaps, or...“inclusion”...or “innovation”...

All of which is fine, right up to the point where you start to wonder what, precisely, you are being held accountable for.

Buckingham and Goodall remind us that when someone asks you what it's like to work at your company, you probably won't pull out the official mission statement or a list of the official corporate values. Instead, you'll just tell them what a day in the life *feels* like.

The authors came up with a list of eight assertions we can all use to check ourselves about how it feels to work where we work. Four are “me” assertions, and four are “we” assertions.

I invite you to read their book for the details around this, but here's a sample of a “me” assertion, to give you a sense of it:

I have the chance to use my strengths every day at work.

And here's a “we” assertion:

My teammates have my back.

You can see this is a far cry from any high-altitude corporate “culture” statement. It's very down to earth. Also, the statements

are not abstract or aspirational; they're about how it actually feels to work there, today, right now, in real life.

If this is organizational culture in the raw - how we actually feel on a day to day basis - then it's got nothing to do with surfing.

It's also pretty clear that we can't make any of these things happen just by saying so. Is it really possible to *begin* a transformation program by dictating a new culture? Wouldn't that prove to be just another high-altitude aspirational statement?

We could declare, "In the new organization, you have the chance to use your strengths every day at work!" But if the structures, processes, practices, and tools necessary to enable you to use your strengths are not in place, will that aspiration be realized?

23.3 | A Systems View of Organizational Culture

Ralph Stacey, who has written extensively on the subject, concludes that organizations of humans qualify as complex adaptive systems, as they comprise independent agents

- who interact to create an ecosystem;
- whose interaction is defined by the exchange of information;
- whose individual actions are based on a system of rules;
- who self-organize in nonlinear ways to produce emergent results;
- who exhibit characteristics of both order and chaos; and
- who evolve over time.

Jim Highsmith based Adaptive Software Development on Stacey's work, among other things. See Highsmith and Stacey (separately) in *References* for more information.

From this perspective, it's hard to see how we could achieve a transformation by establishing a "target culture" for the organization, and then waiting for that culture to generate the required structures, processes, and practices to meet business goals.

If the "culture," or "how it feels to work there," is an *emergent property* of a *complex adaptive system*, then not only can we not *craft* a culture by design, but we would never *find* the existing culture by disassembling the organization into its constituent parts, in exactly the same sense that we can't find an organ in the human body that contains the soul.

23.4 | Stop Worrying About Culture

My conclusion is that "culture change" isn't a practical starting point for an organizational transformation program. If it occurs at all, culture change will emerge naturally as a result of the way things happen in the organization. Even then, it will never be consistent across the whole organization, or stable over time.

It seems to me there's little value in worrying about it. I find it more practical to think about people's *habits* and organizational *standards* than about "culture" as such.

Consider the water pipe analogy, as well as the general approach outlined in Part 4, "Change." The greatest impact comes from changing organizational *structure*. Next comes *process*, followed by *practices*.

With appropriate structures in place, effective processes become possible. Effective processes enable people to apply good *practices*. The combination produces positive *outcomes*. The way people *feel* about working in the organization emerges from these factors. It feels good to achieve positive outcomes using good practices with effective processes within a proper structure.

“Culture,” if you want to call it that, is an *effect* of the transformation, not its cause.

24 | Client-Consultant Relations

Indifference and neglect often do much more damage than outright dislike.

— J.K. Rowling (*Harry Potter and the Order of the Phoenix*)

The health of the relationship between clients (leadership and staff) and consultants (company-to-company as well as individual consultants and coaches) is a critical success factor for organizational transformation programs. Broken or damaged relationships probably account for more canceled or abandoned programs than any other single issue.

24.1 | Factors Resulting in Sensitivity

My observation is that several factors contribute to the sensitivity of these relationships:

- the general stress level during a transformation program
- assumptions on all sides that are not validated
- attempting to manage the transformation as stages with milestones
- assessments that don't reflect reality
- lack of metrics or inappropriate metrics
- poor management of friction and conflict
- earning and maintaining trust

Let's take a closer look at each of these.

24.1.1 | Stress-Related Issues

Everyone involved in a transformation program is already on edge owing to the naturally high level of stress induced by change. Changes may be occurring simultaneously on all levels of the organization, and many of these changes are interdependent.

Unless people are cognizant of the effects of stress and take intentional, informed action to manage them, some may eventually find the course of least resistance is just to call the whole thing off.

24.1.2 | Unvalidated Assumptions

There's a tendency for client executives and management consultants to speak a common language. It's a language full of jargon and buzzwords, verbs used as nouns, nouns used as verbs, and sports analogies.

"We'll leverage the spend for the ask and hit a home run!"

But the language isn't as common as you might think. It's common enough to provide a "feel good" connection during contract negotiations. But when you get down to brass tacks, different individuals may have very different concepts about what the words mean or imply.

As a result, the objectives the consultants think they're aiming for may be quite different from the outcomes client leadership expects. Many consulting engagements devolve into a repeating series of apologies, clarifications, and second chances.

There is finite tolerance for this, and it often ends with premature cancellation of the program or takeover by client leadership and the ouster of the consultants.

A phrase like "Respond to Change," which is one of the competitive capabilities in our sample analysis for the hypothetical bank, can be understood in many different ways. Every detail that plays into the

definition of “Respond to Change” and to the judgment of whether the goal has been achieved must be painstakingly clarified.

People are loathe to appear dim-witted, and prefer not to ask the same question multiple times or to ask for clarification of an answer that the price of their business suit suggests they really ought to know already. But it’s better to go ahead and appear dim-witted early on than to have the contract jerked out from under you after you’ve hired consultants and sub-contractors.

The same pattern applies at all levels. Technical coaches and client team members may have very different concepts of what a given term means or implies.

For example, I was working with a team and noticed a *long method* (a method with more than about 20 or 30 lines) in one of their C# source files. I thought it would be a good opportunity to introduce some basic refactorings.

But the person showing me the code insisted the method wasn’t long at all. He scrolled and scrolled and scrolled and scrolled through hundreds of lines of code, and said, “See? It’s not a long method. It’s just fine.”

The method consisted of hundreds of lines of if/else blocks, each one nearly the same as the others. It was wallpaper from coast to coast. We had very different assumptions about how long was too long, different concepts of the rationale for making that determination, and different opinions about whether there was anything wrong with long methods in the first place.

Every single statement, every single idea, at every level in the organization will be understood differently by different people. If we aren’t diligent about ensuring common understanding, we will repeatedly disappoint clients.

24.1.3 | Managing Change as Phases or Stages

In “Anti-Patterns of Organizational Transformation,” I identify a milestone-driven approach as problematic. In addition to the reasons discussed there, a milestone-driven approach also makes misunderstandings and missed expectations more likely than they are with a directional approach.

The reason is that so much time passes in between milestones or checkpoints or assessments. People move forward on the basis of assumptions that may not be quite right, and the more steps they take the farther off course they wander. By the time the next milestone comes up, they may be so far off course that the client wonders what they’ve been paying for all this time.

It’s much easier to maintain healthy relationships if mis-steps are detected right away, and corrected before too much time is spent on the wrong things.

Thus, taking a directional approach to managing the transformation program is a way to avoid broken or damaged relationships, as well as a way to keep the actual work on track toward the goal.

24.1.4 | Assessments That Don’t Reflect Reality

This issue may be a descendant of both *unvalidated assumptions* and *milestone-driven approach*.

In most cases an “assessment” of client progress consists of nothing more than a list of activities or artifacts that are expected as of the particular milestone or checkpoint.

The question of whether these things are still relevant, mean the same things to all stakeholders, are accurately observed, or are performed with any degree of skill doesn’t enter into the assessment.

One solution is to be explicit about exactly what each item means and how it is to be measured (if assessed quantitatively) or judged (if assessed qualitatively), and crystal clear about how the assessors made their determination.

A better solution is to take a *directional* approach rather than a *milestone-driven* approach, and obtain feedback about progress on much shorter intervals than months-long phases or stages.

24.1.5 | Lack of Metrics or Inappropriate Metrics

This point has been mentioned before, but bears repeating. In many transformation programs, nothing is measured. Sometimes people begin measuring at some point; usually after there's been some sort of friction or conflict between clients and consultants about whether any progress is being made. But without the initial measurement of the baseline state, before any changes were made, we really have nothing useful.

This issue can also result from measuring the “wrong” things. Either we’re measuring activity rather than outcomes, or we’re measuring outcomes other than the business goals of the transformation program.

24.1.6 | Poor Management of Friction and Conflict

Another point that’s been mentioned before is that stress, and its children friction and conflict, are entirely normal and must be expected and managed throughout the transformation program. When people assume they can avoid or circumvent friction and conflict, or that individuals are to blame for it rather than the very nature of change itself, then broken or damaged relationships can result.

A good deal of friction and conflict between clients and consultants arises from:

- different expectations or assumptions
- real or perceived “failure” to hit milestones
- absent or inappropriate measurements

These factors should all be pretty familiar by now. Friction and conflict are just more outcomes from them.

Clarifying assumptions and expectations, clearly documenting the criteria for achieving the desired end state, taking an incremental approach rather than a phased approach, measuring the right things, and taking due account of the humanness of everyone involved will alleviate or eliminate issues arising from conflicts.

24.2 | Earning and Maintaining Trust

This section is from the consultant’s point of view.

When a client organization invites outsiders into their “house” to help correct deep-seated problems with basic matters everyone believes they have already mastered, everyone’s defenses will be raised. The consultants have to earn the trust of the client, and then they have to be proactive in maintaining that trust.

Management consultant Dennis Stevens, one of the founders of the management consultancy LeadingAgile, developed a model of influence and trust through years of observation of numerous client engagements. It looks like this:

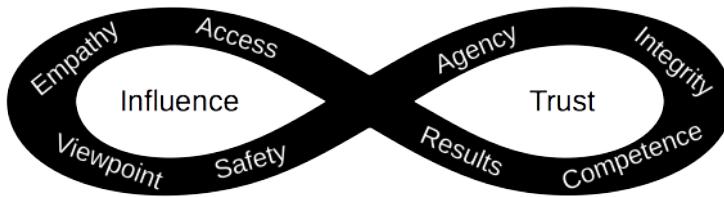


Figure 24.1: Influence and Trust Loop

The general idea is that we begin with *access*. The client lets us into their house for some limited purpose. It's possible the main thing they're looking for at this stage is to find out whether they can trust us enough to work in earnest together.

Building trust begins with *empathy* for the client's point of view, situation, problems, goals, fears, and everything else. When we have a clear understanding of their needs, feelings, and thoughts we can express a credible *viewpoint* about the situation and how to move forward. We also want to instill a sense of *safety*.

At that point, the client grants us *agency* to make changes. It falls to us to do so with *integrity* and *competence*, and to achieve *results*. All of that earns a degree of *trust* from the client.

That level of trust provides greater *access*, which calls upon us to exhibit *empathy* for a wider range of perspectives and people in different roles, to offer a credible *viewpoint* for their problems, and to assure even more people of *safety*. As a result, we gain greater *agency* to effect change. And the loop continues.

If the consultant falls down in any of the eight areas, the result is likely to be a loss of trust, leading to a reduction in influence. The consultant's ability to help the client reach their goals is compromised.

Everything in this section is incumbent on consultants, and isn't a responsibility of client leadership or staff.

24.3 | Who's the Boss?

This section is from the client's point of view.

I've observed that one source of breakdowns in client-consultant relations is that there are *too many bosses*.

Already noted: No single consultancy, however nicely dressed, has the full range of expertise to guide a comprehensive, enterprise-scale organizational transformation program. On a practical level, that means client organizations often engage more than one consultancy.

In turn, each consultancy normally engages independent sub-contractors, as demand for their services is highly variable and it isn't cost-effective to keep a large number of people with rare skills on the bench, waiting for work.

Each of the sub-contractors has a self-concept that they are consultancies, too. They will strive to maintain their independent identities and ideas about organizational change. In fact, this is one of the sources of friction and conflict in transformation programs.

When more than one consultancy is involved, different leadership dynamics can come into play:

- Prime contractor (one of the consultancies) "owns" the program and the direct relationship with the client, and directs the other consultants
- Consultancies jointly engage with the client as peers, and agree on some structured way of collaborating with each other
- Client "owns" the program and directs the consultants

24.3.1 | Prime Contractor Model

The first option (prime contractor model) is compelling, as it's structurally the simplest one to manage, from the client side. There's only one consultancy to deal with, so (in principle) there should be few misunderstandings. There's only one bill to pay, so it's easier administratively.

What I've observed in the field is that the prime contractor seeks to dominate and undermine the other contractors. They want all materials to have their own branding, all consultants and coaches to use their approaches and terminology, and all credit for success.

The typical result is the other consultancies find a way out of the deal as quickly as they can. In the meantime, they absolutely do not surrender their intellectual property to be re-branded by the prime contractor. They absolutely do not abandon their own models and their own terminology.

For the client, it's definitely sub-optimal. In fact, it's unlikely to hold together for the duration of the program.

24.3.2 | Peer Consultant Model

With this arrangement, all the consultancies involved in the program are directly engaged with client leadership. No one of them is deemed "primary," and yet the client is not directly managing them. They are expected to operate collaboratively in the client's interest.

Assuming everyone behaves professionally, rationally, and ethically, in principle this should be the easiest way to go. I suspect there's little doubt about how this works out in practice. With that in mind, let's move on to the third option.

24.3.3 | Client Ownership of the Program

The third open tends to be pretty burdensome on the client organization. They have to dedicate people to keep tabs on what the consultants are doing, set the standards for consistency, and ensure the various consultants are all on message when explaining things to staff.

A second challenge is the reason outsiders were engaged in the first place is client leadership doesn't already know how to manage a transformation program. That places leadership in a Catch-22 position. They must learn enough about transformations to understand what the consultants are up to, and keep them all synchronized.

What I've observed in the field is that client leadership will create an internal group to set standards for the "new" organization. The internal group generally has little or no experience in the new methods, and yet they may be highly enthusiastic because they've been through a little bit of training.

Much of what they define as "standards" will be based on fundamental misunderstandings or be unimplementable in any practical way. These groups tend to become very territorial and defensive, and it's difficult to correct them.

Note this is not the same sort of structure as Kotter's Guiding Coalition as described in "Structures and Responsibilities." A practical Guiding Coalition remains necessary. In this situation, client oversight of the consultants may be implemented through the Guiding Coalition.

Despite these difficulties, this is the only approach that has a fair chance of working at all. So, the bad news is there's no easy "out" for you, if you need outside help to guide a transformation program.

You have to be the boss in no uncertain terms. Furthermore, all the consultancies you engage must be treated as equals. They will try to use your organization as a battleground to gain more business

and more market reputation than their competitors. You can't let them.

It's hard. Sorry.

24.4 | Ensure Common Understanding

Something I've observed in every transformation program I've seen is frequent misunderstandings between client leadership and the consultants who are helping them. It's common to see misunderstandings and misalignment between client leaders and between consultants, as well.

It seems to boil down to the old adage about not making assumptions. In principle, it should be fairly easy to avoid misunderstandings. Yet, they abound in every transformation program.

Even when people use the same words and believe they are talking about the same things, it's very easy for them to come away from a conversation with quite different understandings about what was just said. Both may believe they have reached agreement, and yet each has an entirely different view of what is happening.

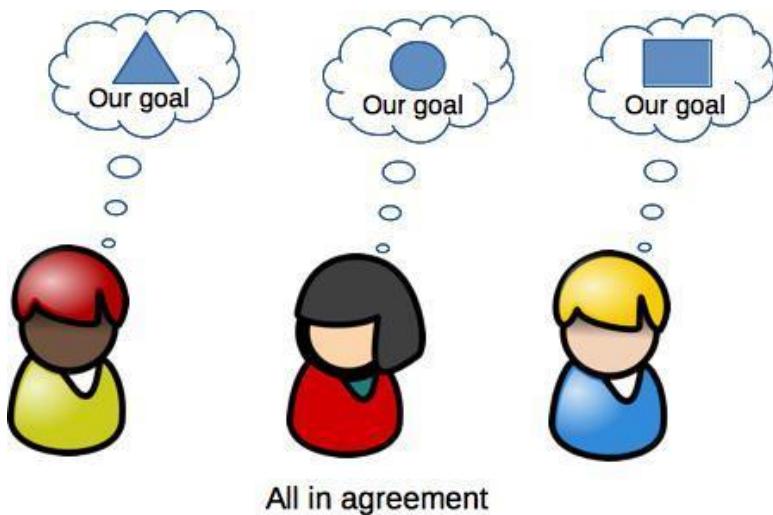


Figure 24.2: All in agreement

I think this must be called out as a distinct aspect of managing a transformation program, and the Guiding Coalition is the group that can best ensure misunderstandings are cleared up quickly and effectively.

Part 4: Change

Nothing is so painful to the human mind as a great and sudden change.

— Mary Wollstonecraft Shelley (*Frankenstein*)

This part deals with effective ways to drive or stimulate improvement. There are a couple of broad points to make on this subject.

First: Developing, delivering, and supporting a product or service require certain structures, processes, tools, and practices. Effecting organizational change is a different activity, calling for its own structures, processes, tools, and practices.

A successful transformation program must recognize these two distinct types of work and create the conditions necessary for both to be performed effectively.

Second: Many change agents are skilled at describing the goal state and identifying the gaps between the current state and the goal state, yet they struggle to define a practical series of steps to move the organization toward the target state. At times, it seems as if change agents expect the organization to flip abruptly to the target state.

A successful transformation program must identify a roadmap from the current state to the target state. The details are subject to change based on lessons learned along the way, but people need to have a clear idea of how to make stepwise progress. In most cases, so many changes are required on so many levels that an abrupt switch-over to the target state is not feasible.

This part addresses the issue on two levels:

- Including specific provisions for change in the structure of the consulting engagement; and
- Making stepwise improvements at the technical level aimed at the desired end state.

25 | Intentional Change

Great things are not done by impulse, but by a series of small things brought together.

— Vincent Van Gogh

Having a clear goal in mind and an accurate understanding of where you're starting are important. What's often missing is clarity about how to go from here to there.

It's been my observation that in the majority of organizational transformation programs, people tend not to make a distinction between product development and organizational change. Yet, these are two very different things.

25.1 | Current and Target Operating States

People usually define some sort of target operating state, often based on a branded framework but sometimes based on an internally-defined model. The defined target state elaborates a set of structures, processes, tools, and practices aimed at product development, delivery, and support. People may take considerable care in defining the target state.

People are also pretty diligent about identifying the gaps between the current state and the target state. So, they begin the transformation program with a clear idea of where they are and where they would like to go.

25.2 | Missing Pieces

But the same people often neglect to define any particular way to move the organization from the current state to the target state. Lacking any sort of roadmap for change, people in the organization may try to leap directly to the target state, or as close to it as prevailing organizational constraints will allow. The result can be a disjointed change effort with small pockets of high performance and numerous dependencies and blockers. The organization may never close all the gaps.

In “Anti-Patterns of Organizational Transformation,” I observe that we can find sound principles to guide organizational change (Kotter, ADKAR) without any roadmap or structure to execute on them. We can find a carefully-elaborated end state model (SAFe, LeSS) without any incremental steps to take us from here to there. We can find general guidelines to establish a continuous improvement system (Kaizen, Kanban Method) without any suggestions regarding exactly what changes we ought to attempt.

25.3 | “How to Change” Doesn’t Mean “How to Implement”

I don’t want to nit-pick individual frameworks or consulting firms. I’ll mention the Scaled Agile Framework (SAFe) because it defines a comprehensive 12-step implementation roadmap that you can read online (SAFe), so it’s a handy example. The roadmap is focused on implementing SAFe as such. It doesn’t begin with a business-focused goal, such as I describe in “Target States: Where Do We Want to Be?”

Similarly, other branded frameworks have implementation guides aimed at setting up their particular framework, but not aimed at any specific business goals that might be identified by analyzing

the company's relationship with its markets, its differentiating products, and so forth.

What's missing is guidance on how to *make change happen* as opposed to definitions for the target operating state.

Some management consultancies offer guidance in driving organizational change, but when you get down to the details the guidance amounts to duplicating a list of supporting structures and processes similar to those defined by Kotter or ADKAR, and possibly re-branding or re-naming them. It's like, "Hey, you need a sense of urgency. Better get some of that. And don't forget to set up a Guiding Coalition while you're at it." The marketing material is slicker than that, of course, but that's the substance.

In a few cases, the consultants tell you what your organizational structure should be to support their model, and then they leave it up to you to set things up before they'll come and work with you. Consultants specializing in Large-Scale Scrum (LeSS) usually expect the client to establish cross-functional teams as a prerequisite to working with them (Larman).

25.4 | Stepwise Improvement vs. Predefined End State

When we address large-scale organizations, we often find it isn't feasible to go from the *status quo* directly to the desired end-state structure in one step. A step-by-step approach seems more practical. If the transformation program has to be stopped for some reason, at least you've achieved *some* degree of improvement, it's quantifiable, and it's sustainable.

But those canonical structures and processes are not the end game; they're only a starting point. Much greater market effectiveness and delivery performance are possible. To achieve them, we have

to think of improvement as an ongoing process, rather than an “implementation” of something.

A model that recognizes the need for stepwise improvement has the power to get people *started* on the right foot as well as providing a path forward beyond the initial, “beginner” level methods.

This book isn’t about Agile scaling frameworks. The topic keeps coming up because the market is saturated with them at the moment. The book is about general approaches to organizational improvement with emphasis on IT organizations. Looking beyond the agile community for useful approaches, we find Lean-based methods can be very effective.

The Kanban Method, originally developed by David Anderson and now guided by Lean Kanban University, also pays attention to the fact organizations aren’t necessarily at a good starting point to make well-reasoned changes. Their approach begins by making the current process visible. This is accomplished by a variety of means, of which process visualization and metrics are key elements. If you don’t know your starting point, how can you tell whether the changes you make are resulting in improved outcomes?

The Kanban Method emphasizes iterative improvement based on models and the scientific method. The same sort of thinking is reflected in the approach described in this book. See the chapter, “The Cycle of Change.”

In my view, approaches of this general shape offer a greater probability of yielding value than approaches based on a final-state concept. Whether a particular approach is based on Agile, on Lean, a little of both, or on something else is less important.

Continuing beyond canonical, beginner-level Agile, we open up possibilities for significant organizational improvement along the lines of Beyond Budgeting and similar models. We also enable selected operations to function in a Lean Startup fashion, if that’s advisable for business reasons. It’s all but impossible to achieve improvements of that scope if we stop with implementing a predefined

framework. It requires constantly looking ahead to new horizons, as suggested in the chapter, “Begin With the End in Mind.”

26 | Initial Changes

Most times, the way isn't clear, but you want to start anyway. It is in starting with the first step that other steps become clearer.

— Israelmore Ayivor

Some of the earlier chapters covered preliminary analysis of the organization's current state that would inform our choices for the initial changes necessary to start the transformation program. In summary:

- identify one or more *competitive capabilities* as the purpose or business drivers of the transformation program;
- identify the products and services that are market differentiating and mission-critical, and those that could be handled through a partner or outsourced to a supplier;
- analyze the dependencies for the *competitive capabilities* to identify the key dependency, or *fulcrum*, to obtain business leverage from the IT area;
- select a short list of key metrics that will provide quantitative measurement of progress toward the goals of the program;
- analyze the interfaces and connections between the various systems to understand how to tease apart the differentiating and mission-critical components from the less-important ones, to facilitate handling them differently in future;
- analyze the flow of data through the organization to understand which systems will be affected by any change in data flows, structure, or content; and
- identify gaps between the current state and the target state.

Armed with the results of this work, we can begin the transformation. The strategy is to make several changes immediately, as a way to set the stage for ongoing incremental improvement.

For a typical large organization, the initial changes are:

- set up physical workspaces for teams to work collaboratively;
- introduce effective time management practices;
- form initial product-aligned cross-functional teams;
- take baseline operational performance measurements; and
- determine initial experiments for incremental improvement.

26.1 | Establish Collaborative Workspaces

Based on the number of products, product lines, or value streams to be supported, and the number of teams necessary to support each of them, create physical workspaces amenable to a collaborative style of work for those activities that benefit from collaboration.

26.2 | Form Product-Aligned Teams

For cross-functional teams aligned with products or value streams. At a minimum, this will include software development teams. It may also include infrastructure or platform engineering teams and production support teams.

The ability to form aligned cross-functional teams may be limited by existing organizational constraints. For example, in larger organizations workers may be segregated by technology stack, such as mobile/web, mid-tier, and back-end. There may also be service teams that can't immediately be folded into development teams.

That's fine initially, provided we plan to refine the team structure incrementally as we proceed with the program. We want to begin by establishing the best team-product alignment that's feasible at the moment.

26.3 | Take Baseline Measurements

Using the metrics we selected for tracking progress with the transformation program, allow the organization to operate with no further changes besides the new team structure, workspaces, and time management guidelines.

Measure the selected factors for a reasonable time to get a baseline for future comparison. This may be one or two release cycles, or a shorter period of time if the organization's current release cycle is more than a few weeks long.

26.4 | Choose Initial Experiments

During the baseline measurement period, guide each team in choosing their first few experiments for incremental improvement. Although the baseline measurements haven't been finalized at this point, some opportunities for improvement will be apparent.

Client personnel will already be aware of aspects of their operations that could be improved. They have been living with "pain points" since before the consultants arrived.

The consultants will also be able to determine some initial step-by-step improvements in team structure, processes, collaboration, and practices. In particular, for larger organizations the initial team structure will not be ideal, and the team members will not be accustomed to working collaboratively. Some typical and obvious directions for stepwise improvement include:

- gradually collapsing functional silos and folding more responsibilities into the product-aligned teams;
- gradually improving the degree of collaboration within and between teams;
- gradually introducing more-effective technical practices; and
- eventually collapsing the largest silos - the front-end, mid-tier, and back-end personnel.

The following chapters offer information about collaborative workspaces, time management, and stepwise improvement in collapsing functional silos, increasing collaboration, and improving processes and practices.

27 | Workspaces for Collaboration

I've got a theory: if you love your workspace, you'll love your work a little more.

— Cynthia Rowley

With respect to teams' *physical* work spaces, we don't want to take an incremental approach to setting up an appropriate environment. We want to guide client leadership to establish the proper physical spaces at the same time as we create the initial product-aligned cross-functional teams.

There are three practical reasons for this.

First, for teams to progress as they should through the series of experiments that will follow the initial restructuring, they need to have appropriate physical work spaces. Otherwise, they will have a hard time learning collaborative work methods. Progress with the transformation program will likely stall.

Second, it isn't logistically or financially practical to "incrementally" change the furniture in the office over and over again. In fact, the idea is absurd.

Third, we can take advantage of the Hawthorne Effect to get teams off to a positive start. They have a long and stressful road ahead. The Hawthorne Effect is generally understood to have the effect that people change their behavior because they know they're being observed (Cherry).

A positive spin on it is team members will feel things are changing for the better *because their managers are paying attention to what's going on for a change*. It's okay if that means the observations

wouldn't pass muster for a well-structured behavioral study. We aren't doing a behavioral study. We're improving organizational performance. If the Hawthorne Effect helps us do that, then let's use it.

In a typical IT organization, managers generally pay little attention to life as experienced by technical team members. During the transformation program, they will pay close attention, and they will actively remove organizational constraints that have been holding people back from doing their best at work. That's a Good Thing.

Contemporary methods for creative work such as software development call for collaboration more often than for solo work. Collaboration calls for people to be located in a common work area. But the characteristics of a collaborative work area seem to be widely misunderstood.

27.1 | The Rise of Office Work

In the early 20th century, businesses faced ever-increasing needs for office workers to operate newfangled machines like adding machines and typewriters, as well as to do calculations and record-keeping using pen and paper. A layout we would now call an "open-plan office" was common, as in this image from officemuseum.com:



Figure 27.1: Office environment in the 1930s

You can imagine these were pretty noisy environments where it would be hard to maintain focus or have a conversation with a co-worker. This layout was especially frustrating for people who did any sort of creative work such as writing, or who had to work with figures, as bookkeepers and accountants did.

27.2 | The Action Office

In 1964, Herman Miller designers Robert Propst and George Nelson created a revolutionary workspace solution: The Action Office.



Figure 27.2: The Action Office (1964)

The Action Office had features we recognize today as leading-edge. The worker could sit, stand, move around, and had different types of work surfaces available. Work spaces were designed to keep work visible, so that important things would not get lost in a drawer or cabinet. There was a “communication center” geared for the telephone era, providing a degree of sound isolation. Each person could reconfigure the space in whatever way they wished at any time.

The solution didn’t catch on with companies. Nikil Saval, writing in *Wired*, speculates part of the reason was the rapid rise in demand for office space. That led companies to seek ways to cram more and more people into whatever space was available.

A second reason may have been that senior managers of the time didn’t want to make work too comfortable for their subordinates. It was an era when hierarchy and formal power dominated leadership thinking to such an extent they often superseded business concerns;

a manager would rather generate less profit than to lose perceived status.

27.3 | The Birth of the Cubicle Farm

Action Office II sought to mitigate business leaders' concerns while maintaining the workers' ability to maintain focus and apply creativity to their work. It featured three padded walls set at 120-degree angles, and had many of the same advanced features as the original version.

This product caught on, but not for the reasons its designers expected. Customers discovered if they set the walls at 90-degree angles, they could create a mass of little squares into which they could stuff many people. Rather than allowing each worker to customize their space, companies standardized the layout to achieve economies of scale and to prevent anyone from feeling superior to anyone else.

The result: The cubicle farm.

The cubicle farm dominated office landscapes throughout the 1980s and well into the 1990s. In hindsight, we know the only positive result of the cubicle farm was the inspiration it provided to engineer Scott Adams to create the comic strip, *Dilbert*.

If working in a cubicle farm doesn't kill your spirit, then you could probably walk right through an army of Dementors without noticing them (Harry Potter Wiki).



Figure 27.3: A cubicle farm

Cubicle farms were highly detrimental to employee morale and productivity. According to Ashley Stahl in *Forbes*, cubicle farms were a great way to cut costs, as they enabled companies to fit many people into a office area without investing in permanent walls. Like so many management fads, she writes that they “seemed like a good idea at the time.”

But they interfered with people’s ability to focus, to collaborate, and to communicate. People were boxed in within drab walls, isolated from others.

27.4 | The Open-Plan Office

With the advent of lightweight management methods and the discovery of the value of collaboration in the late 20th century, companies found a convenient way to save money on furniture: The open-plan office.

It’s hard for members of a team to collaborate when they’re all

sequestered in individual, padded boxes. They need to be able to see and hear each other, to ask each other questions, and to offer help.

No problem: Down came the cubicle walls. Down, too, came the cost of setting up an office space. And up went the number of people who could be stuffed into a building.



Figure 27.4: An open-plan office space

It's a win-win for companies and employees. Except that it isn't.

Open plan offices may be worse for employee health, engagement, creativity, and productivity than any previous set-up. There are so many studies and surveys reporting negative impacts that it would be impossible to list them all in the References section.

Geoffrey James, writing in *Inc.*, provides a summary of key negatives of open plan offices, with links to back them up (James).

I urge you to read the piece, as I could do no better than to copy and paste it here. James concludes, “Here’s the real kicker...open-plan offices are so incredibly destructive to productivity that they’re a net huge loss, even in areas where office space is pricey.”

27.5 | Collaborative Team Work Spaces

The solution has been around a while. The first edition of *Peopleware*, by Tom DeMarco and Tim Lister, came out in 1987, in the heyday of the cubicle farm. It describes effective workspace design for teams that use collaborative methods for creative work such as product development.

In a nutshell, what they found was that a creative team working collaboratively needs:

- a group work space for full-team collaborative activities
- semi-private spaces for brainstorming, design sessions, conversations, etc.
- private spaces for one-on-one meetings, personal phone calls, decompression, etc.

A similar concept known as “caves and commons” recognizes these needs, as well.

27.5.1 | The Common Area

This is a space isolated from other team spaces, so that team members aren’t distracted and stressed by ambient noise, and yet large enough to accommodate the full team, so they can easily collaborate.

Overhearing things that are *relevant to the team’s work* does not have the same negative effects on focus, creativity, and nerves as the random sounds of an open-plan office. Quite the opposite - it enables *osmotic communication*, an interesting phenomenon noted by software development methodologist Alistair Cockburn (Cockburn).

Ideally, this space is reconfigurable by team members as needed. That way, it can accommodate various methods or styles of collaboration the team may choose to use at different times or for different purposes. These can include (from least to most intensively collaborative) working solo but being available to team mates, pairing, working in small groups, or mob programming (also known as “mobbing”). Reconfigurability also facilitates using the space for demonstrations, experiments, code reviews, defect resolution, retrospectives, and team coordination meetings or “stand-ups,” not to mention the occasional birthday party or team celebration.

Many layouts are possible that will meet the needs of a team working together collaboratively. I omit diagrams because I don’t want to prejudice the reader regarding “correct” spaces. As long as the team can reconfigure the space easily and can arrange themselves to suit whatever form of collaboration they’re using at the moment, then it’s good.

Furnishings can include one or two screens for projection, one or more whiteboards, a large table for group collaboration, chairs, and desks suitable for standing or sitting.

27.5.2 | Semi-Private Areas

Semi-private areas are meant to facilitate work by subsets of the team, when they don’t want to distract or annoy the main group. Often, two to four team members will need to discuss something together that doesn’t need to involve the whole team.

These spaces are often used for brainstorming, design sessions, technical debates, and similar purposes.

The semi-private spaces aren’t set up for pairing or mobbing, but rather for discussion. They may contain comfortable seating and two or more whiteboards.

27.5.3 | Private Areas

It isn't always feasible for each individual team member to have their own full-time dedicated private space, as they had in the old cubicle farms. But as they spend most of the day in the common area and semi-private areas, they won't miss their old cubicle.

There are times when the work at hand requires individual focus in a quiet space. There are times when team members need to make a personal call. There are occasions for one-on-one meetings about matters that are none of the business of other team members. There are times when a person needs to get away from people for a few minutes to "decompress" (particularly introverts, but on occasion, anyone).

Private spaces are for purposes like those. They can be shared hotel-style, on an as-available basis, or scheduled in some informal way such as a sign-up sheet. The details will depend on how the team chooses to operate.

27.6 | Considerations for Remote Workers

Remote work is a reality in many companies, and as of this writing indications are it is gaining in popularity.

27.6.1 | Remote Collaboration

Although conventional wisdom in the Agile community is that there's no substitute for collocation, that may not be feasible in all cases. It may not even be *desirable*, if we balance people's overall needs against the purported productivity gains of collocation. It's my view that people's overall needs take precedence.

We won't destroy the company if we're slightly less efficient (in a purely mechanical sense) by working remotely. Other benefits, such as employee morale, engagement, commitment, and ability to focus, will probably cancel out a marginal loss of efficiency.

The point isn't collocation as such; it's *collaboration*. Technology provides a range of options that mitigate the downside effects of remote work, and that enable a fairly high degree of collaboration among remote workers.

One company where I was engaged had technical staff in two locations. In each location, one of the large monitors used for information radiators and demonstrations was dedicated to providing a full-time view into the other location. Working there felt like working in a single large space.

They also set up pairing stations with four monitors and a camera. Pairing between locations was painless.

One day the power went out, We had become so accustomed to seeing the other side of the team on the monitor, it felt as if half the room had been chopped off.

On another engagement, the company (a startup) had no office space for software development. Instead, they had a server room and two engineers working there to keep things up and running. Everyone else worked remotely.

To encourage collaboration, we were officially paired up - not in the way pair programming is done, but formally "locked together" for the duration. I was living in Dallas, Texas, at the time, and my partner was in Ogden, Utah.

The pairs were connected full-time when "at work." When we needed to work with other team members, we used screen-sharing and text messaging software. The setup worked well.

At the end of the project, the company brought everyone together in New York for a celebration and farewell. For the first time, we

met our team mates from China. We had been collaborating with them effectively using text messaging for months.

When we met in person, we realized the collaboration would have been *less* effective in person. Their written English was very good - better than most Americans' - but their spoken English was unintelligible. They learned English for the purpose of reading technical material. They didn't care about conversation, and didn't practice it.

Be careful of making too many assumptions about collocation, collaboration, and effective delivery. Many different situations and setups are possible. The main thing is to enable collaboration. The physical location of team members is less important than that.

27.6.2 | Tools Supporting Remote Collaboration

The key thing is to enable effective collaboration. Technology to the rescue! There are ample screen sharing tools, keyboard sharing tools, collaborative design tools, and other software tools for the purpose, and hardware solutions like cameras, microphones, and screens are reasonably-priced compared with the positive impact on clear communication and product quality they provide.

27.6.3 | Know Your Team Mates

Collaborative working styles are most effective when team members know one another and have a chance to learn to care for one another. If they never meet in person, that kind of team cohesion is unlikely.

Therefore, it's a good idea to schedule times for team members to get together in meatspace occasionally. The travel costs will be repaid in the coin of improved team interactions, engagement, morale, and commitment.

If this isn't feasible for logistical or financial reasons, don't obsess over it. It's beneficial, but if it doesn't happen it isn't a hard showstopper for effective remote work.

27.6.4 | Conference Call Tips

That's not to say there are no challenges in supporting remote workers. Perhaps the most obvious is the notorious *conference call*. Remote participants in an audio-only conference call tend to be forgotten by the people at the main location.

Here are a few tips to make audio-only conference calls effective:

- Be sure the meeting takes place in a quiet area;
- Get everything set up before the meeting starts, so that you can start on time without 15-20 minutes of technical problems;
- Throughout the meeting, actively encourage participation from remote participants; they can't tell when it's safe to interrupt, as they can't see the people in the room; invite them frequently; *a tip*: write the name of each remote participant on a card or sticky note and place the cards around the speaker where the local participants can see them;
- Keep the agenda brief and stick to it; remote participants will lose interest if the discussion drifts aimlessly or if too many different topics are included;
- Record the call and take notes so any decisions, questions, or action items won't be lost in the shuffle;
- Don't hesitate to ask someone to mute their microphone if you're getting feedback or background noise;
- Keep the meeting short; people don't stay focused on phone calls for more than about 30 minutes.

Most of the same considerations apply to video conference calls, too. You probably won't have to invite people to speak up explicitly, as they *can* see what the other people are doing.

With video, you can use visual aids and screen sharing features to enhance clarity of communication.

You can also update documents, spreadsheets, diagrams, presentation slides, project management tools, and source code live, in real time, with everyone involved and able to see what's happening.

There will inevitably be technical issues somewhere. Be ready to use chat features of the conference service so that people who can't maintain a good video feed will have an alternative way to offer input to the meeting.

27.7 | 5S and Software Development

I've written about 5S in the context of software development in the past. Here are some excerpts from a blog post from 2016 on the subject:

Whether you're puttering around the house, working on your car, cooking a meal, doing the laundry, or writing software, everything is easier when the items you need are at hand and in their proper places.

People working in manufacturing operations learned that lesson long ago. A basic idea in lean manufacturing is 5S; a set of guidelines for keeping a work space orderly, so that people don't waste a lot of time looking for tools and materials.

Motion is one of the Seven Lean Wastes; it refers to unnecessary motion on the part of workers to walk around or to find tools or parts during a manufacturing process. The analogue for software teams is unnecessary "motion" to look for things in multiple places, whether physical or electronic.

The five S's stand for five Japanese words:

1. *seiri*, often translated as sort, means to dispose of any items that are not necessary to the work to be done.
2. *seiton*, often translated as set in order, means to arrange necessary items so they can be found and used easily whenever they are needed.
3. *seiso* means to clean or to organize the work area, to help maintain order as well as to ensure worker safety. Cleanliness is also treated as an inspection criterion. A clean work area is more likely than a dirty one to be well-organized and usable.
4. *seiketsu*, often translated as standardize, means to create and follow a set of regular procedures to keep the work area orderly and clean.
5. *shitsuke*, often translated as sustain, means to stay on top of things once you've gotten everything organized. Don't let the situation deteriorate again.

