The Open Organization Guide to IT Culture Change

Open principles and practices for a more innovative IT department

Copyright

Mike Walker's "Introduction to Part 1" originally appeared at https://opensource.com/open-organization/17/5/team-differentiator-not-tech.

Jono Bacon's "What's an IT culture, anyway?" originally appeared at https://opensource.com/article/17/5/what-is-IT-culture.

Gene Kim's "Organizational learning: a new perspective on DevOps" originally appeared at https://opensource.com/business/15/2/organizational-learning-new-perspective-devops.

Jim Whitehurst's "Innovation requires new approaches to feedback and failure" originally appeared at https://opensource.com/open-organization/16/12/building-culture-innovation-your-organization.

Jordan Morgan's "Why a Buffer developer open sourced his code" originally appeared at https://opensource.com/open-organization/16/5/buffer-open-culture.

Gordon Haff's "A user's guide to failing faster" originally appeared at https://opensource.com/open-organization/17/4/accountability-by-design.

Matt Micene's "Changing the way we think of change" originally appeared at https://opensource.com/article/17/4/changing-way-we-think-change.

Chris Short's "Five laws every aspiring DevOps engineer should know" originally appeared at https://opensource.com/open-organization/17/5/5-devops-laws.

Ron McFarland's "How new communication technologies are affecting peer-to-peer engagement" originally appeared at

1 http://creativecommons.org/licenses/by-sa/4.0/

https://opensource.com/open-organization/16/4/how-new-communication-technologies-are-affecting-peer-peer-engagement.

Jackie Yeaney's "What engineers and marketers can learn from each other" originally appeared at https://opensource.com/open-organization/17/1/engineers-marketers-can-learn.

Matt Thompson's "How to strengthen your agile heartbeat with powerful retrospectives" originally appeared at https://opensource.com/open-organization/16/11/checking-your-agile-workflow.

Chad Whitacre's "The benefits of tracking issues publicly" originally appeared at https://opensource.com/open-organization/17/2/tracking-issues-publicly.

Rebecca Fernandez's "Three essential skills for fostering productive debate" originally appeared at https://opensource.com/open-organization/17/5/fostering-productive-debate.

Laura Hilliger's "What to do when your open team has impostor syndrome" originally appeared at https://opensource.com/open-organization/17/5/team-impostor-syndrome.

Allison Matlack's "When innovation trumps process" originally appeared at https://opensource.com/open-organization/17/4/doing-the-right-things.

Lauri Apple's "Better IT culture via the Socratic method" originally appeared at https://opensource.com/open-organization/17/5/better-it-socratic-method.

Stephen Gold's "A formula for running an accountable IT organization" originally appeared at https://enterprisersproject.com/article/2016/7/cvs-health-cio-shares-his-formula-running-effective-it-organization.

Colophon

Typeset in DejaVu[2] and Overpass.[3] Produced with LibreOffice[4]. Cover design by Libby Levi.

Version 1.0

June 2017

2 http://dejavu-fonts.org/wiki/Main_Page

3 http://overpassfont.org/

4 https://www.libreoffice.org/

Also in the series

From Harvard Business Review Press

The Open Organization: Igniting Passion and Performance, by Jim Whitehurst

From Opensource.com

The Open Organization Field Guide: Practical Tips for Igniting Passion and Performance, by the Opensource.com community

The Open Organization: Catalyst-In-Chief, by Jim Whitehurst

The Open Organization Leaders Manual: Instructions for Building the Workplace of the Future, by the Opensource.com community

Additional reading

Every week, Opensource.com publishes new stories about the ways open principles help innovative leaders rethink organizational culture and design.

Visit *opensource.com/open-organization* to read more.

A note on style

This book uses the lower-case term "agile" as a general descriptor for various development methods that embrace flexibility and value responsiveness. It uses the capitalized term "Agile" as a proper noun naming the specific development principles outlined in The Agile Manifesto.

Contents

Part 2: Practices

Appendix

Learn More

Preface

Bryan Behrenshausen

> "Culture is one of the two or three most complicated words in the English language."—Raymond Williams[5]

I doubt I (or cultural theorist Raymond Williams, for that matter) need to convince you of culture's complexity. After all, you've just opened a book offering to guide you through the confusing and confounding nature of a thing that's suddenly front and center in so many pressing discussions. Culture, Williams says, "has now come to be used for important concepts in several intellectual disciplines and in several distinct and incompatible systems of thought"—and he was writing more than 30 years ago. "Culture" hasn't become less puzzling or ambiguous in the ensuing three decades; it's become only *more so* as it pops into conversations anywhere and everywhere. Attempting to understand it, we need all the help we can get.

Hence this book, in which more than two dozen writers, developers, organizational leaders, and influential technologists track the increasing interest in something called "IT culture." That's an evocative term, because it reminds us that "culture" is inseparable from another notion: "technology." Williams knew this too. In his book about a technology he found particularly im-

---

5 See *Keywords*, 1976.

portant for understanding social relationships in the late 20th century, *Television*, he aired his frustrations with the ways people seem to talk about the relationship between the social and the technical.[6] We tend, Williams observed, to imagine one of those forces as animate or active and the other as inanimate or passive. For example, we'll often talk about certain technologies as simply the "byproducts" or "symptoms" of supposedly broader social forces (as if they were just the deposits of more important stuff that goes on between people). Or we'll talk about certain technologies in the opposite way: as something able to single-handedly revamp and reconfigure entire aspects of society just by appearing on the scene (seemingly from a vacuum).

The fact is—and the authors in this volume get it—*neither* manner of talking about the relationship between the social and the technical is ever entirely sufficient. These forces are two inseparable sides of an ongoing process; they're of a piece. Any theory, explanation, or story insisting that they're easily divorced from one another is likely just light on critical details.

This book's contributors hope to convince you that any discussion you have (or any decision you make) regarding technologies *must* account for the social principles and cultural values those technologies embody or represent—and vice versa. The technologies we use both reflect and reinforce certain ways of working together; the ways we desire to work together inevitably shape the technological choices and decisions we make. We can no longer pretend to somehow have one without the other.[7]

---

6 See "The Technology and the Society" in *Television: Technology and Cultural Form*, 1975.

7 DevOps advocates will recognize the logic of this argument, as will fans of the Agile Manifesto. My thanks to Jason Hibbets and Lauri Apple.

So while this book is divided into two *sections*, you should resist the temptation to treat it as consisting of two discrete *parts*. They're intimately related. The first, "Principles," describes several significant changes to the values that have traditionally guided information technology organizations. It explains how open principles are forcing us to rethink our deeply held assumptions about IT's "whys" and "whats." The second section, "Practices," outlines new behaviors we might adopt in order to embrace and express the values necessary for driving innovations from the IT shop to the rest of a business. As the book's title makes clear, these must go hand-in-hand if technologists intend to weather the challenges they're facing today.

One final note: We[8] believe you're reading what is quite possibly the first edited volume developed according to the guidelines of the Open Decision Framework,[9] an architecture for ensuring that both the principles and the practices of openness work in productive tandem to realize the stellar results only openness can. We're grateful to the dozens of writers, advisers, proofreaders, and source matter experts who helped shape this book into the extraordinary artifact you now possess. The book came to life—and continues to live—as a project on GitHub, which readers can visit to flag errors and suggest modifications.[10]

We hope you'll join our community, both there and at Opensource.com, to continue this important conversation about one of the most complicated words in the English language.

---

8 My thanks to colleague and collaborator Jason Hibbets, without whose help this book would not exist.

9 https://opensource.com/open-organization/resources/open-decision-framework

10 https://github.com/open-organization-ambassadors/open-org-it-culture

Bryan Behrenshausen works for Red Hat on Opensource.com, where he's been a writer and editor since 2011. In 2016, he earned his PhD in Communication from The University of North Carolina at Chapel Hill, where he studied the relationship between culture, technology, and other complicated words.

Introduction

Mike Kelly

The IT department is uniquely positioned to handle change. Good IT teams *manage* change. The best ones *lead* change. As the pace of change accelerates today—and at a time when technology is in many respects *the* asset of a company—organizations are demanding their IT departments demonstrate more leadership than ever before.

Today's highest-performing IT teams are leveraging open principles to lead their organizations through monumental technological, social, and economic changes. They're becoming more collaborative and more transparent—and more agile and accountable as a result. They're rethinking organizational boundaries that have constrained them for decades and forming new, productive relationships across the business. They're sharing resources with internal and external stakeholders as they seek to innovate in operationally excellent ways.

This shift to open principles and practices creates an unprecedented challenge for IT leaders. As their teams become more inclusive and collaborative, leaders must shift their strategies and tactics to harness the energy this new style of work generates. They need to perfect their methods for drawing multiple parties into dialog and ensuring everyone feels heard. And they need to hone their abilities to connect the work their teams are doing to their organization's values, aims, and goals—to make sure everyone in the department understands that they're

part of something bigger than themselves (and their individual egos).

In short: Today's IT leaders need to be *culturally* competent as much as they are *technically* competent. That's why a guide like this one is so important, and its chapters reinforce this point. As you read, I hope you'll recognize the powerful role you can play in guiding your organization to success.

Change is intensely emotional. And because the IT department is always at the forefront of change, it's always involved in an organization's most emotional activities. Never forget that. Your effectiveness as a leader depends on it.

Mike Kelly is CIO at Red Hat. He's been recognized in the technology industry as a champion of talent development and leader of large multi-national, diverse, high-performance teams.

Part 1: Principles

Introduction to Part 1

Mike Walker

In 2016, I launched Open Innovation Labs, a place where people seeking to leverage the principles of openness can work with a seasoned team to build innovative software that solves their most pressing business problems. It has been an exciting and daunting undertaking. Today, Open Innovation Labs imparts knowledge and best practices that emerge from the world's most successful open source projects, and we provide a residency-style experience that immerses teams in those practices.

We generally partner with companies looking to do two things: Either they want to move quickly and be disruptive, or they see disruption as an existential threat and seek to adapt their behaviors to facilitate a more rapid pace of change.

Our own team began as one of the former, but we suddenly found ourselves one of the latter. The lessons we learned as we made that transition—lessons about organizational culture and the power of openness to shape it—truly have made us better coaches today.

Here's what happened.

An identity crisis

To launch Labs, I built a small, cross-functional team that immediately set to work doing the very things we encourage lab residents to do. First, we created hypothetical scenarios to

achieve our team's objective—which, in this case, was to accelerate our customers' work in a residency-style engagement and build enthusiasm for building applications the open source way.[11] Then, we applied emerging technologies to build next-generation software that would help us explore those scenarios. We worked relentlessly to create a clever system for accelerating customers' efforts and delivering real value.

We called that system "push-button infrastructure," or "PBI." PBI allows us to spin up a customer-ready Labs environment, built for speed and experimentation, in less than an hour. We took it to market as soon as we could (even before it was fully functional), and the reaction actually took us by surprise. "I want that!" was the most common response we received from customers and internal teams. We were onto something. The excitement was palpable.

About nine months into the endeavor, one of my technical staff pointed out that another open source engineering team in the community had made a major breakthrough with their software—something that allowed them to create a system that could eventually replace much of what we'd built. What's more, this team was easily ten times larger than ours, with a huge ecosystem of partners to boot.

I reached out to one of the project leaders, another open source advocate. He candidly told me that our group was the inspiration for much of their efforts.

While I was flattered, my stomach also sank: Our little gem was already in danger of being disrupted by a more powerful force.

The team took stock of the situation. We faced an identity crisis. If our core product became obsolete, then would we con-

11 https://opensource.com/open-source-way

tinue to exist? In that case, what was our core value proposition? What would we look like?

The primacy of principles

We convened face-to-face to sketch our core value proposition in a group exercise called "mind mapping." It produced two strong revelations.

The first was about PBI. As exciting as the technology was, it didn't actually define much of what made us special. In fact, it was barely on the mind map. If it came from another group or was abandoned entirely, it still wouldn't stop us from achieving our mission. This was a big weight off of our collective shoulders.

The second realization was about something more abstract: The team's most-valued asset was its shared belief system. We all named characteristics like "collaboration," "authenticity," "transparency," "accountability," "open decision making," "meritocracy," "adaptation," "experimentation," and "a focus on impacting people" as the things that made our team truly unique and capable of delivering value to our customers.

These cultural principles came from our experience working in open source communities. We listed principles like:

- Shared problems are solved faster
- Transparency forces authenticity and honesty
- Participative communities are more open to change
- Open standards provide business agility

Our core mission, we decided, was to share with our customers and partners these same principles over the course of our engagements with them—to help them leverage lessons from the open source world to build, better, more adaptive solutions to problems. This insight allowed us to pivot productively; we realized we should focus *more* on imparting new ways of

working together to catalyze organizational (and eventually cultural) change, and *less* on any particular technical solution or offering.

Shortly after that meeting, we made the bold choice to utilize the great work coming from the external community we'd viewed as an existential threat only days before. We adapted our software to take better advantage of that offering. Doing this meant halting development on some of what we initially considered core material in order to ride a bigger, better wave.

I still keep the result of that mind map on my desktop. It reminds me that our shared beliefs endure far beyond any particular experiment or project. It centers our team in a way that allows us to transcend the individual, and become part of a bigger purpose, satisfying a fundamental human desire that lays in each of us.

As you read the first section of this guide, I hope you too will discover the benefits of building an organizational culture on open principles—one that transcends any individual effort and creates an enduring, shared purpose capable of inspiring teams long after we're all gone.