Software teams have both physical and virtual work spaces. I've already touched on the "caves and commons" idea, or DeMarco and Lister's three types of space for collaborative teams.

Those are the physical spaces. Those spaces need to be organized and kept in order. This helps not only with *flow*, by reducing wasted motion, but also tends to enhance morale, as it's a little frustrating to work in a cluttered space.

Some of the things teams may want to be sure are readily at hand are:

- List of phone numbers
- List of chargeback codes
- Batteries for wireless mice and keyboards
- Remote control for the overhead projector
- Keys to the supply cabinet

- Supplies for process visualization (sticky notes, etc.)
- Whiteboard markers and erasers
- Cables, chargers, and similar items

Unlike a manufacturing cell, a software team has *virtual* work spaces, too. These need to be kept orderly, just like the physical spaces. They include:

- Version control system (delete temporary branches that aren't needed)
- Source code (remediate cruft and technical debt)
- Test code (remove meaningless tests; organize test cases by functional area, not by test subject)
- Build scripts / configuration files (delete unused configuration settings and steps)
- Deployment scripts (delete unused steps)
- Environment provisioning scripts (delete unused steps, replace obsolete versions)
- Servers (clear out unused directories, defragment disks)
- Documentation (delete or update obsolete information; keep information organized logically; add indexing feature, if missing)
- Backlog (no need to keep dead backlog items forever)
- Development environments (clear out unused directories, uninstall unused software)
- Test environments (ensure test environments are identical to production environments)
- Staging environments (ensure staging environments are identical to production environments)
- Reports / statistics / metrics (remove obsolete or unused data from reports and logs)
- Backups (keep backups in sync with current configuration; test restore procedures regularly)
- Recovery (test recovery procedures regularly)

Teams may have additional spaces to manage, as well.

27.8 | Gaining Cooperation from Client Leadership

Either you (engaged as a technical coach) or a colleague on the Consulting Team who is engaged at the appropriate level of management in the client organization, must make the connections between proper team work spaces and achievement of the business goals of the transformation program.

Checking the dependency tree presented in “Leverage: Where’s the Fulcrum?” we can trace the dependencies from any one of the target competitive capabilities all the way down to the technical team level.

Let’s start with the simplest one: *Retain Customers*. The immediate dependency is a *delivery capability*, “Predictable release cadence for product updates.”

That capability depends, in turn, on an *operational capability*, “Stable Production Operations.”

That’s the *fulcrum* to gain leverage from IT to support the business goal. It has a number of dependencies of its own. The diagram shows four *operational capabilities* that are required to support “Stable Production Operations.”

Those underlying *operational capabilities*, in turn, depend on a number of low-level organizational structures, processes, tools, practices, and skills. Due to the importance of *collaboration* for most of those things, anything that hinders collaboration will reduce the effectiveness and value of the transformation program while increasing its cost and extending its timeline.

At the start of the transformation program, you won’t have hard numbers to demonstrate the value of collaborative team work spaces. You’ll have to use hypotheticals to make the point.

Here’s a hypothetical team. Call it Team A.

Working under conditions favorable to collaboration, Team A delivers work items with a mean Cycle Time of 10 days. Our Definition of Done means a work item is “delivered” only if it has no known defects and meets all functional and non-functional requirements defined for it. So, in a loose sense, Team A delivers “something” every 10 days or so, most of the time.

What would be the difference if Team A had to work in an open-plan office? Remember all the various negatives that researchers have identified for open-plan offices. Let’s make the assumption that all of that reduces Team A’s effectiveness.

Let’s say under those conditions it tends to take them about twice as long to plow through a given amount of work. That would result in a mean Cycle Time of 20 days.

But that’s optimistic. Owing to the distractions and stress of the open-plan office, Team A isn’t able to deliver defect-free results. Instead, two out of three work items are found to be unacceptable, even when Team A believed they were finished.

That means in the next delivery cadence (“iteration” or “Sprint”, if using a time-boxed model), Team A has to spend half their time fixing the defects they created in the previous cadence. That extends the Cycle Times for the defective items by 10 days, and raises the mean Cycle Time to about 25 days.

But two out of three of *those* work items are found to be defective, as well. As time goes on, the proportion of time Team A spends fixing defects exceeds the time they spend doing value-add work. Progress on product updates slows to a crawl.

We can also assume the testers are working in the open-plan office environment, too. Their effectiveness will also be affected by the working conditions. They will overlook test cases, fail to think of explorations to try, make errors in coding executable checks, and miss regressions. The stability of production operations is further degraded.

Defects make their way into production. That breaks the goal of “Stable Production Operations,” which is a dependency of the business goal, *Retain Customers*. The organization cannot support the business goal without stable production operations, and they cannot sustain stable production operations without clean code from the development teams and correctly-configured server instances from the infrastructure teams.

The infrastructure engineering teams are also affected by the open-plan office. They make errors in provisioning and configuring server instances. They spend most of their time fighting fires, and have insufficient time to script and automate the provisioning and configuration of environments. The pattern of manual work, distractions, stress, and errors continues. Production operations are destabilized even more.

Service teams are also working in the open-plan office. That means degraded quality of network engineering, database management, and other central functions of the IT organization.

A shorter way to describe this is to say the situation is the same as what probably prompted leadership to consider a transformation in the first place. The lesson is that without proper physical work spaces, the result of the entire program will be to leave the organization right where it started.

Client leadership will not like this analysis, because changing the physical work environment is a “big hit” financially, and we’re asking them to do it all at once. But it’s the only way forward.

28 | Time Management

You can have it all. Just not all at once.

— Oprah Winfrey

Already noted: *Structure* has the greatest impact on effectiveness, with *process* in second place; and some initial structural changes have to be made at the outset to enable further incremental improvement by applying the Cycle of Change.

There are also *process* changes that need to be made preemptively to enable further incremental improvement. Chief among these is *time management*.

28.1 | Outlook-Driven Development

Outlook-driven development is a running joke in the software industry. The reference is to Microsoft Outlook, a software product that supports booking conference rooms and scheduling meetings and calls. Outlook and other products like it are ubiquitous in organizations.

In many organizations, people feel overworked in part because of their meeting schedules. They may be triple- or quadruple-booked at all hours of the day, every day. When change agents recommend changes in work processes, people often complain that they don't have time to try the things consultants and coaches suggest because their calendars are already full.

28.2 | Causes of Time Management Issues

It's my observation that people slide into time management issues gradually. They want to contribute and to help others in the organization, so they do their best to juggle all the requests for their time.

Causes may include:

- belief that multitasking is effective
- inability to filter out unimportant requests
- inability to prioritize requests
- desire to please; saying "yes" to everything
- failure to recognize the difference between *manager time* and *maker time*

In reality, people can only focus on one task at a time. When they try to juggle too many things, they can't give their best effort to any one of them. The most effective way to complete a long list of tasks is to tackle them one at a time and finish each one before starting the next.

At times it can be hard to distinguish between requests that are truly important and requests that can be deferred, delegated, or simply not done. With the best of intentions, people try to do everything that is asked of them. The result can be that no task is completed particularly well, and people feel overworked.

Even when people filter out the unimportant requests, they may have difficulty prioritizing the remaining ones. Often people spend time on tasks of lesser importance, leaving more-important ones to wait.

People want to be helpful and to contribute to the success of the organization. It can be hard for them to pick and choose between

meeting overlapping meeting invitations. Yet, it's not possible to be in more than one place at a time, either physically or mentally.

I was giving Lean training at various locations of the client company. At one location, I noticed four people in the room were using their cell phones, had earbuds in place, and were working on their laptops during class.

When I asked what they were doing, I learned they were (a) in another meeting by phone, (b) listening to a third meeting via their earbuds, and (c) doing production support work on their laptops. I asked if they were getting any value from the class, they insisted they were, and it was no problem to multitask that way. *They did it all the time.*

In other instances, client personnel were “attending” one or two meetings “remotely” (although they were physically present in the building) while also working on programming or testing tasks at their desks.

I submit there is *no way* any of these people were carrying out any of the various activities with any degree of focus.

If we are to have any hope of making progress in an organizational transformation program, we have to gain control of *time* at the outset.

28.3 | Manager Time vs Maker Time

Creative work such as software development requires focus for extended periods of time. Any interruption, however brief or trivial, causes a context switch. Recovery from a context switch requires several minutes, in most cases.

The cumulative impact of small interruptions throughout the work day can add up to a serious negative impact on effectiveness, quality, and lead time for software products. Yet, in many organizations,

people engaged in creative work are required to stop what they're doing to participate in meetings and conference calls.

Paul Graham came up with an interesting way to look at it. He distinguishes between a *manager's schedule* and a *maker's schedule* (Graham).

To a manager, a work day consists of one-hour time slots. Organizing one's work means (in part) scheduling meetings in these time slices. Any time a colleague's schedule shows an open time slot is considered available for a meeting.

To a maker, a work day consists of blocks of time long enough to support focused work, in the range of 3 to 4 hours at a stretch. When "maker time" is not respected, makers are interrupted during these periods of focused work, and they experience context-switching events. They cannot do their best work in that way.

It may be counterintuitive for managers, but taking a creative out of their focus zone for 30 minutes in the middle of the morning destroys their morning. Same for the afternoon.

One of the most important changes to make initially is to break the habit of interrupting creative time. If the organization continues with Outlook-driven development, it will be very difficult to make incremental progress with the transformation program.

28.4 | Managing the Calendar

How can people get meeting time under control? Here are some suggestions consultants and coaches can offer to clients.

28.4.1 | Organizers: Is the Meeting Necessary?

Think about the purpose of the meeting.

If it's the sort of thing that's normally handled by events already in the work flow, such as planning sessions, workshops, reviews, and the like, then how does an additional meeting add value?

Is the purpose of the meeting merely to prepare for another meeting? I've seen this pattern when people at one level of management want to be well prepared for an upcoming meeting that will include people at a higher level of management. They don't want to seem uninformed or incompetent.

I've seen situations where people held pre-meeting meetings and pre-pre-meeting meetings to get ready for The Big Meeting. I don't recall a single case when that was at all valuable. Think about whether the preparation can be done without pulling people away from their work.

Is the purpose of the meeting to have a discussion that could be handled by phone or email? A phone call will interrupt a person's focus, but if the nature of the discussion calls for a quick conversation then it could be preferable to a formal meeting. Email will not interrupt a person, but it's a less-effective medium of communication than voice or face-to-face. It's a judgment call.

28.4.2 | Organizers: Don't Invite the World

Once you've determined that the meeting is necessary, consider who really needs to be there to achieve the goal.

In many organizations, it's customary to invite everyone who might potentially have any interest at all in the topic of the meeting, as well as any managers the meeting organizer wants to make aware of the meeting. I've seen meetings with 60 or more invitees, when the goal of the meeting really only required the participation of 4 or 5 people.

Smaller groups have less internal communication overhead than larger groups. If decisions are reached by consensus, it will take

longer to achieve with a large group than a small one. If decisions are made by a senior manager, then there's no need for subordinates to be physically present in the same room to receive the manager's orders.

Besides that, not all stakeholders for projects have the same degree of interest or involvement. People who are peripheral to the project and need to be informed of what's going on may not be needed in the meetings where project participants are doing work or making decisions. They can be informed after the fact.

28.4.3 | Invitees: Don't Be Afraid To Decline

People want to be helpful and they want to participate in and contribute to the things their organization is doing. But the bottom line is no one can do everything. We have to pick and choose.

It isn't really helpful, you aren't really participating, and you can't contribute effectively by accepting a meeting invitation and then not attending the meeting. When you're double-booked or worse, that's what will happen. You can't be in two places at once.

A question to ask yourself is: Do I need to be at this meeting?

If you are expected to provide input, or if you want the opportunity to provide input, then the answer is "yes." Otherwise, it's possible you can be informed of the outcome of the meeting asynchronously and after the fact, without interrupting your "maker time."

This may be easier said than done. In Outlook-driven organizations, it's often very hard to pin someone down even for five minutes just to ask a simple question. You have to wait two or three weeks for an answer, as the other person is completely booked solid.

That's a consequence of the way things have been done in the past, and one of the things we need to halt at the very beginning of the transformation program.

As we restructure the organization to form product-aligned cross-functional teams, it will be easier for people to locate and speak with colleagues they need to work with. In the meantime, we can advise people not to accept every meeting invitation.

28.4.4 | Invitees: Prioritize

The *status quo ante* may be a matrixed organization in which individuals are assigned to multiple projects concurrently, on a percentage basis. In that environment, it's hard to know which meetings to accept and which to decline.

In conjunction with the shift from project-centric to product-centric work and the structural change to institute product-aligned cross-functional teams, individual team members know that nearly all the meetings to which they're invited will pertain to the product their team supports. With that in mind, they can prioritize meeting invitations consistently with the prioritization of the actual work items their team is working on or planning for.

28.4.5 | Invitees: Defer

It's possible the subject of the meeting is of interest to you, but it isn't a task you're currently working on. Unless you need to have a voice in planning for the future task, you can probably skip the meeting and catch up later, when the task comes up in your work schedule.

28.4.6 | Invitees: Delegate

Depending on your role and position in the organization, you may be able to delegate attendance at the meeting to someone else. They can then fill you in on what happened.

28.5 | Protecting Maker Time

With the initial structural changes, each team-level technical coach will be working with one or more cross-functional teams that are aligned with products. The coaches will be encouraging collaborative methods such as pairing and mobbing, work flow improvements such as minimizing batch sizes and WIP levels, and other measures that give the teams room to work properly.

Protecting their “maker time” is key to enabling the other improvements to take full effect.

Define particular times of day when the team is not to be interrupted with meetings, phone calls, or walk-in questions. At other times of day, people outside the team are free to communicate with team members.

This will ensure teams have adequate blocks of time for focused work, either individually, in pairs, or as a group.

In an Outlook-driven organization, one way to support this idea is to block out these times in team members’ calendars to discourage others from inviting them to meetings during those times. However, it’s also a common practice in such organizations to send invitations without checking people’s availability first. So, that strategy may have little effect.

29 | Incrementally Collapsing Functional Silos

“Is that it?”

“No. That’s a wall.”

“It could be disguised.”

“You’re not very good at looking for things, are you?”

“I’m good at looking for walls. Look, I found another one.”

— Derek Landy (*Kingdom of the Wicked*)

Given that changes in *structure* have the greatest impact, changes in *process* the next-greatest, and changes in *practices* the smallest, our series of incremental improvements should be determined mainly by structural changes, with changes in process and practices that support each structural improvement.

29.1 | Initial Organizational Structure

Many IT organizations are structured to maximize *resource utilization* rather than *throughput*. This sort of structure impedes continuous flow by creating numerous cross-team dependencies, leading to hand-offs, miscommunication, delays, errors, back-flows, rework, and unfinished inventory in the form of incomplete work waiting for the next functional silo to pick it up. Any given piece of work bounces around the organization like a Pachinko ball.

This sort of structure also tends to isolate the *makers* of solutions from the *users* of solutions. The result is the makers may have little understanding of, empathy for, or connection with the customers who use their products. They go through the motions of performing their tasks without any real commitment or interest.

The other side of the coin is the organization can enjoy the benefit of only a tiny fraction of the makers' creativity, intelligence, and skill, as they are locked into narrowly-defined roles and expected to repeat the same tasks over and over by rote.

All of that stems from a *utilization* mindset; the belief that results can be obtained efficiently if each person focuses on a single area of specialization, and managers orchestrate their activities. Once people begin to think about *throughput*, they quickly see the *utilization* approach isn't as effective as they had believed.

When we made the initial changes to begin the transformation in earnest, we collapsed functional silos to the extent possible given existing organizational constraints to create product-aligned cross-functional teams. In most large organizations, these initial teams will not include all the responsibilities for supporting their products. Some service teams will remain, and some silos based on technology stacks will remain.

Going forward, we want to continue to break down walls between activities and responsibilities that have become separated over the many years of focusing on *resource utilization*.

Most organizations can't just leap to a perfect structure in a single go. Ironically, to sustain the structure of disconnectedness, too many things are interconnected to allow for a quick or easy fix. The problem is they're not connected in a way that aligns with products or value streams. What might be a reasonable step-by-step approach to collapse functional silos?

29.2 | Starting Point

Let's visualize a starting point for this dimension of improvement in which pretty much all responsibilities are siloed, and see how we could begin to improve the structure.

29.2.1 | A Separate Team For Every Type of Task

Typically, all the work pertaining to a given set of technologies or platforms is isolated from the work for other technologies or platform. For each technology group, all the usual activities surrounding software creation, delivery, operation, and support are duplicated. Within each technology group, each of those activities is isolated or “siloed.” Some responsibilities are global, or enterprise-wide.

Here's a grid that illustrates what I mean. Each X is a separate team or set of teams in the organization.

	Global	Mobile & Web	Distributed & Mid-Tier	Back-End
COTS & ERP Support (1 team per product)	X			
Security	X			
Database (1 team per product)		X	X	X
Network (1 team per zone or subnet)	X			
Operations			X	X
Production Support (1 team per set of apps)		X	X	X
Enterprise Architecture	X			
Solution Architecture (1 team per app)		X	X	X
Analysis & Requirements 1 team per set of apps)		X	X	X
UX, Usability, Accessibility		X		
Unit & Component Test (1 team per set of apps)		X	X	X
Integration Test (1 team per set of apps)		X	X	X
End-to-End Functional Test (1 team per set of apps)		X	X	X
System Test (1 team per set of apps)		X	X	X
Solution Design & Programming (multiple teams, "fungible")		X	X	X
Solution Build		X	X	X
Continuous Integration		X	X	
Infrastructure Engineering		X	X	X
Solution Deployment		X	X	X

Figure 29.1: Teams organized as functional silos

Notice there is no direct connection between any team and any product or service provided to customers. Teams are organized around activities rather than business value.

29.2.2 | Similar Tasks Segregated by Technology Stack

Every category of work involved with software creation, delivery, operations, and support is handled by a different team, and nearly all categories of work are segregated by platform or technology stack.

For example, there may be a team that performs end-to-end functional testing for just one application or a small set of applications

on just one technology stack, such as Mobile & Web or Back-End (i.e., mainframe). These test teams may be somewhat aligned with products, but they are divided across the various technology stacks in the organization.

Similarly, there may be a pool of programmers who are assigned to different projects depending on which projects call for programming to be done. These work groups (not “teams” in any meaningful sense) are temporary, not aligned with any single product, and segregated by technology stack. Within any one technology stack, programmers may be further segregated to work only on front-end or only on server-side code.

A production support team will deal with just one technology stack, and will have to support all the applications (or parts of applications) that run on that stack.

29.2.3 | IT Becomes a World Apart

No one has visibility into a value stream or product line. From a human standpoint, to a person working in one of these teams, the world looks like an endless series of one-off, disconnected requests for small tasks. There’s no sense of purpose or value.

From a political standpoint, this structure begets a thick middle management layer in which career advancement is all but impossible. That leads mid-level managers to compete with each other for visibility, credit, and promotion opportunities. They often pursue this in ways that further impede continuous flow, as they try to undermine one another by delaying selected requests for the teams they manage.

From a mechanical standpoint, every initiative requires services from multiple teams, each of which has its own work queue and priorities. The probability is high that any given work item will be in a wait state for nearly 100% of the time it’s in the system. Cycle

efficiency in organizations that are structured this way tends to be in the 0.5% to 2% range.

29.3 | Organizational Constraints on Silo-Busting

The strategy described in this book calls for some initial changes based on what we already know about what “works well.” With respect to team structure, that means establishing the closest thing to product-aligned cross-functional teams as we can within the prevailing organizational constraints.

What are those constraints? Bear in mind this is a general model, and real companies will differ to some extent. Yet, I think this is pretty accurate on the whole.

I see three levels of constraints with respect to restructuring IT organizations. From large to small, they are:

- Separation of work and people by technology stack, e.g.
 - Mobile & Web
 - Distributed & Mid-Tier
 - Back-End, such as mainframe and legacy midrange systems
- Separation of people by the types of activities they perform
 - Operations
 - Production Support
 - Security
 - Software design, creation, delivery
 - Infrastructure support
- Separation of people by role or tasks
 - Analysis & Requirements
 - Testing (multiple types)

- Enterprise Architecture
- Solution Architecture
- Solution Design & Programming
- Solution Build
- Continuous Integration
- Deployment
- *more*

Notice what's absent here: No team anywhere in the organization is explicitly aligned end-to-end with a system that serves a category of customers.

It's a Matryoshka doll. The largest doll is the technology stack. Inside that one there's a "types of activities" doll. Inside that, there's a "role or task" doll. So, we have (for instance) "end-to-end test" teams for mobile & web, for mid-tier systems, and for back-end systems. It's the same general structure across all roles and tasks. Here's a highly artistic visual representation:

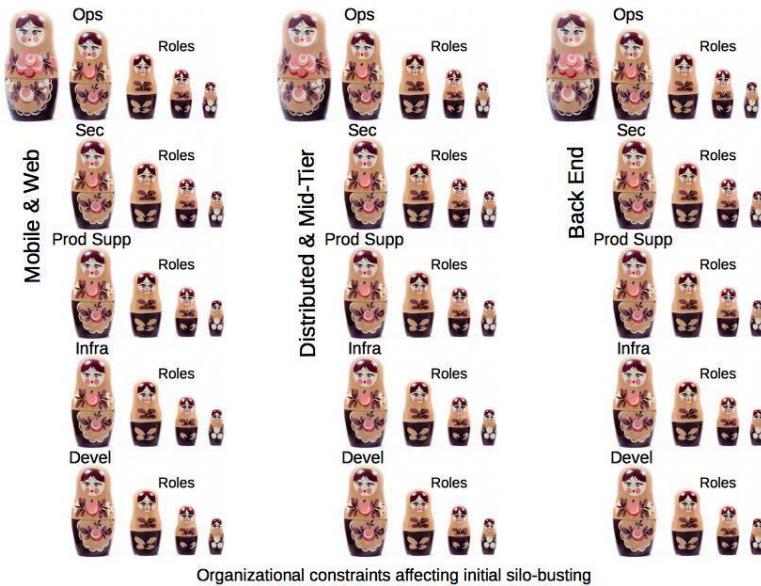


Figure 29.2: Silo-busting constraints

29.4 | Temporary Scaffolding

There is a fundamental concept that most change agents seem to overlook: When you move toward an end state one step at a time, you are not necessarily implementing permanent changes all along the way. You might introduce a change that will become obsolete after further improvements have been made.

I think of this as analogous to ancient construction methods. When people in Egypt, Greece, Rome, or the pre-columbian Americas built pyramids and temples, and when ancient Europeans built structures of large stone monoliths as at Stonehenge, they had to lift very large, heavy blocks to great heights.

To accomplish this, they carved the blocks with *bosses*, or protrusions that stuck out on the sides. They could secure ropes to

these bosses and use various rope-and-pulley systems (depending on the technologies of the particular civilization) to raise the stones into position on top of other stones. Then they carved away the bosses and finished the exterior of the wall or column in whatever way they wished, either by hand-carving the final designs into the surface, or adding a plaster or limestone coating.

This photo shows unfinished column drums on the north wall of the Acropolis in Greece. You can see peg-like protrusions sticking out from the sides of several of them. Those are the bosses. They would have been used to lift the drums to their final position, and then they would have been carved off.



Figure 29.3: Unfinished column drums, Acropolis

To construct columns out of “drums,” the ancient Greeks and Romans cut guide holes in the center of each drum and used wooden pegs to guide the next stone into the proper position on top of the previous stone. Then they finished the exterior of the column so that it looked like a single piece.

Coaches who are guiding teams through a series of “baby steps”

toward a more-effective way of working may have to do something similar.

When we introduce a practice such as defining a “definition of done” for handing off requirements from an analysis team to a programming team, we intend to discontinue that practice once the analysts, testers, and programmers have learned to collaborate directly.

When we introduce an idea such as “relative sizing” to support team-level estimation, we intend to discontinue the practice once the team has learned to slice work items vertically into similarly-sized chunks, and to use forecasting to plan their work.

When we introduce a time-boxed iterative process model to help a team manage its work flow, we expect that eventually the team will learn to operate in continuous-flow mode, and we can dispense with the explicit iterations.

We will not necessarily move directly from the current state to the ultimate target state in a single step, and some of the artifacts we put into place early in the process may be removed as we advance.

So, as you read about step-by-step ways to move toward the goal state, don’t worry if you see practices being recommended that you know aren’t ideal. They are being recommended at particular stages in the progress of the organization or team, for specific reasons; typically, because the team hasn’t yet learned to collaborate at a more-effective level, or hasn’t yet learned some particular technique or practice. Much of this temporary scaffolding can be discontinued at an appropriate point in the future.

29.5 | First Steps in Silo-Busting

The larger the doll, the harder it is to break. Therefore, a sensible starting point for step-by-step structural improvement is the

smallest doll: Collapse specialist teams into cross-functional teams *within* a given category of activity *within* each technology area.

This will affect primarily the *software design, creation, delivery* and *infrastructure support* categories , as they tend to be broken up into more specialized teams than the *operations* and *production support* categories.

Bear in mind this chapter is strictly about silo-busting. There will be corresponding and mutually-reinforcing changes in *process* and *technical practices* happening at the same time. Those will also be done in a measured, stepwise way.

29.5.1 | Analysis, Test, Design, Programming, Build, CI

Let's look at the category of work around analysis and requirements, solution design, low-level testing, programming, solution build, and continuous integration.

29.5.1.1 | Step 1: Create Product-Aligned Development Teams

As part of our initial restructuring, before we begin the recurring series of experiments to drive incremental improvement, we'll preemptively break up some of the functional silos within this area of practice. Where the organization had a matrix structure comprising "fungible" teams for analysis, testing, design, programming, and build scripts and continuous integration, we'll combine those functions in ways that are relatively easy to accomplish given current organizational constraints.

We'll reshuffle people into product-aligned development teams with the following characteristics:

- aligned with products - each team supports a product; they

are not “fungible resources” that can be assigned tasks pertaining to different products; note this goes hand-in-hand with shifting from a project-focused delivery model to a product-focused one;

- stable - the teams are meant to exist indefinitely; rather than forming a team when a project comes up, we prefer to give long-lived teams work pertaining to a product; note: this doesn’t mean individuals have no options to change teams in order to learn new things and advance their careers;
- cross-functional - each team has people on board who have the skills necessary to support the product; in this general category of activity, those skills will mainly concern analysis, testing, application design, and application programming.

Note that these teams will not be able to take on end-to-end responsibility for an application that cuts through the enterprise technical infrastructure. That’s a longer-term goal, but this early in the transformation program the groundwork has not been laid for it. These aligned teams will have responsibility for a given product within a given technology stack only.

The initial teams also won’t be able to support all aspects of the application. This example concerns activities such as solution design, programming, testing, and requirements analysis. It doesn’t include activities around builds, continuous integration, database support, network support, deployment, infrastructure management, operations, or production support. Baby steps.

We’ll establish “service” teams for some of the other functions in this category, with the longer-term goal of folding these activities into the development teams, as well:

- UX, Usability, Accessibility, Localization
- Build Scripts, Continuous Integration support

Other existing service teams will remain in place at this time:

- Database
- Network
- Security

29.5.1.2 | Step 2: Encourage Collaboration Across Certain Roles

Some of the roles in this general area of activity have natural affinities. It's sensible to encourage collaboration between people in these particular roles as a way to begin softening the boundaries between roles.

Functional silos with natural affinity:

- analysis and testing
- design and programming

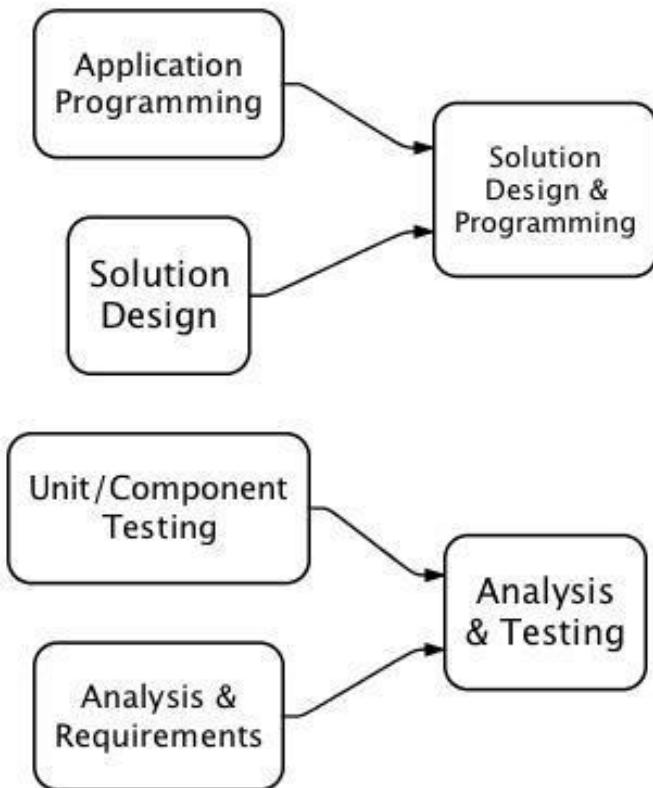


Figure 25.4: Role affinities in development (1)

There's natural affinity between the work of requirements elaboration and the work of creating test plans. It tends to be easy for analysts and testers to collaborate.

There's natural affinity between software design and programming. In organizations that segregate these activities, it tends to be easy to collapse those silos. In many organizations, these activities will not be split to begin with, and there's a little less work to do to break silos.

29.5.1.3 | Step 3: Create Definition of Done and Ready

The terms *definition of done* and *definition of ready* are widely used in the Agile community to refer to explicit rules for handing off entire User Stories. People familiar with kanban systems speak of *explicit policies for pull*.

Here, the idea is generally the same, but the scope is smaller. When we have, say, an analyst and a tester on a development team, and they're still at the level of proficiency where they hand off incomplete items to one another for specialized work, then the usual starting point is that they have no explicit rules for those hand-offs.

Each person hands off work items when they feel they've done what they're expected to do. It's common for the recipient to need further clarification or refinement of the item before they can begin their part of the work. The result is irregularity in flow, and a possibility of miscommunication.

An initial step to smooth this out can be to make the hand-off policies explicit. These written rules can be discarded later, when the parties have learned better ways to collaborate. This is only a baby step to get things moving in the right direction.

29.4.1.4 | Step 4: From Hard Hand-Offs to Safe Hand-Offs

In a 2013 blog post, “How Does Collaboration Begin?” I share a story about that question. It was posted online, and a number of well-intentioned Agile practitioners replied; but they didn’t answer the question. They merely reiterated a description of how well-functioning Agile teams work together. The question wasn’t about end states; it was about *how to begin*. What’s the first step? What’s the second step?

I used an analogy from a handgun safety class my family and I took. Here’s an excerpt:

One of the things you might do with your firearm is to hand it to someone else. As you might imagine from many years of watching movies and television, there are many different ways to hand a firearm to someone else. You could, for example, drop it on the floor and kick it across the room, loaded and cocked. This is analogous to the way software development specialists hand off work between functional silos; you know, the old “throw it over the wall” thing.

...here's what we were taught about handing off a firearm: First, you clear it, then you hand it off. “Clear” means you check the magazine, action, and chamber to make sure there's no ammunition in the gun. Then you ask someone else to look in the magazine, action, and chamber to confirm there's no ammunition in the gun. You're still holding the gun at that point. Making sure the gun is pointed in a safe direction at all times, you can then hand it to someone else. The procedure for that is to look at the person while he/she looks at the firearm. You present it to him/her, pointing it in a safe direction at all times, stock first. You don't let go of it until he/she verbally confirms that he/she has a firm grip on it. You look at the firearm to see for yourself that it appears as if he/she does, indeed, have a firm grip on it. Then you can let go.

On a software development team of specialists, a typical work flow is for an analyst to hand off an artifact to a programmer, who later hands off an artifact to a tester. These various interim artifacts eventually become a piece of working software. Traditionally, the hand-off procedure is to “throw it over the wall” to the next functional silo in line. It's exactly the same as dropping a loaded gun on the floor and kicking it across the room,

spinning in all sorts of unsafe directions and colliding unsafely with all sorts of unsafe obstacles.

With a safe hand-off, we do the same thing. The person handing off work explains in great detail everything about the work, based on the Definition of Done. The person receiving the work makes very sure they understand everything clearly, and their own Definition of Ready has been met.

Depending on circumstances, and particularly for a software development team comprising the roles suggested in this section, another mechanism for safe hand-offs is known as The Three Amigos. Created by Agile consultant George Dinwiddie, The Three Amigos brings together an analyst, tester, and programmer just at the point when the team pulls a work item (such as a User Story). They sit down together and make very sure everyone has clarity about what is to be done. It's a mechanism for the over-communication I mentioned.

29.4.1.5 | Step 5: From Hand-Offs to Joint Work

As people become familiar with what their team mates need from them for each particular hand-off, the over-communication involved in the “safe” hand-off procedure starts to feel overly formal and time-consuming.

At that point, people are probably ready to begin doing the work jointly, working in pairs or slightly larger groups. The analyst now contributes to test plans, and the tester participates in analysis. Their roles are beginning to collapse into a single role.

29.4.1.6 | Step 6: Repeat at the Next Level of Silos

For development teams, there may be two or more “levels” of functional silos that can be broken in this way. Now that the analysts and testers are merged, and the designers and programmers are

merged, we can collapse more siloes and cultivate more generalists on the team.

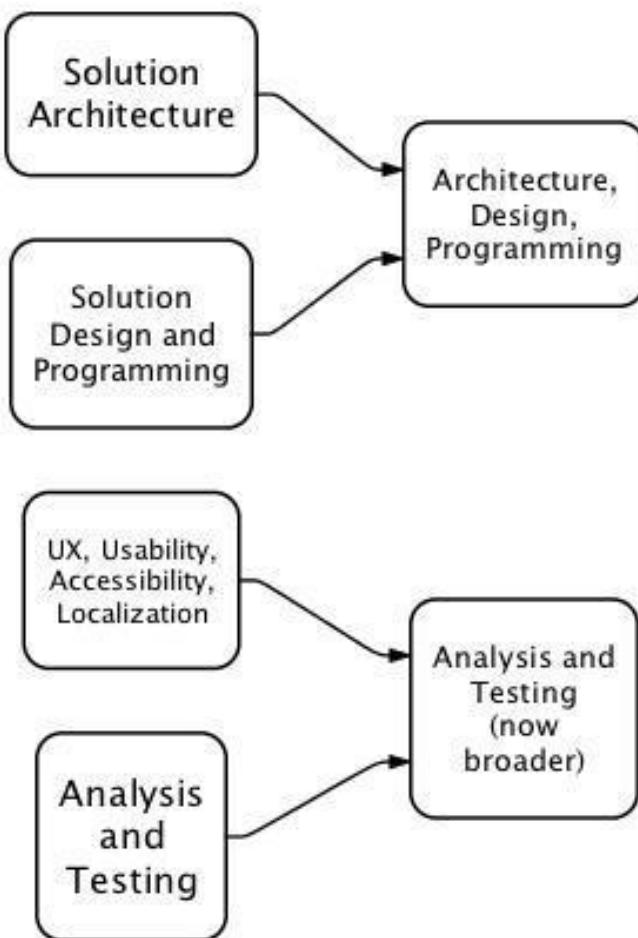


Figure 25.5: Role affinities in development (2)

This iterative process of merging previously-siloed roles on a

development team can be taken to whatever level makes sense in context to support the competitive capabilities that represent the business goals of the transformation.

29.4.1.7 | Step 7: Fold In Support for Builds & CI

For development teams that are doing well after Step 6, the next easiest baby step for silo-busting is to start migrating responsibility for builds and continuous integration support from the separate service team into the various development teams.

There is value to the organization in deciding on a single build/CI toolchain per technology stack, but there is no need for all activity around builds and CI to be isolated in a functional silo. Development teams are well able to handle this work on their own. This will eliminate another cross-team dependency with its attendant delays and inevitable miscommunications.

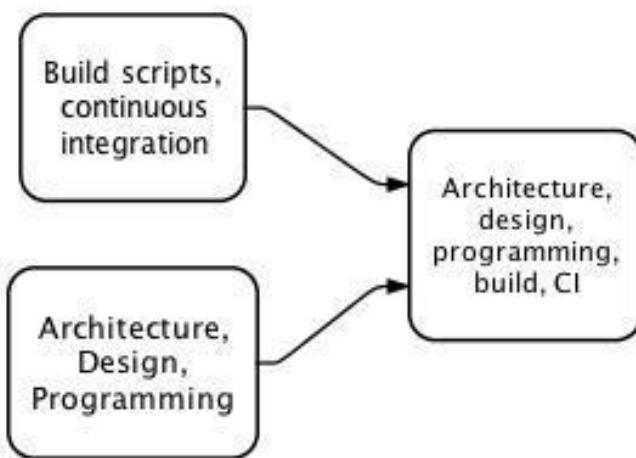


Figure 25.6: Role affinities in development (3)

Build scripts and CI servers are not difficult for product teams to manage, but there are other functions of IT operations that have traditionally been centralized, and for which a degree of central control and standardization is valuable. See section 29.5 below for more on the topic.

29.4.1.8 | Step 8: Collapse Programming and Testing Silos

The next logical step in collapsing silos within product development teams is to start to merge the programming-related and the testing-related activities. We can use the general formula we used before, starting with safe hand-offs and gradually moving toward more-effective methods of collaboration, and finally extending team member skill sets to cover both sets of activities.

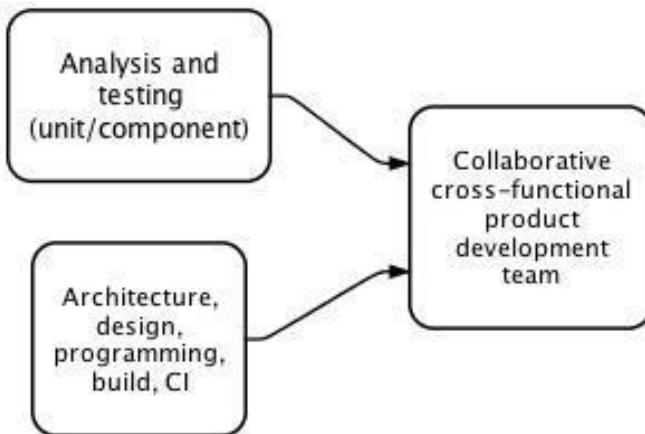


Figure 25.7: Role affinities in development (4)

This chapter doesn't go into improvements in process or technical practices, but the team structure that results from this collapsing of

functional silos can enable good practices such as

- specification by example
- acceptance test-driven development
- test-driven and behavior-driven development
- whole-team exploratory testing
- *more*

It can also give each team member a broader perspective on the work. A programmer who learns to wear a testing “hat” will approach solution design in a way that enables testing. That in itself tends to result in designs that are more robust, resilient, simple, and loosely-coupled, as well as source code that is better organized and more understandable, than when programmers aren’t in the habit of thinking like testers.

Similarly, a tester who learns basic coding skills will be able to take advantage of the many test automation tools available, as well as using automation to support genuine testing activities (as opposed to validation or checking).

An analyst who learns something about testing and programming will be better equipped to write clear and meaningful requirements.

Similar benefits accrue across all the various skills that had been siloed previously.

29.4.2 | Deployment, Release, Operations, Support

Let’s apply the same step-by-step approach to busting silos around another general category of work: deployment, release, operations, and production support.

Most organizations that I’ve seen or heard of have numerous specialized teams, each of which carries out just one type of

task. One team provisions environments. Another team moves applications from one environment to the next. Another team handles production releases and rollbacks. Another team monitors the production environment.

Many large organizations isolate the following activities in separate service teams that support all the development teams:

- migrating code through environments
- releasing applications into production
- production operations
- production support

There may be even further, fine-grained breakdown of the work. For instance, within the infrastructure engineering or platform engineering team, there may be specialized sub-teams that handle just one operating system; a sub-team for Linux, a sub-team for Windows, etc. There's also the same separation of teams per technology stack as we saw for software development activities.

Collapsing all these silos can take time and patience, especially when we consider that nearly all this work is probably being done manually, and nearly all of it will have to be automated or at least streamlined to achieve goals like *Respond to Change*, *Shape the Market*, and *Retain Customers* (see “Leverage: Where’s the Fulcrum?”).

The end state is likely to require a single team or small group of teams to support each product or product line from concept to operations. That means collapsing the broader development and operations areas. For now, we’re concerned with the initial steps we can take to begin the process of collapsing silos. We’re not ready to fold development into operations just yet.

29.4.2.1 | Step 1: Restructure Service Teams

To enable collaboration across previously-siloed specialists, we want to collect closely-related responsibilities together. Responsi-

bilities that have natural affinities may include:

- provisioning environments and moving code through environments
- operations and first-line production support

29.4.2.2 | Step 2a: Collapse Provisioning Environments and Migrating Code

We know that the end-state vision calls for creating environments dynamically as needed (to the extent possible). Ultimately, the work around provisioning environments has to be folded into the development process, picking things up where continuous integration leaves off. That may be too large a step to take at the outset.

A practical initial state may be to collapse the silos for *provisioning environments* and *moving code through environments*.

The *status quo ante* in many organizations is to dictate how certain “standard” environments will look, and requiring application designers to fit their solutions into that.

We want engineers to start thinking in an application-focused way rather than a platform-focused way, to encourage thinking about provisioning from the point of view of each application’s needs.

Engineers responsible for provisioning environments tend to create a server instance and try to keep it stable and operational for as long as possible. This is traditional infrastructure management.

We want engineers to think of server instances as disposable resources that can be created or reset on demand, when applications need them. This is dynamic infrastructure management.

If the same team provisions environments and migrates application code through them, they will naturally start to think about these things in a holistic way. So, this is a baby step in the direction we need to go.

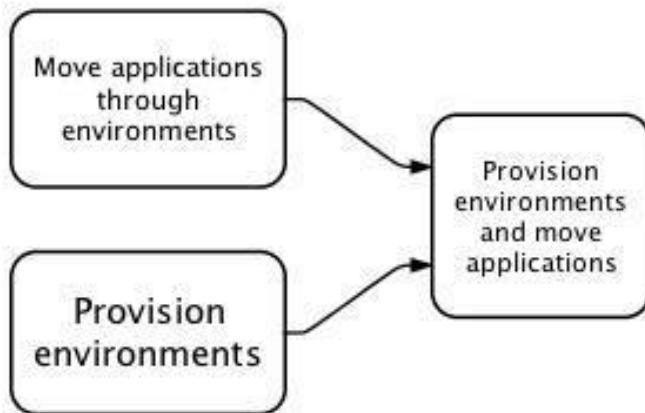


Figure 25.8: Role affinities in engineering

In conjunction with this structural change, we'll be introducing automation for provisioning server instances in a consistent way. Before long, we'll start maintaining the provisioning scripts under version control, moving toward *infrastructure as code* (Morris).

29.4.2.3 | Step 2b: Collapse Operations and First-Line Support

Ultimately, we want a product-aligned team to handle operations and production support, as well as development and deployment, to remove delays and waits from the process.

The *status quo ante* in many organizations is that these two functions are separated. One team monitors production systems, but mainly with the goal of keeping them up and running. Another team receives trouble reports from users, performs some degree of triage, and if they can't solve the problem they hand it off to second level support.

One mechanism to remove delays and waits from the process and

to help ensure the stability of production operations is to adopt the so-called *serverless* mentality. It means we're thinking mainly of the applications and their users, rather than the servers on which the applications run. In contrast with the past, an application is long-lived and the servers that support it may come and go. Standardization and automated provisioning of servers enables this by simplifying and rationalizing that work.

A baby step in that direction is to combine the operations team and the first-level support team. For now, we'll still have second- and third-level support handled by different teams (third-level support is probably the development team). Eventually, the product-aligned cross-functional team will handle everything, but we aren't there yet.

29.4.2.4 | Step *n*: Collapse Operations/Support and Development

This collapse will become feasible after the development area and engineering area have separately improved their structures, processes, practices, and tools to a certain extent. Merging these broad areas of responsibility into a single product-focused team is a big change, and one that requires considerable ground work to be laid before it can happen.

There may be several interim steps here, as test automation grows from the development side forward, and dynamic infrastructure management grows from the engineering side backward (where “forward” means “toward production”). I sometimes visualize this as two sets of tendrils reaching toward each other, gradually entangling, and eventually becoming One.

29.4.3 | Collapsing the Big Dolls

Client leadership may decide to stop collapsing silos at some point, when they’re satisfied with the way the organization is operating. But if silo-busting is taken to the *nth* degree, we will eventually collapse the technology-based teams into a single, end-to-end, product-aligned team. That team will include all the skills needed to deliver product features and modifications from mobile through the backiest of back ends.

It boils down to a repetition of the process of

- putting people together
- encouraging them to collaborate directly
- cross-skilling through collaboration
- blurring the lines of specialization between them

guided by the Cycle of Change, to ensure we don’t go off on a tangent and we remain focused on the business goal.

29.4.4 | Ongoing Iterative Improvement

It will almost always be feasible to find the next “level” of affinity between two roles or teams, and collapse them using very small changes controlled by the Cycle of Change, with clarity about the impact of each change and the ability to quantify the impact of the change. Affinities at “higher” levels depend on our having collapsed teams at “lower” levels of affinity, as suggested by the progression described in this section.

While the team structure in the engineering area may be more complicated than that in the development area, the changes in processes, practices, and tooling are likely to be more challenging for engineering than for development. So, there’s more early transformation work to do in those areas than there is with respect to structure.

29.5 | Special Considerations When Decentralizing Responsibility

Changes in team structure that distribute responsibility for a function from a previously-centralized service team into numerous development teams introduce a new need: Maintaining consistency and simplicity in the technical infrastructure without creating a bottleneck for product delivery.

When we decentralize responsibility for things like build scripts, continuous integration servers, version control systems, provisioning of environments, and other functions, we need to avoid numerous different tools or versions of tools from proliferating in the environment.

We're pulling responsibility for something out of a central group and distributing it across multiple teams. Those teams will be focused on supporting different product lines or value streams. They won't automatically coordinate changes in the infrastructure or platforms they use.

Depending on the nature of the work, this can be managed in one way or another without reverting to 20th-century controls.

29.5.1 | Central Definition With Self-Service

Version control systems, platform engineering, dynamic creation and destruction of virtual machines and containers, and other needs can be defined by a centralized team and then presented to product teams via a self-service system. The self-service system limits choices to approved products and versions of products and assures product teams will make choices consistent with organizational standards.

A pitfall to be aware of is that when there is a centralized team, such as a platform engineering team, their work is often prioritized

below product-related work that has an obvious connection with customers. One way to mitigate this is to ensure every change to any application or to the technical infrastructure is justified on the basis of customer impact.

The naive view that a change has to have an immediate and obvious impact on customers leads people astray. An infrastructure change whose impact is many steps removed from any customer is still necessary to support customers properly. The exercise of justifying those changes in terms of customer impact leads everyone involved to a clearer and more-consistent understanding of customer needs.

It's generally best to provide self-service facilities to product teams to the extent possible. This will minimize cross-teams dependencies with the central team and help avoid delay, miscommunication, back-flows, and rework while making delivery forecasts reliable and predictable.

29.5.2 | Documented Standards With Definition of Done

Continuous integration servers, build tools, server configuration, API design standards, naming conventions and the like can have documented standards. Whenever a product team makes a change that is governed by the standards, they know where to go to to find the standards and possibly working samples that demonstrate how to apply the standard. With that in place, product teams must include checks for standards compliance in their Definition of Done for work items.

Beyond compliance, there will be many times when changes to a standard are necessary. The documented rules must include an appropriate procedure for changing the standards. The guiding “beacon” for the transformation program provides natural guidance for these rules: The key requirement is that changes will not destabilize production operations. (See “Leverage: Where’s the Fulcrum?”)

A second requirement for changing organization-wide standards is that existing solutions must not “break.” Some kind of versioning scheme or other safeguards to assure backward compatibility (within reason) must be developed concurrently with the first change of this kind.

29.6 | Team Size

In 20th-century organizations, team size was determined by adding up the number of people who were assigned to a project, either full-time or part-time. The same individual might be counted multiple times, if they were assigned to multiple projects concurrently. This led to teams sizes that could range over the 100 mark.

But these were not “teams” in the sense Lencioni writes about. They were too large and dispersed to function cohesively as real “teams,” and the members were not dedicated to any single project. We would consider them “work groups” today, rather than “teams.” The work groups were formed when a project was formed, and disbanded when the project was closed.

29.6.1 | Practical Team Size for Collaboration

Today, the idea of a “team” is a little different. It’s a cohesive group of people who interact frequently and collaborate closely to achieve common goals, all dedicated to a single initiative at a time. It has the characteristics of a good team as defined by Lencioni.

To that end, the size of a team has to be amenable to collaborative work, and preferably collocated work. In my experience, teams as small as 2 people have been effective, depending on the nature of the work they performed. Teams as large as 12 to 15 people have also been effective and able to function cohesively.

But once you reach a size of around 12 to 15 people, teams start to divide spontaneously into sub-teams. They generally split up along logical “seams” in the work. For a development team, that could mean people interested in front-end development would gravitate together, leaving people interested in back-end development to form a separate sub-team.

It’s not wrong for teams to be divided, but it’s best to divide them mindfully and with purpose than to allow it to happen at random and without visibility.

For instance, in the example just given, we probably wouldn’t want the front-end developers separated from the back-end developers. If we thought it was appropriate to split the team, we’d create two cross-functional teams with responsibility for some distinct subset of the product, organized in such a way as to minimize cross-team dependencies. That’s unlikely to result from random chance.

One challenge with larger team sizes is that with more people involved, there are more lines of communication. Here’s an image found on StackOverflow that illustrates the point. I don’t know the origin of the image.

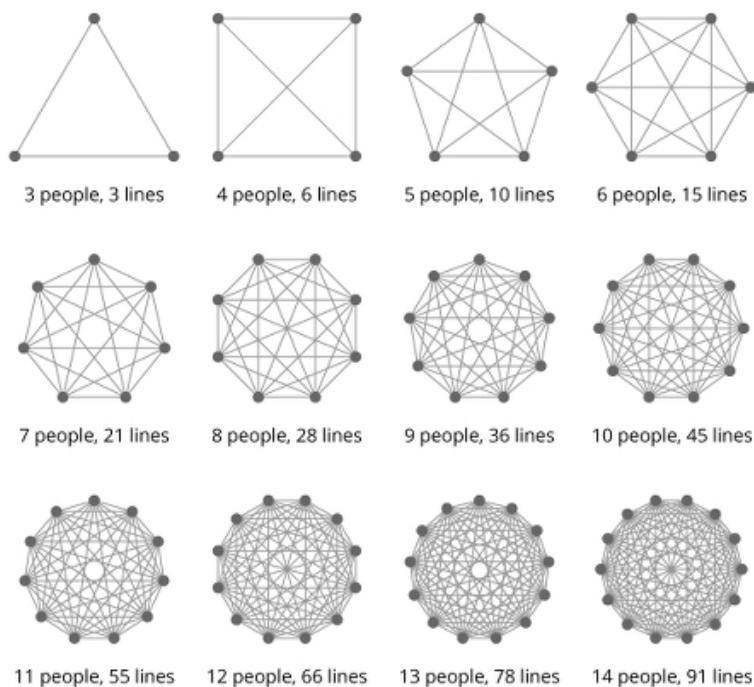


Figure 25.9: Team size and lines of communication

Intra-team communication overhead can overwhelm delivery effectiveness as team size grows beyond natural limits for collaboration.

On one engagement, I was the delivery lead as well as technical coach for a team. (That's not a good idea; the two roles have conflicting mandates! But stuff happens.) My manager approached me one day to remind me that the team was not on track to meet its commitments (this was an agile-in-name-only waterfall project).

He asked me how many more developers I needed to catch up with the delivery plan. I suggested we reduce the team from 8 to 6 developers. He frowned and asked whether some of the developers weren't pulling their weight. He was already reaching for his Firing

Pen.

I assured him everyone was pulling their weight. I wanted to reduce the communication overhead on the team so that they could move faster. He genuinely did not understand the concept. He agreed, but told me that if the idea didn't work, it would be on me.

The team of 6 increased the pace of delivery and completed all the required work before the deadline. The team went from 28 lines of communication down to 15, making it easier to get things done. The improvement in communication efficiency more than compensated for the smaller number of developers.

29.6.2 | Collapsing Silos Increases Team Size

As we collapse functional silos, we'll be increasing the number of individuals on each team. For example, if we combine a team of programmers with a team of analysts and a team of testers, we may end up with a pretty large group:

Original configuration

- Team A = 4 programmers
- Team B = 4 programmers
- Team B = 4 analysts
- Team C = 4 testers

Consolidated team:

- Team ABCD = 8 programmers, 4 analysts, 4 testers

That's 16 people, probably beyond the practical limit for a group to function cohesively as a team in the full sense of the word. It also may not be an ideal mix of skill sets for the work at hand. If we have aligned teams with products properly, this large team may also contain more people than necessary to fulfill its mandate.

29.6.3 | Multiple Teams Supporting One Product

It's possible we can identify a logical breakdown of the product the team supports such that two teams could operate with minimal dependencies on one another to support the product. We could then have:

- Team New-A = 4 programmers, 2 analysts, 2 testers
- Team New-B = 4 programmers, 2 analysts, 2 testers

This means coordination to manage multiple work streams that support the same product.

29.6.4 | Coordinate Changes in Structure, Process, and Practices

It's possible that with corresponding improvements in process and team member skills the same work can be done by fewer people, but initially the teams will be larger with each silo collapsed.

As we go forward with further silo consolidation, Team New-A may end up looking like this:

- Team New-A = 4 programmers, 2 analysts, 2 testers, 2 infrastructure/platform engineers, 1 DBA, 1 UX specialist

That's close to the practical limit for cohesive team collaboration. Improvements in process, team member skills, technical practices, tooling, and progress with automation can alleviate the need for large teams.

For that reason, changes in structure, process, and practices must be closely coordinated. One area must not be allowed to race ahead

of the others. The second level of silo-busting can only proceed when team members have settled into each succeeding level of collaboration and cross-skilling.

29.6.5 | Beware of Magic Numbers

As a technical coach, you may come into an organization that's already been damaged by a previous Agile transformation. You'll have to adapt to the client's actual starting point. Every company won't have the same situation as presented in this chapter.

David Anderson tells a story of a client that told him they had hired 400 new Product Owners, and didn't know what to do with them. When he asked why they had hired so many Product Owners, they explained their Agile coach had told them each team had to comprise 6 people, and each one needed its own Product Owner. Anderson asked whether they had 400 products. They did not.

You may have heard there's a magic number of "seven plus-or-minus two" team members for effective collaboration. That's a rule of thumb, not a rule carved in stone.

Your experience may differ, but I've seen teams ranging from 2 to 15 members working effectively together. It depends on the kind of work they're doing.

Furthermore, there's no "rule" that says each team requires any specific number of people in particular roles. The skillsets needed on a team depend on the kind of work they're doing.

In fact, there are situations in which a *work group* of up to 40 people can be effective, when the work they do doesn't particularly call for close collaboration. A "true" team isn't necessary in all situations.

I've been in large organizations that had 600 to 800 software development teams. When the coaches went out onto the floor to work with the teams, we discovered very few of them were

actually “software development” teams. They performed various tasks relative to the company’s IT function.

Previous Agile coaches had advised management to create a large number of small teams and call them all “development” teams. Each of them was required to have a fixed number of analysts, testers, and programmers.

Therefore, the team members had those titles. None of that reflected the work they actually performed or the skills they actually possessed. The situation made it hard for us to determine how best to help the organization.

Your colleagues on the Consulting Team who work with higher levels of management may not easily relate to this situation. Most people in that sort of role don’t know what the technical work really looks like. Calling everyone a “developer” seems fine to them. So, you can’t necessarily count on the results of the consultancy’s initial assessment to understand the client’s real starting point.

29.7 | Security

If we look at IT operations strictly from a Lean point of view, we would want to incorporate security seamlessly. But for legal and regulatory compliance reasons as well as to minimize the risk of collusion to commit fraud, it makes sense to separate the security function from the rest of IT. This can go so far as to have a separate management hierarchy just for security, that doesn’t report to the CIO, and doesn’t depend on the IT department’s budget.

This can be done in a way that doesn’t introduce excessive delay or bureaucratic overhead for the delivery pipeline. Many security issues are “known” and it’s feasible to build checks for them into the delivery pipeline.

Other security-related issues pertain to application design and server configuration. Those matters can be documented, included

in teams' Definition of Done, and verified as part of automated test suites.

The separate security group would be responsible for defining those things and for helping development and engineering teams understand their role in maintaining security. They would also be responsible for proactive security-related activities and testing.

The security group will have a great deal to say about network perimeter configuration, among other things. But they will never become part of a product-focused cross-functional team.

29.8 | Summary

In this chapter, we've started to see a series of steps for *beginning* to collaborate. So far, in the present section, the steps have been:

1. Establish cross-functional teams. That puts people with different specialties together. It's possible they have never occupied the same physical space at work. Initially, they may continue to work separately and pass work to one another. But just the fact they're in the same space will help them learn a little about each other's work, and about what each is looking for from the other.
2. Create Definition of Done and Definition of Ready for each type of task the specialists on the team carry out. The purpose is to make it explicit just what each specialist requires to begin their work, and just what has to be added to the work item for it to be considered "done" by that specialist. (These formalities can be dropped later in the transformation when teams are collaborating smoothly.)
3. Encourage collaboration across certain roles. Some roles have natural affinities that make it easy for practitioners to relate to one another's work, such as analysts and testers. Initially when they sit together they may watch one another work,

or work independently but near enough to see each other's work and talk about it. It's pretty automatic that they will start to ask what the other person needs, and begin to tailor their work to make the other person's job easier.

4. Encourage the people to collaborate directly together. A baby step in that direction is to shift from hard hand-offs to "safe" hand-offs. That's a semi-formal process for passing work back and forth that involves "over-communicating" details about the work. The people must share a lot of information in the process of performing a safe hand-off, and they begin to learn more about each other's work.
5. Once safe hand-offs have become routine and feel "too formal" to the people involved, a natural next step is for them to *do* the work of both roles jointly. Rather than just sitting next to each other as in Step 1, they now sit next to each other and jointly complete tasks. For example, if they are an analyst and a tester, then at this stage of cultivating collaboration the analyst is helping create test plans and the tester is contributing to analysis.
6. Keep improvements in structure, process, and practices in sync. If one area goes faster or slower than the others, there will be friction.

The same pattern applies to all the specialized silos that may exist in the IT organization.

30 | Incrementally Improving Estimation

The only thing I cannot predict is the future.

— Amit Trivedi

Estimation plays a large role in many endeavors, whether we’re estimating the cost of developing a next-generation fighter jet, forecasting the number of hurricanes to expect next season, or judging whether to exit the highway now to refuel the car or keep driving until the next town. Discussions about estimation methods usually go off the rails because people make different assumptions about the purpose and scope of estimation.

This book is generally about organizational transformation that involves IT operations, and especially about providing coaching to technical teams that are involved in such a transformation program. Therefore, the *purpose* of “estimation” in this context is short-term planning of work, and the *scope* is a single technical team. It’s got nothing to do with fighter jets, hurricanes, or fuel consumption.

30.1 | Who Needs to Estimate?

For our purposes, “who” means “what kinds of teams.”

Software development teams create and modify application code. They may have a list of features and change requests to implement. Their stakeholders want to have an idea of when those features and changes will materialize.

An operations team doesn’t work from a list of features and change requests. They monitor production systems continually and

deal with whatever comes up. Estimation is meaningless in their context.

An infrastructure engineering team, service team (database, network, etc.), or third-party product support team may have a queue of requests, but generally does not have a “project” with defined features to deliver. They may apply patches and product updates and respond to requests and bug reports. Estimation is meaningless in their context.

So, we’re talking about software development teams, and how they estimate the planned work that’s coming up in the near-term future.

In organizations that are still in the early stages of a transformation program, some or all software testing activities may be segregated into specialized testing teams. Until responsibility for testing becomes folded into the development teams, the testing teams will also have to provide estimates for the time they will need to test applications prior to their release.

30.2 | Factors Affecting Estimation

It can be relatively harder or easier to estimate feature delivery time depending on various factors, such as:

- how familiar the team is with the kind of work to be done;
- how frequently the team tends to be interrupted while working on value-add features;
- the degree of variability in work items; and
- how much time may be consumed when the team encounters unexpected difficulties in working with the code base owing to “cruft” or poor software design;

30.2.1 | Familiarity With the Work

At times, the code changes a development team makes to a product will be in the nature of “routine” work. That is, the code they must write or modify is similar to other code they’ve worked on quite a lot. They will be able to estimate this sort of work pretty well.

At other times, the team must do something new to the code base. They will be less certain about how much time the work will take, as they haven’t done the same kinds of things before.

Estimates will improve as the team gains familiarity with the code base and practices general technical skills on a regular basis, through code dojos and similar exercises.

30.2.2 | Interruptions

Interruptions in creative work have two significant effects:

- reduce the proportion of total work time that can be devoted to value-add activity; and
- cause two context-switch events - one at the point of interruption and another at the point when people return to the work they had been doing.

When teams aren’t sure how frequently they will be interrupted or how long the interruptions will last, it’s difficult for them to provide meaningful estimates for planned work.

In this approach to organizational improvement, we protect “maker time” for development teams to limit the number and duration of interruptions from planned work. We also incrementally collapse functional silos to reduce the number of cross-team dependencies, which has the effect of reducing the number of interruptions when one team needs something from another team.

Estimates will improve as the team is interrupted less frequently due to reductions in cross-team dependencies.

30.2.3 | Variability in Work Items

When we measure delivery performance by tracking mean Cycle Time per work item (“task”, “story”, “ticket”, etc.), the *variation* in mean CT may be high or low. When there’s high CT variation, estimates will have a large range or a low confidence level.

Estimates will improve as the team improves its ability to slice the work into similarly-sized chunks and deploy subsets of features controlled by toggles, as these practices will reduce CT variation. Teams will be able to give estimates with smaller ranges and higher confidence levels.

30.2.4 | Effect of Crufty Code

Crufty code affects the team’s ability to provide meaningful estimates, as they never know when a seemingly-minor change will involve ripping up a chunk of badly-designed code, or will involve structural or architectural changes in the application that wouldn’t have been necessary had the design been clean to begin with.

Estimates will improve as the team improves the health of the code base through incremental refactoring and, at times, special refactoring efforts.

30.3 | Incremental Improvement

As teams make improvements in various aspects of their work, they will be able to use simpler and lighter-weight methods to estimate or forecast delivery times.

30.3.1 | Absolute Time Estimates With Interruptions

The typical starting point in large organizations is that teams must provide estimates of delivery times in absolute terms, without regard for any factors that affect the quality of their estimates. The standard coping mechanism for technical workers is to pad their estimates by a factor of 2 to 4 to allow for unexpected delays and interruptions.

30.3.2 | Ideal Time Estimates With a Load Factor

The first step in incremental improvement can be to make estimates in terms of *ideal time* rather than absolute time.

Ideal time is the time people believe the work would take if there were no interruptions or context switches. The *load factor* is the proportion of work time that is typically taken up with meetings and other interruptions.

30.3.3 | Relative Sizing Pegged to Ideal Time

The second step in incremental improvement can be to introduce the concept of *relative sizing* for work items. With that approach, we judge the relative “size” of work items as compared with other work items. So we might have an item of size 3, and the next one seems bigger so we call it a 5, and the next one is smaller so we call it a 2, and so on.

Agile practitioners use a technique called Planning Poker to size their work items during short-term planning sessions. They call the work items User Stories and they size them in terms of Story Points.

Teams that are uncomfortable with using a rubric that isn't tied to clock time may initially use a formula to convert hours into sizes. They might decide one Story Point is the same as half a day, for instance.

30.3.4 | Relative Sizing with Derived Time

A further improvement can be to use relative sizing as it was originally intended; that is, with no explicit reference to time. As the team's delivery performance is observed, the times required to complete work items of a given size is visible.

With this degree of improvement, we can say the team is using *forecasting* as opposed to *estimation* for short-term planning. They are using empirical observation of delivery performance along with gut-feel sizes to forecast delivery performance. This typically provides stakeholders with better information about when to expect delivery of features.

30.3.5 | Counting Work Items

As a team improves in other areas of their work, they learn to define work items that are approximately the same size. At that point, they can simplify their forecasting method by dispensing with relative sizing. Continuing to use empirical observation to forecast delivery performance, they count the work items rather than the "points."

30.3.6 | Absolute Time Estimates Without Interruptions

Once fully cross-functional teams are operating consistently in properly-configured collaborative work spaces with protection of their maker time, they are no longer affected by cross-team dependencies or interruptions.

At that point, they can estimate delivery time for work items in terms of absolute time. That's the way most stakeholders would prefer it. To achieve this, all the external factors that make time-based estimation unreliable in traditional environments must have been solved.

The incremental improvements prior to this have been in the nature of workarounds for external constraints that made it difficult or impossible for the team to control its work time. With the external constraints finally removed, they can do short-term planning based on time.

31 | Incrementally Improving Code Review

My congratulations to you, sir. Your manuscript is both good and original; but the part that is good is not original, and the part that is original is not good.

— Samuel Johnson

31.1 | The Value of Code Review

The value of code review has been reported from practitioners' experience and research studies so many times that I'm at a loss to pick a reference to cite. Suffice it to say that a knowledgeable review by a "second pair of eyes" is a well-established way to catch and prevent errors from moving forward in a software development/delivery process.

31.2 | From No Code Review to Formal Code Review

If the team is not currently doing any sort of code review, then you can introduce them to a method that's as close to state-of-the-art as they can get, in view of other constraints.

You will probably be running at least one Mob Programming session per day with the team. That's a painless way to introduce code review in the context of collaborative work.

If you’re working closely with the team in their normal context beyond the mobbing session, you may have an opportunity to introduce Pair Programming, and show them how to review code continuously during development.

If there are constraints preventing either of those options, you can recommend the team introduce a formal code review step in their work flow as a prerequisite to declaring work items “done.”

31.3 | Use Checklists

Some teams may be using formal code reviews but aren’t ready to move to a highly collaborative technique such as Pair or Mob Programming. If the team is allowing defects to flow through, there may be ways to improve the effectiveness of the code reviews.

A common situation is that errors of the same general type find their way into production, or at least into a testing team’s jurisdiction, again and again. Something as simple as a checklist can alleviate this sort of problem. You can suggest the team create a checklist of things to look for in the code reviews, to help them remember to ensure known problems aren’t present.

31.4 | Limit the Time of Formal Code Reviews

It’s been found that the effectiveness of deep inspection of source code begins to drop off after about one hour. No matter the knowledge and skill of the reviewer, after an hour of closely examining and thinking about code, they will begin to tire. Limit code review sessions to one hour.

This may be challenging when the team has quite a lot of work waiting for review. As a coach, that’s a good problem to have.

It highlights the formal code review as a process bottleneck and provides you with a topic to present to the team for improvement.

Depending on the cause of the bottleneck, a number of valuable improvement opportunities (beyond the review as such) may fall out from this:

- too much work in process (WIP)
- not enough senior team members
- bottleneck could be alleviated by Pair/Mob Programming
- *other* (depending on local circumstances)

31.5 | Refactor to Increase Review Effectiveness

In addition to the time spent in a code inspection, the size of the code under review is a factor that can affect reviewers' effectiveness. As a coach, you can use this as an incentive to refactor code to reduce code smells such as Long Method or God Class. Not only will such refactoring make the code base easier to understand and reduce the risk of change, but it will also enhance the quality and value of code reviews.

31.6 | From Formal Code Review to Pair/Mob Programming

Pair Programming and/or Mob Programming can usually be introduced to any team at any time, regardless of how they are currently working. If teams currently have a formal code review process, you can introduce these collaborative methods as a complement to the review process or as a replacement, depending on how effectively

the team can use them. Eventually it should be possible to dispense with the separate code review step and depend on collaboration to serve the same purpose.

31.7 | Use Static Code Analysis

Static code analysis tools can perform quite a bit of useful code review automatically. Help the team set up and configure a static code analysis tool to help them streamline their process as well as reduce escaped defects and improve code quality.

32 | Incrementally Improving Branching Strategy

We talk about this and that. There's no rest except on these branching moments.

— Jalal Al-Din Muhammad Rumi

The goal of all technical practices is to support *stable production operations*. The ways teams use version control systems and continuous integration processes play a large role in achieving that goal.

There isn't any single “right way” to manage sources, commits, and builds; but there is an unknown (large) number of “wrong ways.” The technical coach has to understand the client’s environment and systems well enough to help guide choices regarding version control systems, branching strategies, and the details of continuous integration processes.

The technical coach, in collaboration with the rest of the consulting team, will be attempting to achieve a few broad objectives with respect to version control and branching strategies, with the overarching goal of *stable production operations* in mind:

- Decide whether a monorepo or polyrepo setup is advisable in context;
- Decide whether and how to split back-end (mainframe) and front-end/mid-tier version control and branching strategies to support different *competitive capabilities* supported by specific applications and services;

- Decide on a consistent branching strategy based on organizational needs and scale;
- Guide client decision-makers in the selection of appropriate tooling to support the goals;
- Ensure continuous integration practices and tooling are appropriate to support the desired branching strategy; and
- Guide development teams and infrastructure/platform engineering teams in appropriate technical practices to use version control and continuous integration properly.

Some of these changes may require temporary scaffolding as teams make incremental, stepwise progress toward the end state. That is:

- Different software products may be installed at different stages of the transformation. Sources may have to be moved from one version control system to another multiple times before the ultimate business goals are achieved.
- Different teams may change their branching strategies multiple times as they move incrementally toward a desired end state; for instance, from “random” to feature branches, and later from feature branches to trunk-based development.

32.1 | Types of Version Control Systems

For the purpose of supporting *stable production operations*, we can categorize version control systems along just a couple of axes:

- centralized vs. distributed design; and
- support for the desired development work flow.

32.1.1 | Centralized Version Control Systems

With a centralized version control system, there is a single central master copy of the repository (Lionetti). Sources are checked out of the master copy, and commits are written to the master copy.

Examples of centralized version control systems include Subversion, CVS, and Perforce.

32.1.2 | Distributed Version Control Systems

With a distributed version control system, developers can *clone* a repository on their local machine. This enables them to work without a network connection and to commit changes locally, without affecting other developers. They can selectively *push* their local commits to a master copy (there is a master copy, even though the general organization of the tool is distributed).

Examples of distributed version control systems include Git, Mercurial, and PlasticSCM.

32.1.3 | Development Work Flow Support

Each version control system is well-suited to one or more development work flows. The features of each system were designed with a certain work flow in mind.

In general, modern version control systems (whether centralized or distributed) can support lightweight, iterative work flows such as those associated with Extreme Programming. Older systems that were developed in an era when long lead times and extended individual work on code modules were the norm may not enable lightweight work flows.

32.1.4 | Mainframe Version Control Considerations

When the systems involved in supporting a service that calls for dynamic interaction with the market include back-end components running on mainframe platforms, it may be necessary to migrate the sources for those systems to a version control system that is suitable for contemporary development work flows. The products typically found in mainframe environments are not suitable - Serena ChangeMan, CA Endevor, PanValet, Librarian, and others.

Those systems were designed in a time when it was common for individual developers to take ownership of a module for the duration of their work, which might take days or weeks. They also are better suited to taking a snapshot of a release for rollback purposes than they are for continuous incremental commits of small changes to drive continuous integration.

While I want to avoid mentioning specific products in this book, I will mention that Git is the only cross-platform version control system suitable for contemporary work flows that runs on mainframes as well as Unix, Linux, and Windows platforms. As of this writing, it is the system of choice for solutions that span multiple platforms. Git has been ported to zOS by Rocket Software.

It is possible that more alternatives have become available subsequent to publication of this book. Technical coaches are advised to look for appropriate tools.

32.2 | One or Multiple Version Control Systems

Our technical choices must be driven or guided by business goals. Consider Figure 6.2, in which we identify specific systems and

categorize them according to market differentiation, core mission alignment, and so forth.

We have *Consumer Banking* and *Commercial Banking* services that are very high in mission criticality and fairly high in market differentiation. If we correlate those services with the *competitive capabilities* shown in the dependency diagram in Figure 7.1, we might decide that *Consumer Banking* needs to support the capabilities, *Respond To Change* and *Shape the Market*, while *Commercial Banking* needs to support *Retain Customers*.

Both those services are built from components that span multiple platforms, including mobile, Web, mid-tier, and back-end (mainframe) components.

32.2.1 | Supporting Dynamic Market Interaction

To support *Respond To Change* and *Shape the Market*, we want to treat *Consumer Banking* as a vertical slice of functionality, and manage it in a way that enables rapid turnaround of changes. That applies not only to the front-end and mid-tier components, but to the entire slice.

Business drivers are in force that will lead us to break the largest Matryoshka doll - that is, to collapse the teams and tooling for *Consumer Banking* across front-end, mid-tier, and back-end platforms - early in the transformation program. To do that, we'll need to migrate the sources for the legacy mainframe components of *Consumer Banking* into a version control system that can support the new development work flow (probably Git, as of 2019).

A developer working on *Consumer Banking* will clone the repo (probably a monorepo) that contains *all* the sources. The product-aligned, cross-functional team will include skills to work on *all* components of *Consumer Banking*, be they Ruby, C#, Java, COBOL, Chef, Bash, PowerShell, or anything else.

32.2.2 | Supporting Stable Market Interaction

On the other hand, to support *Retain Customers*, the *Commercial Banking* service can be managed in a way that provides for reliable, predictable releases on a fixed schedule that may be months long. The back-end personnel and tooling are already optimized to deliver releases every few months. The client probably doesn't want to invest heavily in significant changes to *Commercial Banking* early in the transformation program.

If such changes are advisable, they can wait. For the time being, the front-end and mid-tier components of *Commercial Banking* can be supported by one set of teams and tools, while the back-end components are supported by another. The back-end sources can remain in Serena ChangeMan or CA Endevor or whatever version control system is already in use.

This approach supports our goal of taking incremental steps that can be halted at any point, leaving meaningful and sustainable improvement in place. Should the day come to modernize *Commercial Banking*, then our two-system version control solution can be changed; it will have been in the nature of temporary scaffolding. If that day never arrives, then the two-system solution can remain in place indefinitely. Either way, the client is in a better position than when they started.

The point is that we want to avoid making this decision based on *dogma* or *personal preference* or *coach's current knowledge of tools*. We want to make the decision that will support the *competitive capabilities* of interest to the client.

32.3 | One or Multiple Source Repositories

Within the scope of each version control system in the environment, we have to choose a monorepo or polyrepo strategy. With a monorepo strategy, a single source repository contains the sources for all the systems the organization supports. With a polyrepo strategy, each product has its own source repository. A hybrid or combined strategy is also possible, and often used.

Joel Parker Henderson offers an excellent summary of the pros and cons of, and potential issues with a monorepo or polyrepo strategy, and various mixed approaches (Henderson). There's a lot of material there, and I won't reproduce it here. I recommend reading his write-up.

Henderson writes:

Neither really seems to result in less work, or a better experience. But I've found that your choice just dictates what type of problems you have to solve.

Monorepo involves mostly challenges around scaling the organization in a single repo.

Polyrepo involves mostly challenges with coordination.

If the services to be supported are independent of one another, and each service supports a different *competitive capability* that calls for a different work flow, then a polyrepo strategy may make the most sense.

If the services to be supported are dependent or there are common libraries shared across services, then a monorepo strategy may make the most sense.

A hybrid strategy can be effective as a way to meet in the middle, to mitigate the inherent challenges of either extreme.

While I value the insights Henderson shares in his write-up, I want to offer a word of warning about this statement:

I've found monorepos to be extremely valuable in an less-mature, high-churn codebase

There's an assumption here that a code base settles down and becomes more stable over time. That may or may not be the case.

We're interested in supporting one or more *competitive capabilities*. If the capability involves dynamically interacting with the market (like *Respond To Market* or *Shape the Market*), it will be in the nature of things for the code to be modified frequently and (mostly) in small ways. For a service that calls for that kind of market interaction, it would be a bad sign if the code base settled down.

That said, there are natural scaling challenges with a monorepo and with trunk-based development (addressed below).

32.4 | Branching Strategies

There are many ways to describe the pros and cons of different branching strategies, and many slight variations on a few similar models. In general, many people have experienced problems and tedium when long-lived branches are used, by whatever name. That narrows our choices somewhat.

The reason so many variations exist is that no two organizations, and not all teams, have the same needs or operate in exactly the same way. A key idea is that *trunk-based development* is an enabler for *continuous delivery*. In turn, continuous delivery is an enabler for *stable production operations*; and for our purposes, *stable production operations* is the primary enabler for all the *competitive capabilities* we aim to support.

32.4.1 | Trunk-Based Development

With that in mind, it seems a no-brainer to choose *trunk-based development* as our branching strategy. But there may be particular needs or constraints in the client's world that lead us to use a variation of that strategy. It's part of the technical coach's role to perceive and understand those needs and adjust the recommendations accordingly.

The trunk-based development site defines the strategy as follows (Hammant, 2017):

A source-control branching model, where developers collaborate on code in a single branch [and] resist any pressure to create other long-lived development branches by employing documented techniques. They therefore avoid merge hell, do not break the build, and live happily ever after.

I find that definition pretty clear, and yet broad enough to allow for local customization to meet unique requirements.

That single branch isn't just any old branch. It's the primary one, called *trunk* or *master* or *main* or a similar term, depending on the version control system in use.

I recall a former client where developers worked against a specific branch, but it wasn't main. The main branch was "protected" from developers and could only be updated by a designated engineer. There were many long-lived, team-specific branches. That sort of arrangement might make sense in an organization that lacks knowledge-sharing or training (or lacks trust), but creates a bottleneck for continuous delivery. It is not a solution that will enable the *competitive capabilities* we're aiming for.

There are a couple of "rules" that support trunk-based development, too:

- don't break the build; and
- always be release-ready.

Can you commit only to trunk all the time and satisfy those rules? Well, maybe, sometimes. Trunk-based development doesn't mean you never have other branches; if you're working alone on a small side project. Probably not if you're on a team that supports a product or service.

Trunk-based development doesn't mean you never cut a branch. It means you don't rely on long-lived branches. Temporary branches that are merged or deleted after a short time are okay. You can cherry-pick commits from temporary branches, as needed. Releases are always from trunk.

Some examples may shed light on the factors to consider when choosing a branching strategy that will meet the client's needs.

32.4.1.1 | Example 1: GitHub Flow

GitHub Flow (not to be confused with GitFlow) is documented on the GitHub Guides site (GitHub). They provide an interactive visual that makes the flow very clear. I can't improve on it in book form.

I recommend you look at the GitHub Guides documentation. In the meantime, here's a summary:

1. Create a branch for your work. This will be a short-lived branch.
2. Make changes and commit to your branch.
3. Open a Pull Request to initiate review and discussion of the changes.
4. When everyone is satisfied with the changes, you deploy *from the branch* into production for final testing.
5. If there's a problem, you can overwrite the deploy by deploying again from master, which doesn't (yet) have your changes.