Mike Walker is the Global Director of Red Hat's Open Innovation Labs, whose mission is to make it easier and quicker for customers to bring innovative ideas to life. He has 16 years of IT engineering and consulting experience with emerging technologies, including specializations in application development, cloud computing, data integration, high-performance computing, and distributed systems.

What's an IT culture, anyway?

Jono Bacon

"Culture" is a pretty ambiguous word. Sure, reams of social science research explore exactly what exactly "culture" is, but to the average Joe and Josephine the word means something different than it does to academics. In most scenarios, "culture" maps more closely to something like "the set of social norms and expectations in a group of people." By extension, then, an "IT culture" is simply "the set of social norms and expectations pertinent to a group of people working in an IT organization."

I suspect most people see themselves as somewhat passive contributors to this thing called "culture." Sure, we know we can all contribute to cultural change, but I don't think most people actually feel particularly empowered to make this kind of meaningful change. On top of that, we can also observe significant changes in cultural norms that depend on variables like time and geography. An IT company in China, for example, might have a very different culture from a company in the San Francisco area. A startup in Birmingham, England, will have a different culture to a similar startup in Berlin, Germany. And so on.

Culture is critical. It's the lifeblood of an organization, but it's complicated to understand and shape. The "IT culture" of the 1980s and 1990s differs from "IT culture" today—and it will be different again 10 years from now. Apart from generational

changes, cultural norms for IT practitioners have changed, too. Today, digital technology is more social, more accessible to people with fewer technical skills, and more embedded in our consumer-oriented world than ever. We've learned to cherish simplicity, elegance, and design, and this has reflected the kinds of organizations that are forming.

So in one sense, IT culture is a box of frogs: a variable, changing, and unpredictable entity. In another sense, IT culture is a relatively straightforward issue: It's the connective tissue between *people* and *output*. Organizations need to produce output—products, services, support, events, and more. People drive that work, and they need to be *productive*, *efficient*, *contextually aware*, *evolving*, and *happy*. None of these attributions are optional: When one is missing, frustration starts setting in.

More important than defining IT culture *today*, though, is exploring what an optimal IT culture of *tomorrow* will look like. I want to focus on **five key areas** that I consider to be critical facets of a high-quality IT culture.

Let's do this.

1. Pipelines should be connected

In a typical organization, you have a number of different "pipelines," as people external to the organization get connected to different teams. Examples include:

- **Sales**: prospects → leads → opportunities → customers
- **Community**: users/consumers → advocates → contributors
- **Recruiting**: prospects → candidates → employees
- **Marketing**: broader audience → qualified → connected

You'll also find pipelines that relate to workflow. Examples here include:

- **Engineering**: product features/bugs → specs → code → reviews → product
- **Product**: requirements → ideas → backlog → scoped items
- **Marketing**: key messages/features → ideas → scoped items
- **Support**: requests → triaged requests → engagement

Organizations suffer when people descend into silos, and disconnected pipelines can be a contributing factor to this, especially for IT organizations. As such, explore how you can glue different pipelines together in a sensible and natural (not "forced") way. How can your IT team connect to the community pipeline, for example? How can community members support the sales pipeline? How can engineering and marketing workflow connect together?[12]

Done well, this reduces silos, integrates team cultures, and reduces complexity and road bumps along the way.

2. Workflow should be asynchronous

I spend a lot of time working with companies, helping them to build internal communities and organizational workflow. While many factors influence the start of this work, I always zone in on one key area first: *asynchronous workflow*.

Put simply, asynchronous workflow is the ability for employees to be able to work on *anything*, from *anywhere*, at *any time*. Conventional organizations mix together in-person meetings, whiteboard sessions, online discussions, and other methods of collaboration. But multiple ways of working mean that information often gets lost. For instance, in-person meet-

12 See Jackie Yeaney's chapter in this volume.

ings without clear notes mean that those outside the room have a deficit of context.

Asynchronous workflow helps to solve this issue. When we focus on discussing ideas and projects in an electronic setting, content and discussions are archived and available to everyone. This makes organizations (including IT organizations) more open and transparent. This doesn't mean you can't have in-person meetings, but you have to reinforce a policy of taking notes and decisions in a way that ultimately end up online for the wider team.

Asynchronous workflow is critical for organizations to scale and it is better to get it integrated as early as possible. It requires discipline and training but, done well, it breaks down silos, opens up opportunities across the organization, and creates accountability and a powerful imprint of best practice (and failures) that can be invaluable.

3. Operate a connected meritocracy

I come from the open source world, where the notion of *meritocracy* has been steeped in our culture. In this context, a meritocratic culture is one in which everyone is judged on their *merit*, and it doesn't matter what their gender is, what their skin color is, what car they drive, what their haircut is, and so forth. They're judged on their *contributions*.

Remember that meritocracy is *not* a framework or model. It is a philosophy. Meritocracy can be difficult to put into practice for all kinds of reasons, but I do think it is an important guiding light for our work.

When thinking about your IT culture, think about how you can provide a pathway in which anyone can showcase their capabilities and contributions. This is where being *connected* is important. The best organizations I have seen have the ability

for people from across the organization to contribute. And this is where asynchronous workflow can be hugely helpful. Tracking your project management, engineering, and marketing in an open, online internal system provides an opportunity for people in different teams to feed in and contribute.

When I have seen this in place, I've observed surprisingly valuable contributions: legal feeding into engineering (e.g. such as reviewing licensing/copyright/firmware issues), sales reps feeding into community (e.g. fueling shared knowledge bases and potential customers), product people feeding into support (e.g. coordinating around customer requests), and beyond.

4. Data-driven experimentation is essential

No two organizations are the same. Seemingly similar beasts such as Microsoft and Intel, or Mattermost and Slack, or Canonical and Red Hat, embody totally different cultures. As such, we can learn different lessons from different organizations, but the real insight into what makes *your* organization tick has to be formed with *your* people, processes, and workflow in mind.

As such, to really optimize an IT culture, we need to *experiment*.

The construction and execution of small- and large-scale experiments will help us to discover new insights that we can use as clues to help us make future decisions. With one of my clients, for example, I put in place an experiment to reward contributors with different types of validation (both intrinsic and extrinsic) of their work at different levels of participation. This helped us to determine what kinds of validation people appreciated, and as a boon this mapped well to staff too (who were also wanting validation for their contributions). This was a small experiment, but we analyzed the results to look for clues that

could inform future experiments. We applied what we learned to future work and saw some great results.

The key here is to be data-driven and, frankly, *honest* with yourself while you're experimenting. We need data to suitably determine the success or failure of an experiment, and we need to be honest in peeling away our internal goals and biases to see the experiment's results in an objective light.

We can perform these experiments all over an IT organization, and we should encourage employees to brainstorm ideas for segments. These can be small exercises that involve very limited costs and can deliver incredible insight. They can act as a means of diversifying ideas and limiting risk. I highly recommend you put in place a regular cadence at which you run different experiments across multiple teams. Doing this can deliver great results and offer a wonderfully creative environment for your employees.

5. Accepting failure is not an academic exercise

Many people understand the value of embracing failure as a means to learn from it. Thousands of people read books and articles about this, and with the best will in the world seek to bring this into their organizations.[13]

Sadly, in many cases we can see an *academic* understanding of this but not much of a practical application. Leadership in the majority of organizations rolls downhill. If you have a bitter, nasty leader, you get a bitter, nasty culture. If you have an engaging, respectful, friendly leader, you get a more positive culture. As such, embracing failure needs to cascade through the ranks in a meaningful way.

13 See Gordon Haff's chapter in this volume.

The best IT leaders I've seen embrace failure have been remarkably upfront about failures, both organizationally and personally. They've said, "I screwed this up and this is how I learned and became better from it." These conversations can't be soundbites. They have to be authentic, and this can be tough for leaders of an organization to instill in their daily routines.

Another element here is to reinforce in others the value of embracing failure. You can't preach the value of failure and then hammer people when they fail. Of course, be disciplined in requiring excellence from people in the organization, but base your criticism on a body of constructive next steps. Anger, frustration, and annoyance are normal and to be expected, but you have to augment them with sage, constructive guidance. Our ultimate goal here is to have people look back on their failures and feel like they've grown and improved as a result of them.

Of course, I am merely scratching the surface of what great IT culture is, but I think if you can take these five areas and start building them in your organization, you will see some great results.

Jono Bacon is a leading community manager, speaker, author, and podcaster. He is the founder of Jono Bacon Consulting, which provides community strategy/execution, developer workflow, and other services. He also previously served as director of community at GitHub, Canonical, XPRIZE, OpenAdvantage, and consulted and advised a range of organizations.

Chapter discussion and review

• "Culture is critical," writes Jono. "It's the lifeblood of an organization, but it's complicated to understand and shape." What does "culture" mean to you? What role does it play in your organization?

• Jono stresses the value of *asynchronous work* in fast-moving IT cultures. Does your team or organization leverage the power of asynchronous work? Should it? How?

• Does your team or organization foster a culture of experimentation? Where might experimentation be most beneficial to your work?

• What is a "meritocracy," and how might it work in your IT organization?

Organizational learning: A new perspective on DevOps

Gene Kim

In the DevOps community, we talk a lot about automated deployments, doing multiple deployments per day, and the need for culture. I want to share with you something that isn't talked about nearly as widely, but I think is just as important: the benefits of organizational learning.

Let's take a moment to visualize what an organization that has fully adopted DevOps principles and practices might look like.

We are able to accommodate a high rate of change that allows us to satisfy our organization and out-experiment our competition. Our changes have short lead times, and we can make changes and deploy code at any time of the day (as opposed to only on Friday at midnight), without organization-paralyzing fear that it will cause massive chaos and disruption.

Our code and environments are safe to change (and we can recover from mistakes quickly), ideally without impacting the customer. We have created a high-trust environment where we can rely on our team members throughout the entire value stream, knowing that we are all working together to help the organization win.

When bad things happen—which entropy and Murphy's Law ensure—we have sufficient monitoring in place to quickly

find out what is going wrong, restore service, and resume normal operations. Because we have a culture of relentless improvement, we will figure out how to prevent it from happening again in the future. If we can't, at least enable quicker detection and recovery.

And because we know that more important than the *doing* of our daily work is the *improvement* of our daily work, we are constantly learning as an organization and turning local discoveries into global improvements.

In his book, *The Fifth Discipline*, Peter Senge explains that "knowledge exists at the edges, not at the center." He suggests that we need organizational learning because it enables helping our customers, ensures quality, creates competitive advantage and an energized and committed workforce, and it uncovers the truth.

We therefore must create a culture that rewards learning, even when it comes from failure. Moreover, we must ensure that what we learn becomes embedded into our institutional memory so that future occurrences are prevented.

Encourage and celebrate learning

No amount of command and control management can direct workers to fix each strand, one by one. Instead, we must create the organizational culture and norms so that everyone finds and fixes broken strands, all the time, as part of our daily work.

Our goal should be to maximize our organizational learning from any accident, gain the best understanding of how the accident occurred, and empower everyone to create the most effective countermeasure to prevent it from happening again or enable quicker detection and recovery. In addition, we must foster a culture where the entire organization learns from

accidents, so that any local improvements can be turned into global improvements.

Intuit has a famous monthly ritual where the CEO of the company gives a ceremonial life preserver to the person who made the largest mistake. The recipient signs the life preserver, then tells the entire company what happened and what they can learn from it.

Make it easier to use standards than not

Using standards that encompass the sum of our organizational knowledge should be easier than *not* using standards. One of the best places to put this knowledge is into a centralized source code repository that is shared throughout the organization, allowing the ability to quickly propagate knowledge. Some other characteristics of successful standards include:

- Shared source code repository and thorough documentation that can be searched and widely reused
- Internal discussion groups for each library and service (e.g. "github-users" or "puppet-users"); often people having questions will get responses from other users faster than from the developers
- Widely broadcasted, blameless postmortem reports

Justin Arbuckle, former chief architect of GE Capital once said, "The best architecture document is one that is implemented in code, in a shared source code repository, that anyone can pull from."

Enable the organization to discover its way to greatness

By valuing learning, we create an organization where we no longer expect leaders to plan our way to greatness. Instead, leaders help foster and develop routines, test them in practice,

recognize which don't work, and reinforce those that do. Leaders do this by reinforcing the value of learning and ensuring that obstacles are removed so that whatever got in our way yesterday and today won't get in our way tomorrow.

What does organizational learning look like in a real DevOps journey?

I recently had a chance to hear about organizational learning from Jim Stoneham, CEO of Opsmatic. In 2009, he was the general manager of the Yahoo! Communities business unit, which Flickr became a part of. Stoneham shared:

> "The amount of our organizational learning went through the roof as we increased our deployment frequency at Yahoo! Answers from once every six weeks to multiple times per week. Suddenly, we were able to able to try things out and experiment in ways we hadn't been able to do before. Our team became very much in tune with the numbers: we'd look at them as a team on a daily and weekly basis, and use that to inform feature conversations and plans.
>
> Instead of engineers talking about the product once every six weeks, we'd be talking about it daily. This was exactly the learning that we needed to win in the marketplace—and it changed more than our feature velocity. We transformed from a team of employees to a team of owners. When you move at that speed, and are looking at the numbers and the results daily, your investment level radically changes. This just can't happen in teams that re-

> lease quarterly, and it's difficult even with monthly cycles."

I love how Jim Stoneham talks about the benefits about DevOps that sound very different than how we often talk about it as Dev or Ops. It's this capability of creating organizational learning that enables us to win in the marketplace.