6. If all is well, merge your branch into master and delete it.

Aman Gupta (Gupta, 2015) provides a more detailed walkthrough of the GitHub Flow process.

GitHub Flow starts to slow down at scale due to the repeated deploys from branches and repeated re-deploys or reverts. That might make it more suitable in a polyrepo environment than in a monorepo environment, as each product's separate repo would not be subject to frequent commits by a large number of developers.

GitHub has an internal tool called Hubot that can queue pull requests. This mitigates the scaling challenges by pipelining the merges.

Another factor is that GitHub Flow is optimized for the “pull request” approach to making changes. Many teams in corporate IT shops will not be using that approach. They will be collocated, and will use collaborative work methods to achieve (in effect) continuous code review (through pairing or mobbing). That removes the need for the pull request / review feedback step prior to merging changes to master, and enables very smooth continuous delivery.

32.4.1.2 | Example 2: Release Flow

GitHub Flow works well for GitHub, but isn't suitable for all situations. The Microsoft VSTS team encountered some limitations of GitHub Flow for their needs (Thomson, 2018):

The problem with this development strategy is that it scales extremely poorly to larger teams, because there's contention when you're trying to deploy to production.
[...]

Visual Studio Team Services has hundreds of developers working on it. On average, we build, review and merge

over 200 pull requests a day into our master branch. If we wanted to deploy each of those before we merged them it, it would decimate our velocity. [...]

So instead of deploying every pull request to production, we deploy master to production at the end of each sprint—every three weeks.

This is a valid trunk-based development strategy, as developers don't break the build and releases are always from master. It handles a high volume of changes by a large number of contributors, but doesn't support continuous delivery. However, the VSTS team cherry-picks pull requests to address "hot fixes" immediately. Even then, releases are from master.

Here we have a trunk-based development work flow that handles high volume, so it might be useful when we have a monorepo containing a large quantity of files; but it *doesn't* support continuous delivery, so it *won't* support a *competitive capability* such as *Respond To Market* (consider *Consumer Banking* in the earlier scenario).

That suggests this could be a useful strategy for services that support stable release schedules (like *Commercial Banking*, in the earlier scenario). Releasing to production or staging every three weeks, as in the VSTS example, could be perfectly fine for a service that's on a three-month or six-month release cycle.

32.4.1.3 | Google Trunk-Based Development

Google is famous for keeping all the code for all its products in a single repository. It's the ultimate monorepo.

Unfortunately, some consultants and others take that statement out of context and run with it. They encounter scaling problems.

How does a Google developer fit all that code on their local drive when they check out? Answer: They don't.

Paul Hammant describes how Google subsets the trunk on check-out, and performs pre-commit checks to ensure a developer hasn't affected other code inadvertently with their changes (Hammant, 2014b). That keeps trunk green, in accordance with trunk-based development guidelines. It also enables developers to check out a reasonable amount of code at a time. In some ways, it's similar to having a polyrepo strategy supported by a monorepo.

This requires some pretty sophisticated in-house custom-written tooling. Our clients in the "normal world" will likely be uninterested in investing in this sort of internal development, or supporting such tooling over the long haul.

We need to be thoughtful about how to manage scaling issues with a monorepo strategy. It may be that we switch to a polyrepo strategy for reasons of scaling alone, as a practical matter.

In contrast with GitHub Flow and Release Flow, Google doesn't do pull requests. Instead, they push to master after code review has passed. This is probably the way it would be done in the majority of corporate IT shops.

32.4.1.4 | Facebook Trunk-Based Development

Facebook doesn't have the wide range of different applications and technologies that Google has (Hammant, 2014b), but they do have a very large code base (Goode, 2014). They started to run into scaling limitations with Git in 2013.

Their approach to scaling was to enhance the version control system itself. They investigated Git and decided it would be difficult to enhance. They chose to switch to Mercurial, which is written in Python and has a pretty clean code base.

Copying the history rather than all the source is generally an advantage of distributed version control systems like Git and Mercurial, but in Facebook's case the history was so large that it became the limiting factor for developers checking out code.

They solved the problem by writing an extension to Mercurial that changes *clone* and *pull* to get only the commit metadata and not all the individual file changes.

This is another approach that's unlikely to be welcomed by most clients of organizational transformation services. It involves designing and building very sophisticated internal tooling, and then supporting it over many years. That isn't the reason our clients are in business.

32.4.2 | Release Branches

Trunk-based development has become a sort of “ideal” approach, as of 2019. Companies that must support multiple product releases concurrently often use a slightly different branching strategy. They create a separate branch for each product release. Atlassian’s Dan Radigan describes it this way (Radigan):

Release branching refers to the idea that a release is contained entirely within a branch. [...] All changes for the 1.1 release need to be applied twice: once to the 1.1 branch and then to the master code line. Working with two branches is extra work for the team and it’s easy to forget to merge to both branches. Release branches can be unwieldy and hard to manage as many people are working on the same branch. [...] changes in earlier versions (i.e., 1.1) may need to be merged to later release branches (i.e., 1.2, 2.0).

Release branching can be a fairly clunky approach if everything is done manually. With some custom coding, it’s possible to streamline this process to make it feasible in supporting rapid turnaround of changes.

32.4.3 | GitFlow and Variants

GitFlow (not to be confused with GitHub Flow) was first described in an online article in 2010 (Driesssen). It became quite popular and works well in some circumstances, but there are limitations.

GitFlow defines two long-lived branches: *master* and *develop*. Work is done against the *develop* branch. When ready for a release, teams merge *develop* into *master*. By convention, that constitutes a release. He writes:

...each time when changes are merged back into master, this is a new production release *by definition*. We tend to be very strict at this, so that theoretically, we could use a Git hook script to automatically build and roll-out our software to our production servers every time there was a commit on master.

In addition to the two long-lived branches, GitFlow defines several special types of branches that have distinct purposes:

- Feature Branch
- Release Branch
- Hotfix Branch

In effect, developing against the *develop* branch is much the same as developing against *trunk*. The only event that occurs when *develop* is merged into *master* is that a “release” is defined. Many developers have found this offers no value compared with trunk-based development and an explicit release process.

Ruka (2015) notes another problem with GitFlow: It tends to muddy the history, making it hard to trace how the code has evolved.

The same developer proposed a modification of GitFlow based on a single long-lived branch, making it more consistent with trunk-based development (Ruka, 2017). He calls it OneFlow.

OneFlow feels very similar to GitFlow, only without the complexities created by having two long-lived branches. But there are still Feature Branches, Release Branches, and Hotfix Branches. It still feels a little more complicated than necessary, to me.

32.4.4 | Feature Branches

With feature branches, developers cut a new branch for each feature they work on. “Feature” is a flexible term. It can mean nothing more than a User Story or even a task.

Feature branches are usually used on teams that don’t exploit collaborative methods such as pairing or mobbing. Individual developers create branches as needed to support their own work. The branches live as long as necessary for the developer to complete their task.

With a certain amount of difficulty and self-discipline, a feature branch strategy can support continuous integration. But as Martin Fowler notes, it can lead to something he calls Promiscuous Integration, or PI (Fowler, 2009).

PI happens when developers notice that they’re working on the same section of the code base, and merge into one another’s feature branches without reference to master. Anything can happen. Most likely, (a) merge conflicts and (b) changes falling through the cracks.

Feature branching has led to all kinds of problems on teams. It isn’t a “goal state” for our technical coaching, although we might employ it as a kind of temporary scaffolding to help a team take steps from *no branching strategy at all to some kind of branching strategy*.

Personally, my observation is that feature branching is not a “solution” to anything, but rather a *workaround* that enables teams to survive the day, when their work items are too large and they lack a cohesive and rigorous continuous integration process.

32.5 | Challenges In Scaling

I've already mentioned some scaling challenges, especially for monorepos in conjunction with trunk-based development. There are a few broad areas that present challenges:

- local space in developer environments to store the full repo on checkout (using a monorepo);
- increased probability of merge conflicts (especially when there are many dependencies across products);
- timing issues when applying a large number of commits to trunk; and
- build execution times when many builds are triggered in a short period of time.

Some companies have the revenue, the technical staff, and a business model that make it reasonable to customize or build tooling to counteract specific scaling issues they face. Most “normal” companies don’t.

You've already seen that scaling isn't the same problem in all cases. Google supports many disparate applications from the same monorepo. Facebook supports, basically, one application, but the size of the code base is very large. These are different problems, and they were solved differently in those two cases.

Another example I mentioned earlier is GitHub. They queue the pushes to trunk through their internal tool, Hubot. This approach has been explored and implemented by a team at Uber (Ananthanarayanan, 2019). But there is more to handling scale than just queueing. Their paper reviews several of the practical issues:

- Queuing commits. “This approach does not scale as the number of changes grows. For instance, with a thousand changes per day, where each change takes 30 minutes to pass all build

steps, the turnaround time of the last enqueued change will be over 20 days.” - Batching changes and submitting them together. If the batch passes, we’re done. Otherwise, the batch is split and each part is re-tried. That process continues until everything runs green. “SLAs cannot be guaranteed as they would vary widely based on how successful changes are as well as the incoming rate of changes. In addition, this approach also suffers from scalability issues since we still need to test and build one unit of batch at a time.”

- Commit Queue process (Chromium). Pending changes go through a pre-commit step. Every few hours, all the successful pre-commit changes are picked up by a scheduled process that does the real commit and build. Additional tooling deals with problems. Among other issues, the authors note the “approach does not scale for fast-paced developments as batches often fail, and developers have to keep resubmitting their changes.”
- Optimistic execution. Builds start with the expectation that other builds will succeed. The idea is to get some benefit from parallel execution. “If a change fails, then the builds that speculated on the success of the failed change needs to be aborted, and start again with new optimistic speculation. Similar to the previous solutions, this approach does not scale and results in high turnaround time since failure of a change can abort many optimistically executing builds. Moreover, abort rate increases as the probability of conflicting changes increase.”

An internally-developed tool at Uber called SubmitQueue uses a novel approach to assure trunk stays green at scale without bogging down under high loads. The details are beyond the scope of this book, but make for an interesting read. As of this writing SubmitQueue is not publicly-available, but it seems reasonable to expect products of this its kind to become generally available in the next few years.

In the meantime our approach to scaling for “conventional” client organizations must involve some combination of off-the-shelf version control systems and continuous integration servers alongside well-thought-out branching strategies and sound decisions regarding monorepo, polyrepo, and hybrid repo setups.

32.6 | Typical Starting Points

At the start of the transformation program, technical coaches may find client teams using version control systems in various ways. It’s likely that all teams in the organization are not doing things consistently. Each team may be using different practices and different tools.

Some common starting points:

- No version control system at all;
- Version control used for application sources but not for infrastructure definitions/specifications;
- Multiple version control systems in use;
- Random branching practices per individual developer preferences;
- A standard, but convoluted branching strategy;
- Long-lived feature branches;
- Task branches;
- Feature branches; and/or
- Release branches.

32.7 | Step By Step Improvement

In general, we want to identify the most appropriate combination of

- version control system(s)
- branching strategy(ies)
- development work flow
- continuous integration process and tooling

for each team's situation, always aiming to achieve the overarching goals of the transformation program. The sooner we can get this in place, the easier other aspects of the transformation will be.

The first step for each organization, and often for each team, depends on how the team currently manages these things. Many organizations that call for outside help with technical improvement are in relatively bad shape at the start of the program. Organizations that are already doing reasonably well will not usually engage external consultants and coaches. So, we can expect to find many clients who are operating in the ways listed in the section, *Typical Starting Points*.

If there's no version control in place, then Step 1 has to be to implement a version control system and get sources under its management.

If there's a version control system but no consistent work flow for checkouts and commits, then Step 1 has to be to establish a work flow that will function within the team's existing organizational constraints (possibly temporary scaffolding).

If there's a more-or-less consistent work flow but the branching strategy is random or highly personalized, then Step 1 has to be to rationalize the branching strategy and establish consistent naming conventions and a consistent structure for source artifacts under version control. The initial branching strategy may not be optimal for the long term; it may be no better than Feature Branching (temporary scaffolding, again).

If there's some sort of branching strategy in use, but it's inconsistent or clunky, an incremental improvement could be to move

toward something like OneFlow or GitFlow as an interim measure (temporary scaffolding).

Moving from OneFlow to Trunk-Based Development is an easier step than moving from nothing straight to Trunk-Based Development, as the latter requires fairly robust continuous integration practices to be in place and used knowledgeably by the team.

Some form of Trunk-Based Development, whether based on a monorepo or some other version control structure, is likely to be the goal state for this area of technical practice.

32.8 | Pitfalls and Anti-Patterns

There's an interesting intersection of common problems and anti-patterns in this space. Technical coaches often:

- hold strong personal opinions about the “best” version control systems and branching strategies; and
- tend to “go native” and start to think and act like team members rather than team coaches.

Technical coaches may try to institute their personal favorite branching strategy without going through any analysis of factors mentioned in this chapter, or other pertinent factors that I didn't think to mention. Seasoned coaches bring curiosity, empathy, and patience to engagements, while novice coaches tend to bring predetermined solutions, ready answers, and a desire to look smart.

Unless the coach's favorite branching strategy happens (by chance) to match the parameters of the client situation, their teams start to experience challenges.

Then, when they have slipped into thinking and acting like client team members themselves, the coaches tend to implement *workarounds* to help the team achieve short-term, tactical delivery goals (such as

“meeting the Sprint commitment”) rather than guiding the team toward true *solutions* to issues like “merge hell,” broken builds, and scaling.

That’s how many teams have ended up with convoluted feature branching strategies that hold them back from achieving the goals of smooth, continuous delivery, painless releases, and reliable production operations.

I think it’s important for technical coaches to be self-aware and to take a step back to assess the situation with their teams and with themselves before deciding on a course of action.

33 | Improving Infrastructure Support & Operations

And so he builds, because what is building, and rebuilding and rebuilding again, but an act of faith?

— Dave Eggers

Consider the hypothetical bank we used to illustrate the analysis of which services were market-differentiating, mission-critical, and so forth. The results of the analysis are summarized in Figure 6.2.

The function of “IT infrastructure management” is considered *mission-critical* but not *market-differentiating*. In that example, we decided to move to a mix of in-house and external support for this function.

Infrastructure management and operations are often handled by the same group in traditional organizations. Going forward, it’s useful to separate these functions. In many cases, separating these functions will be in scope for the transformation program.

Infrastructure management as such is an overhead activity. Proactive operational support, ultimately to be handled by product-aligned, dedicated, cross-functional teams rather than by an operations silo, is part of value-add work to provide market-differentiating services to customers. Indeed, that is exactly the function we identified as the *fulcrum* to get leverage from IT to deliver business value, as shown in Figure 7.3.

33.2 | Infrastructure Management Options

Many organizations are moving to solutions like the following:

- external partner to collaborate on infrastructure management (partner relationship);
- external supplier to provide “cloud” infrastructure support (customer-supplier relationship);
- hybrid in-house and cloud-based infrastructure (partner relationship);
- hybrid in-house and cloud-based infrastructure (customer-supplier relationship); or
- in-house infrastructure support.

In my view, a true partnership is worth considering only if the organization has extremely heavy requirements for cloud-based infrastructure; when the level of responsiveness and depth of understanding of the organization’s needs exceed what we would normally expect from a commodity supplier.

When the organization’s needs for cloud-based infrastructure are of a routine nature, a customer-supplier relationship with a commodity service provider is advisable. This will be easier to administer and more cost-effective than a partner relationship or in-house support. It will also be easier to switch suppliers if the need arises than it would be to dissociate from a partner.

Handling infrastructure support internally could be a reasonable approach, as it avoids the overhead of dealing with an external supplier. However, we should remember that managing infrastructure internally will draw high-value technical personnel away from value-add work on market-differentiating products and services, and dedicate them to overhead activity. We have to balance those concerns.

Given the sort of comprehensive transformation program described in this book, decisions regarding these matters will be a joint responsibility of client leadership, management consultants, and one or more technical consultants who collaborate in guiding the program.

Team-level technical coaches may or may not be consulted to choose a direction here. But it's useful for everyone involved to understand the options and the rationale for selecting an approach.

33.2.1 | Stress

The infrastructure or platform engineers are likely to experience the highest level of stress during the transformation program. Technical coaches must be prepared to deal with this.

Consultants almost universally assure clients that the changes they propose will not result in people losing their jobs. Their jobs might *change* - they may need to learn new methods and new tools - but everyone will still be employed.

It's difficult to make that case with respect to infrastructure engineering. Once we have a few changes in place - infrastructure as code, automated continuous deployment, etc. - it's pretty clear that fewer people will be needed in this area.

The reason? The *status quo* in nearly all organizations is that virtually all the work in this area is done "manually." After only a few improvements, people will start to feel the impact of automation.

It's possible some of them can shift their career focus to development, if they are interested in that, but there may not be enough places in the company to accommodate them. So, their stress is quite real and is well-founded. There's no sense in dancing around that.

Technical coaches can serve best by identifying the individuals

who are able to drive and sustain the new ways of working and place them at the forefront of organizational improvements, as internal leaders. At the same time, those who seem unlikely to thrive in the new environment will need guidance to shift focus into programming, testing, or operations or to move into a part of the organization that is not in scope for the transformation program.

33.2.2 | Starting Point for Infrastructure Management

Let's posit a worst-case (and unfortunately common) starting point.

- Server instances (physical and/or virtual) are set up with long-term operation in mind. There's an emphasis on keeping servers up and running, regardless of what happens with the applications they host.
- Each instance is a snowflake, built and configured by hand (typing commands manually, not scripted). Engineers make updates and patches to the existing instances. Configuration drift begins from Day One. The longer this goes on, the more fear there is about losing one of the servers. No one is quite sure how to re-build any instance, so minor problems such as hard drive failures or OS corruption can cause outages with significant business impact.
- Engineers may be hesitant to share information or shell scripts they've written, as a hedge against layoffs.

To support the goal of *stable production operations*, we'd like to move the organization in the following direction (details subject to change as the state of the art advances after publication):

- People focus on customers and the health of the applications they depend on; server instances are seen as transient and

disposable resources, managed in a way that keeps the applications available, functional, and responsive. This is my understanding of the concept, “serverless.” It’s a focus on customers rather than on long-lived server instances.

- Server instances are created on demand from declarative specifications maintained under version control. All instances of a given *type* are consistent.
- Instances may be based on templates that are customized when created, or compiled from source with both the system packages and application code generated from source that is known to be free of viruses or other malware.
- Server instances are immutable once created. Updates and patches are made by destroying existing instances and replacing them with freshly-generated ones. Servers may be destroyed and replaced routinely, rather than replaced only when there’s a change (the so-called Phoenix server strategy).
- Instances are managed in such a way as to minimize the risk of security breaches. This may involve a number of different practices, and the details are out of scope for this book. They can include removing root users and deleting installers once an instance has been created and before it’s opened up to traffic, for example.
- Instances are provisioned with the minimum settings and packages to support their applications. This is another security measure, as well as an operational measure. Responsibility for things like load balancing, logging, and error detection are pushed into the cloud infrastructure rather than being handled by individual server instances.
- Responsibility for the server instances that support any given product resides with the product-aligned software development teams rather than a separate engineering team. This is made possible in part by the fact most of the “engineering” is handled by the cloud infrastructure and needn’t be baked into the server definitions.

33.2.3 | Infrastructure Step 1: Inventory and Categorize the Builds

Given the worst-case starting point, we can begin improvement of infrastructure engineering (also known as platform engineering) by identifying all the builds that are currently in production. Find the build sheets or scripts people are using. Categorize them and group them together in a way that makes sense in context; for instance:

- Linux app servers
- Linux database servers
- Linux Web servers
- Windows app servers
- Windows database servers
- Windows Web servers
- Proxy servers
- Routers
- One-off builds to support COTS products
- One-off builds to support old internal applications

The goal is to group the builds into sets that can potentially be supported by a single build or “family” of builds, possibly from source and possibly by customizing a template instance.

For example, you might find the team is supporting 154 instances including 16 physical boxes and 138 virtual machines. Of those, 8 physical boxes are one-off builds to accommodate limitations of COTS packages or legacy internal applications.

That inventory includes production, staging, test, and development servers. Each of them is built and kept alive for as long as possible, with patches and updates made piecemeal and in an inconsistent way. Every instance is a snowflake, and migrating code from one environment to the next is a hit-or-miss affair.

Given the taxonomy suggested by the list above, let's assume certain numbers of Linux and Windows instances for discussion purposes:

- 1 Linux app server template producing 84 VMs
- 1 Linux database server template producing 16 VMs and 4 physical units
- 1 Linux Web server template producing 32 VMs
- 1 Windows app server template producing 2 physical units
- 1 Windows database server template producing 2 physical units
- 1 Windows Web server template producing 2 VMs
- 1 Linux proxy server template producing 4 VMs
- 2 Router templates producing 8 units of two types

At this point, we're positioned to start reducing the number of unique builds the team must support.

33.2.4 | Infrastructure Step 2: Script the Builds and Version Control Them

The next step for infrastructure engineering is to start scripting the builds for the servers.

We begin with the instances that are subject to the most frequent changes. For example, if we find that the Linux Web servers are changed most often, then we begin with those builds.

Initially, there are 32 Linux Web server instances in the environment, and all of them are configured and provisioned “manually” (via the command line) by following written build sheets.

Over time, they have drifted so that today each is a snowflake, and none of the engineers is confident they could re-build any of the instances quickly if they had to. Every change they make results

in a fire drill and negatively impacts the stability of production operations.

Working collaboratively with the lead engineers, the technical coach guides the team to identify the common elements in the configuration of the Linux Web servers and to start writing a build script. This may take some time and several iterations to get right.

Eventually, we learn what a “standard” Linux Web server looks like for this environment, and how to provision and configure one from a script that we can store in the version control system.

We reduce the number of unique builds from 154 to 123.

We continue to work through the list of builds. On each pass we address the most-frequently-modified type of build still remaining. Eventually, we end up with:

- 1 scripted build for standard Linux app servers
- 1 scripted build for standard Linux database servers
- 1 scripted build for standard Linux Web servers
- 1 scripted build for standard Windows app servers
- 1 scripted build for standard Windows database servers
- 1 scripted build for standard Windows Web servers
- 8 scripted builds for one-off instances

We’ve reduced the number of unique builds to keep track of from 154 to 14. All are scripted, including the one-off builds, and all are under version control.

33.2.5 | Infrastructure Step 3: Stop Updating and Patching

This step overlaps Step 2. As each category of builds is scripted and placed under version control, we can begin to introduce the practice of re-creating instances whenever a patch or update is required,

rather than modifying existing instances. We can modify the builds so the instances do not have a root account once they are live.

Any change to a server definition means a new instance is created. Existing instances are never modified. This eliminates configuration drift and reduces the risk of security exploits.

33.2.6 | Infrastructure Step 4: Phoenix Server Strategy

This step overlaps Step 3. As the team becomes comfortable with dynamic infrastructure management, we can introduce a phoenix server strategy. That means servers are destroyed and re-created routinely, on a timed basis, in such a way that users do not experience any interruption in service.

The details are beyond the scope of this book. Suffice it to say that at this point we have set things up in a way that can enable the phoenix server strategy, if the team and the technical coach consider it advisable in context.

33.2.7 | Infrastructure Step 5: Self-service for Development Teams

An eventual goal is to bust the silo walls between development and infrastructure management. With server configuration and provisioning scripted and under version control, and environments managed by cloud service providers (whether in-house or external), there's no reason a development team can't create the server instances for its product as needed, without having to request services from another team.

To set the stage for physically merging infrastructure/platform engineers into development teams, it may be advisable to create a

self-service system to enable development teams to request server instances on demand.

This is an interim step between Step 4 and physically merging the teams (see *Busting the Final Silos*, below). It's an example of *temporary scaffolding* to enable the organization to take baby steps forward.

The self-service arrangement enables development team members to learn the skills necessary to handle this work properly, and for infrastructure/platform engineers to prepare themselves to become members of a product-aligned team. In the meantime, the infrastructure/platform engineers still have the ability to ensure instances are defined properly.

33.3 | Leading Practices in Operations

At the time of writing, two areas of software-related work are undergoing significant and rapid change - operations and testing. *Circa* 2019 those are definitely the most exciting areas to be involved in.

The main shift in operations is from a *reactive* to a *proactive* approach. For a long time, we've been reacting to support tickets opened by...well, *someone*...and poring over log files to try and figure out what went wrong. It's not the most satisfying way to work.

In the few years prior to this book, there's been a strong move toward automation in software delivery. That has included automation of building, testing, packaging, and deploying application software.

The next logical step is proactive monitoring of operations. As Steve Waterworth put it in an article on "Observability vs. Monitoring"

(Waterworth):

To run the CI/CD process effectively, there must be some form of feedback. It does not make any sense to continually push out changes without knowing if they make things better or worse.

The earliest reference I've found to the term *observability* is in a 2013 blog post on Twitter Engineering (Watson), mentioned in a 2017 article on the subject by Cindy Sridharan (Sridharan). In any case, *observability* has become a new “-ility” for application designers. It enables proactive monitoring of operations, and a highly effective approach to production support. It converts the CI/CD *pipeline* into a *loop* connecting everything from version control to users.

At the same time, there's been a trend toward cloud computing. A natural outgrowth of cloud computing is microservices; and when your solution comprises a large number of microservices living in an elastic cloud environment, the production environment becomes a *complex adaptive system*, in the Systems Thinking sense, rather than a statically-defined (and hence predictable) platform.

If the production environment is a complex adaptive system, then it won't be feasible to predict the effect of any change, or even to predict the behavior of the system as workloads shift and server instances come and go, by using linear cause-and-effect analysis methods.

A new technical discipline has emerged to meet these needs, starting around 2017 or slightly earlier in some companies. It's called *Site Reliability Engineering*, or SRE. Even from the name, you can see the affinity with our “fulcrum” for effecting meaningful and valuable organizational transformation - *stable production operations*.

The online book from Google, *The SRE Book* (Google), has become

the *de facto* practical handbook for engineers interested in site reliability. The book describes a site reliability engineer this way:

The result of our approach to hiring for SRE is that we end up with a team of people who (a) will quickly become bored by performing tasks by hand, and (b) have the skill set necessary to write software to replace their previously manual work, even when the solution is complicated. [...] SRE is fundamentally doing work that has historically been done by an operations team, but using engineers with software expertise, and banking on the fact that these engineers are inherently both predisposed to, and have the ability to, design and implement automation with software to replace human labor.

Before the term SRE was coined, people were finding other ways to express the idea. The term *observability engineering* was used at Twitter even before 2017 (Asta):

These are the four pillars of the Observability Engineering team's charter:

- Monitoring
- Alerting/visualization
- Distributed systems tracing infrastructure
- Log aggregation/analytics

Initially, engineers rolled their own tools to help them manage operations proactively. Google's description of the SRE role emphasizes both the ability and the predisposition to do just that. But we can't depend on individual initiative industry-wide for sustainable, scalable, and consistent operations.

Enter Charity Majors and a new type of software tool she pioneered, now known as Honeycomb. Applications are instrumented

to enable Honeycomb to capture runtime data in great detail. Engineers can drill into this data in any way they can think of, in real time, to spot anomalies or emergent behaviors of the production environment.

As production environments become more complex, and applications consist of more and more small pieces, not all of which were developed by the same teams, tooling that enables us to deal with the environment as a complex adaptive system are becoming critically important.

By the time you read this there may be other players in this market besides Honeycomb, but that was the first tool of its kind (as far as I know).

Connecting this with the Agile movement, we're interested in forming stable, cross-functional teams that are dedicated to a single product or family of products. In principle, these teams are supposed to take full responsibility for the product they support.

Until recently, there has been a practical limitation in that regard: Teams could assume responsibility for requirements, design, solution architecture, testing, documentation, packaging, and deployment...but not for operations. Operations support has been relegated to a central team. Traditionally, that team has worked in a reactive mode to respond to issues.

The concept of Site Reliability Engineering and the availability of Honeycomb (and presumably more tools of that kind, going forward) make it feasible to form cross-functional, product-aligned teams that can take full responsibility for a product. These teams will be equipped to notice emergent problems in production and make any changes necessary to solve those problems, both to survive the immediate impact and to adapt the system to avoid future occurrences.

The final piece to the puzzle in this area is the concept of *serverless*. Writing in May, 2018, Mike Roberts notes (Roberts):

Like many trends in software, there's no one clear view of what Serverless is.

He then goes on to provide one definition. His definition is about designing applications without worrying about the server environments that host the application components, but focusing on the functionality.

I'd like to offer another "take" on the term, and I'm not the first person to think of it this way: *Serverless* is an attitude or mindset that focuses on the user's experience with the application (or even better, their experience doing whatever they're using the application to do) rather than on keeping the server alive at all costs.

It seems to me this fits perfectly with the other trends mentioned in this section. Our server instances become commodity resources that are easy, cheap, fast, and safe to change. In a typical cloud environment, server instances can come and go without interrupting application service. They don't have to be the focus of our support effort. Application availability and service can become the focus instead.

As Charity Majors put it in a 2017 piece on the changing world of operations (Majors 2017):

Compared to the old monoliths that we could manage using monitoring and automation, the new systems require new assumptions:

- Distributed systems are never "up"; they exist in a constant state of partially degraded service. Accept failure, design for resiliency, protect and shrink the critical path.
- You can't hold the entire system in your head or reason about it; you will live or die by the thoroughness of your instrumentation and observability tooling

- You need robust service registration and discovery, load balancing, and backpressure between every combination of components
- You need to learn to integrate third-party services; many core functions will be outsourced to teams or companies that you have no direct visibility into or influence upon
- You have to test in production, and you have to do so safely; you cannot spin up a staging copy of a large distributed system

With *stable production operations* as the “fulcrum” to gain leverage from the IT department for business value, this is the model we’re aiming for as technical coaches. All technical work must become operations-centric.

All we have to do is find a path from here to there.

33.3.1 | Starting Point for Operations

The *status quo* in most organizations is very different indeed from the scenario described in the previous section. The *SRE Book* describes it pretty well:

Historically, companies have employed systems administrators to run complex computing systems. This...approach involves assembling existing software components and deploying them to work together to produce a service. Sysadmins are then tasked with running the service and responding to events and updates as they occur. [...] Because the sysadmin role requires a markedly different skill set than that required of a product’s developers, developers and sysadmins are divided into discrete teams: “development” and “operations” or “ops.”

That's operations in most IT shops in a nutshell. How does it compare with the goal state?

- Site Reliability Engineering - not exactly. There's a focus on keeping each server alive; performance assessment is often based on percentage of server up-time. That's different from the *reliability* of the entire *site*. It's different from our "fulcrum" - *stable production operations*.
- Observability - not exactly. Of the four "pillars" of observability from Twitter, there may be log aggregation but not analytics, and there may be some form of notification or alerting. Even log aggregation is becoming obsolete, *circa* 2019, in favor of observability tools like Honeycomb.
- Serverless mentality - no. The focus is on the servers, and not on the users. When a problem is reported, and first-line triage discovers it may be due to an application issue, the ticket is handed off to another team for resolution.
- Product-aligned cross-functional team - no. Operations is a silo, not directly engaged with the development teams that support the applications.

So, the technical coaches have a lot of work to do. The difference between the starting point and the goal state is larger than in any other area of the IT organization...and this is the *most critical* area.

And yet, it may not be the most challenging area to address. The changes are so profound and so radical that there's no easy path back to the old *status quo*. That, in itself, may simplify the technical coach's task.

A programmer who is struggling with test-driven development (TDD) can easily set it aside. She has written code for the past 15 years without it, and she can build one more feature using familiar methods. Maybe she'll try TDD again next week.

A tester who can't quite get that Selenium or Cucumber thing running can set it aside. She has tested applications for the past

8 years without it, and she can test this one again today. Maybe she'll try automating those tests again next week.

An infrastructure engineer who hasn't worked out the last details in that Chef recipe or Bash script can spin up the server with a root account, ssh into it, and finish the configuration manually. Maybe he can try scripting the build again next week.

But the operations engineer who used to wait for a ticket to be opened, and then looked at aggregated log files to try and find the source of the problem, is now looking at a whole different set of tools. He also has a whole different set of expectations to meet. Even if he *wanted* to back-slide, there's no path. The only way to walk is *forward*.

33.3.2 | Operations Step 1: Measure Stability of Operations

Earlier in the book I mentioned that we need to take baseline measurements in order to know whether we're making progress in the transformation program. For the operations area, we're interested mainly in tracking the time required for certain things to happen, such as:

- proportion of issues that are caught by the team before a customer notices anything is wrong;
- mean time between issue occurrence and issue discovery;
- mean time between issue discovery and team action;
- mean time between team action and identification of the root cause;
- mean time to resolve issues; and
- proportion of resolutions that fix the underlying cause vs. those that "put the fire out."

One thing to notice about these measures is that none of them points a finger at any individual or team. It's the effectiveness of the

organization that's important. Individuals must feel free to make mistakes and learn from them, with political and psychological safety.

33.3.3 | Operations Step 2: Increase Collaboration With Development Teams

When I started in the IT industry, it was normal for people to support the applications they wrote. We didn't have functional silos for analysis, design, architecture, programming, testing, documentation, packaging, deployment, support, and maintenance. It was all one job.

You could argue that things were simpler then. Those of us who worked on IBM mainframes wrote applications using COBOL or maybe PL/I or RPG, Assembly language, JCL, a filesystem such as ISAM or VSAM, sequential files, maybe a database management system such as IMS/DB, Datacom, or DB/2, a handful of utilities like SORT/MERGE and good old IEFBR14, and possibly a short list of third-party products. The only different target environments we had were batch and CICS or IMS/DC.

Today, we use a handful of languages of different kinds (object-oriented, functional; statically- or dynamically-typed, etc.), write for multiple target platforms (Unix, Linux, Windows, and still good old zOS), various kinds of containers or managed code environments (Docker, CLR, JVM, and still good old CICS), use multiple scripting languages, multiple database systems, we write and use APIs, assorted automation tools, and have significantly more complicated UI issues to deal with than we did in the “green screen” era. Testing can be substantially more challenging when it involves setting up test data on multiple platforms and validating that everything is updated as expected across multiple components running on disparate platforms. Cloud environments bring a level of complexity to the table that didn't exist before.

And yet, it isn't out of the question for a small group of people to manage all that stuff. The longstanding tradition of functional silos, and the habit of maintaining fine-grained, narrowly-focused sub-professions within IT - each with its own job titles, career progression, conferences, university degrees, certifications, and books - has given rise to the assumption that a human being is capable of learning and doing just one or two things. Once they've learned those things, their brains are full forever.

But that isn't so. The realities of software in the modern era are drawing us back to the model where the same group of people designs, builds, deploys, and supports an application. Charity Majors writes (Majors 2018):

Developing software doesn't stop once the code is rolled out to production. You could almost say it *starts* then. Until then, it effectively doesn't even exist. If users aren't using your code, there's no need to support it, to have an oncall rotation for it, right? The thing that matters about your code is how it behaves for your users, in production - and by extension, for the people who are oncall for your code running in production.

In the course of the transformation program, we're guiding the development teams toward building an automated CI/CD pipeline. Conventionally, the pipeline stops at production. Now it's time to connect it up.

This step in improving operations involves bringing the operations team, infrastructure/platform engineering team, and development teams together to collaborate more closely. It's time to end the practice of throwing the application over the wall to the operations team. We're breaking the mid-level Matryoshka dolls.

An abrupt change to a true end-to-end product-aligned cross-functional team may not be feasible, especially in organizations that have more than a few hundred people in the IT area. So we can

begin the process of increasing collaboration at this point, with the intention of continuing it as we progress with the transformation program.

Here, we want developers and testers to start learning about operations and support; and we want operations people to start learning about development and testing.

At this stage, we aren't breaking the largest Matryoshka dolls. This process of increasing collaboration takes place separately in the front-end, mid-tier, and back-end areas.

In some organizations, it will be feasible to treat the front-end and mid-tier areas as a single unit. In most large organizations that have significant legacy technologies in place, it will probably be necessary to treat the back-end area separately, at least.

33.3.4 | Operations Step 3: Introduce Tooling and Practices for Proactive Support

When the changes introduced in Step 2 have stabilized, and corresponding improvements in development and infrastructure areas are in place, we expect to see:

- improvement in the metrics around operational stability (established in Step 1);
- increased knowledge and skill in operations on the part of development and infrastructure team members;
- increased knowledge and skill in scripting and rudimentary programming on the part of operations and infrastructure team members;
- infrastructure definitions in version control alongside test suites and application source;
- increased ownership of server configuration specifications by development teams aligned with products;

- infrastructure/platform engineers migrating (by choice) into development teams or operations;
- increased attention to *observability* in application design;
- increased feeling of ownership on the part of everyone who is aligned with a product; and
- a more proactive attitude toward production operations, indicated by a tendency to look for problems rather than waiting for tickets.

This is the time to introduce new tooling and practices around that tooling to enable a more proactive approach to operations and support.

I want to avoid naming specific products in this book, as those tend to change fairly rapidly over time. However, as of this writing (2019) there really aren't many products on the market to support operations in this way. I will mention Honeycomb.io here, with the expectation that other products will be available as time goes on.

This is the time to introduce such a tool, or multiple tools as appropriate to the situation, and to bring the teams up to speed in using them. It will take time for people to learn to look for patterns and trends in emergent behaviors in the complex production environment. This is the time to begin.

33.4 | Busting the Final Silos Within Each Technology Stack

This is the point where we phase out the siloed infrastructure / platform engineering and operations teams, folding their responsibilities into product-aligned, cross-functional teams alongside analysts, programmers, testers, and others.

But there are limits to this.

First, we're talking about the mid-level Matryoshka dolls only. We haven't yet folded together the front-end and back-end teams.

The learning curve for a Web developer to pick up legacy COBOL and VSAM skills is steep; and the learning curve in the other direction is just as steep. It will be easier for people to merge into well-functioning cross-functional teams than for them to jump in at this stage, when there are so many other changes in play.

Second, there will still be someone, somewhere, who is responsible for the underlying infrastructure. That may be a cloud provider, an internal group, or a combination of the two.

We've separated the gory details of keeping the hardware alive from the people who want to focus on user experience and customer value. That doesn't mean the hardware magically runs itself, somehow.

The expectation is that the hardware environment (except for legacy mainframe and mid-range computers) is designed and built to be resilient, flexible, and user-definable within reason.

33.5 | Special Considerations for Mainframe Teams

The steps outlined for front-end and mid-tier technologies may not apply one-for-one to the mainframe area. Several unique considerations (or a subset of them) are likely to apply:

- more bureaucracy and finer-grained separation of technical tasks into functional silos;
- general lack of familiarity with contemporary software development methods such as test automation, owing to a dearth of tools to support those things;
- defensiveness about job security; “ownership” of some small piece of the work may be seen as job protection; and

- technical differences in infrastructure configuration.

33.5.1 | The Rise of Bureaucracy

In the late 1970s and early 1980s, it was common for developers in mainframe shops to write their own JCL, configure and start/stop their own resources (such as VTAM and CICS), define their own databases and files, perform their own testing, deploy their own code, support their own applications, and interact with their own users.

Through the rest of the 1980s and 1990s, that gradually changed. The various tasks involved in software-related work became segregated into different teams. Separation was enforced administratively by the introduction of “heavyweight” processes to request services from other teams, and by sharply limiting access rights to the bare minimum assets an individual needed to get through the day (and sometimes less than that).

33.5.2 | Job Insecurity

Insecurity about job safety is probably an effect of many years of chatter about the impending doom of the mainframe world. Instead of dying out, mainframes have continued to evolve until they now stand as one of the most compelling cloud hosting platforms around.

Experienced mainframers have nothing to fear with respect to job security, but that may be a hard message to get across as a technical coach who has been brought in to introduce new technologies and methods.