Gene Kim is a multiple award winning CTO, and researcher. He was founder and CTO of Tripwire for 13 years, and is an author of both The Phoenix Project *and* The DevOps Handbook.

Chapter discussion and review

• Gene argues that "we therefore must create a culture that rewards learning, even when it comes from failure." Does your team or organization cultivate a culture that rewards learning? How? How could it improve at doing this?

• Does your organization maintain some kind of knowledge commons, a place to store the collective wisdom it builds? If not, do you think one would be useful? Why or why not? And how could you build one, if necessary?

• "Using standards that encompass the sum of our organizational knowledge should be easier than *not* using standards." Gene writes. What processes for standardization does your team or organization have in place? What can it standardize to enhance efficiency and reliability?

Innovation requires new approaches to feedback and failure

Jim Whitehurst

"Organizational culture" is something plenty of people are puzzling over today, and with good reason. More and more leaders are realizing that the culture permeating and guiding their organizations will determine whether they succeed or fail.

The term "organizational culture" refers to an alignment between two forces inside an organization: values and behaviors. Aligning those forces productively is one of the most difficult and important tasks facing leaders today.

Customers and partners routinely tell me they want to create a "culture of innovation" in their organizations. By this, they usually mean that they want to create contexts where certain actions—those that generate new and unforeseen sources of value capable of fueling growth—are not only expected but also commonplace.

I certainly understand why. Today, a culture of innovation is a strong indicator of an organization's ability to weather the kinds of constant disruption nearly every industry seems to be experiencing. Creating one is easier said than done.

Here's how I'd recommend an organization approach that challenge.

A new method

One method for creating a culture of innovation involves focusing on how your organization treats *feedback* and *failure*.

In innovative organizations, feedback is continual and frank—in other words, it's open.[14] Dialogue about associates' ideas must be ongoing, constructive, and, above all, honest.

To foster innovative environments, leaders must model the kinds of feedback behaviors they want to see in their teammates and associates. They need to be open to even the most difficult conversations.

Innovation is one product of creativity. Despite the way we tend to think about it on most days, creativity is *very difficult*; it's the product of intense collaboration and sharing. Actually, Ed Catmull and Amy Wallace discuss creativity this way in their book *Creativity, Inc.* Innovative teams and organizations, they say, must have some way to simply separate the wheat from the chaff—to call a bad idea a bad idea—and move forward. Creating a culture of respectful, frank disagreement is key to this.[15] The opposite of this kind of culture is one where feedback is a rarity—or, worse yet, where it's only positive. (As I wrote in *The Open Organization*, it's possible for organizations to be "terminally nice.")

One of the things people receive feedback about is their failures. Cultures of innovation take a specific approach to failure: They celebrate it.

Without question, being innovative involves taking calculated risks. People in innovative organizations must feel like they can try something novel and unexpected without fear of in-

14 See Jimmy Sjölund's chapter in this volume.

15 See Rebecca Fernandez's chapter in this volume.

tense, negative blowback; otherwise, they'll never attempt anything new.

Traditionally, we've treated failure as a sign of personal failing. Someone faced with a tough choice didn't make the "right" decisions, so we need to punish the behavior that led to a certain outcome.

In cultures of innovation, where everyone is expected to experiment, how can anyone possibly know what the "right" and "wrong" decisions will be if the problem is so new that few people have any concrete experience with it?

Instead, I like to think about failure the way Jeff Bezos once described it in a letter to Amazon shareholders.[16] He said:

> Most large organizations embrace the idea of invention, but are not willing to suffer the string of failed experiments necessary to get there . . . Given a ten percent chance of a 100 times payoff, you should take that bet every time. But you're still going to be wrong nine times out of ten. We all know that if you swing for the fences, you're going to strike out a lot, but you're also going to hit some home runs.

The trick to making this approach to failure an organization's *default* approach is changing the way we think about evaluation.

Traditional management is management by objective. It examines *outcomes* to see if they're aligned with expectations someone set out *before* undertaking a task. If these don't align, then someone, somewhere, has failed—and that's a bad thing.

In innovative cultures, we need to balance that approach with one that actually rewards failure. Leaders must be able to

16 https://www.sec.gov/Archives/edgar/data/1018724/000119312516530910/d168744dex991.htm

encourage certain *motivations*, which are a key source of innovation. They're not as overt or quantifiable as outcomes, which is why traditional management theory struggles to account for them.

How can leaders assess people who might have failed, but who've demonstrated exciting new ideas and approaches along the way? And how can they encourage others to actually emulate those people?

If you can get there, you'll know you have a culture that rewards risk-taking.

A focus on structure

This approach to creating a culture of innovation isn't a foolproof and complete plan for changing the way your organization functions today. I don't think such a comprehensive plan actually exists. (If it does, please let me know!)

But I do believe that focusing on the organizational structures that govern approaches to feedback and failure is a promising way to begin—much better than simply telling people to "be more innovative."

Jim Whitehurst is President and Chief Executive Officer of Red Hat, the world's leading provider of open source enterprise IT products and services.

Chapter discussion and review

• "Organizational culture" is an important topic to IT leaders today. What does the term mean to you?

• How does your team currently handle feedback and address failure? How might you be able to change your approach to these issues?

• What do you think are your team's or organization's largest impediments to creating cultures of innovation?

• Jim writes: "How can leaders assess people who might have failed, but who've demonstrated exciting new ideas and approaches along the way? And how can they encourage others to actually emulate those people?" Can you think of strategies for doing this on your team or in your organization?

Transparency, failure, and other things I've learned to enjoy

Nick Hall[17]

We've all heard statements that begin with phrases like these:

- "Full disclosure . . ."
- "I'll admit . . ."
- "I'll be honest . . ."[18]

They preface a moment of clarity, bringing everyone to a place of common ground through complete and utter transparency. They promise listeners a window into what's really going on.

And they're not always comfortable.[19,20]

This discomfort isn't the result of some aversion to honesty; it's just that the full story is rarely convenient or glamorous. When someone starts with one of those phrases, we

---

17 Abbreviated "NH" in the footnotes. Edited by Bryan Behrenshausen ("BB" in the footnotes).

18 Love the idea of jumping right in with these all-too-common turns of phrase. In fact, I think we can work them into the narrative strategically and stylistically.—BB

19 Would be good to have an editors note or something with how many edits/revisions/changes/whatever went into the final form of this.—NH

20 We can totally do that. I'll leave the marginalia in the chapter as footnotes.—BB

react immediately: *This conversation is taking a different path than I anticipated*. When the tide turns from truisms to this *other thing*, we have to be prepared for *anything*, and if that *other thing* is complete transparency, these conversations have a way of exposing things that we might not be so proud of. They can shine a light on our vulnerabilities. And even if the light isn't focused on us, our self-reflection can still bring those vulnerabilities to the surface.

It might not be comfortable, but it can be incredibly rewarding.

I'm going to explain that positive outcome—how transparency and failures can work hand-in-hand, and how coming from a place of discomfort, vulnerability, and disappointment can be a good thing, not only for us individually, but for our projects and our teams.[21,22]

Consider this chapter one of my projects.

The transformative power of transparency

One key benefit of transparency is its transformative power.[23,24]

---

21 Readers would benefit from a clear and concise encapsulation of your chapter's argument right here. What are you going to argue, prove, or teach? Bonus points if you can tie that work to the previous statement and explain (earlier) why those statements actually serve to make people uncomfortable, rather than the other way around.—BB

22 I expanded a bit here, referenced those earlier remarks, and added a transition that I think ties to the next section as well as the close of the article—which hopefully will allow us to keep that idea on how this article was shaped over time and incorporate that if it still works.—NH

23 Another editorial option would be to actually begin the chapter here, which the reflexive approach of speaking to the chapter itself, as that's how you've currently ended the piece.—BB

Undoubtedly, by the time you read this chapter, it will be in much better shape than it was when I originally typed it. If the final product is lacking, rest assured you are still faring better than you would have otherwise. And that is because of the tireless work and patience of an editor tweaking, cutting, revising, suggesting—perhaps providing some "tough love" now and again.

As someone who has not been writing like this for some time (*full disclosure*: I once worked as a newspaper reporter), I am not only expecting it, but also looking forward to it. To edit and be edited wears down those barriers we erect to shield ourselves from criticism, not unlike the way a coarse rock tossed across a river becomes smooth over time. It doesn't injure you (at least, not in the long run), but it does mold you. Failures are part of the process; you will see them called out, and you will make fewer mistakes as you go along. The process improves not only the deliverable, but also the people (or person) responsible for it. And if I continued to write—and our paths crossed as writer and reader months down the road—I'd imagine we would both see the progress, the positive transformation over time, formed by cycles of iterations, project after project.

It's a cycle that is built on failure and transparency. Thus, one way to think about failure is to consider it the ultimate form of development.

Failing to develop, developing failure

I was discussing all of this with one of my mentors recently. She mentioned a development exercise she'd undertaken, which involved self-assessing and soliciting feedback from oth-

24 I kept the opening you have, but I think we highlighted the chapter itself a bit more with some of the changes.—NH

ers. The goal was to identify areas for professional improvement.[25]

The results, in her case, were not particularly useful. They didn't really give her any clear area for improvement. And she said that made her uncomfortable.

I'll be honest: I could see why. I suggested that "either you're perfect, or they're delusional," and we both laughed. Those probably aren't the only two options, if we're being fair, but we agreed on the point: Everyone can improve. But the only way you can improve is to be open and honest about those areas of weakness, to be transparent when you make mistakes, and to continue to work on refining those areas.

The value of transparency—especially in an atmosphere where people can be honest and open about failures—is absolutely vital to truly growing and improving. Anything else leads to stagnation (and sometimes delusional behavior). It's something that we've all seen.

An inability or unwillingness to be open and honest about our own personal shortcomings (or those of others) is damaging to ourselves and to everyone we work and live with. A person who cannot acknowledge their own mistakes or failures does not see the opportunities for growth. And if they see their mistakes but refuse to take ownership of of those mistakes, the situation could be even worse. They *could* identify opportunities for improvement, but *they might not care*. This kind of attitude leads to a culture lacking in accountability and engagement—one that emphasizes appearance over substance, talk over action, and the comfortable *status quo* over the uncomfortable change.

25 I'd like to see you elaborate this section somewhat, to really make the takeaway more concrete for readers. How can it function as a better bridge for the sections that come before and after it? How can it connect them?—BB

Like we said earlier: It might not be comfortable, but it can be incredibly rewarding.

And what does that look like?

You may have heard the term "growth mindset," which Carol Dweck coined. It's the result of transparency and openness around failure. One short summary would be: Someone possesses a growth mindset when they believe that they can improve if they put forth the effort, dedicate themselves to improvement, and respond to feedback from others. Doing this requires that others provide constructive, open feedback; it also requires a willingness to thoughtfully accept that feedback and act on it.

According to Dweck, that mindset can permeate an entire culture of teams, even organizations. Those negative qualities we mentioned above—the unwillingness to be open and honest about shortcomings, lack of accountability, lack of engagement, and sometimes complete indifference—turn those on their head.

Transparency is the key to personal growth and development, and it is critical for cultivating a culture where people are interested in improving together.

That leads us to our next point about the great interpersonal value in transparency and openness around failure.[26]

The strength in weakness

So there's value in growth and development—actual, concrete improvement never fully realized without a culture of transparency and a willingness to be open and honest about failure. There's also value in *the failure itself*, especially when you

26 I added quite a bit to this section, and I think the transition from the previous section and the transition out of this section make the connections more clear. Let me know what you think!—NH

yourself are failing and are willing to take ownership of those failures and speak openly and freely about them.

In other words, then, failure has both *personal* and *interpersonal* value.

Failure is personally valuable when it spurs you to overcome hardship and become better in the process (think of all those iterations of that old adage: "What doesn't kill you makes you stronger").

That's all well and good, but failure's interpersonal value might be even more substantial.

Failure, specifically when combined with that culture of transparency, can be incredibly motivating and engaging for others who witness it.

It's not that failure causes onlookers to think: "Wow, I better step up my game because that person really screwed up!" Instead, failure creates an *actual connection*—a bond, a sense of trust and commitment between team members.

When someone tries, fails, owns, and vocalizes their failure to their peers—well, *I'll admit*, that can be very powerful.

Showing vulnerability not only requires strength from the person who's become vulnerable; it also encourages others to become stronger in supporting them, and encourages more of that behavior in the future. Eventually, repeat displays of vulnerability can potentially form the backbone of a strong team.

In *The Five Dysfunctions of a Team*, Patrick Leoncini highlights the importance of this willingness to show vulnerability. Leoncini arranges his five dysfunctions in a triangle. From top to bottom, they are: *inattention to results, avoidance of accountability, lack of commitment, fear of conflict, and absence of trust* (the foundation that gives rise to all other forms of dysfunction). Key to developing trust is showing vulnerability, and here lead-

ers are crucial for displaying the behaviors they expect from their team members.

It makes sense: By creating that trust and mutual support, you'll see how the dysfunctions can begin to right themselves.[27] By showing vulnerability, you erase the myth of perfection. You show your team that you are, in fact, a mere mortal, and you create a healthy environment where people can fail and then trust that they won't be ostracized, singled out, or considered weak or incapable. From your example, they'll learn to trust that you will similarly be as open the next time things don't go as planned. The more your team sees it, the more willing they will be to show it.

It's easy to see how transparency and trust go hand-in-hand. Soon you'll be able to tackle other forms of dysfunction—when people can trust one another, they're more willing to engage in *healthy conflict*, because they're not afraid to speak their minds. When they're more willing to engage in healthy conflict, they feel like their opinions have value and they can have their say in the direction of their team, and they become more *committed and engaged*. When they are more committed and engaged, they invest themselves more heavily in their work and take ownership of it, accepting *accountability* for the good and the bad. And engaged, accountable teams are deeply *committed to the results* of their actions and their work.

It sounds easier than it is in practice, but that foundation of trust—built on transparency—is the first step.[28]

---

27 Can you perhaps expand this just a bit? Elaborate? It's an important point; drive it home.—BB

28 Alright, I sort of walked up the triangle starting with trust. I think this works to illustrate how that foundation relies on transparency and how important it is to a healthy team.—NH

Embracing a culture of failure

Alright. Recap: We've seen the value of direct, concrete improvements that emerge from growth and development, all of which is dependent on a transparent culture. We've also seen the forms of *personal* and *interpersonal* value we gain from the mutual trust and support arising from the vulnerability we demonstrate by maintaining a transparent culture.

What else is there to say about the value that comes from a culture where you can be transparent and open about failing?

Let me be clear: It can be fun.

Certainly it isn't *always* fun. But embracing the inevitability of things that do not go as we expected can be rather valuable.

For example, I was recently in a class where one participant, a manager with a moderately large group of direct reports, was explaining how he embraces failures as a leader. The manager acknowledged that he's rarely the person with the most knowledge of any team function—just like anyone else in a leadership role who works with a diverse group of people embodying their own skillsets, interests, and strengths. That's the whole point of recruiting and developing a team of specialists.

Whenever team tensions were high—when people were stressed, and the job just became a bit too much—this manager would take the opportunity to get his hands dirty and pitch in wherever he could.

Sometimes that would mean he'd fail spectacularly. But he also found that it boosted morale. It was disarming, it lowered stress, and it lightened the mood. To his team, it communicated that he supported them, that he was willing to work as hard as he needed to in any number of areas to help the job get done, and that he was willing to look foolish in the process.

I thought it was a great idea. It produced a strong connection among the team members and encouraged others to do the same, to fill in as needed, to embrace their collective strengths and weaknesses, and to have some fun with it.

Getting there from here

Cultivating a transparent culture is no easy task. But I hope I've shown you the value of doing it: the development and growth opportunities at the individual and group level, the interpersonal connections and group trust that can come from vulnerability and honesty, and the positive and productive work atmosphere that results from a willingness to try new things and not be afraid to fall on your face every now and again.

So what are some useful takeaways and tips to get *there* from *here*—wherever *here* is?

- **Communicate all failures as an opportunity for growth.** Whether you're in a leadership position or are working through your own or a peer's failure, use failure as a chance to help others develop a new skill, shore up a weakness, or maybe re-align talents in another area, if that's more appropriate.
- **Lean on those with an editor's mindset.** Use those more open and willing to communicate their failures as a mentor and potential keeper of the culture within your group. Showing vulnerability has a way of connecting others, and they can help pave the way for that behavior.
- **Take on tasks that are outside of your comfort zone.** Assist others when necessary, and encourage others to branch out and explore different functions in the team (even if it's not familiar to them). It's a way of politely nudging people into the unknown, but by providing

support, it can be a great way to facilitate experimentation in a safe environment.

- **Don't be afraid to have the discussion.** Failure and accountability go hand-in-hand. Having difficult discussions about accountability and becoming transparent about failures are fundamental to a successful team. When you have that discussion, come from a place of support and don't be afraid to make yourself an example. Walk through the intent, the result, and the impact, and then brainstorm and iterate. How and why did something go wrong? How do we adjust for next time? Should there even be a next time? Ultimately, keep things in the greater context. Chances are the failure was not in a life-or-death matter, so perspective is important. You can always take corrective steps, and there are always opportunities for getting back on the right path. This is true no matter what your role.

Start small, and start with yourself. Cultures do not develop overnight, and some atmospheres are more forgiving of failures than others. But all atmospheres are made better by transparency and open discussion about failures.

As are all chapters.[29]

Nick Hall is a project manager in Global Partner and Technical Enablement at Red Hat, where he focuses on team processes and standards. He manages the go-to-market process for course development and release for both internal and external enablement programs and assists with project management, communications, and reporting for the Red Hat Online Partner Enablement Network (OPEN) program.

29 This document received approximately 240 total edits.—BB

Chapter discussion and review

• How transparent is your team? How transparent is your organization? Can you identify areas in which greater transparency could enhance your collective work?

• Do you agree with Nick when he argues that increased transparency leads to increased accountability in an organization? Why or why not? What limits to transparency must you acknowledge in your organization?

• Nick says transparency is the foundation of trust in an organization. Do you agree or disagree? Does your team or organization experience issues with trust? And would greater transparency solve them?

• Are members of your team open and honest about their failures as much as they are about their successes? Could they be? Should they be?

Why a Buffer developer open sourced his code

Jordan Morgan

If you look for the official definition of open source, you'll likely stumble upon this outline[30] from the board members of the Open Source Initiative. If you skim through it, you're sure to find some idea or concept that you feel very aligned with. At its heart, openness (and open source) is about free distribution—putting your work out there for others to use.

It's really about helping others and giving back.

When we started to think about open source and how we could implement it at Buffer, the fit seemed not only natural, but crucial to how we operate. In fact, it seemed that in a lot of ways we'd be doing ourselves a disservice if we didn't start to look more seriously at it.

But what I didn't quite realize at the time were all the effects that open source would have on me.

Open source has positively impacted me as a developer, as an employee at Buffer, and even as a person. Those are the things I'd love to share with you here—to show you how we stumbled upon open source at Buffer.

---

30 https://opensource.org/osd-annotated

Acting on your values

At Buffer, we're known just as much for the way we operate as much as our product. We believe that making your values and culture wildly transparent gives you an extra sense of responsibility to act on them. As someone who works at Buffer, I often wonder how I can be a good steward of what we're all about. How can I promote our ideas, failures, successes and experience in a way that helps other people?

As a company, we value transparency and put a premium on it. We think it helps us operate, and we hope that other people can look at our data and derive real, lasting value from it. That's why you can find all of our salaries[31] in a public Google Docs spreadsheet, open up a Trello board and see our product roadmap, or even go to a realtime dashboard showing all of our revenues.

After thinking about this one day, I came to realize that I wasn't fully taking advantage of perhaps the biggest opportunity Buffer was affording me to give back: our own code. I spend hours every day writing it, testing it, and thinking about it to make sure the work I do solves real problems for people, and generally makes their life easier or better.

So why wasn't I sharing it?

From the top down

I think values like this tend to flow from the top of organizations. Sharing the code you write daily for a company might be difficult if that company didn't feel the same way about the code! To that end, our CEO, Joel Gascoigne, seemed to sense this opportunity, and was also passionate about it. I remember

31 https://opensource.com/open-organization/16/3/social-startup-buffer-transparency-reigns

reading an email thread he started, where he raised some strong points in favor of using open source at Buffer.

Here is some of what Joel had mentioned:

> "I'd love to share something that's been on my mind for several years at Buffer. As you all know, one of our values is to Show Gratitude. Since the very beginning, we've been super fortunate to be building Buffer in a time where open source is a big part of the world of software development.
>
> There's no way that we'd be as big as we are today without open source. In fact, we probably wouldn't even be here at all. The internet is very much built on the generosity of those who lead and contribute to open source. We are quite literally standing on the shoulders of giants, and in many ways, what we've done ourselves is minute in comparison to the incredible technologies we're lucky enough to rely on and make use of.
>
> I believe that contributing more towards open source as a company is a key part of our future, and almost a duty we have. With our value of transparency, I think it's something people likely expect and should expect from us."

Once I read that, I felt reaffirmed. Getting involved with the open source community felt exactly like the right thing to do for Buffer.

Buffer's CTO, Sunil Sadasivan, is also a passionate open source champion. Sunil has the best "big picture" of engineering

at Buffer, and was quick to help us get open source initiatives moving at Buffer.

Recognizing the power of open source, Sunil helped us facilitate many important things—from a Slack channel specifically for open discussions, to an open calendar for suggestions, and a habit of leaving comments on our open source documents. Sunil was on board and helping us push forward.

When the CTO takes time to provide a larger vision for open thinking in a company, developers like me can more easily act on it. It's a symbiotic relationship, and it takes several of us to execute on the vision we have for open source. And seeing our leadership promoting our open source efforts really was amazing.

Committing to open source was a gut check for all of us. We knew we could be doing better here! Our values tend to promote personal growth, gratitude, and openness. By the same token, the open source community also advocates a lot of the same ideas.

It felt like a perfect fit for our workplace and culture.

Personal growth

At that point, I started to think about how I could help. With so much code and opportunity, I realized the challenge really lied within finding the right things to share. I came to the realization that, first and foremost, open source code should help someone. So what is most helpful?

We could, of course, open source the entirety of Buffer. That would certainly hold true to our values, but it also may not be the most beneficial move for the community. It seemed like the right choice to get started with the open source movement at Buffer would be to release some focused and individualized components.

As an iOS developer at Buffer, I'm most familiar with our iOS codebase. It's what I know best, so I started there. Around this time, I'd been working on a modular component to view images easily within our app. It was easy to use and solved a real problem that developers on the platform often face. It felt like the perfect place to start.

Eventually it was. But first, I experienced an open source reality check: This was code that I wrote, and I didn't write it thinking that the world would one day examine it. Impostor syndrome and doubt quickly crept in.[32] I started asking myself:

- What if this isn't any good?
- What if there are some mistakes?
- What if people think I didn't write the best parts (it was based on an existing open source project)
- What if I missed important shortcuts, like using the right APIs?

In only a matter of hours, I experienced some important growth as an engineer. And that growth stemmed directly from two things:

- Working at a company who believed in us to share our code, and that it was the right thing to do
- Open sourcing that code to the world

Sometimes, developing with only your team is easiest. They know you, and they are likely quite familiar with your coding tendencies. There's much comfort there (as well there should be). Contrast that with coding for potentially thousands of people, and your mental state can quickly change from comfort to doubt.

I think this experience is an important one for software engineers to encounter. It made me realize that I had an incredi-

32 See Laura Hilliger's chapter in this volume.

ble learning opportunity in front of me. As the old adage goes: "Nobody bats a thousand." There were bound to be mistakes or rough edges, and that was completely okay. So, I shared the code.[33]

Showing gratitude

That experience directly correlates with the second benefit I derived from open thinking: gratitude. When I posted the open source project I previously mentioned, community reception was very positive. Other developers mentioned some tweaks, made some edits to our README file—and most of all, they were just thankful we released it!

This was such an important reminder of how much developing is a community driven task. No single developer has all the answers. There are experts, but I've constantly seen those experts point to the fact that the community helped them get where they are.

Open source helps other developers work and accomplish great things, but inherently it's also an act of knowledge transfer. I remember when Apple made Swift open source. It was an exciting day for me. I was elated to look through Apple's code and learn from the industry experts on the language. I picked things up that I may not have otherwise, and learned a lot of what best practices were.

In short, I was very grateful for that!

Beginning a journey

With open source at Buffer, we are very much in our infancy. We're still asking some questions to help put us on the right path, like "What is the most helpful code to open source?"

33 https://github.com/bufferapp/buffer-ios-image-viewer

"How do we tell people about it?" and "How do we develop with an open source mindset?"

Throughout the process, though, we've constantly been reminded that the internet is actually a very sharing and generous place. As Joel said, we are only where we are today at Buffer because of the brilliant code of other developers who were kind enough to share their hard work with the world. And what an amazing bar they've set.

All I can think about is how I want us to be like that. We want to learn from those people who are doing it so much better, and we'll strive to hit that high bar. We want to give back and help solve problems, too. We want to save other people time. We want to share all of our work in the open.

That's what lead us to open source, and it's already had an incredible impact on the way we think about work and culture. I'm excited to see where it takes us next.

Jordan Morgan is an iOS developer at Buffer. He is from Ozark and also founded Dreaming In Binary. He is focused on helping the community, creating things that inspire others, doing talks over iOS, and constantly being a student of any form of software engineering.

Chapter discussion and review

• Jordan writes that values like transparency "tend to flow from the top of the organization." Do you agree or disagree? What values do you see flowing from your organization's leadership?

• Jordan describes how he one day "came to realize that I wasn't fully taking advantage of perhaps the biggest opportunity Buffer was affording me to give back: our own code." Are you or your teammates missing an opportunity to share valuable knowledge and resources with each other—or with the rest of the organization? How can you begin doing that?

• Sharing allowed Jordan's team "to learn from those people who are doing it so much better, and we'll strive to hit that high bar." What do you think you could learn from others if they decided to share with you and your team? What's the best way to facilitate that kind of sharing?

A user's guide to failing faster

Gordon Haff

Failure. Now that's a word with a negative vibe. Among engineering and construction projects, it conjures images of the Titanic sinking, the Tacoma Narrows bridge twisting in the wind, or the space shuttle Challenger exploding. These were all failures of engineering design or management.

Most failures in the pure software realm don't lead to the same visceral imagery as those above, but they can have widespread financial and human costs all the same. Think of the failed Healthcare.gov launch, the Target data breach, or really any number of multi-million dollar projects that basically didn't work in the end. In 2012, the US Air Force scrapped an ERP project[34] after racking up $1 *billion* in costs.

In cases like these, playing the blame game is customary. Even when most of those involved don't literally go down with the ship—as in the case of the Titanic—people get fired, careers get curtailed, and the internet has a field day with both the individuals and the organizations.