33.5.3 | Positive Traits Common in Mainframers

It's also likely you will find some positive characteristics in this area, such as:

- deep concerns about security and operational stability;
- greater sensitivity to and awareness of the potential impact of changes to other systems; and
- better code quality;
- greater willingness to try new things.

System security and operational stability have been front-of-mind for mainframers for decades. The whole system is designed with those goals in mind. That's a Good Thing. You won't find the general carelessness that we see in younger programmers who can sling code but who lack experience with large-scale, complex systems.

Possibly connected with that attribute, mainframers tend to think carefully about what could happen as a result of any change - to data formats, data values, logic, job scheduling...you name it. They live in a world where no one fully understands the way data flows through the various systems. Any change introduces risk to other systems in the environment. That's a great mindset to have when we're making sweeping changes.

33.5.4 | High Quality Code

In my experience, code quality tends to be higher in legacy mainframe applications than in front-end and mid-tier applications. An exception is code that the organization inherited from a third-party vendor, when the vendor went out of business. That code tends to be pretty bad.

But code developed in-house is often of a very high standard. I believe it's because people who chose to stick with the mainframe platform in the days when everyone was predicting its demise were those who cared about it and liked working on it. They have continued to hone and refine their skills and the code they support through all these years.

So, as a technical coach, you won't be facing significant legacy code remediation on these applications compared with the front-end and mid-tier code. It's one less thing to worry about in a complex transformation program.

33.5.5 | Openness to New Ideas

Perhaps counterintuitively, most mainframers are more open to new ideas and more willing to try unfamiliar techniques and tools than their younger counterparts. I'm not sure how to explain the phenomenon. It may be that they simply enjoy technology and programming, while many of the younger generations of software professionals chose the field strictly because it offers marginally higher salaries than other white-collar occupations. They've learned one way to do their jobs, and they're satisfied with that.

This is another factor that may make your work as a technical coach a little easier. Of course, there will be plenty of other challenges to counterbalance it.

33.5.6 | Technical Differences in Infrastructure Configuration

Mainframe systems can be used as cloud host platforms. zSeries machines can run thousands of Linux VMs concurrently while handling mixed workload types (batch, CICS OLTP, CICS DSS, and Linux) with good throughput.

But the basic components of mainframe systems aren't as easy to "spin up" as resources in cloud environments or in server rooms. Workloads are handled by Logical Partitions (LPARs). Starting an LPAR from scratch and configuring it for normal use can take (as of this writing) around 20 to 25 minutes. There are also pricing considerations for activating resources on the zSeries platform.

The short story here is we won't use the phoenix server strategy for LPARs, and development teams won't be able to create LPARs on demand.

What we *can* do is specify a fixed number of LPARs for specific uses, such as development or testing, and "script" the provisioning of the LPARs. Provisioning doesn't mean starting an LPAR from scratch; it means deleting files that are left over from its previous use and resetting everything to a known base state so that another team can use the resource. Then we can assign LPARs to teams as needed.

There will be waiting periods for teams with this scheme, but it's more flexible than the traditional approach of defining a fixed number of LPARs whose configurations are inconsistent and possibly not quite what each team requires, and then having multiple teams modifying the resource concurrently.

CICS systems can be executed as batch jobs. Normally, production systems will be executed as MVS subsystems and they will comprise multiple regions. For development and test purposes, we can allow teams to provision a set of regions from a predefined collection of "standard" ones. Each team can have its own CSD file and can stop/start their region(s) as needed, while they have ownership.

Like the LPAR-sharing scheme, this is not as flexible as spinning up VMs and containers on the fly in a cloud environment or on a laptop, but it's better than the traditional approach to managing infrastructure.

We can take a similar approach to other large-scale resources such

as DB2 databases, so that we can assure test, stage, and production resources are all configured consistently.

Part 5: Coaching

Coaching, so you are no longer needed.

— Steve Chandler (*Hands Off Manager*)

Coaching is an important part of guiding an organizational transformation. With particular emphasis on technical coaching, this part deals with the working definition of “coaching,” the skill sets needed by a coach, and the reasons for the severe shortage of qualified technical coaches.

I also describe a practical approach to technical coaching that is becoming more widely used *circa* 2019, and that avoids some of the issues the technical community has experienced with ineffective or non-sticky technical coaching in the past.

In addition, I mention specific communication frameworks, collaboration techniques, and “internalities” to help coaches deal with stress and maintain focus during engagements.

34 | Coaching Skills

A good coach can change a game. A great coach can change a life.

— John Wooden

People who work in the organizational transformation field mention the words “coach” and “coaching” quite a bit. So do I, throughout this book. What does it actually mean?

In my view, to function effectively as change agents we need to cultivate skills in the following general categories:

- Guiding teams and individuals
- Facilitating events and meetings
- Skills in practical application (process and technical)
- Skills in effecting change as such

34.1 | Coaching Competencies

The *Agile Coaching Competency Framework* developed by Lyssa Adkins and Michael Spayd provides a comprehensive model of skills relevant to helping people change and improve and for guiding organizational transformation. Although the framework has the word “agile” in its name, it’s broadly applicable.

They list several important areas of skill development for people who want to work in this sort of role. Here they are, correlated with the categories I listed above (those categories are not part of the framework):

- Coaching (Guiding)

- Mentoring (Guiding)
- Teaching (Guiding)
- Facilitating (Facilitating)
- Technical (Application)
- Business (Application)
- Transformation (Change)

Most of the following definitions aren't taken directly from the framework, but represent my understanding of the terms. The meanings are consistent with those of the framework.

34.1.1 | The Guiding Competencies

Coaching is helping others discover their strengths and find their own way forward. It's important that the coach not impose any personal growth objectives on the coachee. The coach's role is to help the coachee progress toward their own objectives.

That's a general definition. In the context of a transformation program, the coach must guide coachees toward the defined goals of the transformation, so it isn't solely about their personal growth goals. This can involve helping people recognize how their personal goals may align with the organization's goals.

The word *coach* is overused in our field. In the context of working with technical teams, people usually use the word *coach* when all they really mean is someone who has a higher level of skill than the rest of their team, and can show them how to do things. The majority of people who work as "technical coaches" have no awareness of *coaching* skills or techniques as such.

Mentoring is showing people how to do specific things; for example, demonstrating how to do refactoring, and working side-by-side with someone as they learn it.

Teaching consists of explaining concepts about a topic, to provide foundational knowledge. This may be done individually or in groups, formally or informally.

Coaching, mentoring, and teaching are often interleaved in the day-to-day activities of a team-level technical coach. I think of them as part of the *guiding* function of a coach.

34.1.2 | The Facilitation Competency

Facilitation is a *guiding* skill, too, but is sufficiently distinct from the other guiding skills that it warrants its own section.

Adkins and Spayd define a *facilitator* as a

Neutral process holder that guides the individual's, team's, or organization's process of discovery, holding to their purpose and definition of success.

I often use facilitation skills when guiding teams in process-related improvements, and sometimes in connection with technical practice improvement as well; particularly in a group setting such as a mob programming session or randori-style dojo.

The facilitator gets a session started and allows participants to run it, only speaking up when necessary to keep things on track. Sometimes participants aren't sure what to do, or the session veers away from the goal, and requires some correction.

A facilitator never "owns" the meeting and is not the "boss." At the team level, the coach may facilitate team activities such as the *team coordination meeting* or *daily stand-up*, gradually relinquishing control until the team learns to handle these events on their own.

Under normal circumstances, the need for a facilitator fades away naturally as teams learn to handle routine activities independently.

However, when there is a conflict, there may be a need for an outsider to facilitate a discussion to help resolve the conflict.

Eventually, client personnel should be able to handle conflict situations without an outside facilitator, as well. This may require more learning curve time than normal activities.

34.1.3 | The Application Competencies

The ability to *apply* the skills in which we're guiding client teams is especially important in the technical coaching role, as technical people have little confidence in coaches who can't "walk the walk."

Technical coaches typically guide teams in both technical practices and process activities. This calls upon both the *technical* and *business* application skills, as well as sufficient domain knowledge to be able to function in context.

The particular applied skills necessary will vary based on the kinds of work each team performs.

For a software development team, technical skills could include things like software testing, test automation, requirements analysis, programming, design, architecture, database, continuous integration, build scripts, and so forth.

For an infrastructure support team, technical skills could include shell scripting, server provisioning, working with virtual machines and containers, system administration skills, operations skills, and so forth.

For a team that supports a third-party product such as a Business Rules Engine, ETL product, Data Warehouse product, application platform (Oracle, PeopleSoft, Siebel, SAP, etc.), technical skills include general knowledge of the category of functionality the product supports and, possibly, some experience working with the particular product (although that may not be necessary).

On the process side, strong understanding of Lean principles will help the coach guide teams in focusing on value, maintaining continuous flow, and removing waste, regardless of any specific methods or frameworks in use.

Practical working knowledge of any specific methods or frameworks the team must use will be helpful for guiding them on the process side.

In the case when the technical coach is engaged as part of a larger program that's led by a prime contractor, and the prime contractor is introducing a specific model or framework for the client, then the technical coach has a professional obligation to support that model or framework and to provide advice consistent with that of the prime contractor. If for philosophical or ethical reasons the technical coach is opposed, then they should withdraw from the engagement.

34.1.4 | The Transformation Competencies

Transformation skills pertain to specific techniques for driving *change* in the organization, on a team, or by an individual. They also involve coordination to ensure consistent messaging when more than one consultant or coach is engaged.

Many people working in the role of technical coach can demonstrate techniques such as test-driven development, design by contract, etc. That is *mentoring*. Mentoring does not automatically translate into "sticky" change.

The coach may explain the value proposition, underlying philosophy, mechanics, and tooling around a particular technical practice; say, trunk-based development, continuous integration, or deployment automation. That is *teaching*. Teaching does not automatically translate into "sticky" change.

The coach may encourage and support individual team members'

progress in learning to apply various technical skills. That is *coaching*. Coaching does not automatically translate into “sticky” change.

Transforming or changing people’s habits is a different matter from guiding them in learning new technical practices. It requires a range of “soft” skills and a certain amount of field experience, including making many mistakes, to cultivate transformation skills.

“Coaching Approach,” touches on some of these skills. For the moment, it’s sufficient to mention that this is a *bona fide* skill area for technical coaches.

Another transformation competency is the ability to convey a consistent message to coachees. In large-scale transformation programs, team-level technical coaches often work as part of a consulting team that is engaged at multiple levels in the organization’s management hierarchy.

In that case, the technical coach must coordinate with consultants and coaches who interact with various levels in the organization to ensure a consistent message and direction are provided *vertically*. They also must coordinate with other team-level coaches to ensure consistent messaging *horizontally*.

Many transformation programs have been compromised when different client personnel received, or *thought* they received, conflicting messages from different consultants or coaches.

Sometimes just using a different word for the same concept can lead to misunderstanding and friction. Remember that everyone is under stress due to the changes happening all around them. It doesn’t take much to set off an emotional response.

34.2 | Self-Awareness and Empathy

An important aspect of coaching skill is *self-awareness*, or being conscious of how you are perceived and how your guidance is

received by coachees. Mindful use of communication models can be helpful here, as can the “internalities.”

Practice in growing your EQ, or Emotional Intelligence Quotient, is also helpful for cultivating the ability to notice how you are perceived as well as detecting how others are responding on an emotional level.

Part of the job is to ensure psychological safety for coachees, so they will feel confident in trying new things and making mistakes. People must feel safe before they will adopt unfamiliar practices.

Some consultancies and some client companies place a high value on standard personality tests. They may conclude that individuals are “hard wired” to behave in certain ways, and cannot learn to behave differently.

In my view, this is incorrect. Everyone can learn self-awareness and empathy. People’s underlying tendencies might make that sort of learning easier or harder for different individuals, but that is not the same as assuming people literally can’t learn or improve. You can.

Another factor that seems related to this topic is language. In different organizations, different words may be “trigger” words that cause people to shut down or to oppose the advice we offer. Many coaches are fond of particular words, and spend time (often, weeks or longer) trying to make client personnel adopt their understanding of a particular word.

In my experience, it’s more effective to adapt our language to whatever “works” in the client organization’s culture. There’s more than one way to express an idea, and at the end of the day it doesn’t really matter what labels people apply to various concepts, provided they achieve the business goals of the transformation.

34.3 | Course of Least Resistance

My observation has been that people tend to do whatever is *easiest*. They don't necessarily do what is *best*.

34.3.1 | Case 1

A story comes to mind about a company where the software development manager and most of the people on development teams said they were very interested in adopting pair programming as a standard practice.

Management set up pair programming stations at the end of each row of cubicles in the software development area, where team members could go and pair up whenever they wished. No one ever used them. Yet, in every team retrospective, every team stated they really, really wanted to start pair programming.

What was the problem? It was easy enough to pair. Just find a partner and to sit at the end of the cubicle row.

The thing is, “easy” isn’t sufficient. If you want people to do something different, it has to be *the single easiest thing they could possibly do*. It has to be the thing they will *do by default* because doing anything else would require effort.

34.3.2 | Case 2

In another company where a team kept saying they wanted to try pair programming, the eight developers would come in each day and sit around the big table in the middle of the team space and work solo at one of the eight workstations.

One day after hours, their ScrumMaster and I remained behind and removed four of the workstations.

The next morning, as soon as team member #5 showed up, they started pair programming. With four workstations and eight people, it would have been more difficult to *avoid* pairing than just to do it.

34.4 | Make Things Visible

Many coaches spend a lot of words trying to explain concepts in great depth. Sometimes this is effective and sometimes it leads to confusion, boredom, or disengagement.

I've found that it's often sufficient to make a key thing visible to the team. Once they see it, they tend to take effective action on their own initiative, even if they haven't received any sort of formal introduction to the underlying concepts.

The *coaching skill* in this case is the ability to recognize *which things* to make visible, in order to shine a light on the problem you want the team to address.

34.4.1 | Case 3

On an engagement at a large client, I wanted to capture Cycle Time and Process Cycle Efficiency metrics for four teams that I was coaching. I wanted to do it in a way that would not be too intrusive on their normal work flow.

I set up a Lego plate for each team, and asked them to place bricks in a line corresponding to each User Story. The hours of the day were represented across the top of the plate.

I asked them to place a green 1x2 brick under each hour when they spent *any time at all* moving that User Story forward, and a red brick otherwise. When they completed a User Story, they placed a white 1x1 brick to show that they hadn't simply forgotten to place a brick for that hour.

I didn't worry about getting down to minutes for the PCE data. A crude level of detail was sufficient for the purpose. So, if a team paid attention to a User Story for five minutes out of an hour, that hour got a green brick. The PCE numbers looked a little high, but that was not a problem in context.

At the end of each day, I copied the data from the Lego plate into a spreadsheet and cleared the plate. Here's how the plates and spreadsheet looked for a team called "Cleaners" for the first couple of days:

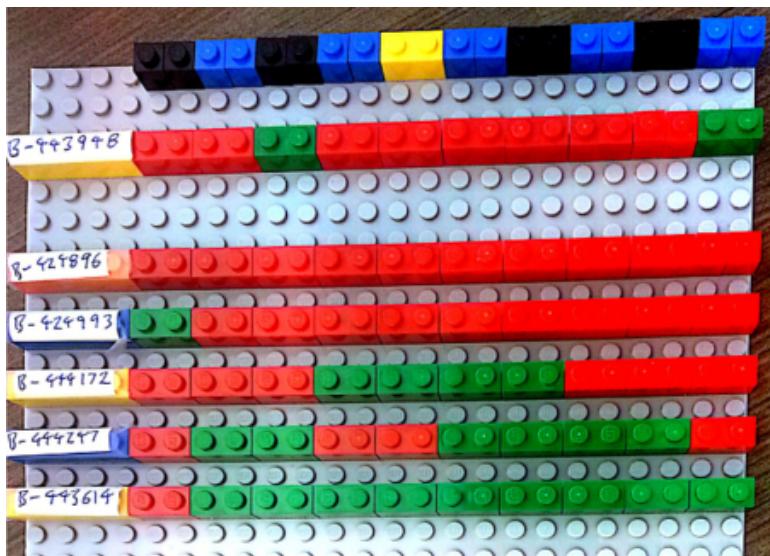


Figure 16.1: Raw data as Lego bricks, day 1

	A	B	C	D	E	F	G
1	Updated: 6/18/14						
2	Story	Team	External Dependency?	Value add hours	Total hours	PCE	Done
3	B-443948	Cleaners		1	8	12.50%	
4	B-424896	Cleaners		0	8	0.00%	
5	B-242993	Cleaners		0	8	0.00%	
6	B-444172	Cleaners		4	8	50.00%	
7	B-444247	Cleaners		2	8	25.00%	
8	B-443614	Cleaners		9	10	90.00%	
9							

Figure 16.2: Collected CT and PCE data, day 1



Figure 16.3: Raw data as Lego bricks, day 2

	A	B	C	D	E	F	G
1	Updated: 6/19/14						
2	Story	Team	External Dependency?	Value add hours	Total hours	PCE	Done
3	B-443948	Cleaners		7	17	41.18%	
4	B-424896	Cleaners		0	16	0.00%	
5	B-242993	Cleaners		1	9	11.11% Yes	
6	B-444172	Cleaners		11	17	64.71%	
7	B-444247	Cleaners		2	16	12.50%	
8	B-443614	Cleaners		13	19	68.42%	
9	B-443982	Cleaners		7	9	77.78%	

Figure 16.4: Collected CT and PCE data, day 2

Don't be alarmed that the team appeared to be working 10 solid hours a day. Different team members arrived and left early or late, and some liked to work through lunch or take lunch at different times of day. So, there was activity throughout the day.

In two of the four teams, I noticed a response to the Lego plates that I had not anticipated. Here are excerpts from my blog post describing the observed behaviors (Nicolette).

I was surprised to see the natural response when a team member reaches for a red brick. Others on the team immediately ask what the impediment is and how they can help. These were already practicing "agile" teams, so they are already stable teams working in team spaces, and collaboration was not a new concept at the start of the engagement. However, the immediacy of their reaction to seeing a red brick is a radical improvement in the speed with which teams respond to emerging issues.

A second natural reaction to the boards is that when a team notices a large swath of red on their board, they start exploring the reasons why. Without any formal training in Lean concepts, they quickly conclude that they have too many User Stories in play. They limit their WIP, even without knowing that term. Before

the impact of high WIP was visible, team members often said they did not understand the “big deal” about pulling just one or two User Stories at a time.

Nearly all teams in the organization have the classic “hockey stick” burndown chart, and a CFD that looks like a single block of color representing “in progress.” The teams that have started to notice the impact of high WIP thanks to their Lego boards are already starting to show burndowns that look like actual burndowns. They are pulling User Stories through to completion rather than starting everything on the first day of the sprint. Within days, it became a sort of game to see if the team could eliminate all the red bricks from their board.

Just making visible the difference between value-add time and non-value-add time triggered a natural response in two of the four teams, even with no further explanation or training, or any verbal encouragement to do anything differently.

I trust you noticed that the effect did not occur naturally in the *other* two teams. It goes to show that no technique or approach will be universally effective. We need to build up a toolkit of coaching methods so that we have more than one way to try and drive change.

34.4.2 | Case 4

On another engagement, a team I was coaching expressed the desire to raise their unit test coverage numbers. Coverage is a negative metric; that is, if it seems too low, it’s a negative indicator, but there’s no “magic number” that tells us things are going well. In fact, when we see 100% coverage, it almost always means the team is gaming the metrics. So, coverage is a metric to be used with care.

But this team had unit test line coverage of 5%. They understood that was not good, and they wanted to get into a more reasonable range, like 70% or 80%, so they could feel as if they were taking the codebase in a good direction. They weren't fixated on a number, as if it were a target. That was a healthy way to think of it, in my view.

Yet, day after day they didn't extend their unit test suite. They were in the habit of working in a certain way, and habits can be hard to change even when we want to change them.

I started to post the unit test coverage percentage on the wall next to the door, just *outside* the team room. I plotted dots to create a line chart that showed the day-by-day change in line coverage, based on the final build of the day.

Without any further discussion of the matter, the team began to add unit test cases to their suite incrementally in the course of completing User Stories. Just having the statistics visible was sufficient incentive for them to cultivate new working habits.

34.5 | Manipulation

“Manipulation” is quite rightly seen as a negative thing, as masters of manipulation tend to be found mostly on the Dark Side. But sometimes coaches have to use some of the same techniques to get things moving in a transformation program.

On the bright side, many manipulation techniques are really the same as general social interaction skills. People think of them as “manipulation” only when they’re used for nefarious purposes.

34.5.1 | Connect Organizational and Personal Goals

People involved in the transformation program may feel uneasy about changing anything about their work. They aren't certain the transformation will really happen. If they've worked in a large company for very long, then they've already experienced and witnessed attempted transformations that were abandoned or reversed.

One way to encourage people to try new things is to help them see how it could benefit them personally. Arguing logically from the point of view of what benefits the company may not resonate with everyone on a personal level.

The coach has to cultivate a positive working relationship with each coachee. In a private conversation, or a series of them, find out about the person's goals, fears, and any other concerns they may have that are blocking them from moving forward with new practices.

Then think of ways the proposed organizational changes will alleviate their worries and/or advance their personal agenda. I've often found this an effective way to turn people around from hardline refuseniks into valuable allies in the transformation effort.

This really isn't an "evil" use of manipulation. It helps the other party see what's in it for them; it helps them achieve their own personal goals.

34.5.2 | Build Relationships

Some of the world's most effective technical coaches make a point of having lunch with different client team members, as well as various individuals external to the team who can affect the team's work, each day. During these times, they explicitly don't talk about work.

The connection with learning new skills and being willing to try new things under the stressful conditions of a transformation program may not be obvious, and yet it's a powerful way to encourage change. When people feel a personal connection, they're more willing to accept suggestions and take (small) risks.

We're still well clear of the Dark Side. We aren't *pretending* to get to know the people; we're *actually* getting to know them. And we don't know whether we're making temporary working relationships or long-term friendships. Time will tell.

34.5.3 | Mirroring

Mirroring is a way to make the other party feel at ease during any sort of discussion. It's often recommended to people for job interviews and business negotiations.

If the other person crosses their legs, then so do you. If they lean forward when they speak, then so do you. If they position their hands or arms in a certain way, then so do you.

The effect is thought to be that subconsciously the other party identifies with you because they see themselves reflected in you, to an extent.

The manipulative aspect of mirroring comes when you gradually shift who is mirroring whom. You start by mirroring the other person, and once they're subconsciously doing it, you start guiding them into mirroring you instead.

You can do this verbally, as well. When the coachee expresses reasons or fears about changing some aspect of their work, you agree with them and reflect their own language back to them. Gradually build their connection with you by agreeing with them and even extending their argument with additional points.

Then, using the same words, start to guide the conversation toward positive aspects of the proposed change, and negative aspects of the

status quo.

You're definitely trying to manipulate them at this stage. You may choose to *omit* negative aspects of the proposed change and positive aspects of the *status quo* as a way to heighten the perceived benefit of the proposed change.

As the proposed change is deemed to be “good” in the context of the transformation program, this form of manipulation used in this context doesn’t qualify as “evil.”

34.5.4 | Playing Dumb

This is a way of guiding people’s thinking by asking leading questions. It’s useful when you want the other party to reason through a situation and arrive at a good answer. It can be useful when the coachee is unwilling to try a given technical practice.

On one engagement I was working with a team to show them various practices from Extreme Programming. The team lead had a strongly negative opinion about test-driven development (TDD). The rest of the team respected him highly and followed his lead.

One day, I approached him at his desk and asked if I could sit with him. I didn’t call it “pair programming,” as he had a strongly negative opinion about *that*, as well. I told him I wanted to watch him work as a way to get familiar with the code base. That made sense to him, and he agreed.

We picked up the next ticket to work on, and I pretended I didn’t understand the requirement. He explained it to me patiently a couple of times, but I just didn’t get it.

Acting as if I had just thought of it out of the blue, I asked whether he thought it would be a good idea to write some client code that would interact with the planned code, to get an idea of what it would look like to call that code and get return values from it. He thought that was a reasonable thing to try.

We wrote what was, in effect, a kind of “unit test,” although the scope was larger than we would normally prefer, and we didn’t use any sort of testing framework. Even so, when he saw how easily and clearly the exercise defined the requirements for calling the code, dealing with return values, and handling possible exceptions, he really liked it.

Almost immediately, he became a strong proponent of TDD and encouraged his team mates to try it. Best of all, it was his idea. He gathered the team members together and explained the benefits of TDD to them, without asking me first. I didn’t have to convince each team member individually. It was a good day for all.

34.5.5 | Nouns Over Verbs

We’re often advised to focus on *behaviors* rather than criticizing *people*. But to encourage people to move outside their comfort zone, it’s sometimes more effective to do the opposite.

It can be done in a subtle way. It seems that people tend to think about *behaviors* when they hear verbs, and to think about their *self-identities* when they hear nouns.

When describing people who don’t do the things you’re recommending as a coach, structure your statements around nouns that have mildly negative connotations. When describing people who already use the techniques or practices you’re recommending, use nouns that have mildly positive connotations.

Avoid extreme words, as these will cause people to snap to attention. You want to lull them along, so they don’t think too deeply but just allow the subconscious response to happen.

Now we’re moving into less benign manipulative territory, but not “evil” just yet.

34.5.6 | Don't Allow Time for Arguments

From this point forward, we're drifting ever closer to the Dark Side. These last three manipulation techniques are to be used advisedly, and only if nothing else has worked. They will probably work no more than once per team, if indeed they work at all.

This is a variation on a standard verbal manipulation technique, adapted for technical coaching situations. There are two aspects to it.

First, you must have established a positive working relationship with the coachees. This will cause them to want to please you, or at least not want to disappoint you.

The second aspect is a variation on the fast-talking used car salesperson technique. You're encouraging the team to try an unfamiliar technical practice, and they're hesitant. They want to debate it.

Instead, you quickly take the keyboard and "just do it," with a slight air of annoyance. This works for small things, like beginning to refactor a long method, for instance.

They see the result of doing what you were recommending before they have time to formulate verbal arguments against trying it. They're also a little unhappy that they've disappointed you.

Afterwards, they will be more willing to try it themselves; in part, because they've seen the benefit, and in part to win back your approval.

Important note: If you haven't established a relationship that causes them to care about your opinion, they'll just watch you type with the same degree of engagement as a house-cat watching a car crash.

34.5.7 | Fatigue As a Motivator

Sometimes, when people are tired and want to get out of the office at the end of the day, they'll be more willing to try new things than

they are in the middle of the day. They don't want to waste time arguing, and they'll do what you suggest just to get you out of their hair so they can leave.

This works with *some* people *some* of the time for *some* new skills. You have to know the people well enough to judge whether it's likely to be effective.

The point is to get a person try the new thing at least *once*, to give themselves a fair chance to experience the benefit of it.

Now you're just a tiny bit evil. You deliberately waited until they were ready to leave work before you sat them down to do one more task.

34.5.8 | Fear As a Motivator

A last resort manipulation technique is to prey on people's fears. Trying the new skill might be slightly less frightening than something else, such as a negative performance review or a tongue-lashing by their manager.

Some fears are benign while others are emotionally significant.

Some people might care enough about code quality to fear introducing defects into production. Some people might feel competitive, and will respond to the suggestion that another team will outperform their own team.

You know your teams, so you know what different individuals might respond to. But don't overuse this technique. It's pretty evil. And the second time around, it's totally obvious, too.

34.6 | Acting

This might seem an unusual skill to cite for technical coaches, but it actually applies in a couple of different ways.

34.6.1 | Improv Techniques Applied to Coaching

Techniques from *improv* are useful both to increase our self-awareness, empathy, and deep listening skills, and to help client personnel break down internal barriers of fear or nerves to reach a point where they're willing to try new things (Fidei). The latter is more appropriate to a workshop setting than *in-situ* team coaching.

If you have seen or used the popular exercise, “Yes, but vs. Yes, and,” then you’ve had a taste of how improv techniques help us with listening and empathy. You can take it much further by practicing improv techniques guided by a skilled facilitator. If you haven’t participated in this sort of workshop at a conference or coach camp or other event, I recommend it.

34.6.2 | When You Can’t Be Yourself

The second application of acting skills to coaching is to “present” in a way that resonates with coachees and may help influence them to change in desired ways.

The *manipulation* techniques “Mirroring,” “Playing Dumb,” “Don’t Allow Time for Arguments,” and “Fear As a Motivator” (and possibly others) require us to pretend. In addition, when an introverted coach must present as an extravert (or *vice versa*) in order to interact effectively with coachees, some basic acting skills can be helpful.

34.7 | Removing Organizational Constraints

In most cases, the project manager, team lead, ScrumMaster, or some other person has primary responsibility to remove orga-

nizational constraints that affect their team's ability to deliver, and to get any necessary resources, system access rights, or other requirements the team needs to get their work done.

During an organizational transformation program, the technical coach may have to take on the responsibility for this, at least initially. Be prepared to go to client management and/or collaborate with other consultants on the consulting team to help teams with these issues.

In the meantime, mentor and coach team members to take on this responsibility going forward.

Elsewhere I shared the story of a client where I was coaching a single team that had five bosses. I stepped outside the formal boundaries of my contract to ask those five people to collaborate to manage the team's backlog, prioritizing requests according to how they supported the company's official list of business goals for the year. That's an example of acting to remove organizational constraints that interfere with a team's ability to deliver.

But I didn't approach them out of the blue. I had established at least minimal working relationships with each of them before then, without asking them for anything. Otherwise, they probably wouldn't have met with me at all.

34.8 | Let the Team Stumble

A coach's role is to help people improve. Sometimes, people will listen with an open mind when you warn them away from problematic practices and steer them toward effective practices. Sometimes, they need to experience the natural consequences of a practice in order to understand its effects.

There are times when the best thing a coach can do to help a team is to let them experience problems, and then help them connect the dots that led to the problem (either with immediate feedback or via

a facilitated retrospective). A key point is not to let them crash and burn entirely. That would have negative ramifications for you, the team, and the organization.

But the coach is not there to support *delivery*; they're there to drive *improvement*. Improvement sometimes involves deeply understanding how the dominoes are likely to fall, depending on which one you push first. Nothing beats direct experience for that.

There is a special consideration here. You or other consultants on your team have to prepare client management for this technique. Management often expects (or *hopes*) improvements can be made with little or no downside impact on current delivery performance. That's not realistic, but many managers will balk at coaching methods that slow teams down, even temporarily. This has to be socialized and agreed vertically in the organization, or it will likely backfire.

34.9 | Be the Rock

At the risk of belaboring the point, I'll repeat that everyone involved in a transformation program is under stress nearly all the time. I've noticed an interesting phenomenon when coaching teams in this context. When everyone around me is getting stressed out, if I remain calm (even if only outwardly), it tends to defuse the situation.

Occasionally, a team member has told me after the fact that they appreciated my calmness. One day, a coachee said they wouldn't have made it through the day had I not been there with the team, even though I said very little that day. In the midst of some serious challenges at a large client, a colleague from our consulting firm said, in front of the others, that it seemed to him that stress didn't affect me.

Well, stress affects me plenty. But I think it's part of the job to

handle it. We have to provide psychological safety for our coachees. We can't do that very well if we, ourselves, are obviously stressed. We have to guide our coachees through challenging days when things aren't going well. We can't do that very well if we, ourselves, are visibly affected by the challenges of the day.

It surprised me the first few times it happened, but clearly if one member of the consulting team can maintain calm, it gives a form of non-verbal emotional support to the other consultants. This can make the difference between a day that erodes client relationships and a day that strengthens them.

For those reasons, I include this as a coaching skill we should all cultivate and practice. You may find the “internalities” to be useful for this purpose.

34.10 | Conflict Resolution

Friction and conflict will be frequent in any transformation program. Part of our role as coaches is to resolve conflicts among client staff members and, if necessary, among our own colleagues on the consulting team.

To do so, we need to bring to bear a range of different skills, including:

- relationship-building;
- facilitation skills;
- specific conflict resolution models;
- various communication models; and
- the “internalities.”

In addition, it's within the scope of our role as coaches to teach and mentor the same capability with our coachees, so that they can learn to resolve conflicts themselves.

Details of specific techniques are out of scope for this book, but in summary they include:

- Forcing - imposing one party's will on the other
- Collaborating - seeking a win-win solution
- Compromising - finding a less-than-perfect solution that is acceptable to both parties
- Withdrawing or Avoiding - agreeing to set aside the conflict temporarily or permanently, and living with any leftover friction
- Smoothing - accommodating the other party's wishes as a way to avoid further conflict (possibly temporary)

Each of these, including the ones that *sound* completely negative, have advantages and disadvantages and might be selected based on context.

See (separately) Bolton, Human Metrics, and Weeks in *References* for more information.

You might think a technical coach can comfortably remain within the warm embrace of technology, and not have to deal with conflict. But in fact it's a problem that comes up quite often.

34.10.1 | Case 5

There was a case many years ago when a company needed to connect two disparate systems - a DEC VAX and an IBM mainframe. (I *told* you it was many years ago.) Applications, user documentation, and everything else was ready to roll, but the two systems were still not talking after months of delay.

Long story short, the engineer on the DEC side and the engineer on the IBM side sat in cubicles about 3 meters apart. They never spoke to one another. It turned out we needed to make simple configuration settings on both systems, and then they could talk.

I had to convince the two of them that it was in their personal interest to get it done. We ended up with a *collaborating* or *win-win* solution.

The two engineers were not hostile to one another. Strangely enough, they *didn't know about each other*, and neither of them knew enough about the other system to know how to connect them.

34.10.2 | Case 6

In a similar story at a different company, a Java application needed to talk to a CICS application. Again, there had been months of delay. The two engineers sat in the cubicles right next to each other, and never spoke. Each insisted that the other system had to be configured to the liking of their own system.

This time, by the way, the two engineers *were* hostile to one another.

I managed to convince the Java guy to “be the bigger man” and configure his side to conform with the IBM engineer’s preferences. It was a *smoothing* solution, in which one side concedes to the other. I wouldn’t have guessed at that outcome based on the initial conversations. You never know how these things will turn out.

34.10.3 | Case 7

More recently, I was visiting colleagues in another company and one of them invited me to shadow him as he worked with a client. I was not formally engaged there; I was a guest and observer only.

Toward the end of a working session with a certain team, there was a blow-up between team members. The coach was taken by surprise; he hadn’t been aware of tension between them before. He suggested an “intervention.” We all went to another room to work things out.

Unfortunately, the people involved in the conflict pointed to me and said, “We want to know what *that guy* thinks. He hasn’t been here before, so he’s objective.”

Well, maybe, for some definition of “objective.” It was an uncomfortable position for me. I had to use *facilitation* skills and *active listening* to try and draw out people’s real thoughts and feelings about the situation.

Their coach had described one of the individuals as “sad,” and I could see her shut down. When an opportunity arose, I remarked that I didn’t see her as “sad,” but rather as “angry.” She immediately lit up and vented her frustrations for several minutes. Using the correct word, even if it wasn’t a “soft” word, broke the impasse. Difficult but necessary conversations followed.

Ultimately the team agreed to a *compromising* solution they felt would be good enough to get them through the next month of working together, and then they would see how things felt at that point.

A lesson from this is that we mustn’t let friction continue; we need to confront it as soon as we notice it. I say so because in this case the team members had “checked out” and buried themselves in their individual tasks for months. They all had made unverified assumptions about the others, and tension had steadily mounted until it reached a breaking point.

The team had suffered a significant loss of trust and was not functioning cohesively. As coaches we need to be aware of these things and prevent them from reaching that stage.

Under the stressful conditions of an organizational transformation program, conflict is normal and frequent. It isn’t something any coach can avoid; not even a technical coach. We must cultivate the skills to handle it.

34.11 | Principles-Based Adaptation

While most organizations and teams experience broadly-similar issues, it's also true that every team is unique. Each team has its own work to do, likely different from the work of other teams, and its own organizational constraints, similar to other teams insofar as management style and organizational culture are concerned, but otherwise unique, usually based on what cross-team dependencies or resource limitations they have.

With that in mind, one coaching skill is the ability to apply *principles* to a wide range of different situations, to help determine effective ways to improve performance. As this is written (2019), the industry is flooded with freshly-minted ScrumMasters who have little to no practical experience in IT, and whom client leaders think are qualified as team-level coaches because they hold a certification.

Many of these coaches know how to apply exactly one model or framework in a rote way. When they are trying to help a team whose unique circumstances don't quite fit that model, they're at a loss.

34.11.1 | Case 8

I was engaged as a team-level process/technical coach as part of a IT-wide transformation program. It was one of those “agile adoption” programs, this time based on the client’s internally-defined Agile scaling framework. It was as good as any of the branded frameworks that are sold commercially. Maybe better. At least it was crafted around their business needs, rather than a “canned” model.

One of the other team-level coaches, who was a process coach and not a technical coach, came to me and asked if I would visit her team to try and see why they were rejecting all her advice.

We all met in a conference room. I told the team I wasn't familiar with their work or their situation, and asked them to fill me in. They explained what they did and how their team interacted with other teams in the organization.

It became clear they were not a "conventional" software development team. They did various kinds of work, some of which was of a "technical" nature, but they didn't fit the canonical "development team" pattern. Their coach was advising them to do things like "daily stand-up" and "two-week iterations" and "slice stories to a small size." The usual.

They said "agile" wouldn't work for them. I said, okay, tell me what it is about "agile" that won't work for you. They took turns, and I made notes. It was obvious they needed to vent, so I didn't interrupt them. If anything, I asked clarifying questions to draw out even more of their frustrations.

Then I affinity-grouped their comments, and it boiled down to six specific things about "agile" that didn't fit their situation. I don't recall what the six things were, but that's not relevant to the coaching pattern.

I told them how I had grouped their comments, and they agreed with the categorization. Then I went through the items one by one, and told them I agreed in each case that canonical "agile" didn't support any of them. Their coach grew increasingly nervous during this part of the process. The team had seemed tense and adversarial up to that point, and now they were relaxed and open.

Then I suggested we think about the guiding principles of "agile" and see if we could get any value from them as applied to their own work. We were able to think of at least one thing in each category they believed would help them. They came away feeling cautiously optimistic about "agile" and how to apply it in context.

34.11.2 | Case 9

I've had a couple of engagements where the Siebel platform was involved in a team's work. It's a "legacy" product designed to host multiple applications. There are other products like it. That was a good architecture in the day when centralized systems were the way to go, and when there was less need for systems to be ready to interact with new clients that were unknown at design time. It was also an era when solutions were delivered in big-bang fashion with long lead times (compared with today's expectations).

Owing to those design considerations, a Siebel instance is fully defined in a single file, known as an SRF, or Siebel Repository File. It's a binary file that contains the "compressed" or "compiled" definition of all the applications hosted by the particular instance. A perfectly good solution in its day.

Today, we generally expect each application to be independently deployable on a much shorter cadence than was normal in those days. Teams that support this type of platform have challenges in adopting lightweight methods.

In one situation, the organization had six applications running on a single Siebel instance. They had three Siebel environments - development, test, and production. They wanted to "go agile," and that meant each of the six teams had to be able to define User Stories and deploy software changes independently of the other five. In fact, there were no *bona fide* dependencies between the applications; the only thing they had in common was that they shared a single Siebel instance.

None of the six teams could deliver any faster than the time required to push changes through all three environments for all six teams. In that case, it was around 12 weeks. They were stressed out because they were being told they had to deliver a production-ready solution increment in 2 weeks.

For a little context, this is just the sort of situation Sean Dunn

reports in his article, “Eating Your Own Dogfood: From Enterprise Agile Coach to Team Developer,” in part 6 of the article - “Sometimes stories cannot be split” (Dunn)

The area manager asked for a working session. We stepped through all the issues and challenges, and I made a recommendation: Set up a separate Siebel instance for each application. That meant buying 18 Siebel licenses - development, test, and production times six.