But how do we square that with the frequent demand to embrace failure in your DevOps culture?[35] If we should embrace failure, how can we punish it?

---

34 http://www.computerworld.com/article/2493041/it-careers/air-force-scraps-massive-erp-project-after-racking-up--1b-in-costs.html

35 https://www.veracode.com/blog/secure-development/why-you-should-embrace-failure-your-development-culture

Failing well

Not all failure is created equal. Understanding different types of failure and structuring the environment and processes to minimize the bad kinds is the key to success. The key is to "fail well," as Megan McArdle writes in *The Up Side of Down*.

In that book, McArdle describes the Marshmallow Challenge, an experiment Peter Skillman, the former VP of design at Palm, originally concocted. In this challenge, groups receive 20 sticks of spaghetti, one yard of tape, one yard of string, and one marshmallow. Their objective is to build a structure that gets the marshmallow off the ground, as high as possible.

Skillman conducted his experiment with all sorts of participants from business school students to engineers to kindergarteners. The business school students did worst. I'm a former business school student, and this does not surprise me. According to Skillman, they spent too much time arguing about who was going to be the CEO of Spaghetti, Inc. The engineers did well, but also did not come out on top. As someone who also has an engineering degree and has participated in similar exercises, I suspect that *they* spent too much time arguing over the optimal structural design approach to take.

By contrast, the kindergartners didn't sit around talking about the problem. They just started building to determine what works and what doesn't. And they did the best.

Setting up a system and environment that allows and encourages such experiments enables successful failure in agile software development. It doesn't mean that no one is accountable for failures. In fact, it makes accountability easier because "being accountable" needn't equate to "having caused some disaster." In this respect, it changes the nature of accountability.[36]

36 See Stephen Gold's chapter in this volume.

Designing for accountability

We should consider five principles when we think about such a system: scope, approach, workflow, incentives, and culture.

Scope. The right scope is about constraining the impact of failure and stopping the cascading of additional failures. This is central to encouraging experimentation because it minimizes the effect of a failure (and if you don't have failures, then you're not experimenting.) In general, you want to decouple activities and decisions from each other. From a DevOps perspective, this means making deployments incremental, frequent, and routine events—in part by deploying small, autonomous, and bounded context services (i.e. microservices or similar patterns).

Approach. The right approach is about continuously experimenting, iterating, and improving. This is the philosophy that DevOps and Agile development[37] bring from the Toyota Production System's kaizen (continuous improvement), and other manufacturing antecedents. The most effective processes have continuous communication—think scrums and kanban—and allow for collaboration that can identify failures before they happen. At the same time, when failures *do* occur, the process allows for feedback to continuously improve and cultivate ongoing learning.

Workflow. The right workflow repeatedly automates for consistency and thereby reduces the number of failures attributable to inevitable casual mistakes like a mistyped command. This allows for a greater focus on design errors and other systematic causes of failure. In DevOps, much of this takes the form of a Continuous Integration/Continuous Delivery (CI/CD) work-

37 See Jen Krieger's and Hina Popal's co-authored chapter in this volume.

flow that uses monitoring, feedback loops, and automated test suites to catch failures as early in the process as possible.

Incentives. The right incentives align rewards and behavior with desirable outcomes. Incentives (such as advancement, money, recognition) need to reward trust, cooperation, and innovation.[38] The key is that individuals have control over their own success. This is probably a good place to point out that failure is not always a positive outcome. Especially when failure is the result of repeatedly not following established processes and design rules, actions still have consequences.

Culture. The right culture is, at least in part, about building organizations and systems that allow for failing well—and thereby make accountability within that framework a positive attribute rather than part of a blame game. This requires transparency. It also requires an understanding that even good decisions can have bad outcomes. A technology doesn't develop as expected. The market shifts. An architectural approach turns out not to scale. Stuff happens. Innovation is inherently risky. Cut your losses and move on, avoiding the sunk cost fallacy.[39]

Properly dealing with accountability and failure in agile IT does require appropriate architectures, tools, and processes. Low-impact experimentation on a fragile, monolithic application will be difficult. Avoiding costly failures and subsequent blame will be difficult. However, the culture of an organization still plays an outsized role. Legendary management consultant Peter Drucker once famously said that "Culture eats strategy for breakfast." Culture has a similar appetite for many aspects of the software development process.

---

38 See Jim Whitehurst's chapter in this volume.

39 https://pdfs.semanticscholar.org/e456/4b88ca2349962a707b76be4c75076ad6bd43.pdf

Gordon Haff is Red Hat's cloud evangelist. He is a frequent and highly acclaimed speaker at customer and industry events, and helps develop strategy across Red Hat's full portfolio of cloud solutions. He is the author of Computing Next: How the Cloud Opens the Future*, in addition to numerous other publications.*

Chapter discussion and review

• What's the relationship between failure and accountability?

• Do you agree with Gordon when he asserts that one can "fail well"? What does "failing well" involve? What does it look like?

• Gordon lists five elements that teams should consider when building robust systems of accountability: scope, approach, workflow, incentives, and culture. How does your team currently balance these factors in its approach to accountability? Would you add anything to the list?

Changing the way we think of change

Matt Micene

Think about the last time you tried to change a personal habit. You likely hit a point where you needed to alter the way you think and make the habit less a part of your identity. This is difficult—and you're only trying to change *your own* ways of thinking.

So you may have tried to put yourself in new situations. New situations can actually help us create *new* habits, which in turn lead to new ways of thinking.

That's the thing about successful change: It's as much about *outlook* as *outcome*. You need to know *why* you're changing and *where* you're headed (not just how you're going to do it), because change for its own sake is often short-lived and short-sighted.

Now think about the changes your IT organization needs to make. Perhaps you're thinking about adopting something like DevOps. This thing we call "DevOps" has three components: people, process, and tools. People and process are the basis for *any* organization. Adopting DevOps, therefore, requires making fundamental changes to the core of most organizations—not just learning new tools.

And like any change, it can be short-sighted. If you're focused on the change as a point solution—"Get a better tool to do alerting," for example—you'll likely come up with a narrow vision of the problem. This mode of thinking may furnish a tool

with more bells and whistles and a better way of handling on-call rotations. But it can't fix the fact that alerts aren't going to the right team, or that those failures remain failures since no one actually knows how to fix the service.

The new tool (or at least the idea of a new tool) creates a moment to have a conversation about the underlying issues that plague your team's views on monitoring. The new tool allows you to make bigger changes—changes to your beliefs and practices—which, as the foundation of your organization, are even more important.

Creating deeper change requires new approaches to the notion of change altogether. And to discover those approaches, we need to better understand the drive for change in the first place.

Clearing the fences

> "In the matter of reforming things, as distinct from deforming them, there is one plain and simple principle; a principle which will probably be called a paradox. There exists in such a case a certain institution or law; let us say, for the sake of simplicity, a fence or gate erected across a road. The more modern type of reformer goes gaily up to it and says, "I don't see the use of this; let us clear it away." To which the more intelligent type of reformer will do well to answer: "If you don't see the use of it, I certainly won't let you clear it away. Go away and think. Then, when you can come back and tell me that you do see the use of it, I may allow you to destroy it."—G.K Chesterton, 1929

To understand the need for DevOps, which tries to recombine the traditionally "split" entities of "development" and "operations," we must first understand how the split came about. Once we "know the use of it," then we can see the split for what it really is—and dismantle it if necessary.

Today we have no single theory of management, but we can trace the origins of most modern management theory to Frederick Winslow Taylor. Taylor was a mechanical engineer who created a system for measuring the efficiency of workers at a steel mill. Taylor believed he could apply scientific analysis to the laborers in the mill, not only to improve individual tasks, but also to prove that there was a discoverable best method for performing *any* task.

We can easily draw a historical tree with Taylor at the root. From Taylor's early efforts in the late 1880s emerged the time-motion study and other quality-improvement programs that span the 1920s all the way to today, where we see Six Sigma, Lean, and the like. Top-down, directive-style management, coupled with a methodical approach to studying process, dominates mainstream business culture today. It's primarily focused on efficiency as the primary measure of worker success.

If Taylor is the root of our historical tree, then our next major fork in the trunk would be Alfred P. Sloan of General Motors in the 1920s. The structure Sloan created at GM would not only hold strong there until the 2000s, but also prove to be the major model of the corporation for much of the next 50 years.

In 1920, GM was experiencing a crisis of management—or rather a crisis from the lack thereof. Sloan wrote his "Organizational Study" for the board, proposing a new structure for the multitudes of GM divisions. This new structure centered on the concept of "decentralized operations with centralized control." The individual divisions, associated now with brands like

Chevrolet, Cadillac, and Buick, would operate independently while providing central management the means to drive strategy and control finances.

Under Sloan's recommendations (and later guidance as CEO), GM rose to a dominant position in the US auto industry. Sloan's plan created a highly successful corporation from one on the brink of disaster. From the central view, the autonomous units are black boxes. Incentives and goals get set at the top levels, and the teams at the bottom drive to deliver.

The Taylorian idea of "best practices"—standard, interchangeable, and repeatable behaviors—still holds a place in today's management ideals, where it gets coupled with the hierarchical model of the Sloan corporate structure, which advocates rigid departmental splits and silos for maximum control.

We can point to several management studies that demonstrate this. But business culture isn't created and propagated through reading books alone. Organizational culture is the product of real people in actual situations performing concrete behaviors that propel cultural norms through time. That's how things like Taylorism and Sloanianism get solidified and come to appear immovable.

Technology sector funding is a case in point. Here's how the cycle works: Investors only invest in those companies they believe could achieve their particular view of success. This model for success doesn't necessarily originate from the company itself (and its particular goals); it comes from a board's ideas of what a successful company should look like. Many investors come from companies that have survived the trials and tribulations of running a business, and as a result they have different blueprints for what makes a successful company. They fund companies that can be taught to mimic their models for

success. So companies wishing to acquire funding learn to mimic. In this way, the start-up incubator is a direct way of reproducing a supposedly ideal structure and culture.

The "Dev" and "Ops" split is not the result of personality, diverging skills, or a magic hat placed on the heads of new employees; it's a byproduct of Taylorism and Sloanianism. Clear and impermeable boundaries between responsibilities and personnel is a management function coupled with a focus on worker efficiency. The management split could have easily landed on product or project boundaries instead of skills, but the history of business management theory through today tells us that skills-based grouping is the "best" way to be efficient.

Unfortunately, those boundaries create tensions, and those tensions are a direct result of opposing goals set by different management chains with different objectives. For example:

Agility ⟷ Stability

Drawing new users ⟷ Existing users' experience

Application getting features ⟷ Application available to use

Beating the competition ⟷ Protecting revenue

Fixing problems that come up ⟷ Preventing problems before they happen

Today, we can see growing recognition among organizations' top leaders that the existing business culture (and by extension the set of tensions it produces) is a serious problem. In a 2016 Gartner report, 57 percent of respondents said that culture change was one of the major challenges to the business through 2020. The rise of new methods like Agile and DevOps as a means of affecting organizational changes reflects that recognition. The rise of "shadow IT" is the flip side of the coin; recent

estimates peg nearly 30 percent of IT spend outside the control of the IT organization.[40]

These are only some of the "culture concerns" that business are having. The need to change is clear, but the path ahead is still governed by the decisions of yesterday.

Resistance isn't futile

> "Bert Lance believes he can save Uncle Sam billions if he can get the government to adopt a simple motto: 'If it ain't broke, don't fix it.' He explains: 'That's the trouble with government: Fixing things that aren't broken and not fixing things that are broken.'" — Nation's Business, May 1977

Typically, change is an organizational response to something gone wrong. In this sense, then, if tension (even adversity) is the normal catalyst for change, then the *resistance* to change is an indicator of success. But overemphasis on successful paths can make organizations inflexible, hidebound, and dogmatic. Valuing policy navigation over effective results is a symptom of this growing rigidity.

Success in traditional IT departments has thickened the walls of the IT silo. Other departments are now "customers," not co-workers. Attempts to shift IT *away* from being a cost-center create a new operating model that disconnects IT from the rest of the business' goals. This in turn creates resistance that limits agility, increases friction, and decreases responsiveness. Collaboration gets shelved in favor of "expert direction." The result is an isolationist view of IT can only do more harm than good.

40 https://thenewstack.io/parity-check-dont-afraid-shadow-yet/

And yet as "software eats the world," IT becomes more and more central to the overall success of the organization. Forward-thinking IT organizations recognize this and are already making deliberate changes to their playbooks, rather than treating change as something to fear.

For instance, Facebook consulted with anthropologist Robin Dunbar[41] on its approach to social groups, but realized the impact this had on internal groups (not just external users of the site) as the company grew. Zappos' culture has garnered so much praise that the organization created a department focused on training others in their views on core values and corporate culture. And of course, this book is a companion to *The Open Organization*, a book that shows how open principles applied to management—transparency, participation, and community—can reinvent the organization for our fast-paced, connected era.

Resolving to change

> "If the rate of change on the outside exceeds the rate of change on the inside, the end is near."—Jack Welch, 2004

A colleague once told me he could explain DevOps to a project manager using only the vocabulary of the Information Technology Infrastructure Library framework.[42]

While these frameworks *appear* to be opposed, they actually both center on risk and change management. They simply present different processes and tools for such management. This point is important to note when to talking about DevOps outside

41 http://www.npr.org/2017/01/13/509358157/is-there-a-limit-to-how-many-friends-we-can-have

42 https://en.wikipedia.org/wiki/ITIL

IT. Instead of emphasizing process breakdowns and failures, show how smaller changes introduce *less* risk, and so on. This is a powerful way to highlight the benefits changing a team's culture: Focusing on the *new* capabilities instead of the *old* problems is an effective agent for change, especially when you adopt someone else's frame of reference.