The manager balked. He worried about the cost of the licenses. I walked him through a quick-and-dirty analysis of the costs of the current setup. An hour later, he understood that the way they were currently doing things was costing far more than the price of the additional licenses, due to delayed delivery and the impact of bug-fixes for any one application on the release schedules of the other five applications.

34.11.3 | Case 10

On another large-scale transformation program (also in the nature of an “agile” implementation), I was asked to work with a particular team the other coaches felt was “impossible.” When I asked why they said so, they looked at one another with knowing smiles and replied, “You’ll see.”

I joined that team during one of their daily stand-up meetings. They gathered in a room that was just a bit small for them. The usual “agile” things were taped up on the walls, placed there by their former coach. The team basically ignored all that stuff. They were extremely cynical.

I asked them to explain what work they did and how they interacted with the rest of the organization. It turned out they were on a long-term project to upgrade all the company’s communications equipment at 160 locations in several countries.

The work involved physically inspecting each site to inventory what was actually installed there, rather than what the official

records said was *supposed* to be there. Those were two different things.

Then they would return to each site to install and configure the new equipment. They could monitor the entire network from the home office, but the information was not very useful, as the external contractors who installed the gear and wiring didn't follow specifications, so things like MAC addresses weren't what they were supposed to be. Contractors would also take short-cuts like using indoor CAT-5 for outdoor applications, and pinching or twisting the cables such that they didn't perform as CAT-5 anymore.

Several details were different from conventional “agile” software development teams. For one, the pace of change in their project was much slower than on a software development project. They were frustrated with the daily stand-up, because nothing changed day to day. I clarified the *purpose* of the stand-up - briefly, to ensure everyone knew what was going on, and could ask for and offer help as needed - and asked them how frequently they felt they should touch base as a team to achieve that purpose, in their context.

They settled on twice a week. I actually thought that was more than they needed, but if they were comfortable with it, that was fine. I reminded them they could adjust that schedule at any time, if it wasn't serving their needs. That pleased them, as they had been led to believe nothing in “agile” was flexible.

They were immediately more relaxed and open, the moment an option was offered to them. Their first coach had imposed canonical “agile” on them without collaborating with them. That was one cause of their frustration. From that point on, it became much easier to work with that team, because they were solving their own problems. I only offered guidance and suggestions as needed. They were genuine engineers as opposed to “software engineers,” so I was the least intelligent person in the room by a long shot.

Some of my suggestions weren't “agile” things, particularly, but I tried to tie them in with “agile” principles if possible, as that

was the overarching purpose of the transformation program. For example, I used the idea of “test-driving” to suggest they clearly specify requirements for the external contractors, and pay them only if they met the requirements; not just because they showed up and drove a truck around a site.

The MAC addresses had to be per spec and the CAT-5 cables had to respond as CAT-5 when tested. Otherwise, no check-ee. They liked that one a lot. It saved them considerable post-installation time and effort.

Coaches need to be able to adapt whatever model they’re teaching to different contexts based on fundamentals, as every team will have its own context, and sometimes that context will be very different from software development.

34.12 | Having Multiple Ways to Explain Things

In the first installment of Cutter Consortium’s *Cutting-Edge Agile* series, Agile Manifesto author Alistair Cockburn notes:

We are now in the “post-Agile” age.

Post-something means that we are in an in-between period, where one something has been incorporated into our culture and the next something has not yet fully formed to the point of naming.

As of this writing (2019), Cockburn is quite right: There’s no name for what we think we ought to be doing and showing other people how to do. What most coaches do is continue to use the old “agile” words and phrases.

By now, pretty much everyone has experienced some flavor of “agile,” or something labeled as such. Many of those people did not like the taste.

When freshly-minted ScrumMasters and the like come bounding into the room full of energy and enthusiasm, they’re often taken aback by the lukewarm (or cooler) reception they get.

The thing is, the various “agile” and “lean” buzzwords and phrases have become *triggers* for people out in the world. When we first visit a client organization, we don’t know which words are triggers there. They might not be the same words as the ones that are triggers at the company across the street.

For that reason, I consider it a basic coaching skill to be able to describe the benefit and the mechanics of any given method or practice in more than one way. If we can describe a practice without recourse to buzzwords, and still convey the essence of that practice, then we’ll avoid tripping any verbal landmines.

Why worry about verbal landmines? Remember the high stress level. Remember that people are on the alert for reasons to avoid changing their habits. When we trip a verbal landmine, what follows may be days or weeks of explaining and re-explaining to try and recover to the position where we started.

Try explaining the value and/or mechanics of these things, and any others you can think of, without using any of the usual buzzwords:

- sprint
- pairing or pair programming
- mobbing or mob programming
- test-driven development
- test automation
- hold accountable
- limiting work in process
- batch size

- buffer management
- feedback
- retrospective
- emergent design
- technical debt
- Scrum
- agile

It may be challenging at first, but once you get going you'll find you can describe these things using words that won't trigger a negative reaction. Different clients have different triggers, so you'll need two or three different ways to express these ideas without buzzwords. I suggest practicing your explanations as if rehearsing for a play.

34.13 | Awareness of Context

There are two general scenarios for clients to engage team-level technical coaches:

- technical coach engaged by a team manager (or area manager responsible for a handful of teams), possibly to introduce good practices with the aim of general improvement in skills, product quality, and/or delivery effectiveness; or
- technical coach engaged as part of an organizational transformation program; one member of a consulting team that interacts with the organization at multiple levels to provide cohesive and comprehensive guidance toward one or more business goals.

There is a popular notion among team-level coaches that one must be "invited" by a team to offer coaching advice to them. The coach must earn the trust of the team before he/she will be welcome to offer advice. This is appropriate for the first case.

When we are engaged as part of an organizational transformation program, we may not be able to wait for each team to come to trust us enough to invite us personally to coach them. Our “invitation” was already sent, from the CEO or CIO or other client leader who called for the transformation.

In that case, our job is to *make change happen*, as opposed to waiting until team members are good and ready to consider changing. As it’s ineffective to *tell* people what to do, that may call for a wide range of different ways to influence people.

It’s still good if teams trust our advice - and we’re dead in the water if they *distrust* us - but it isn’t necessary for them to “invite” us in quite the same way as when we’re engaged to help a single team. Context matters.

34.14 | Knowing When To Quit

No matter how beneficial everyone believes the recommended technical practices may be, there will always be a few individuals who just don’t want to work that way.

As coaches, I believe our responsibility to these individuals is to ensure they understand the trade-offs they may be making so that they can make an informed professional judgment about whether to adopt the recommended practice.

We need to cultivate in ourselves the “internalities” that I call *Eye of the Hurricane* and *Non-Attachment*. The former gives us the inner peace to accept the fact other people are in charge of their own lives, and need not accept our advice. The latter reminds us that we don’t “own” our clients’ outcomes; we can shine a light on the path, but we can’t walk for them.

34.14.1 | Case 11

In my experience, one of the most fundamental software development practices is *test-driven development* (TDD). It always surprises me when good developers want to argue against it.

At one client, the senior-most developer on a team was dead-set against TDD. No amount of explaining or demonstrating would move him.

One day we were planning a learning event for developers in which we would run a code dojo around TDD. He volunteered to facilitate the event. With no training or preparation from any of the coaches, he ran the event like an expert. He understood as much about TDD as any of the coaches did, and he could explain it well and demonstrate it effectively. He helped 24 colleagues learn and practice the technique.

There was no doubt that he knew TDD well. Yet, his personal choice was not to develop code that way.

As a coach, I thought that was fine. He made an informed professional judgment. It was different from my own professional judgment, but it was a completely valid choice on his part.

We have to accept the fact that not everyone will choose to do things the way we recommend, and that's okay.

What we want to avoid is to let people dismiss unfamiliar techniques just because they don't understand them or because they've heard negative stories. We need to help people understand the choices they're making. As long as that's true, there's no "rule" that says they have to make the same choices we would make.

35 | Shortage of Technical Coaches

No one learns as much about a subject as one who is forced to teach it.

— Peter Drucker

A technical coach performs a range of different services to support the client's transformation program. These include:

- Coaching
- Mentoring
- Teaching
- Facilitating
- Resolving conflicts
- Influencing change
- Removing organizational constraints
- Applying technical skills
- Applying process skills
- Mitigating client stress
- Coordinating work with other consultants and coaches

The majority of people who are engaged as technical coaches today have at least reasonably strong technical skills. Some percentage of them also have reasonably good knowledge of one or more process frameworks commonly used at the team level, such as Scrum, XP, or Kanban. A smaller percentage also have *mentoring* skills. A few have genuine *teaching* skills.

Almost none have any significant skills in the areas of facilitating, coaching (in the true sense), resolving conflicts, influencing change, mitigating client stress, or coordinating work vertically and horizontally in a large-scale transformation program.

35.1 | Availability of Technical Coaches

The combination of those skills - in other words, the complete “technical coach” package - is *extremely* rare. Most of the good ones have more work than they can handle, and aren’t usually available on demand, especially if extensive travel is required.

This highlights a significant logistical challenge for IT organizational transformations: There aren’t enough qualified technical coaches.

Note the word, *qualified*. To coach a technical team effectively, a person needs strong expertise in at least a handful of different technical disciplines, and enough general experience to feel comfortable working with new languages and tools. Of course, it’s unrealistic to expect any single individual to have deep expertise in *every* technical discipline. The point is that some breadth of skill is necessary for this role.

A process coach can’t show technical teams how to apply unfamiliar technical practices. It isn’t sufficient to *tell* teams to use particular technical practices, and still less sufficient to *encourage* teams to discover good practices through trial and error; the coach has to *show* team members how to do it. That requires a qualified technical coach.

In addition to technical expertise, a technical coach needs expertise in the craft of coaching as such. Coaching and mentoring others in non-trivial skills, in a situation where they are understandably uneasy about the impact of the organizational changes on their own positions and careers, is a completely separate skill set from technical matters.

35.2 | A Note on Agile Coaches

The popularity of “agile” software development and of branded processes and frameworks based on it have led to the proliferation of “agile” and Scrum coaches. As of this writing, some 500,000 people have earned the Certified ScrumMaster credential.

Most of them possess no technical knowledge, experience, or skill whatsoever. Most of them, as well, have never studied or practiced *bona fide* coaching skills. Many have never worked in the IT field at all, and have no background in *other* methods from which one might draw useful ideas and techniques to supplement the “agile” ones.

Be careful to distinguish between an “agile coach” and a “technical coach,” in the way we have defined the latter. Some “agile” coaches are quite good, but most of them are interested in implementing a favorite framework (usually Scrum or a derivative).

Their goals for your transformation will be to see that your staff follow the rules of Scrum or some other branded framework. Often there is no reference to business outcomes or to target operational models. The “agile” framework *is* the target operational model, as far as they are concerned. I observe this is another cause of transformation failures.

35.3 | Tailored Approaches

I tailor the approach to handling the shortage of qualified technical coaches to the client’s circumstances. The key factors to consider in crafting an approach include:

- size of organization; scope of transformation
- target operational states

- availability of interested client personnel to become internal coaches

35.3.1 | Small

The approach for a small organization is straightforward. If only a small number of technical teams will be involved in the transformation (say, ten or fewer), then we can use typical coaching methods to achieve the transformation goals, where “typical” means one or two coaches per team, working full-time with the team.

35.3.2 | Medium

Given a larger number of teams, but not a very large number (say, up to about 40), we adopt a slower-paced coaching model in which a single technical coach can work with as many as six teams at a time, alternating between sets of three teams every two weeks. There is a specific model for this approach, developed by Llewellyn Falco, which is described in “Technical Coaching Approach.”

35.3.3 | Large

For large organizations (perhaps hundreds of teams), we may use one or more mitigation strategies to deal with the shortage of qualified technical coaches. These can include

- sequencing - generally starting with market-differentiating operations and then addressing other areas
- slicing - addressing vertical slices of the organization in turn
- training internal staff as entry-level coaches, with senior coaches then rotating among them to monitor their work

Obviously, any of the three large-scale strategies will extend the timeline of the transformation beyond what client leadership may hope for. Some consultancies will cheerfully promise that they can deliver this degree of change effectively on a tight schedule. They expect to be able to hire coaches quickly to fulfill the contract.

They cannot. There aren't enough *qualified* technical coaches in the world for that to be possible. Larger clients are urged to accept that reality and work with it as best they can.

35.3.4 | Adjustments for Target Operational State

The target operational states may influence our technical coaching approach, as well. We have noted that we do not ask technical teams to adopt particular practices just because they are generally “good” or because they comply with a given published framework. Everything we do is aimed at enabling the desired *competitive capabilities*.

Therefore, if the organization does not require any highly dynamic market capabilities - that is, all their systems can serve customers well without continuous delivery and market experiments, provided changes can be released safely on a predictable schedule - then the technical staff can “get away with” a less-robust set of technical practices. This can simplify and speed up the transformation.

36 | Technical Coaching Approach

In order to change an existing paradigm you do not struggle to try and change the problematic model. You create a new model and make the old one obsolete.

— R. Buckminster Fuller

It seems to me the majority of transformation models focus on training and coaching executive leadership, management, and team leads in a given framework or method, and training delivery teams on their role in the method, and then “a miracle occurs” with respect to technical infrastructure, technical practices, automation, and so forth.

36.1 | Problems With the *Status Quo*

There are several reasons why I think technical coaching has to take a more central role in IT transformation programs, and why the approach to technical coaching has to be well thought-out. These are listed more-or-less in order of importance, as I see it.

36.1.1 | Misunderstanding the Importance of the Role

A failure mode in transformation programs is that both client management and consultancies who hire technical coaches undervalue the role, underestimate the difficulty of becoming qualified to perform the role, overestimate the availability of qualified coaches

in the job market, and assume the work the coaches perform is in the nature of “staff augmentation” or a low-end “commodity” service.

Almost invariably, transformation programs are started with too few technical coaches, underqualified technical coaches, and insufficient time planned to produce “sticky” improvement in client teams.

36.1.2 | The Fulcrum Is In the Technical Area

According to the general case analysis in “Leverage: Where’s the Fulcrum?” the focus of the transformation resides within the IT sphere rather than the business sphere: *stable production operations*.

Without appropriate guidance in the technical area, all the great ideas at the executive and management levels can never be executed.

36.1.3 | The Key Need Lies in an Emerging Area

Approaches, team composition, tooling, and practices for production operations and support are currently undergoing significant and rapid evolution.

Existing staff skills don’t align with the emerging practices, and focused coaching is needed in that area to a greater extent than the usual areas of programming, testing, and delivery pipeline automation.

This exacerbates the shortage of qualified technical coaches, as most of them specialize in either application programming or testing rather than in production operations.

36.1.4 | Global Shortage of Technical Coaches

There is a severe shortage of truly *qualified* technical coaches worldwide. As an industry, we can't afford to pile multiple coaches on each delivery team on a full-time basis because there simply are not enough of them to meet demand.

We need to find practical ways to enable fewer technical coaches to support more teams while maintaining effectiveness.

36.1.5 | Technical Learnings Don't "Stick"

As far as I have been able to learn from experience reports, comments from staff at companies that have undergone transformations, colleagues in the technical coaching arena, blogs, social media posts, and my own direct field experience, the technical practices that were taught and coached have not "stuck" for very long after the coaches left, except in a vanishingly small number of cases.

The fact this continues to be an industry-wide problem lends credence to the observation that we haven't solved it. Logically, then, something about conventional approaches to technical coaching hasn't been working. We need to understand how to change our own practices to achieve better results.

36.1.6 | Coaches Lack Agency (or Believe So)

Another failure mode for technical coaching is that many coaches lack skills in communicating and collaborating with non-technical stakeholders to gain assistance removing organizational constraints limiting the impact of their technical coaching.

Many act as if they were helpless leaves in the wind, with no agency to function as true consultants. When client personnel say

a recommended action would be difficult or impossible, the coach stops working for the change and tries to work around the problem in some way rather than solving it.

36.1.7 | Coaches “Go Native”

One failure mode for technical coaching is that when coaches remain with the same team for an extended time, they start to feel like members of the team rather than coaches.

They “take ownership” of the team’s work instead of their true work of guiding the team forward.

36.1.8 | Teams Grow Bored

Another failure mode for technical coaching is that when coaches remain with the same team for an extended time, the team either grows tired of them or starts to take them for granted, and no longer listens to their advice.

36.1.9 | Coaches Become Part of the Background

When coaches are always available (assigned to the same team full-time), team members feel no sense of urgency to try the things the coaches suggest, as they “know” the coach will still be around tomorrow and the next day and the next.

36.1.10 | Teams Become Dependent on Coaches

Another failure mode is that when coaches remain with a team full-time, the team has no opportunity to think about, stumble through

and learn from, and finally internalize any of the new practices the coach shows them. When the coaching engagement ends, the teams are still not self-sufficient.

36.1.11 | Coaches As *De Facto* Team Leads

Most technical coaches enjoy the technical work themselves, and they like to solve problems. There's a tendency for them to become too actively involved in team tasks. Sometimes this results in teams looking to the coach as a kind of technical lead. They don't learn to make choices or take chances on their own initiative, but turn to the coach for leadership.

When visiting teams to observe or to coach the coach, I've often noticed non-verbal exchanges between team members and their coach in the nature of "asking permission." It blocks teams from becoming self-sufficient.

36.2 | Addressing the Problems

Let's consider some changes we might make in the way we approach technical coaching to address these problems.

36.2.1 | Balance Business and Technical Improvements

I hear many circular debates among consultants along the following lines:

- It's important to determine the *right things to build* to deliver business value to customers. There's no value in being able to build and deliver software effectively if we don't know what to build. You end up with a lot of "stuff" nobody wants.

- It's important to establish the capability to build and deliver software effectively. There's no value in understanding the right things to build if there's no way to execute. You end up with a list of pretty things you'd like to have, but can't obtain.

There may be a couple of reasons for this type of debate.

First, people are human. They tend to focus on the subjects they understand best, and to de-emphasize other subjects. This may be particularly true for people who see themselves (or portray themselves) as “experts.” They don’t want to spend a lot of time talking about things they don’t understand well, especially in front of clients or prospects.

Second, there’s been a sort of “pendulum” effect in our line of work over the years. People focus on technical excellence for a few years, they begin to neglect customer value, and there’s a backlash. The backlash leads to the opposite situation: People focus on business excellence for a few years, they begin to neglect technical excellence, and there’s backlash in the other direction.

I suggest this type of debate is not useful. Both arguments are right, and both are wrong. They’re right in that it’s important to determine the right things to build, *and* to be able to build them. They’re wrong in that there is no case when the “other” capability is secondary. The two considerations are of *equal* importance.

As consultants, we need to structure our proposals for organizational transformation in a way that places equal emphasis on the business and technical areas. This may require some education for consultants as well as for prospects.

36.2.2 | Focus On the Key Dependency: Leadership Challenges

One concept in my approach that differs from conventional wisdom is that I distinguish between the *business driver* for the transfor-

mation and the *focus for improvement* during the transformation program.

The key dependency that enables *every* market-facing competitive capability is *stable production operations*. The business driver for the transformation may be one or more of those competitive capabilities. The primary focus for improvement efforts has to be the key dependency.

Given the typical approach to transformations by most consultancies today, the key dependency in our model would receive short shrift. It falls into the bubble on the flowchart labeled, “And then a miracle occurs.”

Instead, the key dependency has to be front and center. It’s the “beacon on the hill” that lights the way to achieving the business goals of the transformation.

36.2.3 | Focus on the Key Dependency: Coaching Challenges

If we can get consultants and client leaders to recognize the centrality of *stable production operations* to the success of the transformation program, that’s a good start. Then we run into a couple of interesting challenges.

First, the global shortage of qualified technical coaches makes it hard to staff up a transformation program appropriately to address this need. I needn’t belabor the details again.

Second, the key dependency happens to fall in an area that’s currently undergoing rapid change. Most people working in operations aren’t well-versed in emerging solutions. They are still operating in a reactive mode, responding to support tickets as they are opened. Most technical coaches, scarce as they are, know little to nothing about proactive approaches to supporting applications in production.

Most technical coaches have strong expertise in application software development; their skills are primarily in areas like programming, testing, and continuous integration. They aren't prepared to guide teams in applying new methods of managing production operations.

The general solution will require a shift in emphasis throughout the technical coaching industry. Coaches need to learn about new methods of monitoring and supporting operations, merging development and operations teams, building observability into application designs, and designing work flows for teams that take these considerations into account. This is a significant learning curve. It's not a question of reading an article or following a five-minute tutorial.

Development practices such as trunk-based development, continuous integration, test automation, and so forth are dependencies of the key dependency. That is, stable production operations require all the underlying technical practices to be applied in ways that ensure deployments will not break production. Production, rather than development, becomes the focus of our thinking regarding technical practices.

A focus on the health of production operations has to become core to coaching all technical teams, regardless of their particular roles or activities.

36.2.4 | Mitigating the Technical Coach Shortage

The long-term solution is to shift the technical coaching industry toward a more-comprehensive and meaningful definition of the range of skills a technical coach needs. This is connected to the information in "Coaching Skills."

Based on a broad consensus among technical coaches regarding the specific skill sets involved, we (as an industry or within individual

consultancies) can develop training programs for coaches to build the missing skill sets.

There are a couple of shorter-term solutions that can mitigate the shortage.

First, an approach to technical coaching developed by Llewellyn Falco, and now used by quite a few of the top technical coaches in the industry, enables one coach to work with up to six teams concurrently.

That is not the way Falco applies his approach, but there is potential to use it to gain some leverage from a relatively small number of technical coaches on a transformation engagement. It promises to be a practical approach for medium-scale transformation programs involving tens of teams.

The approach is described in more depth later in this chapter.

Second, we can train a group of client personnel to an entry-level technical coaching capability. The external coaches can then rotate among them to make course corrections and “coach the coaches” throughout the transformation program.

This less-than-ideal approach may be a practical workaround for the coach shortage for large-scale transformation programs involving hundreds of teams.

36.2.6 | Building Coaching Competencies

The remaining problems listed in section 33.1 boil down to a lack of coaching competencies. Technical coaches today have strong technical skills and some ability to mentor people in effective practices, but generally they lack a broad coaching skill set.

Stickiness is reduced when coaches take too active a hands-on role in a team’s work, and when they sit with a team full-time such that the team has no “alone time” for new learnings to sink in.

In my view, this reflects a lack of clarity about where the coach's responsibilities end and the team's responsibilities begin.

Coaches who tend to lead or drive team events such as retrospectives and stand-ups effectively take ownership of the team's outcomes. Team members never have a chance to learn to take ownership of their work. This reflects a lack of *facilitation* skills, which is a key area of coaching competence as defined in the Agile Coaching Competency Framework. This is another factor that affects stickiness.

Coaches who behave like "victims" of current organizational constraints in the client's environment may be unclear about the things a consultant normally does to drive change. It seems to me this reflects the mentality that the coach is "just another team member." This is related to building and maintaining influence and trust with the client.

The other issues mentioned in section 33.1 can occur when coaches lack skills in interpersonal interactions, such as communication models, conflict resolution, and ways of encouraging or nudging people to change their habits.

As an industry - or perhaps within forward-looking consultancies to create a competitive advantage - we need to build these skills among technical coaches in an intentional way.

36.3 | Team-Level Technical Coaching Model

In this section, I'll describe a pattern for effective technical coaching. It addresses the problem of "stickiness" of new learnings, and also enables one coach to influence more than one team at a time.

Independent technical coach Llewellyn Falco arrived at this model as a way to improve the effectiveness of his own coaching work,

after noting that many teams he had helped did not continue their improvement on their own after he left. Others who do technical coaching subsequently found the method useful, and it's gaining momentum.

36.3.1 | A Scalable Approach

In my view, an additional benefit of the model is that we can scale a small number of technical coaches to support a medium-scale transformation program. This offers a practical mitigation strategy for the general shortage of qualified technical coaches.

Do we need to “scale” coaching? I can share an anecdote from experience. A certain consultancy did good work on a small transformation engagement for a large client. On the strength of their success, they won a larger contract from the same client.

The second contract included technical coaching for 30 development teams (among other things at the management and executive levels). The consultancy did what they all do: They scrambled to hire some technical coaches at the last minute, to avoid making commitments to people before they had a signed contract in hand.

They were able to find three people who had reasonably strong technical skills, but none of the broader coaching skills, and very little experience in coaching as opposed to direct delivery work.

Using conventional coaching methods, that provided the consultancy with sufficient coaching capacity to support three client teams out of 30, and the level of coaching skill was not high. This was not a Good Thing. One of the consultancy’s senior coaches played a “coach the coaches” role, and expressed frustration to me privately.

Had they applied Falco’s approach, the three coaches could have supported a maximum of 18 client teams; far short of the 30 teams included in the transformation program, and probably not enough

to smooth over cross-team dependencies in order to get anything of significance done.

This underscores the severity of the shortage as well as the inadequacy of conventional methods of coaching.

36.3.2 | Foundations of the Coaching Model

There are several foundational ideas underlying Falco's approach, including:

- consistent schedule for activities
- consistent locations for activities
- alternating touch time and alone time for teams
- blend of coaching, mentoring, facilitating, and teaching
- preference for experiential learning
- training in short, focused segments rather than lengthy classes
- use of a strict form of mob programming and strong-style pairing for coaching sessions
- a slow pace of change that gives people time to work through the S-curve for new practices, and to internalize new learnings
- one new thing at a time, to enable *fluency*
- frequent small retrospectives with action taken on feedback
- explicit measures to prevent teams growing bored with coaches
- explicit measures to reach beyond the team to address organizational constraints
- assess progress when team is absent and team is working independently

Having a consistent schedule for activities that involve the coach helps teams manage their time. Without this, coaches are at the mercy of the client's existing schedule. Typically, in organizations that are in need of change, that meeting schedule is at the root of

many problems. To be effective, this is one of the first things a coach has to break.

The daily routine Falco describes is as follows. This is from the point of view of a technical coach.

Start Time	End Time	Event
9:00	11:00	Coaching Team A
11:00	12:00	Learning Hour
12:00	1:30	Lunch
1:30	3:00	Coaching Team B
3:00	5:00	Coaching Team C

Having a consistent location for recurring activities makes it easier for team members to remember when and where the events will take place and to make room in their schedules to participate. Without this, recurring events occur in different locations based on the availability of conference rooms and other facilities. Client leadership may have to exercise formal authority to make this happen.

Alternating touch time and alone time gives teams the opportunity to take ownership of their own continuous improvement, to practice and stumble with new practices, and to internalize new learnings. Ultimately the teams can become self-sufficient and independent of the coach.

This alternation also helps teams avoid taking the coach for granted, as they are not always around, and lengthens the time it takes for teams to become bored with the coach.

If the coach has a tendency to “go native,” being away from the team periodically also helps with that problem.

In Falco’s model, there are two cadences of alternation between coach touch time and team alone time.

Each day, the coach works with any given team for an hour in a mobbing session and team members participate in the Learning

Hour. The rest of the day, the team works without the coach present (Zuill).

During the mobbing session, the coach acts as facilitator. One team member has the keyboard and one other is the navigator. Roles are switched at frequent intervals, such as 3 to 5 minutes (Falco).

The coach works with up to three teams for two weeks, and then leaves the teams alone for two weeks. The purpose is to give teams time to internalize and gain fluency with new skills (Larsen).

In a nutshell, when a person has a *fluent* skill, they can perform that skill without having to think about how to do it. When they are working under stress, they can perform the skill effectively. When they lack fluency in a skill, they have to think about how to do it while they're doing it.

The model recognizes that a human can deal with one source of stress at a time. The effort to learn a new skill is a source of stress. The implication is people can't effectively gain fluency in a new skill if they're already under stress from some other source.

In the context of an organizational transformation program, the program itself is the primary source of stress. That means a crucial part of the transformation engagement is to alleviate the stress arising from the transformation. Otherwise, people will be unable to gain fluency in new skills, as their minds will be occupied with handling the stress of change.

When the sole source of stress is the new skill being learned, then people can practice the skill until they gain fluency with it. Then they can move on to learning another new skill.

The pace of learning in this model takes these factors into account. Progress is slow and deliberate, so that people have time to gain fluency with each new skill.

During the day, the coach balances coaching, mentoring, facilitating, and teaching as needed to help each team. The differences among these skills are described in "Coaching Skills."

When in teaching mode, the coach exploits experiential learning techniques. This approach is thought to be more effective for inculcating practical skills than the conventional lecture-then-practice approach (Kolb).

Each event concludes with a brief retrospective lasting between about 5 and 10 minutes. The scope is kept small so that action items can be addressed without delay and everyone can see progress.

For any single team, the model looks like this:

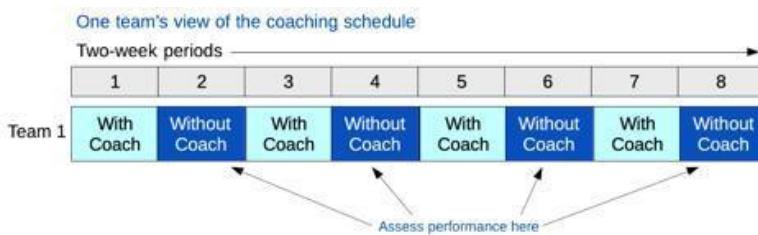


Figure 33.1: Single team's view of coaches

To enable the coach to address organizational constraints and obtain cooperation from team stakeholders, Falco makes a point of having lunch with one individual each day. These are not members of the teams he is coaching, but they are close to the teams in the organization's universe.

Lunchtime conversations are not usually about work. There are no immediate tactical goals for the conversations. The objective is to build positive working relationships. As challenges arise in the course of the engagement, these relationships can open doors for resolving issues.

Finally, it's important to measure team performance at the times when the coach is absent. When the coach is present, teams tend to do what the coach asks or to do things they believe the coach would like to see. This skews the results of any observations we might make about the team's level of fluency or internalization of new practices.

36.3.3 | What Happens in the Mobbing Sessions

During the regularly-scheduled mobbing sessions, the technical coach works with a single team. The team works on actual items from their work queue or backlog.

Work items are not cherry-picked to illustrate any particular technical practices or techniques. Techniques are introduced as the work demands them. That keeps all new learning in a real-world context, and limits the number of new concepts that are introduced at any one time.

The room is set up in the usual way for Mob Programming, with a single workstation at the front of the room, connected to a projector. The driver stands there.

Another team member is designated the navigator. They usually stand near the front of the room, possibly off to one side, so they can see the screen and everyone else can see and hear them.

Many variations are possible for Mob Programming. In this context, work is done using a strong-style pairing model in which the person at the keyboard isn't supposed to think. The only reason they type anything is that the navigator guides them. Everyone else is an observer.

Roles are switched every 3 to 5 minutes, so that everyone on the team has multiple opportunities to be the navigator and the driver, as well as ample opportunity to observe and learn.

The coach may act as a facilitator most of the time and as a teacher, mentor, or coach part of the time as appropriate.

Toward the end of the session, work stops and the group has a brief retrospective.

An important point is that the goal of the mobbing sessions is not to get work finished quickly. The goal is to learn. To that end, progress

is deliberate and the rationale and details of every new technique is explained carefully.

36.3.4 | What Happens in the Learning Hour

The Learning Hour is set aside for presenting specific topics in a more formal way than the Mob Programming sessions. Topics are short, so they will fit into the one-hour time slot. The lessons are designed around the experiential learning pattern to maximize learning effectiveness.

Some of the sample topics Falco lists in his “day in the life” video are:

- Koans
- Katas
- TDD
- Fake it till you make it
- Approval tests
- Combinatorial Testing
- Git
- Peel and slice
- User story writing
- Sparrow decks
- *many more*

As you can see, the topics are narrowly focused and learning goals are of limited scope so that people can get value within an hour.

Falco’s observation from field work is that one hour of focused learning per day yields significant improvement over time. In a presentation, he suggests 5x improvement can be achieved in one year’s time in this manner.

On the other hand, he doesn’t mention how that is measured. For me, the key point is that this approach is more effective than the

“heavy” classroom approach. Exactly how many X’s it yields is less interesting, and probably highly variable from case to case.

Three-to-five-day formal classes in which participants attend 8 hours a day are far less effective. It’s been known for many years that when new skills aren’t applied in practice within a couple of weeks of training, people forget what they saw in class. Besides that, people get worn out from sitting in a class all day long, day after day, and lose focus.

36.3.5 | Avoiding Burnout and Boredom

Falco’s model includes a mechanism to hand teams off to new coaches. Advantages include avoiding team boredom, avoiding coach burnout, and bringing additional perspectives and coaching styles to bear.

The mechanism is called the Guest Coach Program. A new technical coach joins the current one for a couple of weeks. First, the new coach shadows the original one. Then, the new coach facilitates team events while the original one observes and guides. Finally, the new coach takes over, following the same schedule and routine as the original coach did.

Technical coach Emily Bache describes this in her talk at SCLConf 2018. At the time of this writing, she is also preparing an ebook on LeanPub entitled *Technical Agile Coaching*, that covers Falco’s approach (Bache).

36.4 | Scaling the Coaching Model

As any single technical coach works with three teams in the course of a day and alternates two weeks on and two weeks off with the teams, the model offers a straightforward way to scale. A coach

can work with three teams for two weeks and with three different teams the following two weeks.

For example, two technical coaches could support up to 12 teams in this way:

Coaching Schedule: 12 teams, 2 coaches								
	Two-week periods →							
	1	2	3	4	5	6	7	8
Coach A	Teams 1,2,3	Teams 4,5,6	Teams 1,2,3	Teams 4,5,6	Teams 1,2,3	Teams 4,5,6	Teams 1,2,3	Teams 4,5,6
Coach B	Teams 7,8,9	Teams 10,11,12	Teams 7,8,9	Teams 10,11,12	Teams 7,8,9	Teams 10,11,12	Teams 7,8,9	Teams 10,11,12

Figure 33.2: 12 teams, 2 coaches

Depending on how many qualified technical coaches are available, this approach could scale linearly to an arbitrarily large transformation program. Given the current shortage of qualified technical coaches, it is probably feasible for medium-scale engagements involving no more than some tens of teams. For large-scale programs involving hundreds of teams or more, there are simply not enough qualified technical coaches available.

We have to find creative workarounds for that problem, at least in the short to medium term.

36.5 | Developing Internal Technical Coaches

As a workaround to the shortage, to support large-scale transformation programs we can train client personnel to serve as entry-level or apprentice technical coaches under the guidance of qualified technical coaches.

The downsides are:

- longer timelines to effect meaningful improvement in large-scale transformation programs;
- impact of shifting some of the best technical people in the client organization away from business as usual to become apprentice technical coaches;
- overall coaching effectiveness will not be as high as it could be if all the technical coaches had senior-level coaching competencies; and
- as the key dependency lies within the technical sphere (stable production operations), it will be the bottleneck for achieving the overall transformation goals (competitive capabilities).

The qualified technical coaches will perform the training. Afterwards, the apprentice coaches will play the role described above with one to three technical teams.

The qualified coaches will rotate around the organization, visiting each mobbing session and Learning Hour and consulting with the individual apprentice coaches to ensure they are continuing to hone their skills and teams are making progress.

This is clearly not an ideal solution, but may be a practical way to scale technical coaching in the face of the current shortage of qualified coaches.

36.6 | Barriers to Adoption

We are years away from having a strong pool of qualified technical coaches in the job market. Until we can build up the community in this area, this level of technical coaching expertise will remain a “boutique” offering.

Adoption of Falco’s model is progressing very slowly among practitioners. Just as client personnel are reluctant to move outside their comfort zones, technical coaches are reluctant to move outside

their own. The idea of such a light touch with teams doesn't seem practical to many of them.

They may also be impatient to introduce a great deal of information quickly, thinking this will help the teams pick things up faster. In my view, if that were the case it would have happened by now, after all these years of conventional coaching methods.

A couple of other market trends may be contributing to the shortage of qualified technical coaches. For one, team-level coaching is seen as a low-end commodity staff augmentation service rather than as a genuine coaching service. Large consultancies are offering such coaching on that basis and at low rates. Client leaders tend not to understand the nature and importance of this level of coaching, and aren't interested in paying for it.

For another, most technical people are unaware of the importance of cultivating the other coaching-related skill sets. There is a common assumption that strong technical skills are sufficient to provide useful guidance to client teams.

The next point may be more a *trend* than a *barrier*, but it does affect the shortage of technical coaches. Understandably, many of the more-senior technical coaches feel as if they can add more value for clients if they focus higher in the organization. They transition from team-level technical coaching to mid-management or executive coaches. That makes the hole at the team level even bigger.

37 | Communication Models

The single biggest problem in communication is the illusion that it has taken place.

— George Bernard Shaw

Here are some communication models that we've found useful in working with client personnel, and that we often teach and mentor to help improve communication in client organizations. We've also mentioned caveats to be aware of, as some consultants and coaches tend to go overboard with some of these techniques. The techniques are listed alphabetically by name, and not in order of usefulness or recommendation. You may notice that most of these models or frameworks are nearly the same, substantively.

- Active Listening
- Appreciative Inquiry
- Clean Language *
- Crucial Conversations
- Emotional Intelligence
- Getting To Yes
- Mindful Kindness
- Powerful Questions
- Radical Candor *

* Use with caution - see caveats below

37.1 | Active Listening

This is a highly effective, practical, and learnable technique for engaging in meaningful conversations and clearly understanding the other party. Our coaches practice this technique for use in their coaching, mentoring, teaching, and facilitating work, and they teach and mentor the technique for client personnel to improve the quality and effectiveness of communication in the client organization.

The technique specifies particular verbal and non-verbal techniques to signal that we are engaged and listening carefully to the other party. Some of these feedback techniques can be “gamed,” so it’s important to practice truly listening and not just mimicking the techniques.

The *Skills You Need* website contains excellent material about Active Listening. You can also find an amusing and yet accurate introduction from the television series, *Everybody Loves Raymond*, on YouTube. See *References* for links.

Caveats

None.

37.2 | Appreciative Inquiry

Developed by David Cooperrider and Suresh Srivastva at the Weatherhead School of Management, Case Western Reserve University, Appreciative Inquiry is what AI stood for before Artificial Intelligence came along.

The Center for Appreciative Inquiry describes it as

a way of being and seeing. It is both a worldview and a process for facilitating positive change in human systems.

The site notes that

[e]very human system has something that works right—“things that give it life when it is vital, effective, and successful. AI begins by identifying this positive core and connecting to it in ways that heighten energy, sharpen vision, and inspire action for change.

You will recognize these points as consistent with other observations various researchers and authors have made, and described using different words. In particular, the core idea in the book *Switch: How to Change Things When Change Is Hard*, is to identify what is already working well and build on that to effect further improvement. The connection with organizational transformation consulting is probably obvious.

Many books have been published on the subject of Appreciative Inquiry. It is not a communication framework as such, but if we can learn to approach problems from a positive standpoint, it becomes much easier to find constructive paths forward.

Caveats

None.

37.3 | Clean Language

Clean Language is a therapeutic interviewing technique developed by psychologist Dr. David Grove. The technique involves asking a set of standard questions in a particular order and with specific

voicing and pacing. The technique seems to help patients discover and express thoughts and feelings related to their emotional or psychological issues.

Used in therapy, the questions are asked in a sort of monotone, and at a steady pace that leads the patient to “zone out” and respond automatically, without having time to rehearse answers or worry about what sort of answers the therapist might be fishing for. The intent is to guide the patient toward a good outcome without polluting their thinking with the therapist’s own assumptions or beliefs.

The technique has been adapted in business circles as well as among consultants as a way to interview people about work-related matters without introducing their own biases. The goal is to help the questioner learn about people’s thoughts and feelings without implying anything about what kinds of answers might be “good” or “bad.”