Change isn't just about *rebuilding* the organization; it's also about new ways to cross historically uncrossable gaps—resolving those tensions I mapped earlier by refusing to position things like "agility" and "stability" as mutually exclusive forces. Setting up cross-silo teams focused on *outcomes* over *functions* is one of the strategies in play. Bringing different teams, each of whose work relies on the others, together around a single project or goal is one of the most common approaches. Eliminating friction between these teams and improving communications yields massive improvements—even while holding onto the iron silo structures of management (silos don't need to be demolished if they can be mastered). In these cases, *resistance* to change isn't an indicator of success; an embrace of change is.

These aren't "best practices." They're simply a way for you to examine your own fences. Every organization has unique fences created by the people within it. And once you "know the use of it," you can decide whether it needs dismantling or mastering.

Matt Micene works at Red Hat, evangelizing Red Hat Enterprise Linux. He has more than 10 years of experience in information technology, where he's worked on Solaris and Linux architecture and system design as well as data center design. He's also spent many long, coffee-filled nights performing system maintenance for various web-based service companies.

Chapter discussion and review

• What is Matt's argument about the relationship between "culture" and "tools" in organizations?

• What kinds of "received wisdom" or "legacy thinking" guide your team's or organization's approaches to work today? Are these still beneficial? Should you change or reframe them? How might you begin doing this?

• How would you say your organization currently feels about the issue of "change"?

• Matt demonstrates what he feels is the value of "refusing to position things like 'agility' and 'stability' as mutually exclusive forces." Are you seeing unproductive dichotomies at work on your team or in your organization? How can you work to resist or undo them?

Five laws every aspiring DevOps engineer should know

Chris Short

"A good engineer is a lazy engineer," some may say. And to a certain extent, it's true: Laziness is a great quality if you're automating repetitive tasks. But laziness flies in the face of learning new technologies and getting new work done. Somewhere between Junior Systems Administrator and Senior DevOps Engineer, laziness no longer becomes an advantage.

Let's discuss the five laws aspiring DevOps engineers should follow if they want to become *great* DevOps engineers.

1. Forget "I don't know"

The first thing great engineers should do is to banish the phrase, "I don't know" from their vocabularies. The impression that phrase makes is the verbal equivalent of throwing your hands up in defeat (before you ever start). Banishing the phrase is difficult. Saying, "I'll have to do some research," or "I know someone that might be able to point me in the right direction," sounds much better. The point is: If you're discussing something as a possible task, chances are that you'll end up doing it. The fact you or anyone else in the room does not know anything about it is irrelevant.

Treat every task as an opportunity to learn. Dedicate the time necessary to become the resident expert in the task at

hand. Prove to yourself that you can teach an old dog new tricks. Prove to your peers that you can enable the team to go further. Seek out new knowledge and improve upon the things you build and maintain. Do not be afraid to dive headlong into something you know nothing about. Your thirst for knowledge should be unquenchable. You might not know it today but you can know it tomorrow.

2. Read the documentation

Documentation is everywhere, and solutions to complex problems are at our fingertips. Make an effort to *not* ask your peers how something works without reading its documentation first.

Your peers spent time writing that documentation for a reason.

Time is life's most precious resource, so don't waste others' time. If you have questions *after* reading the documentation, then feel free to ask. The same goes for man pages. Developers spent time creating those documents, and OS vendors put the tooling in place for you to install and read them. The more effort spent on something, the more important reading becomes.

In the absence of documentation, read the code. It's bound to contain comments or notes on decisions that affected how it works. At the very least, make sure you understand the contents of the code repository. If the repo is not following the methodology of a 12-factor app[43] make sure you understand where it falls short. When you end up asking questions, make sure you do so in a positive manner. Being positive is sometimes difficult to do. As an outsider you are missing the context that

43 https://12factor.net/

lead to the decision. Never forget that iterative improvement is the modus operandi.

3. Search before asking

How many times have you read how to do something—and then needed to ask how to do it? The answer is likely "zero." You probably have team members who can answer most of your questions. On the rare occasion that you must consult your manager, make sure you have at least searched for possible answers. Someone once told me, "Don't bring me problems; present solutions to me." It's quite a simple statement that has such a deep meaning in DevOps. If you are discussing a problem, you'll likely play a part in its solution. Instead of going to your leadership with a problem, present the problem and your solution to it.

The solution to your problem will not fall out of the sky. Solving new problems requires searching for new answers. We live in an amazing time. A vast majority of human intellect is available to us with a few keystrokes. If you are turning to leadership without at least searching for an answer, you have failed them.

You are in your role to do work that your leadership has determined they need someone other than themselves to solve. The least you can do is self-manage solutions to problems.

4. Anything is possible. Never say never. Trust but verify.

Too often I sense that team members feel something isn't possible. The beautiful thing about working in DevOps is that physics is the only limit in your environment. You can't send more electrons over connections than what's *physically* possible. You can't store more blocks on a hard disk than what's *physically* possible. You're also limited by time, with business

deadlines being the most common limiters. This means you have an amazing capacity to do anything to which you apply your effort. Anything is possible in this space with proper time, coordination, and effort. You and your team members should remind yourselves of that on a regular basis.

When it comes to complex, distributed systems (or even simple scripts) you should never assume anything. Remember, anything is possible. This means great solutions can end up in production as well as poor ones. Almost every place I have worked has had a team that has made an assumption about the way their systems work. There are various reasons why these assumptions exist, but the fact that no one has ever performed a deep dive to ensure the systems work the way they *assume* they do should be perplexing. You should always trust your teammates. Yet if something feels weird or doesn't work as expected, you need to verify whether the assumption is actually true.

5. Acknowledge technical debt

Technical debt is the result of decisions that made sense at the time someone made them. Those decisions are likely causing issues *now* because they no longer make sense. What got the product out the door a year ago under a tight deadline is likely going to hinder you from doing the same thing this year. If you are on a DevOps team, you are either helping to eliminate technical debt or you are pushing it to production. Often times you have to be the voice of reason in the planning sessions, the one willing to say why something won't work long term. This can make you an outcast if you are not careful. Treat these moments as opportunities to teach others around you something new. Do not act surprised people don't understand why what they are discussing will add complexity later on. It is your job, not theirs, to understand the complexity of the systems and stacks you are

supporting. Put your foot down if you have to. Keep in mind that if you are having to put your foot down, then chances are you need to align yourself closer to the beginning of the project's feedback loops.[44]

You exist because of technical debt. Whether and how that debt exists *after* your time on the project is up to you.

Conclusion

An unquenchable thirst for fundamental systems knowledge is *necessary* for success in DevOps. Great DevOps engineers constantly seek answers to questions and solutions to problems.[45] To become one of them, make preventing future technical debts a constant focus of your work.

Never stop learning. Laziness just won't get you there.

Chris Short has more than two decades of experience in various IT disciplines, from textile manufacturing to dial-up ISPs to Senior DevOps Engineer. He's been a staunch advocate for open source solutions throughout his time in the private and public sector. He's a partially disabled US Air Force Veteran living with his wife and son in NC. Read more of his writing at chrisshort-.net and devopsish.com.

44 See Jimmy Sjölund's chapter in this volume.

45 See Chrissy Linzy's chapter in this volume

Chapter discussion and review

• Chris cites an old adage that "a good engineer is a lazy engineer." What does this mean? Was it ever true? Is it true today? Why or why not?

• Chris suggests that "you are in your role to do work that your leadership has determined they need someone other than themselves to solve. The least you can do is self-manage solutions to problems." Do you feel like you're able to do this for your team? What opportunities or resources would make self-management easier for you?

• "Technical debt is the result of decisions that made sense at the time someone made them," Chris says. "But those decisions are likely causing issues now because they no longer make sense." Do you struggle with forms of technical debt? What are they? If you could eliminate them today, what would be the result? How would your work change?

Why you should build a team of boundary spanners

DeLisa Alexander

The traditional proprietary software world limits developers' ability to collaborate with others outside their own companies. But developers in the open source software world collaborate beyond the walls of the company. And that collaboration isn't limited to software development; it also extends to collaborating in multiple ways with customers and partners.

We can learn a lot from this kind of open collaboration, and it's rapidly becoming an essential capability for IT teams and organizations.

Creating a culture that nurtures collaboration—both inside and outside of various functions, as well as outside of the corporate walls—is a difficult task. But when we prime our organizational cultures for collaboration, I've noticed an interesting side effect: People tend to more willingly step outside their comfort zones, span boundaries, and take on new responsibilities. And sometimes they find that becoming a "collaborator" or "boundary spanner" can result in unexpected career opportunities.

I'll offer my story as an example.

A brave new world

I joined Red Hat as the second lawyer on a team of two. A few years in, my then-manager and Red Hat's former general counsel, Mark Webbink (open source licensing guru) approached me about participating in an internal leadership development program called "Brave New World."

In the Brave New World program, associates from around the globe and from different functions within the organization came together to select and work on a strategic problem facing the company. The program created an opportunity not just to collaborate with others in different parts of the organization, but also to contribute beyond your own day to day role.

In my case, it was a huge turning point. No one was asking me to use my lawyer skills or to be a consultant to the business. Instead, they wanted me to contribute outside of my normal skill set and comfort zone and to be a member of a larger team.

My Brave New World team took on the challenge of trying to create a culture of recognition at Red Hat. During the project, I started learning about "HR stuff"—compensation, engagement, recognition, rewards, etc. I developed new capabilities and relationships, and we ended up developing a spot recognition program called the Reward Zone that Red Hat associates around the world still use (more than a decade and a few evolutions later).

Why do I tell this story? Because it was an experience in collaboration and boundary spanning that pushed me beyond my core skills, helped me to develop a broader perspective of the needs of other functions, and made me better understand the points of view of others. It also gave me the introduction to an entirely new field. In fact, I attribute to Brave New World my eventual move from Legal to HR.

Building an IT culture of boundary spanners

You don't need a leadership development program like Brave New World to inspire boundary spanning. When your organization promotes a culture of collaboration, this sort of career transformation quits being a happy accident and instead becomes part of the master organizational plan. You'll also see increased trust and respect between departments.

That's why, over the years, we've applied a similar cross-functional, action learning project model to solve many other types of challenges at Red Hat, including a redesign of our performance management process and system. We've also brought together our IT and Engineering teams to tackle a number of technical challenges.

Let's explore three ways that your team can get started with building a culture of boundary spanners.

1. Start small, and build on the partnerships you already have

As an IT team, your customers likely include a number of different teams or departments. Consider what options you might have to deepen those relationships and inspire associates from your own team and your customers' teams to span boundaries and collaborate.

If your project teams currently include only IT representatives, extend an invitation to someone from each of your customers' or partners' teams to join the project meetings. Give them clear roles and responsibilities within the project.

If your project teams are already cross-functional, consider trying an "embedded team" or "associate exchange" model for your next project, where members of the project team sit together for a defined period of time (typically a few months). This

can build stronger relationships between teams, and spark new collaboration opportunities.

2. Facilitate cross-functional mentorships

Another way to encourage boundary spanning is to reach out to one of your peers in another department, and find potential mentors and mentees to pair up between your teams.

The cross-functional aspect of these relationships is particularly powerful when the mentors' and mentees' roles and responsibilities are different, as the conversations go beyond technical topics, and instead focus on career development.

This experience can encourage both teams to think more broadly and cross-functionally, as well as spark ideas and connections for boundary-spanning projects and collaboration opportunities.

3. Offer to collaborate with other departments to solve a challenging IT problem

In many organizations, IT teams shy away from tackling challenging business problems that span multiple departments or where decision-making authority will be split between leaders. But these kinds of projects, if approached with a genuine desire to help and a clear commitment to transparency and inclusiveness, can be some of the most culturally transformative experiences for your team.

When you come together with the shared purpose of solving a tough challenge, your team members gain new insights into how the organization beyond IT works. By working through a problem together, and taking the time to align everyone on clear roles and responsibilities, each team member learns how to span boundaries, navigate conflict, and work together to deliver valuable solutions.

Let the open source way take you beyond your comfort zone

My experience at Red Hat has been that these investments in deliberate boundary spanning pay big returns in the long run. I've personally benefited from learning how to step out of my comfort zone and span boundaries. And as the pace of technology projects continually speeds, it's a capability that will greatly benefit IT organizations.

DeLisa Alexander is Executive Vice President and Chief People Officer at Red Hat.

Chapter discussion and review

• What opportunities for "boundary spanning" do you see in your current team or organization?

• If you could shadow someone on your team or in your organization for a day, who would you choose? Why? What would be the most valuable insight you'd achieve by doing this?

• DeLisa suggests developing an "'embedded team' or 'associate exchange' model for your next project, where members of the project team sit together for a defined period of time (typically a few months)." Is this possible in your organization? What do you think its effects might be?

• Does your organization facilitate cross-platform mentorships? Could it? What might be the value of doing (or not doing) this?

A new approach to operations

Chrissy Linzy

IT operations teams handle much of the heavy lifting for the company's infrastructure. This means deploying new hardware, patching existing hardware, securing the network from threats, and handling daily issue resolution. Nearly every project across an IT organization needs an operations team whose members can get new services deployed and ensure they're deployed in a way that can scale and be maintained easily. These are the teams that help automate deployments for new applications or updates to existing applications, and they handle the monitoring for all of these applications, databases, and the physical hardware, too.