You can download the standard list of questions from www.e-russell.com. Here are a couple of examples of Clean Language sessions, to give you an idea of what it looks like:

- What Is Clean Language. URL: cleanlearning.co.uk/resources/faq/what-is-clean-language
- Mindtools - Clean Language. URL: www.mindtools.com/pages/article/david-grove-clean-language.htm

Caveats

As they seek better ways to understand client needs, aspirations, and fears, consultants and coaches have looked to a wide range of therapeutic and communication models and techniques, and they have tried to learn a bit about psychology and human nature.

But *this is not their field.*

Business consultants and technical coaches *are not therapists* and their clients *are not patients*. Beware of this sort of thing being taken too far and becoming intrusive, insulting, or just plain silly.

37.4 | Crucial Conversations

This is an approach to guiding your conversations with others to ensure you stay on topic and avoid ego-driven language that shuts people down or unnecessary distractions into unrelated topics. It's very practical for everyday use in a business setting.

The technique is fully described in the book, *Crucial Conversations: Tools for Talking When Stakes Are High*.

Coaches practice this technique to help them function as effective communicators on the job. They also teach and mentor client personnel in the technique, as it's quite useful in everyday work.

Caveats

None.

37.5 | Emotional Intelligence

According to Wikipedia, Emotional Intelligence refers to “the capability of individuals to recognize their own emotions and those of others, discern between different feelings and label them appropriately, use emotional information to guide thinking and behavior, and manage and/or adjust emotions to adapt to environments or achieve one’s goal(s).”

This is not a communication technique as such. It is a framework for understanding our emotional nature and recognizing the emotional responses of those we wish to communicate with.

The concept is described in the book *Emotional Intelligence 2.0* by Travis Bradberry and Jean Greaves. It's also described on the PsychCentral website at psychcentral.com/lib/what-is-emotional-intelligence-eq/.

The model defines five categories of emotional intelligence:

- Self-awareness
- Self-regulation
- Motivation
- Empathy
- Social skills

Each area is explored in some depth in the book.

Caveats

None.

37.6 | Getting To Yes

This is a negotiation framework derived from work by the Harvard Negotiation Project. It's described in the book, *Getting To Yes: Negotiating Agreement Without Giving In*, by Roger Fisher, William L. Ury, and Bruce Patton.

The original context of the framework is business negotiation, but it is also very useful for conflict resolution. It provides a way to discuss controversial or divisive issues without getting tangled up in personalities or details that aren't relevant to the needs of the participants.

The framework comprises six integrated techniques:

- Separate the people from the problem

- Focus on interests, not positions
- Learn to manage emotions
- Express appreciation
- Put a positive spin on your message
- Escape the cycle of action and reaction

I find this framework to be quite practical and effective as well as relatively easy to learn.

Caveats

None.

37.7 | Mindful Kindness

A study reported on National Public Radio in the United States found that when people intentionally try to be empathetic and connect with others, they receive unexpectedly kind treatment in return. At the end of the day, they feel happy and energized.

I was unable to locate the study for reference. As memory serves, participants were asked to apply mindful and extreme “niceness” in every encounter they had with any person for any reason, for a full day. Without exception, participants reported very positive results.

For a coach involved in a stressful exercise such as an organizational transformation, this can reduce stress and avoid friction and conflict with client personnel, as well as help maintain the coach’s own clarity of mind and positive attitude.

37.8 | Powerful Questions

As described on the website of the International Coach Academy (coachcampus.com/resources/powerful-questions/), a Powerful Ques-

tion

is one that gives the client greater clarity, awareness, exploration. It gets them outside the box of what they're thinking and helps them shift...perspective.

Powerful Questions have three key characteristics. They are

- open-ended
- challenging
- free of judgment

To meet these characteristics, the questions generally start with the word, "What." They are not binary, yes-or-no questions.

A technical coach might try to initiate a team discussion about a broken build by asking, "What caused the build to fail?" rather than "Who broke the build?" or "Why is the build broken?" which carry a judgmental undertone. During the discussion, the question "What can we try to prevent the same problem happening again?" could come up.

It's okay if the question is a little uncomfortable for the client; part of the purpose is to help free them from a conceptual rut they may be in so that they can visualize alternative solutions. But "Who" and "Why" questions tend to put people on guard, and when that happens they can't easily consider alternatives, as they're focused on defending themselves.

This is a highly useful technique coaches practice intentionally, apply on the job, and teach and mentor client personnel to use in their own work.

Caveats

None.

37.9 | Radical Candor

Radical Candor is a communication framework developed by Kim Scott to improve the effectiveness of managers' communication with subordinates. Consultants and coaches use it to improve clarity of communication between people in all roles.

The basic idea is to “challenge directly” while “caring personally.” The goal is to avoid going beyond the point of diminishing returns in the attempt to be “nice,” but without going so far as to become a “jerk.” The following quadrant diagram summarizes the model:

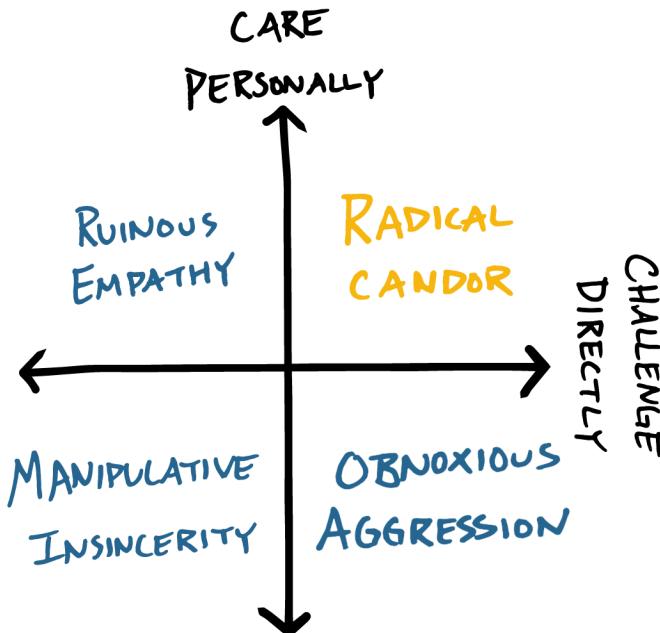


Figure 34.1: Radical Candor quadrants

In a nutshell, if you care a lot but are afraid to broach an important subject for fear of offending or hurting the other person, you succumb to “ruinous empathy.” If you don’t care about the person

or about the problem at hand, you might try to manipulate the other person for your own purposes; what Scott calls “manipulative insincerity.” If you want to address the problem but you don’t care about hurting the other person, you may practice “obnoxious aggression.” The desired balance is “radical candor.”

The book is *Radical Candor: Be a Kick-Ass Boss Without Losing Your Humanity*, by Kim Scott.

Caveats

The value in this model lies in the basic point that you can’t effectively solve problems if you’re afraid to mention them at all, and that there’s a way to discuss problems head-on without prevarication or waffling, and yet also with due empathy and respect for the other party.

In practice, it’s quite challenging for people operating under stressful conditions (such as those of an organizational transformation initiative) to remember to behave well and balance emotions perfectly. Radical Candor can be taken as a license to be rude and thoughtless, even if that is not Scott’s intention.

38 | Collaboration

Alone we can do so little; together we can do so much.
— Helen Keller

Merriam-Webster defines *collaborate* as

to work jointly with others or together especially in an intellectual endeavor

Collaboration is central to contemporary methods of working. It's a broad concept that can apply at different levels of abstraction in an organization, and that can be practiced with different levels of intensity.

It's unlikely we will be introducing collaboration as a completely new concept. People in the client organization are already collaborating in various ways.

38.1 | Improving Collaboration Improves Flow

There are many forms of collaboration, but some are more effective than others. Let's see how to improve continuous flow by improving collaboration.

38.1.1 | Collaboration In a Linear Process

Consider a linear software delivery process, sometimes called a “waterfall” process. Several teams perform discrete types of work

to prepare a software product for release. As each team completes its contribution, they pass the work along to the next team in the series.

- An analysis team prepares requirements and passes them along to a testing team.
- The testing team prepares a test plan and passes the requirements along to a programming team.
- The architecture team designs or modifies the solution architecture for the application, taking into account the new and changed requirements, and hands off to the programming team and the operations team.
- The operations team reviews the architecture to inform their capacity planning work. They may begin the process to add capacity to the environment to accommodate the application.
- The programming team writes code and passes it along to the testing team.
- The testing team and programming team circle around until everyone is satisfied with the code. Then they pass it to an infrastructure team for packaging.
- That team may pass it along to another infrastructure team responsible for provisioning and configuring environments.
- If database changes are involved, the work passes through a database team.
- Similarly, if other specialized work is called for, each specialist team contributes to the work and passes it along.
- A second testing team, responsible for testing at a larger scope than the first testing team, runs the application through its paces before approving it for production release.

There could be many more steps like these. Eventually, the application is passed along to the team responsible for placing into production. Following a “warranty” period, the programming team and testing team bequeath the system to a production support team.

That's a form of collaboration. All the specialist teams collaborate to prepare the application for production, and then to support it.

It might take some time, but the teams eventually deliver a working application to the production environment. The application provides business value to customers.

So, we're good, right? We're collaborating and we're delivering business value to customers. It's time for Happy Hour.

38.1.2 | Competitive Capabilities and Hand-Offs

Let's remind ourselves why we're here:

- Respond to Market (competitive capability for products that interact dynamically with the market)
- Shape the Market (competitive capability for products that interact dynamically with the market)
- Release on a dependable and predictable cadence (competitive capability for products that interact in a stable way with the market)
- Assure stable production operations (the key dependency for all the target competitive capabilities)

Can we support that using a linear process with hand-offs between specialist teams?

The short answer is, “no.” The most obvious reasons are:

- Each hand-off in a linear process may incur wait time until the receiving team is ready to work on the item. In practice, this is true for just about every hand-off.

- As discrete types of work are performed by different teams based on indirect communication (usually documents), the probability of miscommunication, misunderstanding, or missing information at each hand-off is high. This leads to back-flows and rework in the process, increasing cost and delay.
- The hand-offs and delays introduce high variation in cycle time, such that it becomes impossible to predict the release cadence. Customers are never sure when changes they require will be ready to use.

Our goal is not to “implement” collaboration just because we think it’s a Good Idea. Our goals are those listed above. We want to improve collaboration because that will improve our ability to meet the goals.

At least, that’s the assumption. Let’s test it.

38.1.3 | Competitive Capabilities and Continuous Flow

To achieve continuous flow, we need to remove all those wait states from the software delivery process. To maintain customer satisfaction and stable production operations, we need to do that in a way that keeps the application and the production environment healthy.

In the sample linear process above, we have about a dozen hand-offs. In that sort of environment, each specialist team will have its own list of things to do, its own priorities, and quite possibly its own management hierarchy that makes plans independently of other parts of the organization.

It’s highly likely that when any team hands off the work to the next team in line, the receiving team is already occupied doing something else. The work will have to wait.

With linear project planning, we can identify dependencies and try to plan out the duration of each step. But what we can't do, on a practical level, is know how long each of the wait states will be. That's because the blocker causing the wait state is not in our project plan; it's part of a different project.

For example, when our work is handed off to the second testing team in the series - the one that does the "heavy lifting" sort of testing that requires fully-configured test environments - they're already busy testing some other application. They're not available right away, and neither is the test environment.

For purposes of discussion, let's assume the *average* wait time between teams is three weeks. For many larger and older organizations, that's actually pretty optimistic, but it will serve for example purposes.

There are other delays lying in wait, too. In that sort of environment, it's typical for database schema, server provisioning and configuration, application configuration, software packaging, promotion of code through environments, testing, production deployment, deployment validation, and production monitoring and support to be performed manually. Every single task takes an unpredictable amount of time and is subject to human error.

That adds up to late discovery of issues, lengthy analysis to debug code and track down configuration errors, and a reactive approach to production support. More delays, more errors, more back-flows, more rework.

All of that represents irregular flow. When we have irregular flow, we don't have a reliable way to predict how long it will take to prepare a modified application for release into production. We fall down on one of our competitive capabilities - we can't release a stable product on a reliable and predictable cadence.

It goes without saying that if we can't even maintain a predictable release cadence for stable products, we surely can't provide quick

turnaround of dynamic products. So, we fall down on Respond to Market and Shape the Market, too.

38.1.4 | Structure First, Then Process, Then Practices

Improvements in organizational structure have the greatest impact, followed by improvements in process, followed by improvements in practices. How can we change structure in a way that will improve flow?

What would happen if we combined some of the teams? Let's say we combined the responsibilities of the analysis team, the first testing team, the architecture team, and the programming team into a single team.

38.1.5 | Reducing Hand-Offs Improves Flow

What's the effect of collapsing those teams? It reduces the number of hand-offs by four. That improves *flow* right away.

Earlier we agreed to assume average hand-off delay was three weeks. Removing four hand-offs reduces lead time by twelve (12) weeks.

We haven't done anything difficult yet. All we did was collapse some of the specialist teams into a single team. We didn't cross-train anyone. We didn't change any formal procedures, except the ones governing the four hand-offs that were eliminated. We didn't automate anything. We didn't improve any technical practices. We didn't add any tools.

This is what some consultants call "low-hanging fruit."

From the standpoint of *collaboration*, what we've done is improve the level of collaboration. The former members of four teams are

now working together directly. They don't have to wait for each other anymore. They were collaborating before, in a sense. Now they're collaborating better.

38.1.6 | Reducing Formality Improves Flow

What happens in real organizations is the individuals on the new team continue to interact in a pretty formal way initially. Now that we've simplified the structure of the teams, we can simplify the process as well.

In many organizations, particularly larger ones, there's an assumption that the only way to communicate information is through documents. The documents may not be printed on paper, but they're still documents. They may live on a SharePoint site, a Wiki, or a shared network drive.

The thing about documents is that they are an *indirect* communication channel. Indirect channels are well-suited to information that remains stable over time. Documents are great for specifying things like standard naming conventions, because that information doesn't change frequently. People can look up the relevant document when they need to know something about standard naming conventions. Otherwise, the document stays out of the way.

Live work on software changes involves quite a lot of information that is transient. That is, it changes frequently, if not almost constantly. When indirect communication channels like documents are used to communicate transient or volatile information, it tends to slow things down as people must wait until the relevant documents have been updated, reviewed, approved, and copied to the place where people can look them up.

For transient information, a *direct* communication channel is better. And the most direct communication channel of all is face-to-face.

When we collapsed the four teams into one, we eliminated the need for them to use formal procedures to share transient information about the work in progress. Now we can remove the standard procedures so team members will feel free to speak to one another directly.

We just improved collaboration a little bit more, reduced delay, and improved flow.

38.1.7 | Increasing Cross-Functional Collaboration Improves Flow

We have a development team that includes responsibilities for analysis, architecture, programming, and the initial levels of testing. They use informal, direct communication methods to share information about work in progress. Is there more to be improved with respect to collaboration?

In most organizations, when we take teams this far in the transformation, the individuals on the team are still working in their original functional silos. They're sitting together and speaking directly, which is all good, but they're still doing individual work and doing mini-hand-offs within the team.

We could improve collaboration a bit more if we ask people in different roles to sit down together and directly collaborate on work. How might that look?

Maybe an analyst and a tester could jointly work out the requirements and the test plan at the same time.

Maybe an analyst and an architect could jointly clarify their mutual understanding of how users will interact with the application, and what the architectural implications might be.

Maybe an architect and a programmer and a tester could jointly work out a high-level application design that will meet user needs, work well in the enterprise architecture, and be testable.

Maybe a tester and a programmer could jointly perform various kinds of testing, to eliminate the old-school back-and-forth between the two roles as they work the kinks out of the code.

You can probably see that there are a number of small delays, back-flows, and rework that could be eliminated in this way. Less impactful than simplifying the process, which was in turn less impactful than streamlining the team structure, but still quite valuable.

38.1.8 | Expanding Technical Skills Improves Flow

At this point, team members have learned to collaborate much more closely than when the transformation program began. But individuals are probably still doing their work the same way as they did before, for the most part.

That is, programmers are still writing code without executable functional checks. Testers are still following a documented test plan and executing test cases manually. Both of them are still reading requirements documents to understand what to do, even if they can sit with analysts whenever they need clarification.

You might think improving individual technical practices has no connection with collaboration. Actually, they're already in the habit of sitting together and creating results jointly. While they're at it, they can also help each other improve their own technical practices.

If a tester and a programmer are sitting together, the tester can help the programmer learn testing skills and the programmer can help the tester learn programming skills.

If a tester with programming skills is sitting with an analyst, then instead of jointly producing a test plan they can jointly produce a suite of executable test cases.

Programmers working from executable test cases can be more confident of handling all the necessary edge cases and of avoiding over-engineering the solution.

Eventually there's no difference between an analyst, a tester, and a programmer. They're all *developers* now.

38.1.9 | Continuous Improvement

That team can take themselves much further through collaboration. We can loop back and pick up other specialist teams, as well.

Using the Cycle of Change, we can break down most of the walls between specialist teams and smooth over most of the gaps in individuals' skill sets.

38.2 | Team Cohesiveness

Sounds good, doesn't it? If people would just sit down together and collaborate closely, they could eliminate pretty much all needless delay and rework from their process. Nothing to it!

So, why don't people do that?

People are *human*. Humans have many admirable qualities. They have other qualities, too.

When you read material about "coaching" and "collaboration," most of it concerns emotional and psychological aspects of collaboration.

As logical as it may be for people to sit down together and work jointly, it doesn't happen naturally.

The starting point for this sort of transformation is usually an organizational culture characterized by finger-pointing and punishment. That environment doesn't give people confidence that it's safe for them to collaborate, or share information, or be honest about their concerns, or a lot of other things.

38.2.1 | Functional and Dysfunctional Teams

In Patrick Lencioni's landmark book, *The Five Dysfunctions of a Team*, the author presents a triangle or pyramid diagram that illustrates the way "dysfunctions" build on one another. The result is a team that doesn't perform well and whose members are not very happy working together.

The base of the pyramid has *absence of trust*. In Lencioni's model, absence of trust is the root of all team dysfunction.

Coaches and organizational change specialists devote a great deal of attention to building trust on teams.

Jean Tabaka's *Collaboration Explained: Facilitation Skills for Software Project Leaders*, provides practical facilitation techniques for people in leadership roles to use in various kinds of meetings and working sessions.

Many practicing coaches have an assortment of games and exercises they use to break down interpersonal barriers on teams as a way to foster trust.

Trust is key. It depends on team mates being able to rely on one another's word. It depends on team members being willing to help each other, and willing to request help when they need it.

It's hard to establish trust in the midst of an environment that lacks psychological and political safety. Hence the importance of safety.

In Lencioni's model, the next level of dysfunction built atop *absence of trust* is *fear of conflict*. People will avoid mentioning problems or contradicting anyone because conflict is *unsafe* in a dysfunctional environment. You could be branded a troublemaker, or thought to be trying to undermine someone for personal gain, or (ironically) "not a team player."

So, you don't step up. You just watch things fall apart, and shrug. And that's the next level in the pyramid: *lack of commitment*.

Why should you care, if the most likely outcome is that you'll be punished?

People who aren't committed don't really care about outcomes. They don't take responsibility for results. They aren't accountable. It leads to the next level in the pyramid: *avoidance of accountability*.

The top of Lencioni's pyramid is *inattention to results*. Nobody really cares what happens. That completes the picture of the dysfunctional team, according to him.

38.2.2 | Free Advice Is Worth Every Penny

I've seen articles and blog posts that offer sound advice to deal with Lencioni's five dysfunctions. It usually goes something like this:

- Absence of trust? Trust each other!
- Fear of conflict? Learn how to disagree constructively!
- Lack of commitment? Commit to the team's goals!
- Avoidance of accountability? Be accountable!
- Inattention to results? Pay attention to results!

Most of the advice about building trust on teams reminds me of the sketch from *Monty Python's Flying Circus* entitled, "How To Do It," in which Alan explains how to play the flute:

Well you blow in one end and move your fingers up and down the outside.

I assume that's technically correct. I don't find it especially useful.

38.2.3 | The Tuckman Model

In 1965, psychologist Bruce Tuckman published an article proposing a sequence of stages through which small groups pass when they are organized into a team. The stages are:

- Forming
- Storming
- Norming
- Performing

The model is widely cited and has been used as a basis for many variations, some simpler and some more complicated than the original.

While this is a somewhat subjective model, it does seem to align well enough with observations of teaming behaviors to be useful for discussion purposes. Plus, it rhymes. That's got to be worth something.

I've seen some teams that didn't pass through these stages, but I've seen more that did so.

Many team-level coaches use this or a very similar model to guide their approach to building trust, accountability, commitment and so forth on teams.

38.2.4 | Breaking Through Initial Barriers

When we come into an existing team environment where people have had to adapt to a toxic organizational culture, we often find teams operating at the Storming level on a permanent basis. They have low mutual trust and are very poor at resolving conflict. In many cases, they don't even admit there *are* any conflicts.

They prevaricate, side-step, walk out, work separately, gossip, hide information, and don't communicate.

In the situation I share in Chapter 34, section 31.10.3, Case 7, that's the state the team was in. They had been working together for about a year. They had never progressed beyond the Storming stage in Tuckman's model. They were stuck at the *fear of conflict* level in Lencioni's model.

In that single working session I was unable to make a start toward building trust. I was pretty happy with the fact people started to express their true feelings (even if somewhat explosively), and to reach some sort of tentative agreement to work together to improve the situation.

Their agreement lacked real commitment or any sense of responsibility, but it was better than nothing. At least two of the team members had admitted publicly that everything wasn't fine and dandy after all. Others checked email or pretended to work.

I mention that here because when you come into a situation like that, the initial steps toward building trust are going to be the most challenging of all. Once you get the ball rolling, it will become a little easier to make progress.

Many coaches have favorite exercises or games to break the ice with teams. The idea is that if people start to get to know one another on a personal level, they'll be more open to collaboration in the work context. There's some merit to the idea.

These games usually take the form of having each team member share something of a personal nature with the group. It isn't meant to be anything horrific, like "I killed my neighbor 35 years ago and they never found his bones in my back yard." It's meant to be something fairly benign, like "I enjoy country music" or "I like cheeseburgers."

Another popular game with many coaches is the truth-or-lie game, in which each team member says three personal things of which one is false. Others have to guess which one. So, someone might say, "I play the guitar, I was born in the back seat of a taxi in the middle of a traffic jam during an earthquake, and I have six toes on my left foot," when in truth he plays the banjo.

But this game is a bit risky, as many people aren't comfortable saying that much about themselves. Saying one thing is hard enough. Saying two things, and then having to muster the creativity

to make up a credible lie on the spot, is too much pressure for some. I would never be able to do it, myself. I'm not that creative.

Games like this can work, and they can backfire. My observation is it has a fair chance of working when the team is newly-formed and the general atmosphere in the company is not too terrible. In a situation like the one in Case 9, it will be like pouring gasoline on a fire. The last thing any of those people wanted to do was get to know each other better. They would have used games like those to shred each other verbally.

In those situations, you have to take a much slower approach. Your best tools won't be games and exercises that take five or ten minutes. Those will be a waste of time, or worse. Instead, you'll have to build working relationships with the team members individually, and make use of communication tools like those described in "Communication Models," while relying on internal controls for yourself, like those described in "Internalities."

You might even try teaching some of the internalities to the team. "The Four Agreements" is a good one that's pretty easy to convey, if they're open to anything at all at that stage. Do *not* expect quick wins.

38.2.5 | Team Development and Coaching Stances

Every individual, team, and organization will be unique enough that there's no way for me to provide a realistic standard template for taking your teams forward. If you're using Falco's model and working with 3 teams, or a "scaled" version of it and working with 6 teams, you'll be dealing with 3 or 6 quite distinct situations simultaneously. If you're using conventional coaching methods and you're spread across 3 or 4 teams (a pretty common situation, unfortunately), you'll have 3 or 4 distinct situations. There's no "standard" approach to this.

A very high-level and generic rule of thumb: Mapping Tuckman team development stages to coaching stances, we might try something like this:

- Forming - teaching stance (mostly)
- Storming - coaching stance (mostly)
- Norming - facilitating stance (mostly)
- Performing - letting go to promote independence

But that's a very generic picture. Your mileage may vary.

39 | Internalities

In all things have no preferences.

— Miyamoto Musashi

By *internalities* I mean skills, methods, and practices that pertain to the internal state of mind of a coach. I believe these things help us to remain focused and balanced under the stress of guiding an organizational transformation.

These are things I would like to ask coaches to become familiar with and, if they choose, to practice mindfully. They may also choose to teach and mentor some of these techniques with client personnel, if it makes sense to do so in context.

This is the aspect of professional practice that is farthest afield from the crude mechanics of organizational structure and process, and technical methods and techniques. Yet, it can help consultants and coaches become more effective in helping client organizations.

Let's consider a few of these.

39.1 | Beginner's Mind

Wikipedia (as of May 7, 2019) defines *beginner's mind* this way:

Shoshin (初心) is a word from Zen Buddhism meaning “beginner’s mind.” It refers to having an attitude of openness, eagerness, and lack of preconceptions when studying a subject, even when studying at an advanced level, just as a beginner would.

The idea of beginner's mind was popularized in the West by Shunryu Suzuki's book, *Zen Mind, Beginner's Mind*. The basic idea is to approach a new situation or a learning opportunity with all the openness and curiosity of a beginner. It does not mean you discard everything you've learned in life so far. It only means you "empty your cup" so that you can receive more knowledge, more insight.

Nithyananda Sanga describes the concept in less pedestrian terms:

By definition, having a beginner's mind means having an attitude of openness, eagerness, and freedom from preconceptions when approaching anything. Beginner's mind is actually the space where the mind does not know what to do. It is that delicious state when you are sure of nothing, yet completely fearless, totally available to the moment.

Scott Jeffrey has an excellent write-up about beginner's mind on his site (Jeffrey). It is highly recommended. He describes four simple exercises that can help us cultivate (or *re-discover*, as it's the natural state of our child mind) our beginner's mind:

- Mindful Breathing
- Grounding
- Mindful Observation
- Dropping Labels and Identities

Read the descriptions and guidelines on Scott's site.

The beginner's mind has practical application in our work. It enables us to be open to learning about the client's situation, context, point of view, constraints, vision, hopes, and fears. It helps us imagine solutions to problems or, at least, directions for exploration.

I consider beginner's mind to be the most fundamental and powerful internal tool for work and life.

39.2 | Eye of the Hurricane

This is a metaphor or visualization to help enable spiritual centering so that the mind can focus on the task at hand without distraction or negative self-talk.

A hurricane is powerful, spectacular, violent, and massive. Wind, water, and debris swirl around at high speed, leaving total destruction in their wake.

At the center of all this activity is an area of utter calm - the eye. Within the eye, all is quiet and still. The sky above is blue and peaceful.

When we are in the midst of an organizational transformation, there is frenetic activity all around us. Much of that activity is not going smoothly. People are worried, frustrated, stressed, angry. It's easy for consultants and coaches to become caught up in the fury.

It's helpful for us to have a way to re-center ourselves so we can think straight. After all, *someone* has to think straight in the midst of all the noise and fuss. The eye of the hurricane may be a useful visualization tool to clear the mind and settle the spirit.

This is not a published technique, and there is no book reference for it.

39.3 | Non-Attachment

This is a concept from Zen that can be helpful for people in the role of coach, mentor, or teacher. Thich Thien An introduces it like this:

Our world is a world of desire. Every living being comes forth from desire and endures as a combination of desires. We are born from the desire between of our father and mother. Then, when we emerge into this

world, we become infatuated with many things, and become ourselves well-springs of desire. Through desire we give rise to attachments. For every desire there is a corresponding attachment, namely, to the object of desire.

Non-attachment is a useful concept in our work because of the tendency for coaches to “go native.” After working with a team for a while, a coach may start to feel as if he/she is actually a member of the team, and the team’s deliverables and responsibilities are the coach’s own.

In fact, the coach’s role is to help the team grow and learn. The coach should *care*, but should never *own* the team’s outcomes. The coach must not become *attached* to the team’s outcomes, in the Zen sense. You demonstrate, you teach, you facilitate, you guide, you mentor, you coach. Then you let go.

39.3.1 | The Wind and the Sun

One day, the wind and the sun were arguing about which was the stronger.

The sun said, “See that man over there? Let’s see who can get him to remove his jacket in five minutes’ time. That will prove which of us is the stronger.”

The wind agreed, and began to blow. He tried and tried to blow the man’s jacket off, but the man only hugged it tighter around himself. The five minutes ran out, and the wind stopped.

Then it was the sun’s turn. He shone and shone, and soon it grew hot. The man removed his jacket well before the five minutes expired.

The wind had to admit the sun was the stronger.

Client team members have their jackets - their habits. If you try to blow the habits away by force, they will only cling all the harder to them. Let the overarching goals of the transformation program shine like the sun.

Team members will choose whether to remove their jackets, remain uncomfortable where they are, or go to a cooler place. It's their choice, not yours.

Those who wish to be part of the new organization will stay. Those who don't will leave. Everyone will get what they need. None of it is bad or good. It just is.

As the coach, you don't own their choices or their outcomes. Remove your own jacket.

39.4 | Nonviolent Communication

Despite the name, this topic is listed here rather than under Communication Models because it's more in the nature of an internal spiritual centering than a communication "technique."

According to the Center for Nonviolent Communication, NVC

is based on the principles of nonviolence - the natural state of compassion when no violence is present in the heart. NVC begins by assuming that we are all compassionate by nature and that violent strategies - whether verbal or physical - are learned behaviors taught and supported by the prevailing culture.

The idea originated with Dr. Marshall Rosenberg, and was first popularized through his book, *Nonviolent Communication: A Language of Life*.

NVC is defined in contrast to so-called "violent" communication, which is characterized by "judging others, bullying, having racial

bias, blaming, finger pointing, discriminating, speaking without listening, criticizing others or ourselves, name-calling, reacting when angry, using political rhetoric, being defensive or judging who's 'good/bad or what's 'right/wrong' with people."

NVC has three goals:

- Increase our ability to live with choice, meaning, and connection
- Connect empathically with self and others to have more satisfying relationships
- Sharing of resources so everyone is able to benefit

To achieve those goals, NVC integrates four concept/practices:

- Consciousness
- Language
- Communication
- Means of influence

The concepts that underlie these points are fully consistent with the communication models I prefer to use in interactions with clients and colleagues.

Consciousness as defined in NVC directly supports all the communication models I use.

Language as defined in NVC relates to Active Listening, Clean Language, Crucial Conversations, Getting To Yes, Powerful Questions, and Radical Candor.

Communication as defined in NVC directly relates to Crucial Conversations, Getting To Yes, and Powerful Questions, and indirectly supports other communication models, as well.

Means of influence is mainly about sharing power with others rather than using power to coerce others. In this way, NVC supports

Appreciative Inquiry, Crucial Conversations, Emotional Intelligence, Getting To Yes, Powerful Questions, and Radical Candor. It is also helpful to team coaches who want to encourage effective self-organization among client personnel, as well as for reducing barriers to trying unfamiliar technical practices.

In addition to supporting communication models that are useful in consulting and coaching work, NVC is highly effective for resolving conflict. The book contains stories of Rosenberg's approach that will astonish readers who are unaccustomed to the idea. In situations where one would expect him to be murdered or beaten up, he used NVC to defuse tension and open meaningful dialog.

39.5 | The Four Agreements

The four agreements are agreements that we make with ourselves. They come from the book, *The Four Agreements: A Practical Guide To Personal Freedom*, by Miguel Ruiz.

The agreements are:

- Be impeccable with your word.
- Don't take anything personally.
- Don't make assumptions.
- Always do your best.

The advice seems pretty simple; even obvious. But it can be challenging to apply.

Be impeccable with your word means to be careful not to overpromise, and if you do promise something, you must deliver on it or be prepared to explain why you couldn't (after having *done your best*). Mean what you say and say what you mean.

In the midst of a transformation program, everyone involved will be under stress. People will say thoughtless things. Some will not

speak honestly to your face. Some will speak dishonestly behind your back.

It's all due to the stress. To maintain focus on what you're doing and avoid useless distractions, it's helpful to avoid *taking things personally*.

Easier said than done, of course.

Don't make assumptions is another piece of “obvious” advice that can be hard to put into practice. “Client-Consultant Relations” discusses the damage that can result when people go forward on the basis of unvalidated assumptions.

Always do your best means to do the best work you can given

- your present level of understanding
- the resources available
- the constraints under which you must operate

It doesn't necessarily mean you must do the best that you could theoretically do under ideal circumstances.

Another angle on doing your best comes from the engineering world. It ties back to *your current level of understanding*. With every experience, every observation, every lesson, every mistake, you increase your understanding. Your “best” as of today will not be the same as your “best” as of five years ago.

I've seen signs on the walls of engineering labs that read, “Don't do anything stupid on purpose.” You'll do plenty of stupid things. That's one of the main ways we learn, after all. The key is not to continue doing the stupid thing after we know better. *That* would amount to doing it “on purpose.”

Part 6 | Summary

This section contains a summary of key points for technical coaches and a few parting thoughts regarding implications of this kind of organizational change.

As the book is about technical coaching to support organizational transformation programs, there's a fair amount of contextual material that isn't directly related to day-to-day team-level coaching. Technical coaches will be part of a larger team addressing, together, larger issues than team-level activity.

Take-aways for technical coaches

Skills - Learn and practice the full range of skills associated with "coaching." The work requires more than technical skills alone.

Keeping Up - Every aspect of software-related work is constantly changing and evolving. A technical coach who offers the same advice today as ten years ago isn't serving customers at the highest professional standard. Keep up with what's going on in the industry.

Context - When you're engaged as part of a larger initiative, you have to support the goals and methods of that larger initiative, and you may not have complete freedom to act as you might do when you are engaged as an independent team-level coach.

Invitation - When engaged as part of an organizational transformation program, your "invitation to coach" does not come from each individual team; it comes from the client executives who are

driving the transformation program. When you start to work with a new team, you are already invited.

Pull - It's more effective to *pull* change from coachees than to *push* recommendations onto them. Remind them of the overarching goals of the transformation program, and encourage them to think about how they can support those goals. That way, they have nothing to "resist" and they're open to your guidance.

Pace - Historically, technical coaches have tried to move technical teams along faster than they were capable of absorbing change and internalizing new information and unfamiliar techniques. Effective technical coaching requires a slow pace of change, so that people have time to gain *fluency* with new practices.

Away Time - Alternate touch time and away time with the teams you're coaching. This gives them a chance to think about, practice, and internalize the things you are showing them. When coaches are always with their teams, the tendency is for the teams to revert to the *status quo ante* as soon as the coach leaves.

Non-Attachment - You don't own your coachee's outcomes. They choose what they wish to do. You can offer advice to guide them toward achieving the goals their leadership has established for the transformation program. You can shine a light on the path, but you can't walk for them or carry them.

Appendix A | IBM Mainframe Considerations

In organizational transformation programs, most people tend to regard the IBM mainframe system as something to phase out and remove from the environment. In some cases, that may be the best option from a business standpoint. However, the system has capabilities many consultants are unaware of.

A.1 | Mainframe UNIX Support

IBM zOS ships with a level of UNIX support. It isn't a full-blown UNIX implementation, but rather a command shell, called UNIX System Services (USS), that supports most of the usual UNIX commands. The native shell language is korn, and tcsh is also included.

Other UNIX tools are available from the IBM zOS Tools and Toys project on Github, and from Rocket Software.

USS is not a separate operating system, but part of the base distribution of MVS on zOS. You have full access to files and programs that live on the "MVS side" of the box. So, you can read and write VSAM files using Java executed from a shell script, for instance.

Far from obsolete, the system runs everything it used to run plus everything that runs on other mainstream platforms, as well; and old and new can access each other.

A.2 | Mainframe ASCII Support

Natively, the zSeries machine uses EBCDIC encoding. Today, zOS handles both EBCDIC and ASCII data. I have run Ruby and Cucumber on a zOS system, and I know of others who have done similar things.

The implication is that just about any of the familiar cross-platform tools will work on the mainframe platform. You may have to find an obscure configuration setting somewhere, or make a minor change or two to the code, but in general you can do anything you need to do on the platform.

A.3 | Mainframe as Cloud Host Environment

An IBM mainframe system can function as the host environment for a cloud infrastructure. zSeries machines can run thousands of Linux VMs concurrently and provide good throughput for the VMs as well as legacy workloads running concurrently.

IBM has been enhancing the ability of the zOS platform to support contemporary, vendor-neutral technologies such as Docker and Kubernetes (Vizard). This is compelling in view of the stability, security, performance, and reliability of the mainframe platform.

As a technical coach or consultant, you will find managers in the mainframe area to be leery of IBM pricing. In many cases, organizations are not taking full advantage of the capabilities of the platform because doing so would incur excessive usage charges.

In 2019, IBM announced Tailor Fit Pricing for IBM Z, which mimics cloud providers' usage-based pricing models (Mauri). This will make it more feasible for organizations to take better advantage

of the platform they already have in-house to support emerging technologies.

A.4 | Legacy Applications and IBM Z Cloud Features

Pure legacy applications written in COBOL, PL/I, Assembly language, and others will not work “out of the box” with IBM Z hybrid cloud features. As of this writing (2019), it is possible to containerize a COBOL application using GnuCOBOL with IBM Cloud Kubernetes (Picarelli).

To migrate a legacy COBOL application to this environment will almost certainly require source code changes. If the application uses non-SQL data stores such as VSAM or QSAM, additional modifications will be needed. Be careful not to over-promise what is possible to your client.

A.5 | Test Automation and Test-Driven Development

Support for test automation and test-driven development is the same on a mainframe as for other platforms, when we are working with contemporary tools such as Java or Ruby. It’s straightforward to run JUnit or Rspec.

For pure legacy applications written in languages like COBOL or PL/I, the available tooling is limited. A certain amount of hand-rolling of test frameworks may be necessary. IBM zUnit and related products provide some support for TDD, but it isn’t at the level of granularity you may be accustomed to with other languages on other platforms.

In many mainframe environments, staff are quite good programmers and generally interested in trying new things and doing enjoyable projects such as developing test frameworks. They are not accustomed to finding tools for free on the Internet, so they are not as reluctant as front-end staff to build tools. In addition, the mainframe environment is a kind of programmer's playground, for those who know the "rules" (that is, assembly language).

So, you may well find strong support and enthusiasm for building testing tools.

Appendix B | Glossary

- B -

bottom-up transformation (n.) - an attempt to spark an organizational transformation by changing one team or a small set of teams and allowing it to serve as a positive example in hopes the same changes will spread virally through the organization.

branching strategy (n.) - a work flow for checking out sources from version control and committing changes back into version control. (See *version control system*.)

- C -

capability (n.) - in organizations, the ability to achieve a business objective or carry out a function effectively, consistently, and sustainably.

collaboration (n.) - cooperative joint work carried out by two or more people together to achieve a common goal.

competitive capability (n.) - a capability that directly affects markets, customers, and/or competitors.

cross-functional team (n.) - a team comprising members who possess various different skills necessary to achieve the team's goals.

cycle of change (n.) - a repeating pattern of actions intended to effect organizational change in a step-by-step, controlled manner. The steps are (1) measure, (2) decide, (3) change, (4) stabilize.

- D -

dedicated team (n.) - a team whose sole or primary function is to support one particular product, service, or asset.

definition of done (n.) - at the level of a Story, this is a description of how to know when a given option has been implemented sufficiently to enable its release to production. At the level of an in-flight work item to be handed off between specialists, this is a description of what must be true to enable the hand-off.

definition of ready (n.) - at the level of a Story, this is a list of prerequisites that must be met before a delivery team has sufficient information and resources to implement the functionality. At the level of an in-flight work item to be handed off between specialists, this is a list of prerequisites that must be met before the receiver can properly work on the item.

delivery capability (n.) - a capability pertaining to an organization's ability to deliver new or modified products and services.

differentiator (n.) - a capability, product, or service that distinguishes the organization from others competing in the same market.

dynamic (adj.) - anything that changes frequently and in real time while continuing to function, such as a market, a technical infrastructure, an application architecture, etc.

- H -

heavyweight (adj.) - characterized by high overhead and significant formality; typically used in reference to a process or method of doing something.

- I -

incremental (adj.) - work that is done in a series of small steps.

- L -

leader (n.) - someone who guides an organization toward the achievement of its defined goals, creates new organizational goals, sets a tone for internal culture and management style, represents the organization to the public, and inspires staff members to bring their best selves to their work.

lightweight (adj.) - characterized by low overhead and minimal formality; typically used in reference to a process or method of doing something.

- M -

maker schedule (n.) - a term coined by Paul Graham to describe a daily work schedule geared for people who “make” the product; it features long blocks of time reserved for uninterrupted, focused work. See *manager schedule*.

manager (n.) - the steward of a process, having responsibility to ensure people carry out the process in accordance with organizational standards and/or external regulatory requirements.

manager schedule (n.) - a term coined by Paul Graham to describe a daily work schedule geared for people in management roles; it features one-hour time slices in which meetings can be scheduled. See *maker schedule*.

mobbing (n.) - a format for whole-team collaborative work on a single task at a time.

monorepo (n.) - a “single repository” for storing sources; all sources in an organization are stored in the same repository. (See *polyrepo*, *version control system*.)

- O -

operational capability (n.) - a capability that affects the internal operations of an organization.

operational model (n.) - a pattern of operations that can apply to multiple different products and services, and that may be observed in situations with similar characteristics.

option (n.) - a possible new feature or modification in a customer-facing product or service, that may be selected for refinement and implementation by a product manager.

options list (n.) - a list of options that may be selected for implementation by a product owner. Generally equivalent to the Master

Story List in Extreme Programming, the Work Queue in Kanban, or the Product Backlog in Scrum. Options may be elaborated to different levels of detail depending on how near in time they are to being selected for implementation.

outcome-oriented transformation (n.) - an organizational transformation program driven by the cultivation of one or more business capabilities or market objectives. Contrast with an “adoption” or “implementation” program, meant to install a process framework without reference to business outcomes.

- P -

pairing (n.) - the practice of two people collaborating directly to complete a single task.

perfection (n.) - (1) the enemy of “good enough;” (2) the perpetual and unreachable target of continuous improvement.

polyrepo (n.) - multiple repositories for storing sources; sources for each product are stored in their own repositories. (See *monorepo*, *version control system*.)

practice (n.) - in organizations, a specific and repeatable way of performing a given type of task or operation.

present (v.) - how you put yourself forward when interacting with others.

process (n.) - in organizations, a defined set of steps or a protocol for carrying out a particular type of work.

process model (n.) - a pattern of processes that have the same general form; e.g., linear process model, iterative process model, time-boxed process model, continuous flow process model.

- S -

stable team (n.) - a team whose membership remains unchanged for a long period of time.

static - (adj.) anything that remains stable for a long period of time, changing infrequently and with external controls.

story - (n.) a description of how an option would affect users or customers of a product or service, should the option be implemented. A story also provides a Definition of Done to help guide development and implementation of the functionality it describes.

structure (n.) - in organizations, a pattern of arranging and orchestrating resources and people, intended to support a given business operation or function.

- T -

team (n.) - a group of people who work cohesively and collaboratively to achieve common goals.

team-building (n.) - activities designed to enhance a team's cohesiveness and effective collaboration.

technical (adj.) - any activity, skill, or person that directly creates, modifies, configures, or operates a hardware or software asset.

tool (n.) - an implement, document, computer file, computer program, process, method, technique, framework, or conceptual model that helps people perform one or more specific actions or functions. Examples: laptop computer, A3 template document, whiteboard, Excel spreadsheet file, customer information database, sticky notes, Scrum, Test-Driven Development, Java Server Faces, Kano Model.

top-down transformation (n.) - an organizational transformation program that is driven by executive leadership, typically “imposed” on staff.

trigger (n.) - a word or phrase that elicits a negative reaction from people. Trigger words are often unique to an organization; the same words are not triggers universally.

trunk-based development (n.) - a branching strategy in which changes are committed to the master or trunk or main branch by all developers. (See *branching strategy*.)

- V -

version control system (n.) - a software product that manages sources, keeping track of changes to the sources and enabling developers to analyze the history of changes and to extract specific versions of selected sources.

viral change (n.) - a change that spreads through an organization in a manner similar to the way a virus spreads through an organism.

Appendix C | References

Note: All online references were available as of April 20, 2019.

- A-

Adams, Scott. *The Dilbert Principle: A Cubicle's-Eye View of Bosses, Meetings, Management Fads & Other Workplace Afflictions*. United Media Pub, 2014.

Adkins, Lyssa; Michael Spayd. *Agile Coaching Competency Model*. URL: <http://agilecoachinginstitute.com/agile-coaching-resources/>

Adzic, Gojko. *Impact Mapping: Making a Big Impact With Software Products and Projects*, Provoking Thoughts, 2012.

Adzic, Gojko. *Specification by Example: How Successful Teams Deliver the Right Software*, Manning Publications, 2011.

Ananthanarayanan, Sundaram et al (2019). “Keeping Master Green At Scale,” EuroSys ‘19, March 25–28, 2019, Dresden, Germany.

URL: http://delivery.acm.org/10.1145/3310000/3303970/a29-ananthanarayanan.pdf?ip=69.244.54.207&id=3303970&acc=TRUSTED&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2EE47D41B086F0CDA3&acm=1558555682_5dfed9090cc0e58908bf4f6f22fa8b3

Asta, Anthony. “Observability at Twitter: Technical Overview, part 1”, on Twitter Engineering, March 18, 2016.

Avery, Christopher. *The Responsibility Process*. Online as of April 20, 2019. URL: <https://www.christopheravery.com/responsibility-process>

-B-

Bache, Emily. “Effective Technical Agile,” video of a session at SCLConf 2018.

URL: <https://www.youtube.com/watch?v=CuO6dbdQup4>

Bache, Emily. *Technical Agile Coaching*, LeanPub, 2019. Ebook.
URL: <https://leanpub.com/techagilecoach>

Barr, Adam. *The Problem With Software: Why Smart Engineers Write Bad Code*. MIT Press, 2018.

Binder, T. “Wie gut verstehen Berater ihre Kunden? Ich-Entwicklung – ein vergessener Faktor in der Beratung,” in S. Busse, S. Ehmer (Hrsg.): *Wissen wir, was wir tun? Beraterisches Handeln in Supervision und Coaching*, Vandenhoeck & Ruprecht, Göttingen 2010.

Bogsnes, Bjarte. *Implementing Beyond Budgeting: Unlocking the Performance Potential*. 2nd ed. Wiley, 2016.

Bolton, Chris. “Campbell’s Law and Why Outcome Measurement Is a Dead Cobra,” What’s The Point, May 19, 2019.

URL: <https://whatsthepoint.com/2019/05/19/campbells-law-and-why-outcome-measurement-is-a-dead-cobra/>

Bolton, Robert. *People Skills: How to Assert Yourself, Listen to Others, and Resolve Conflicts*. Touchstone, 1986.

Bradberry, Travis; Jean Greaves. *Emotional Intelligence 2.0*. TalentSmart, 2009.

Brownson, Tim. “4 Cognitive Biases Every Coach Should Understand,” on the Coach the Life Coach site, October 3, 2017.

URL: <https://www.coachthelifecoach.com/cognitive-biases-life-coach/>

Buckingham, Marcus; Ashley Goodall. *Nine Lies About Work*, Harvard Business Review Press, 2019.

Burgo, Joseph. “The Difference Between Guilt and Shame,” in *Psychology Today*.

URL: <https://www.psychologytoday.com/us/blog/shame/201305/the-difference-between-guilt-and-shame>

Campbell, G. Ann. "Cognitive Complexity: A new way of measuring understandability."

URL: <https://www.sonarsource.com/docs/CognitiveComplexity.pdf> (PDF)

Center for Appreciative Inquiry. "Appreciative Inquiry" Undated.

URL: <https://www.centerforappreciativeinquiry.net/more-on-ai/what-is-appreciative-inquiry-ai/>

Center for Nonviolent Communication. "What is NVC?" Undated. URL: <https://www.cnvc.org/node/6856>

Cherry, Kendra. "The Hawthorne Effect and Behavioral Studies," on *Very Well Mind*, November 11, 2018.

URL: <https://www.verywellmind.com/what-is-the-hawthorne-effect-2795234>

Cockburn, Alistair. "Cutting-Edge Agile - Opening Statement," on Cutter Consortium, April 22, 2019.

URL: <https://www.cutter.com/article/cutting-edge-agile-opening-statement-503026>

Cockburn, Alistair. *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional, 2004.

Cooper, Alan. *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How To Restore the Sanity*. Sams-Pearson Education, 2004.

-D-

DeMarco, Tom; Tim Lister. *Peopleware: Productive Projects and Teams*, 3rd ed. Addison-Wesley, 2013.

Dettmer, H. William. *The Logical Thinking Process: A Systems Approach to Complex Problem Solving*. 2nd ed. American Society for Quality, 2007.

Diana, Frank. "A Closer Look at Transformation: Sense and Respond Systems," in *Reimagining the Future: Through the Looking*

Glass, September 16, 2013.

URL: <https://frankdiana.net/2013/09/16/a-closer-look-at-transformation-sense-and-respond-systems/>

Dichter, Steven F.; Chris Gagnon; Ashok Alexander. “Leading organizational transformations,” in *McKinsey Quarterly*, February, 2993. URL: <https://www.mckinsey.com/business-functions/organization/our-insights/leading-organizational-transformations>

Dinwiddie, George. “The Three Amigos Strategy of Developing User Stories,” AgileConnection, April 21, 2014.

URL: <https://www.agileconnection.com/article/three-amigos-strategy-developing-user-stories>

Driessen, Vincent. “A successful Git branching model,” on nvie.com, January 5, 2010. URL: <https://nvie.com/posts/a-successful-git-branching-model/>

Dunn, Sean. “Eating Your Own Dogfood: From Enterprise Agile Coach to Team Developer,” on the AgileAlliance site, undated.
URL: <https://www.agilealliance.org/resources/experience-reports/eating-your-own-dogfood-from-enterprise-agile-coach-to-team-developer/>

-E-

Everybody Loves Raymond. Active Listening segments on YouTube.
URL: https://www.youtube.com/results?search_query=everybody+loves+raymond+active+listening

-F-

Falco, Llewellyn. “A Day In the Life of an Agile Coach,” August 20, 2017. Video.

URL: <https://www.youtube.com/watch?v=7iPybzyYZbU>

Falco, Llewellyn. “Llewellyn’s Strong-Style Pairing,” June 30, 2014.
URL: <https://llewellynfalco.blogspot.com/2014/06/llewellyns-strong-style-pairing.html>

Fidei, Kelly. “What Improv Offers to Agile Coaching,” on Agility Transformation (undated).

URL: <https://agilitytransformation.com/what-improv-offers-to-agile-coaching/>

FindLaw. “Libel, Slander, and Defamation Law: The Basics”.

URL: <https://injury.findlaw.com/torts-and-personal-injuries/defamation-law-the-basics.html>.

Fisher, Roger; William L. Ury; Bruce Patton. *Getting To Yes: Negotiating Agreement Without Giving In*. Penguin Books, 2011.

Forsgren, Nicole; Jez Humble; Gene Kim. *Accelerate: Building and Scaling High-Performing Technology Organizations*. IT Revolution, 2018.

Fowler, Martin (2009). “Feature Branch,” on MartinFowler.com, September 3, 2009.

URL: <https://www.martinfowler.com/bliki/FeatureBranch.html>

-G-

GitHub. “Understanding the GitHub Flow,” GitHub Guides (undated).

URL: <https://guides.github.com/introduction/flow/>

Goode, Durham. “Scaling Mercurial At Facebook,” F Code, January 17, 2014.

URL: <https://code.fb.com/core-data/scaling-mercurial-at-facebook/>

Google. *The SRE Book*, 2017. URL: <https://landing.google.com/sre/sre-book/toc/>

Graham, Paul. “Maker’s Schedule, Manager’s Schedule,” on paulgraham.com, July, 2009.

URL: <http://www.paulgraham.com/makersschedule.html>

Gupta, Aman. “Deploying branches to GitHub.com,” GitHub Blog, June 2, 2015.

URL: <https://github.blog/2015-06-02-deploying-branches-to-github-com/>

-H-

Hammant, Paul (2014a). “Google’s vs. Facebook’s Trunk-Based Development,” Trunk-Based Development, 2014.

URL: <https://paulhammant.com/2014/01/08/googles-vs-facebooks-trunk-based-development/>

Hammant, Paul (2014b). “Googlers Subset Their Trunk,” Trunk-Based Development, January 6, 2014.

URL: <https://paulhammant.com/2014/01/06/googlers-subset-their-trunk/>

Hammant, Paul (2017). “Trunk Based Development,” Trunk-Based Development, 2017-2018.

URL: <https://trunkbaseddevelopment.com/>

Harry Potter Wiki. “Dementor”.

URL: <https://harrypotter.fandom.com/wiki/Dementor>

Heath, Chip; Dan Heath. *Switch: How To Change Things When Change Is Hard*. Crown Business, 2010.

Henderson, Joel Parker. “Monorepo vs. Polyrepo,” on Github.

URL: https://github.com/joelparkerhenderson/monorepo_vs_polyrepo.

Hesselbein, Frances. “The Key to Cultural Transformation,” in *Leader to Leader* 1999:12, pp 1-7.

Highsmith, James. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Addison-Wesley Professional, 2013.

Hope, Jeremy; Robin Fraser. *Beyond Budgeting: How Managers Can Break Free From the Annual Performance Trap*. Harvard Business Review Press, 2003.

Human Metrics, Inc. “Conflict Management Techniques” Undated.

URL: <http://www.personalityexplorer.com/FREEResources/ConflictManagementTechniques.aspx>

-I-

Ilama, Eeva. “Creating Personas,” on UX Booth, June 9, 2015.
URL: <https://www.uxbooth.com/articles/creating-personas/>

-J-

James, Geoffrey. “9 Reasons That Open-Space Offices Are Insanely Stupid,” in *Inc.*, February 25, 2016. URL: <https://www.inc.com/geoffrey-james/why-your-company-will-benefit-from-getting-rid-of-open-office-spaces-first-90.html>

James, Michael. “An Agile Approach to Metrics,” CollabNet whitepaper. URL: http://www.danube.com/system/files/CollabNet_WP_Macromeasurements_061710.pdf

Jeffries, Ron. “Connections, ‘Transformation’, Agile Software Development,” on ronjeffries.com, November 27, 2018.
URL: <https://www.ronjeffries.com/articles/018-01ff/connections-transformation/>

Jeffrey, Scott. “How to Adopt a Beginner’s Mind to Accelerate Learning and Increase Creativity,” on the CEOsage site. Undated.
URL: <https://scottjeffrey.com/beginners-mind/>

-K-

Kelly Warner Law. “Defamation Laws in Finland”.
URL: <http://kellywarnerlaw.com/finland-defamation-laws/>

Kerievsky, Joshua. “Anzeneering” on the Industrial Logic website, posted January 21, 2014.
URL: <https://www.industriallogic.com/blog/anzeneering/>

Kerth, Norman. *Project Retrospectives: A Handbook for Team Reviews*. Addison-Wesley, 2013.

Kolb, David A. *Experiential Learning: Experience as the Source of Learning and Development*, Prentice-Hall, 1984.

Kolbert, Elizabeth. “Why Facts Don’t Change Our Minds,” in *The New Yorker*, February 19, 2017.

URL: <https://www.newyorker.com/magazine/2017/02/27/why-facts-dont-change-our-minds>

Kotter, John. *Leading Change*. 1R ed. Harvard Business Review Press, 2012.

-L-

Larman, Craig. “An Introduction to LeSS: Large-Scale Scrum,” at the Phoenix Scrum User Group meeting, December 8, 2015 (Unpublished).

Larsen, Diana; James Shore. “Agile Fluency Model.”

URL: <https://www.agilefluency.org/model.php>

LegalMatch. “Future Loss of Earnings or Wages”.

URL: <https://www.legalmatch.com/law-library/article/future-loss-of-earnings-or-wages.html>

Lencioni, Patrick. *The Five Dysfunctions of a Team: A Leadership Fable*. Jossey-Bass, 2011.

LeSS. “LeSS Overview”. Undated. URL: <https://less.works/>

Lionetti, Giancarlo. “What is version control: Centralized vs. DVCS,” Atlassian Blog, February 14, 2012.

URL: <https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs>

Loevinger, Jane. *Ego Development: Conceptions and Theories*, Jossey-Bass, 1976.

Logically Fallacious. “Gambler’s Fallacy,” on Logically Fallacious.

URL: <https://www.logicallyfallacious.com/tools/lp/Bo/LogicalFallacies/98/Gambler-s-Fallacy>

-M-

Majors, Charity (2018). “Oncall and Sustainable Software Development,” Honeycomb.io Blog, February 15, 2018.

URL: <https://www.honeycomb.io/blog/oncall-and-sustainable-software-development/>

Majors, Charity (2017). “Ops: It’s Everyone’s Job Now,” Open-Source.com, July 28, 2017.

URL: <https://opensource.com/article/17/7/state-systems-administration>

Mauri, Ross. “IBM Z defines the future of hybrid cloud,” IBM IT Infrastructure Blog, May 14, 2019.

URL: <https://www.ibm.com/blogs/systems/ibm-z-defines-the-future-of-hybrid-cloud/>

Miller, Jon. “Exploring the ‘Respect for People’ Principle of the Toyota Way,” on the Gemba Academy blog, February 3, 2008.

URL: https://blog.gembaacademy.com/2008/02/03/exploring_the_respect_for_people_principle_of_the/

Monty Python’s Flying Circus. “How To Do It,” transcript

URL: <http://www.montypython.net/scripts/howtodoit.php>

Morris, Kief. *Infrastructure As Code*, O'Reilly Media, 2016.

Murphy, Bill. “11 Psychological Tricks to Manipulate People, Ranked In Order of Pure Evilness,” in Inc. for December 6, 2015. URL: <https://www.inc.com/bill-murphy-jr/evil-psychological-tricks-to-manipulate-people.html>

-N-

Nickolaisen, Niel. “Breaking the Project Management Triangle,” in *informIT*, August 20, 2009.

URL: <http://www.informit.com/articles/article.aspx?p=1384195&seqNum=2>

Nicolette, David. “5S for Software Teams,” LeadingAgile blog, September 8, 2016.

URL: <https://www.leadingagile.com/2016/09/5s-for-software-teams/>

Nicolette, David. “Feature Teams and Vertical Slices in a Large Corporate IT Environment,” on Wordpress, May 13, 2015. URL <https://davenicolette.wordpress.com/2015/05/13/feature-teams-and-vertical-slices-in-a-large-corporate-it-environment/>

Nicolette, David. “How Does Collaboration Begin?” on Wordpress, August 2, 2013.

URL: <https://davenicolette.wordpress.com/2013/08/02/how-does-collaboration-begin/>

Nicolette, David. “Introversion and Agile,” on the NeopPragma blog, March 24, 2019.

URL: <http://neopragma.com/index.php/2019/03/24/introversion-and-agile/>

Nicolette, David. *Software Development Metrics*. Manning Publications, 2015.

Nicolette, David. “Using Lego to capture raw data for Cycle Time and Process Cycle Efficiency,” on Wordpress, June 20, 2014.

URL: <https://davenicolette.wordpress.com/2014/06/20/using-lego-to-capture-raw-data-for-cycle-time-and-process-cycle-efficiency/>

-P-

Patterson, Kerry; Joseph Grenny; Ron McMillan; Al Switzler. *Crucial Conversations: Tools for Talking When Stakes Are High*. 2nd ed. McGraw-Hill Education, 2011.

Patton, Jeff; Peter Economy. *User Story Mapping: Discover the Whole Story, Build the Right Product*, O'Reilly Media, 2014.

Personality Max. “Extraversion vs. Introversion Preference”. Undated. URL: <https://personalitymax.com/personality-types/preferences/extraversion-introversion/>

Picarelli, Ennio. “Mainframe application dockerization - Part 2 - How to deploy a GNU cobol container with SQL capabilities by using the IBM Cloud Kubernetes environment,” IBM developerWorks Recipes, January 3, 2018.

URL: <https://developer.ibm.com/recipes/tutorials/mainframe-application-dockerization-part-2-how-to-deploy-a-gnu-cobol-container-with-sql-capabilities-by-using-the-ibm-cloud-kubernetes-environment/>

-R-

- Radigan**, Dan. “Feature Branching Your Way To Greatness,” Atlassian (undated).
URL: <https://www.atlassian.com/agile/software-development/branching>
- Roberts**, Mike. “Serverless Architectures,” on MartinFowler.com, May 22, 2018.
URL: <https://www.martinfowler.com/articles/serverless.html>
- Rosenberg**, Marshall. *Nonviolent Communication: A Language of Life*. 3rd ed. PuddleDancer Press, 2015.
- Ruiz**, Miguel. *The Four Agreements: A Practical Guide To Personal Freedom*. Amber-Allen Publishing, 2011.
- Ruka**, Adam (2015). “GitFlow Considered Harmful,” on End of Line Blog, May 3, 2015.
URL: <https://www.endoflineblog.com/gitflow-considered-harmful>
- Ruka**, Adam (2017). “OneFlow - a Git Branching Model and Workflow,” on End of Line Blog, April 30, 2017. URL: <https://www.endoflineblog.com/one-a-git-branching-model-and-workflow>
- Russell**, E. *The basic Clean Language Questions*. Undated. URL: http://www.e-russell.com/images/The_basic_clean_language_questions1.pdf (PDF)
- S-
- SAFe**. “SAFe Implementation Roadmap”. Undated.
URL: <https://www.scaledagileframework.com/implementation-roadmap/>
- Sanga**, Nithyananda. “What is a beginner’s mind?” Undated. URL: <https://www.nithyananda.org/article/what-beginner-mind#gsc.tab=0>
- Saval**, Nikil. “The Cubicle You Call Hell Was Designed To Set You Free,” in *Wired*, April 23, 2014. URL: <https://www.wired.com/2014/04/how-offices-accidentally-became-hellish-cubicle-farms/>
- Scott**, Kim. *Radical Candor: Be a Kick-Ass Boss Without Losing Your Humanity*. St. Martin’s Press, 2017.

Skills You Need. “Active Listening”. Undated.

URL: <https://www.skillsyouneed.com/ips/active-listening.html>

Smart, John Ferguson. “Feature Mapping - A simpler path from user stories to executable acceptance criteria,” on johnfergusonsmart.com, January 25, 2017.

URL: <https://johnfergusonsmart.com/feature-mapping-a-simpler-path-from-stories-to-executable-acceptance-criteria/>

Spiral Dynamics. Website at URL: <https://spiraldynamics.org/>

Sridharan, Cindy. “Monitoring and Observability,” Medium, September 4, 2017.

URL: <https://medium.com/@copyconstruct/monitoring-and-observability-8417d1952e1c>

Stacey, Ralph. *Complexity and Creativity in Organizations*, Berrett-Koehler Publishers, 1996.

Stahl, Ashley. “New Study Reveals That Cubicle Farms Are Ruining Employee Morale (and Output),” in *Forbes*, July 28, 2016.

URL: <https://www.forbes.com/sites/ashleystahl/2016/07/28/new-study-reveals-that-cubicle-farms-are-ruining-employee-morale-and-output/#367463d3ee2c>

Star Trek. “The Immunity Syndrome,” broadcast January 19, 1968. Synopsis URL: https://www.startrek.com/database_article/immunity-syndrome

Stevenson, Seth. “Polka Dots Are In? Polka Dots It Is!,” in *Slate*, June 21, 2012. URL: <https://slate.com/culture/2012/06/zaras-fast-fashion-how-the-company-gets-new-styles-to-stores-so-quickly.html>

Suzuki, Shunryu. *Zen Mind, Beginner’s Mind*. Shambhala, 2011.

Szabo, Peter W. *User Experience Mapping: Enhance UX with User Story Map, Journey Map, and Diagrams*, Packt Publishing, 2017.

-T-

Tabaka, Jean. *Collaboration Explained: Facilitation Skills for Software Project Leaders*, Addison-Wesley Professional, 2006.

Thich, Thien An. “Non Attachment,” on the Purify Mind site. Undated. URL: <http://purifymind.com/NonAttach.htm>

Thomson, Edward. “Release Flow: How We Do Branching On the VSTS Team,” Microsoft DevBlogs, April 19, 2018.

URL: <https://devblogs.microsoft.com/devops/release-flow-how-we-do-branching-on-the-vsts-team/>

Tuckman, Bruce. “Developmental sequence in small groups,” *Psychological Bulletin* 63:6(384-99), 1965. doi:10.1037/h0022100. PMID 14314073.

-V-

Vedantam, Shankar. “What Can a Personality Test Tell Us About Who We Are?” on *Hidden Brain*,

National Public Radio, April 19, 2019. Recording and transcript.

URL: <https://www.npr.org/templates/transcript/transcript.php?storyId=712876949>

Vizard, Mike. “IBM Advances Mainframe Container Strategy,” Container Journal, May 16, 2019.

URL: <https://containerjournal.com/2019/05/16/ibm-advances-mainframe-container-strategy/>

-W-

Waterworth, Steve. “Observability vs. Monitoring,” in DZone, March 19, 2019.

URL: <https://dzone.com/articles/observability-vs-monitoring>

Watkins, Michael D. “What Is Organizational Culture? And Why Should We Care?” in Harvard Business Review, May 15, 2013. URL: <https://hbr.org/2013/05/what-is-organizational-culture>

Watson, Cory G. “Observability at Twitter,” Twitter Engineering, September 9, 2013.

URL: https://blog.twitter.com/engineering/en_us/a/2013/observability-at-twitter.html

Waytz, Adam. “The Illusion of Explanatory Depth,” on *Edge*. URL: <https://www.edge.org/response-detail/27117>

Weeks, Dudley. *The Eight Essential Steps to Conflict Resolution: Preserving Relationships at Work, at Home, and In the Community*. Tarcher Perigee, 1994.

-Y-

Yronwode, Catherine. *Throwing the Bones: How To Foretell the Future With Bones, Shells, and Nuts*. Lucky Mojo Curio Company, 2012.

-Z-

Zuill, Woody. “Mob Programming Basics,” November 14, 2012.
URL: <https://mobprogramming.org/mob-programming-basics/>

Appendix D | Picture Credits

Cover Art

“Metamorphosis of a Butterfly,” Maria Sibylla Merian, 1705. Public domain image from <https://www.wikiart.org/en/maria-sibylla-merian/metamorphosis-of-a-butterfly-1705>.

Figures

Figure 1.1: Connect the dots. Original, made with LibreOffice.

Figure 2.1: A miracle occurs. Found on the Internet, unattributed.

Figures in Chapter 5: Original, assembled from public-domain images found at <https://publicdomainvectors.org>.

Figure 6.1: Purpose Alignment Model. Original, made with LibreOffice. Based on work of Niel Nickolaisen.

Figure 6.2: Sample analysis using PAM. Original, using public domain sticky note image from <https://publicdomainvectors.org> and LibreOffice.

Figures 7.1, 7.2, Original, made with FlyingLogic Pro (licensed).

Figure 7.3: Leverage. Original, made with LibreOffice.

Figure 9.1: The Cycle of Change. Original, made with public domain infinity symbol from <https://publicdomainvectors.org> and LibreOffice.

Figure 10.1: The Cycle of Change. Original, made with public domain infinity symbol from <https://publicdomainvectors.org> and LibreOffice.

Figure 11.1: Aggregate code health, point in time. Original, made with Microsoft Excel.

Figure 11.2: Application 1 code health. Original, made with Microsoft Excel.

Figure 11.3: Cognitive Complexity trend, Application 1. Original, made with Microsoft Excel.

Figure 27.1: Influence and Trust Loop. Original, made with with public domain infinity symbol from <https://publicdomainvectors.org> and LibreOffice. Based on work of Dennis Stevens, LeadingAgile.

Figure 27.2: All in agreement. Original, made with “free” clipart from <https://clipartix.com> (terms of use not specified) and LibreOffice.

Figure 16.1: Raw data as Lego bricks, day 1. Original photograph.

Figure 16.2: Collected CT and PCE data, day 1. Original, screenshot from spreadsheet.

Figure 16.3: Raw data as Lego bricks, day 2. Original photograph.

Figure 16.4: Collected CT and PCE data, day 2. Original, screenshot from spreadsheet.

Figure 27.1: Office environment in the 1930s. Found at <http://officemuseum.com>. Terms of use not specified.

Figure 27.2: The Action Office (1964).

Found at <https://www.wired.com/2014/04/how-offices-accidentally-became-hellish-cubicle-farms/>.

Terms of use not specified.

Figure 27.3: A cubicle farm. Found at <http://indianpublicmedia.org>. Terms of use not specified.

Figure 27.4: An open-plan office space. Found at <http://stxnext.com>. Terms of use not specified.

Figure 29.1: Teams organized as functional silos. Original, made with LibreOffice.

Figure 29.2: Silo-busting constraints. Original, made with public domain image of Matryoshka dolls from <https://www.publicdomainpictures.net> and LibreOffice.

Figure 29.3: Unfinished column drums, Acropolis.

From <http://www.stoa.org/athens/sites/northslope/source/p08080.html>. Terms of use not specified.

Figure 29.4: Role affinities in development (1). Original, made with FlyingLogic Pro (licensed).

Figure 29.5: Role affinities in development (2). Original, made with FlyingLogic Pro (licensed).

Figure 29.6: Role affinities in development (3). Original, made with FlyingLogic Pro (licensed).

Figure 29.7: Role affinities in development (4). Original, made with FlyingLogic Pro (licensed).

Figure 29.8: Role affinities in engineering. Original, made with FlyingLogic Pro (licensed).

Figure 29.9: Team size and lines of communication. Found on StackOverflow at <https://stackoverflow.com/questions/984885/how-do-i-explain-the-overhead-of-communication-between-developers-in-a-team>. Original author not identified. Terms of use not specified.

Figure 33.1: Single team's view of coaches. Original, made with LibreOffice.

Figure 33.2: 12 teams, 2 coaches. Original, made with LibreOffice.

Figure 34.1: Radical Candor quadrants. From <https://www.topsimages.com> (terms of use not specified)

39.{:: encoding="utf-8" /}

Index

The Index provides chapter and section numbers rather than page numbers because LeanPub doesn't have an index-generation facility. Thus, the reference 5.2.8 means Chapter 5, Section 2, Subsection 8.

-Numbers-

5S 27.7

-A-

access 24.2

accountability 16.1

acting 34.6

Active Listening 37.1

Action Office 27.2

agency 24.2

Agile 0.5, 2.2, 2.6, 2.8

Agile scaling frameworks 2.7, 2.9, 2.10, 2.12, 4.4.2

Agile Transformation as red flag 2.2

alerting 33.3

analytics

 in operations 33.3

anti-patterns 2

 big-bang change vs incremental change 2.9

 conflating business agility with agile software development 2.6

 in coaching branching strategies 32.8

 inappropriate organizational structures 2.8

 incomplete solutions 2.1

 lack of direction 2.3

 milestone-driven approach vs directional approach 2.11, 10.1.2, 11.2, 24.1.3

misalignment of goals 2.2
misunderstanding the nature of the work 2.5
no meaningful monitoring 10.1.1
reintroduction of the old 2.7
rigid interpretation of frameworks 2.10
separation of process and technical improvements 2.4

Appreciative Inquiry 37.2**assessments** 10.1.2, 24.1.4**authority**

view of 21

-B-**back-end systems** 8.2**Beginner's Mind** 39.1**big-bang change** 2.9**branching strategies** 32.4**business agility** 2.6**-C-****capability** 3.1

competitive capability 3.1.3, 8.3.1, 11.3

delivery capability 3.1.2

operational capability 3.1.1

change 25**Clean Language** 37.3**client-consultant relations** 24**cloud computing** 33.3**coaches**

Agile 35.2

shortage of 32

skills for 31

coaching

approach 33

context awareness 34.13

defined 34.1

- knowing when to quit 34.14
 - skills for 31
 - code health** 11.8
 - cognitive biases**
 - confirmation bias 19.2
 - gambler's fallacy 19.3
 - illusion of explanatory depth 19.6
 - negativity bias 19.5
 - sunk cost fallacy 19.4
 - cognitive complexity** 11.8.1
 - collaboration**
 - and flow 38.1, 38.1.3
 - and hand-offs 38.1.2
 - in a linear process 38.1.1
 - incremental improvement 29.5.1
 - remote 27.6
 - collaborative workspaces** 26.1, 27
 - communication models** 34
 - competence** 24.2
 - competitive capability** 3.1.3, 8.3.1, 11.3
 - conference call tips** 27.6.4
 - confirmation bias** 19.2
 - conflict resolution** 24.1.6
 - consulting team** 4.4.4.2, 10.5, 12.1.2
 - continuous improvement** 2.2, 10.5
 - continuous integration** 32
 - cross-functional teams** 2.8
 - cross-team dependencies** 2.8
 - Crucial Conversations** 37.4
 - cubicle farm** 27.3
 - culture** 23
 - cycle of change** 3.3, 4.4.2, 4.4.4, 10.4.2
- D-
- dependencies** 7, 10.4.1

defamation 22.2
delivery capability 3.1.2, 11.3
Digital Transformation as red flag 2.2
directional approach 2.11
directional metrics 10.2, 11.8.2
dynamic market interaction 2.6, 6

-E-

ego development 20
Emotional Intelligence 37.5
empathy 24.2, 34.2
estimation 30
 and familiarity with the work 30.2.1
 and interruptions 30.2.2
 and variability in work items 30.2.3
 and crusty code 30.2.4
 by absolute time 30.3.1, 30.3.6
 by counting work items 30.3.5
 by ideal time 30.3.2
 by relative sizing 30.3.3, 30.3.4
 factors affecting 30.2

Evo 2.12

excuses 22.3

experiments 26.4

extraversion 18

Eye of the Hurricane 39.2

-F-

facilitating 34.1

Feature Branches 32.4.4

flow

 and collaboration 38.1
 and formality 38.1.6
 and hand-offs 38.1.5
 and improving technical skills 38.1.8

forecasting *see* estimation

Forrest Gump 2.12

Four Agreements 39.5

fulcrum 3.2.1, 4.1, 8.3.3, 11.6

functional silos 2.8

 collapsing incrementally 29

 organizational constraints 29.3

 service teams 29.4

 starting point 29.2

-G-

gambler's fallacy 19.3

Getting To Yes 37.6

GitFlow 32.4.3

GitHub Flow 32.4.1.1

****governance** 2.7

guiding coalition 2.1, 4.4.4.1, 10.5, 12.1.1

-H-

hold accountable 16.1

-I-

illusion of explanatory depth 19.6

incremental change 2.9

 collaboration 29.5.1

 silo-busting 29.5

 temporary scaffolding for 29.4

influence 24.2

infrastructure as code 29.4.2.2, 33.2

integrity 24.2

internalities 36

 Beginner's Mind 39.1

 Eye of the Hurricane 39.2

 Four Agreements 39.5

 Non-Attachment 39.3

 Nonviolent Communication 39.4

introversion 18

invitation to coach 34.13

-K-

Kaizen 4.4.3

kanban 4.4.3

Kanban Method 4.4.3, 25.4

-L-

Lean 2.8, 2.12, 4.4.3

leverage 3.2

linear SDLC 2.12

log aggregation 33.3

-M-

maker time 28.3, 28.5

management consultant defined, 0.2

manager time 28.3

manipulation techniques

 build relationships 34.5.2

 connect organizational and personal goals 34.5.1

 don't allow time for arguments 34.5.6

 fatigue as a motivator 34.5.7

 fear as a motivator 34.5.8

 mirroring 34.5.3

 nouns over verbs 34.5.5

 playing dumb 34.5.4

market differentiators 4.4.4.3, 6, 8.3

market interaction

 dynamic 2.6, 6

 stable 6

Matryoshka Doll organizational structure 29.3

meetings 28.4

mentoring 34.1

metrics

 baseline 26.3

inappropriate 24.1.5
cycle efficiency 11.1.2
cycle time 11.1.2
directional 10.2, 11.2
for steering delivery 11
for tracking improvement 11
identification of 8.3.4, 10.4.1
indirect 10.4.1
lack of 24.1.5
lean 11.1.2
process-agnostic 11.1
qualitative 10.5
quantitative 10.5
throughput 11.1.2
velocity, problem with 11.1.1

microservices 33.3
milestones 2.11, 24.1.3
Mindful Kindness 37.7
mission-critical systems 8.3.1
monitoring 33.3
monoculture 22.4
monorepo 32

-N-

negativity bias 19.5
Non-Attachment 39.3
Nonviolent Communication 36.4

-O-

observability 33.3
OneFlow 32.4.3
organizational slices 8.2
open-plan office 27.4
operational capability 3.1.1, 11.3
operations, stability of 7, 11.6

Outlook-driven development 28.1

-P-

PDSA - *see* plan-do-study-act.

phoenix server strategy 33.2.6

pigeon-holing 22.1

plan-do-study-act 3.3, 4.4.3

polyrepo 32

portfolio management 2.7

Powerful Questions 37.8

practices

improvement in 8.1

proactive operations 33.3

process

improvement in 8.1

products

alignment of teams with 4.4.4.4

market-differentiating 4.4.4.3, 6, 8.3

mission-critical 4.4.4.3

profiling 22

-R-

Radical Candor 37.9

Release Branches 32.4.2

Release Flow 32.4.1.2

remote collaboration 27.6

resources 13

respect for humanity 14

responsibility

challenges when decentralizing 29.5

of client 12.3

of client and consultancy jointly 12.4

of coach 12.5

of consultancy 12.2

of consultant 12.5

- process (Avery) 16.2
- results** 24.2
- RUP 2.12
- S-
- SAFe** - *see* Scaled Agile Framework
- safety** 11.8.2, 15, 24.2
- Scaled Agile Framework** 2.9, 3.4, 4.4.2
- scaling framework** 2.1, 2.9
- Scrum** 2.12
- security** 29.7
- self-awareness** 34.2
- self-service infrastructure** 33.2.7
- serverless** 29.4.2.3, 33.3
- service teams** 29.4
- Site Reliability Engineering (SRE)** 33.3
- software quality** 11.7
- SonarQube** 11.8.1
- Spiral process** 2.12
- Spiral Dynamics** model 21
- stable market interaction** 6
- static code analysis** 11.8.1
- stepwise improvement** 25.4
- stress** 17
 - and client-consultant relations 24.1.1
 - related to change 17.1
- structure**
 - improvement in 8.1
- sunk cost fallacy** 19.4
- T-
- team**
 - alignment with products 4.4.4.4, 8.3.2, 26.2
 - and coaching stances 38.2.5
 - cohesiveness 38.2

functional and dysfunctional 38.2.1
size 29.6
Tuckman model 38.2.3
teaching 34.1
technical coach defined. 0.2
technical coaches, shortage 4.1
technical consultant defined. 0.2
Theory of Constraints 4.4.3
time management 28
Toyota Production System 4.4.3, 14
toxic culture 22.5
TPS - see Toyota Production System
tracing
 in distributed systems 33.3
trigger words 34.12
trunk-based development 32.4.1
trust
 earning and maintaining 24.2

-V-

-V-Model 2.12
value streams 8.3.2
velocity, problem with 11.1.1
version control 32
viral change 2.1
visualization
 in operations 33.3

-W-

waterfall 2.11