As more applications enter the infrastructure, operations teams are expected to become experts in all levels of support and application management. In recent years, the role of the jack-of-all-trades IT professional began transitioning into something more specialized. Gone are the days when one team can support everything from email to financial applications. The inner workings of *each* of these tools require in-depth understanding, and a dedicated team must ensure that the applications are being implemented and maintained properly.

This need for specialization has driven many IT organizations to begin focusing on a DevOps model, or a model that incorporates both developers and infrastructure with operations team members in one area. This model allows these team mem-

bers to better understand the environment from end to end, while allowing for a more collaborative workplace, one with more opportunities for everyone.

Traditional approaches to operations are changing. Today, it's much more likely for IT professionals to have a specific area of focus, whether that be database design, monitoring and metrics design, or the development of specialized applications to solve business problems. Operations teams are also beginning to specialize, supporting the shift to a DevOps model. This model changes the nature of several team dynamics—including team composition, knowledge-sharing practices, and automation. This chapter explains those changes.

The full stack team

The industry has now come up with a few ways to redefine the work that teams are doing, like the current shift to so-called "full stack engineers." Unfortunately, most definitions of a "full stack engineer" are not realistic. A developer who can clarify requirements from customers, develop code, design the infrastructure for the application, design a simple and intuitive user interface for the application, and complete all quality testing on the application is a rare creature, to say the least. When you add operational support for the application and the infrastructure to the mix, the full stack engineer fallacy becomes even more apparent.

Expecting someone to be proficient in all of these areas does not prepare this employee to be successful. Having a full stack *team*, however, gets everyone one step closer to a DevOps world. By sharing responsibilities across a team capable of supporting multiple applications and infrastructures, developers can release more quickly and can own their deployments, end to end. The missing piece in most development teams is (histori-

cally speaking) the support, or operations, work. Keeping the infrastructure for the application stable is a critical element for these teams. Siloed operations teams struggle to understand the interactions of multiple applications, especially when those operations engineers are not included in conversations around the purpose of the applications being released.

Before considering how DevOps can help resolve some of these problems that IT teams face, consider the responsibility of operations. IT departments are currently at a crossroads: Are the infrastructure teams now responsible for the successful deployment and maintenance of applications running in the cloud? How can these teams migrate more applications to a hosted environment, and how do these teams decide what can be moved safely? Whose responsibility is this? More importantly, as applications migrate to these shared environments, how are application development teams handling the support and scaleability of the environments while keeping an eye on the true cost of their applications? These sorts of infrastructure requirements have historically been handled by the operations teams. Today, these teams must work closely to allow development to happen more quickly.

Sharing knowledge

A DevOps mindset can also resolve some of the common knowledge-sharing challenges operations teams face today. Spreading knowledge across a specialized team that consists of developers and operations engineers can help the team work collaboratively and deliver results quicker.[46] This makes managing new technology that is just coming to the market easier to adopt.

46 See Chris Short's chapter in this volume.

If a developer is already familiar with the functions of a specific application, including the ways it integrates with the underlying infrastructure, then sharing this knowledge can help a team pivot quicker. A full-stack team of developers and operations engineers can work together to find potential pitfalls with the implementation of these new tools. If these teams are not working together, finding issues and identifying the proper deployment process becomes more difficult.

At the other end of the application lifecycle is the task of retiring outdated technology. All too often, the organizational focus is on deploying new tools to solve pressing business problems, and IT departments end up with a mix of old and new technologies to support. Without a team focusing on the entire application lifecycle, operations teams can get bogged down in the ongoing support of too many applications. By using the DevOps model, teams have more control over their offerings, and they can ensure that there are no applications in need of retirement. Having an operations engineer on the team helps to identify any potential gaps when migrating to a new tool, and it helps streamline the entire lifecycle process.

By allowing the operations teams to be part of the application development lifecycle, teams are in a better position to understand the impact of neglecting to retire applications in a timely manner. Operations teams are familiar with the issues involved in supporting applications that have fallen out of favor. All too often, these applications are still part of a business process, but no developer remains to provide updates or the final feature needed in the new application that would allow full retirement of the outdated application. When the developers and operations teams work together to support applications with similar functions, ensuring that all features are included in a replacement application (while managing the workload of the

entire DevOps team) becomes much easier. When these teams are not working together, this kind of ongoing support can become someone else's problem.

Doing more with less

IT operations teams have always worked to find the balance in maintaining existing infrastructure while keeping up with improvements in the industry. They're continually working to learn new technologies while updating and maintaining legacy environments. These environments are still a large part of most companies' infrastructures, so operations teams hear a common refrain: "We'll figure it out—we just need to do more with less." Teams dig deep, find a way to keep the lights on, and continue making progress on new projects. IT professionals are some of the most resourceful problem-solvers in the world, and operations teams are at the top of the list.

This may mean a focus on automation, streamlining supported tools and services, or asking engineers to perform multiple tasks. By allowing teams to have end-to-end support of their applications and services, IT departments can now consolidate applications with similar functions into smaller teams. And by then standardizing tools across these teams, developers and operations engineers can function as a team to help meet development deadlines or deployment projects.

As DevOps teams consider how to manage application lifecycles, these are some of the more common problems they must solve. Focusing on standardization means that the infrastructure teams will not need to learn how to configure an environment for multiple test environments or code repositories, for example. As the number of environments to support continues to grow faster than the size of the team, automating monitoring and deployment processes free up operations engineers to focus on

persistent issues in the infrastructure, on retiring applications that are no longer needed, or on ensuring that the environment can scale to meet the needs of the development teams as new items are deployed. By having operations team members working directly with developers, the operations engineers get to be one step closer to the business needs. Understanding the issues that business partners are facing gives the engineer valuable information about the direction of the infrastructure.

As developers and operations teams move to align with the DevOps model for working together, fostering accountability for a particular application across the entire team becomes important. Developers should strive to understand how their application interacts with the infrastructure, just as the operations engineer should be working to understand the nuances of the application and the business problems that it will solve. Without this common understanding, these historically separate teams won't truly become one unit, and the DevOps model will struggle. Commitment to this education brings everyone closer to creating the full stack team.

Chrissy Linzy is the Manager of the IT Application Lifecycle Management team at Red Hat, the world's leading provider of open source solutions. Chrissy has more than 20 years of experience in IT, including managing operations, software delivery, and application development teams. Her favorite part of working in IT is partnering with the business to solve problems and streamline processes using technology.

Chapter discussion and review

• What is a "full stack team," in your opinion? Is your team a full stack team? What would it need if it were to become one?

• What coordinated knowledge-sharing practices does your team employ? How does your team store and share knowledge it gathers from all points in an application's lifecycle? Are your teams current methods adequate?

• Chrissy suggests that "focusing on standardization means that the infrastructure teams will not need to learn how to configure an environment for multiple test environments or code repositories." What are your team's methods of standardization? Is anything *not* standardized that *should* be standardized? How does your team strike a balance between standardization and innovation?

How new communication technologies are affecting peer-to-peer engagement

Ron McFarland

Both *The Open Organization* and *The Open Organization Field Guide*[47] discuss ways new communication technologies are changing the nature of both work and management. I've seen these changes firsthand during my nearly three decades working for Japanese corporations. Over time, I've been able to classify and characterize some of the impacts these technologies —particularly new telecommunication technologies and social media—are having on daily life in many organizations. More specifically, they're effecting the way peer-to-peer decision-making practices function in organizations today.

Four approaches to communication technology

In Japan, I see companies that heavily promote today's communication technologies, as well as some that avoid them. Imagine four types of companies currently making use of today's communication technologies as they compete with other firms. These technologies are key, because they influence the environment in which certain peer-to-peer communities must work, and this, in turn, affects members' enthusiasm, desire, and engagement—so *investment* and *utilization* are critical considerations.

---

47 https://opensource.com/open-organization/resources

In fact, we can actually chart the four types of technology-adopters according to those two variables: investment and utilization.

Some companies are underinvested in new communication technologies, considering their needs and the relatively lower costs of these technologies today. And they're not using to capacity what they *do* have. I call these companies communication technology "slow movers" (low investment/low utilization). Others buy whatever is available at any cost, but don't fully put to use what they've purchased. I call these communication technology "fashion followers" (high investment/low utilization). Still other companies invest in the very minimum amount of communication technology, but what they do have they use to full capacity. I call these communication technology "conservative investors" (low investment/high utilization). Lastly, there are some companies that invest heavily in communication technology and work very hard to put it to full use. I call these communication technology "communication superstars" (high investment/high utilization).

These "communication superstars" have the ideal environment for peer-to-peer, front-line discussions and decision-making. But in Japan, particularly among smaller companies, I'd say more than 70 percent are "slow movers" or "conservative investors." If companies would pay more attention to investing in communication technology, and simultaneously increase their efforts at training staff to use the technology at its full potential, then peer-to-peer, front-line employees could explode with creativity. These technologies affect four aspects of information today: volume, speed, quality, and distribution.

Increased capacity for decision-making (volume)

In "communication superstar" environments, communication technologies can actually increase in the amount of information that can be made available quickly. Gone are the days in which only researchers or professors have access to in-depth information. Now, front-line people can obtain volumes of information if they know what they are looking for. With more and greater in-depth information in communication superstar company environments, front-line people working there can have more educated discussions and can make the type of decisions that only top management (supported by consultants and researchers) could have made in the past.

Faster pace of decision-making and execution (speed)

New technologies in these "communication superstar" companies are leading to quicker information acquisition, feedback, and flow between front-line members in the organizations, even if they are very widely disbursed. Using the metaphor of adjusting the temperature of water coming out of a faucet, I would describe the effect this way: If you move the handle but the temperature changes slowly, then finding the temperature you want becomes difficult, because the pace of change is very slow, and differences between settings are difficult to determine. But if you move the handle and water temperature change is more immediate, you'll find that getting the correct temperature is much easier; you're moving quicker and making more rapid adjustments.

The same logic applies to peer-to-peer discussions and feedback. I have a five-minute-to-twenty-four-hour goal when replying to my worldwide customers. That means that if I receive

an email from a customer (something that arrives on my desktop computer at home, my desktop computer in the office, or on my mobile phone), I like to reply within five minutes. This really surprises customers, as they're probably still sitting in front of their computer! In the worst case, I try to reply within 24 hours. This gives me a competitive advantage when attempting to get customers to work with me. Front-line, peer-to-peer communities in these "communication superstar" companies can have that same competitive advantage in making quality decisions and executing them faster. The capacity for speedier replies allows us to make more adjustments quicker. It keeps both employees and customers involved, motivated and engaged. Information arriving too slowly can cause people to "turn off" and direct their attention elsewhere. This weakens the passion, dedication and engagement of the project.

Toward wiser decisions (quality)

Information travels more quickly when the business communication channels are adequate. It's also subjected to more scrutiny. People can share second opinions and gather additional empirical data using these technologies. Furthermore, new communication technologies allow employees and managers to deliver data in new ways. With my years in sales training around the world, I've learned that using multiple visual aids, infographics, and so forth have greatly enhanced communication when English language barriers could have impeded it. All this can lead to high levels of peer-to-peer, front-line engagement, as up-to-date status reports can be quickly distributed and easily understood, making everyone more responsive.

Maximal reach (distribution)

Not long ago, teammates had to be physically close to one another and know each other well in order to communicate successfully. That's no longer the case, as people literally all over the world can develop communication channels. Good communication is the outcome of developing a trusting relationship. For me, building trust with people I've never met face-to-face has taken a bit longer, but I've done it with today's technology.

Let me explain. Good communication starts with an initial contact, whether meeting someone in person or virtually (via social media or some tele-communication format). Over some period of time and through several exchanges, a relationship starts to develop, and a level of trust is reached. People evaluate one another's character and integrity, and they also judge each other's competencies and skills. With this deepening of trust over time, greater communication can evolve. At that point, open and in-depth discussions on very difficult, complex, and sometime uncomfortable topics can take place. With the ability to communicate at that level, peer-to-peer discussions and decisions can be made. With today's communication technology, groups with widely disbursed members can participate in greater information exchange. I currently have approximately 20 customers around the world. Some I have never met in person; most I have just met in person once. Being stationed in Japan can make regular get-togethers with Europeans and Americans rather difficult. Fortunately, with today's communication technology, I can find solutions for many problems without physically getting together, as I have built a trusting relationship with them.

Concluding comments

With all the benefits of this "communication superstar" working environment, in open organizations that promote peer-to-peer discussions, decision-making and management, I recommend the other three groups to move in that direction. The "slow movers" more than likely have managerial barriers to open information exchange. They should be convinced of the benefits of a more opened organization and the value of greater information exchange. If they don't improve their communication environment, they may lose their competitive advantage. The "fashion followers" should more carefully study their communication needs and time their investments with their in-company training capacities. The "conservative investors" should study their communication bottlenecks and find the technologies that are available to eliminate them. That's the path to super-stardom.

Ron McFarland has been working in Japan for 40 years, and he's spent more than 30 of them in international sales, sales management training, and expanding sales worldwide. For the last 14 years, Ron has established distributors in the United States and throughout Europe for a Tokyo-headquartered, Japanese hardware cutting tool manufacturer. He's worked in or been to more than 80 countries.

Chapter discussion and review

• Do you think your organization adequately fosters front-line, peer-to-peer activity and engagement? If not, what could it do to better cultivate this?

• Ron says that effective decisions consider four factors: volume, speed, quality, and distribution. Can you think of others? Would you revise this list in light of the decisions you and your teammates must make regularly?

• How would you describe your team's approach to peer-to-peer communication technologies? Does it utilize them sufficiently? Too little? Too much?

What engineers and marketers can learn from each other

Jackie Yeaney

After many years of practicing marketing in the business-to-business tech world, I think I've heard just about every misconception that engineers seem to have about marketers. Here are some of the more common:

- "Marketing is a waste of money that we should be putting into actual product development."
- "Those marketers just throw stuff against the wall and hope it sticks. Where's the discipline?"
- "Does anyone actually read this stuff?"
- "The best thing a marketer can tell me is how to unsubscribe, unfollow, and unfriend."

And here's my personal favorite:

"*Marketing is all fluff.*"

That last one is simply incorrect—but more than that, it's actually a major impediment to innovation in our organizations today.

Let me explain why.

Seeing my own reflection

I think these comments from engineers bother me so much because I see a bit of my former self in them.

You see, I was once as geeky as they come—and was proud of it. I hold a Bachelor's in electrical engineering from Rensselaer Polytechnic Institute, and began my professional career as an officer in the US Air Force during Desert Storm. There, I was in charge of developing and deploying a near real-time intelligence system that correlated several sources of data to create a picture of the battlefield.

After I left the Air Force, I planned to pursue a doctorate from MIT. But my Colonel convinced me to take a look at their business school. "Are you really going to be in a lab?" he asked me. "Are you going to teach at a university? Jackie, you are gifted at orchestrating complex activities. I think you really need to look at MIT Sloan."

So I took his advice, believing I could still enroll in a few tech courses at MIT. Taking a marketing course, however, would certainly have been a step too far—a total waste of time. I continued to bring my analytical skills to bear on any problem put in front of me.

Soon after, I became a management consultant at The Boston Consulting Group. Throughout my six years there, I consistently heard the same feedback: "Jackie, you're not visionary enough. You're not thinking outside the box. You assume your analysis is going to point you to the answer."

And of course, I agreed with them—because that's the way the world works, isn't it? What I realize now (and wish I'd discovered out far earlier) is that by taking this approach I was missing something pivotal: the open mind, the art, the emotion—the human and creative elements.

All this became much more apparent when I joined Delta Air Lines soon after September 11, 2001, and was asked to help lead consumer marketing. Marketing *definitely* wasn't my thing, but I was willing to help however they needed me to.

But suddenly, my rulebook for achieving familiar results was turned upside down. Thousands of people (both inside and outside the airline) were involved in this problem. Emotions were running high. I was facing problems that required different kinds of solutions, answers I couldn't reach simply by crunching numbers.

That's when I learned—and quickly, because we had much work to do if we were going to pull Delta back up to where it deserved to be—that marketing can be as much a strategic, problem-oriented and user-centered function as engineering is, even if these two camps don't immediately recognize it.

Two cultures

That "great divide" between engineering and marketing is deep indeed—so entrenched that it resembles what C.P. Snow once called the "two cultures" problem.[48] Scientifically minded engineers and artistically minded marketers tend to speak different languages, and they're acculturated to believe they value divergent things.

But the fact is that they're more similar than they might think. Research[49] from the University of Washington (co-sponsored by Microsoft, Google, and the National Science Foundation) identified "what makes a great software engineer," and (not surprisingly) the list of characteristics sounds like it could apply to great marketers, too. For example, the authors list traits like:

- Passion
- Open-mindedness

48 https://en.wikipedia.org/wiki/The_Two_Cultures#Implications_and_influence

49 https://faculty.washington.edu/ajko/papers/Li2015GreatEngineers.pdf

- Curiosity
- Cultivation of craft
- Ability to handle complexity

And these are just a few! Of course, not every characteristic on the list applies to marketers—but the Venn diagram connecting these "two cultures" is tighter than I believe most of us think. *Both* are striving to solve complex user and/or customer challenges. They just take a different approach to doing it.

Reading this list got me thinking: *What if these two personalities understood each other just a little bit more? Would there be power in that?*

You bet. I saw it firsthand when I was Executive Vice President of Marketing and Strategy for Red Hat, where I was surrounded by people I'd have quickly dismissed as "crazy creatives" during my early days. And I'd be willing to bet that a marketer has (at one time or another) looked at an engineer and thought, "Look at this data nerd. Can't see the forest beyond the trees."

I now understand the power of having both perspectives in the same room. And in reality, engineers and marketers are *both* working at the *intersection of customers, creativity, and analytics*. And if they could just learn to recognize the ways their personalities compliment each other, we could see tremendously positive results—results far more surprising and innovative than we'd see if we kept them isolated from one another.

Listening to the crazies (and the nerds)

Case in point: *The Open Organization*.

During my time at Red Hat, I spent much of my day thinking about how to extend and amplify our brand—but never in a

million years would I have thought to do it by asking our CEO to write a book. That idea came from a cross-functional team of those "crazy creatives," a group of people I rely on to help me imagine new and innovative solutions to branding challenges.

When I heard the idea, I recognized it right away as a quintessentially Red Hat approach to our work: something that would be valuable to a community of practitioners, and something that helps spread the message of openness just a little farther. By prioritizing these two goals above all others, we'd reinforce Red Hat's position as a positive force in the open source world, a trusted expert ready to help customers navigate the turbulence of digital disruption.

Here's the clincher: *That's exactly the same spirit guiding Red Hat engineers tackling problems of code.* The group of Red Hatters urging me to help make *The Open Organization* a reality demonstrated one of the very same motivations as the programmers that make up our internal and external communities: a desire to share.

In the end, bringing *The Open Organization* to life required help from across the spectrum of skills—both the intensely analytic and the beautifully artistic. Everyone pitched in. The project only cemented my belief that engineers and marketers are more alike than different.

But it also reinforced something else: The realization that openness shows no bias, no preference for a culture of engineering or a culture of marketing. The idea of a more open world can inspire them both equally, and the passion it ignites ripples across the artificial boundaries we draw around our groups.

That hardly sounds like fluff to me.

Jackie Yeaney is Chief Marketing Officer at Ellucian.

Chapter discussion and review

• How would you describe the relationship between the engineers and the marketers in your organization? Are you satisfied with that relationship? Would you change it? If so, how?

• When Jackie reflected on her experience in an open organization, she found that "in reality, engineers and marketers are both working at the intersection of customers, creativity, and analytics," and striving to address the same issues and problems. Have you ever discovered that a team you thought was different from yours was actually more like you than you realized? When did this happen? And what was the result of your realization?

• Jackie writes that "the engineers" and "the marketers" often appeared to embody separate cultures she assumed made them unable to effectively collaborate. Do you see similar teams in your organization with similar, stark cultural differences? How can you begin to recognize and address those differences?

Part 2: Practices

Introduction to Part 2

Jason Yee

In Part 2, we'll share ways you can convert ideas into behaviors—principles into practices.

Culture isn't only a set of ideas; it's a set of ideas *that guides collective action*. The key word there is "collective"; culture is only as effective as the *group* of people enacting it. So as someone attempting to catalyze cultural change, you face a challenge: convincing others to follow you in this transformation.

It can seem intimidating at the outset. You may even be wondering if you're the right person to lead this change. But as Chris Short writes in Part 1 of this book, "If you are discussing a problem, you'll likely play a part in its solution." I'd like to extend that: If you're considering an open culture, then you'll have to play a part in that transformation.

I've seen cultural transformation ignite at every level of an organization, from upper management leading inspiring top-down changes, to motivated engineers initiating grassroots movements. I've even seen middle managers fight—and win—two-front battles, transforming both the teams under their direction and the leadership to which they report.

No matter where you are in the hierarchy of an organization, *you can affect change*.

And I believe that others in your organization want you to succeed. Creating an open culture is, as Allison Matlack writes,

about "doing the right thing." It's about valuing the business over the processes. When we follow Jonas Rosland's advice to "assume positive intent," then we have to assume that others also want to do the "right thing" and want the benefits that an open culture brings to the organization. They just may not realize it!

Building an open culture and convincing teammates to join your effort require the same skills. As you read Rebecca Fernandez's chapter on productive debate, consider how you can apply that practice to both general business decisions and also to potential critics of cultural change. Similarly consider how you can make cultural decisions publicly as you would the technical decisions that Chad Whitacre addresses. Also keep in mind Lauri Apple's lessons from the Socratic method and ask if you're setting up your own "invisible walls and imaginary authority figures" who will stop you from trying to implement change.

Matt Thompson leads Part 2 of this book by writing about agile heartbeats, but he mentions the word "ritual" with respect to regularly occurring practices. I love that word. Rituals draw people together. They turn groups into communities and provide opportunities for celebrating personal growth. Consider, for example, the way harvest rituals transform farming (the tedious work of gathering crops) into a celebration of community and sharing, or the way coming-of-age rituals strengthen cultural communities by inspiring younger members as they celebrate the growth of older members. As coworkers and teammates join you in your cultural transformation, reflect on the rituals you create—either purposefully or unintentionally—to welcome them and inspire them to spread change.

Cultural transformation is a process. Work openly and continue to iterate. And as Jimmy Sjölund writes, "Little by little, you and the organization around you will improve."

Jason Yee is a technical writer and evangelist at Datadog, where he works to inspire developers and ops engineers with the power of metrics and helps write the technical documentation to enable them to harness that power. He's also a co-organizer of DevOpsDays Portland.

How to strengthen your agile heartbeat with powerful retrospectives

Matt Thompson

If you work in an open organization for any length of time, you're likely to hear someone mention "sprints" or "heartbeats" at some point. Understanding these terms is simple: Take a big goal, then break it into small pieces that help you get there.

The practice derives from Agile development and its various (funnily-named) flavors like "Scrum" and "Kanban," but the underlying logic is simple. Break big jobs into small time-bound sprints, then design a process and ritual for unpacking:

- what you accomplished in the last sprint,
- what you learned from it, and
- what you're going to tackle in the next one.

From 'sprints' to 'heartbeats,' finding a healthy cadence

Many of us have found that replacing the word "sprint" with "heartbeat" is helpful for explaining the value to new colleagues. It implies a steady, *healthy* cadence or rhythm—as opposed to endless "sprinting" or panting against a series of arbitrary deadlines.

Heartbeats can create a great sense of purpose, and ebb and flow in your team. They can be set to any length—a week,

two weeks, a month. It's really just about bringing people together in a regular, predictable cycle, with a ritual and set of dance steps to ensure everyone's on the same page, headed in the right direction, and learning and accomplishing important things together (as opposed to the "make it up as you go along" / gazillion emails and meetings / Bataan Death March of Multi-Tasking that gobbles up most projects by default.)

Too busy to think

A big reason people love working in heartbeats is that it makes work more *mindful*. It invites or even *forces* regular moments to step back, reflect, adjust your goals, and share real insights with colleagues that don't typically fit in the hurly burly of email and status updates. That process is generally called a "retrospective"—and lately I'm finding it to be the single most valuable part of the process. Done right, it can help you work smarter instead of harder.

But: Retrospectives are also often the most neglected or easily skipped over part of the process—especially for time-starved teams already suffering from Too Many Meetings Syndrome. Here's why the retrospective is one meeting you don't want to skip.

A regular ritual for reflection

A retrospective at the end of each heartbeat helps you unpack what you've accomplished and learned together, and where you might want to improve together in the next cycle. They can be dead simple; at the end of your heartbeat, just ask each team member to share:

1. What went well in the last heartbeat?
2. What could have gone better?
3. What do we want to improve in the next heartbeat?

Good retrospectives generate surprises

I find that when I'm part of a great retrospective, I leave the meeting feeling *surprised*. I leave knowing something I didn't know going in. I've had my perspective shifted in some way—particularly around what our *priorities* should be in the next heartbeat, or a key learning someone shares that helps me spot a new opportunity. In particular, retrospectives can help to:

- **Re-prioritize** stuff that seemed urgent a week or month ago but doesn't anymore. That's great! Let's consciously de-prioritize or set it aside. By the same token: Little things that didn't *seem* important suddenly reveal themselves as highly leveraged in the week or month ahead, little keys or springboards that emerge out of the haystack.
- **Punt!** What can we push out to the next heartbeat, so that we can narrow our focus in this one? Retrospectives make you more conscious of *time* and the value of *phasing*. Not everything needs to be done all at once; it's liberating to push stuff out. If it's not on this train, it'll go on the next one.
- **Do less work!** Yes, I said it: Great retrospectives should help you do *less* work. Less work means faster, better work. Eliminate the clutter and distractions that grow like weeds around your team's feet; it's amazing how good that feels—and your teammates will love you for it.
- **Unpack learning.** You're learning great stuff as you go that you didn't know when you started the project. Retrospectives are a chance to share and write this stuff down. Without a regular ritual or invitation to do so, this usually slides to the bottom of everyone's to do list. But

these are valuable diamonds and nuggets you don't want to slip away.

- **Pull up.** Good retrospectives invite altitude adjustment. Go back to your original strategy/roadmap and remind yourself what you said was important: the stuff that actually *matters,* as opposed to just being "busy." How are we doing? How has our thinking changed? How do we re-connect our big picture goals to day-to-day tasks?
- **Re-energize.** Feel proud. Most of us walk around feeling guilty and stressed about how "behind" we are. Retrospectives remind the team that, no matter how imperfectly, you *really are* accomplishing and learning a lot together. You're not just hamsters on a treadmill.
- **Continuously improve.** Get better at getting better. Small improvements add up to powerful change over time, like compound interest. You don't have to move mountains; just feel the trust and momentum that builds after your team makes an agreement and actually sticks to it.

Bland retrospectives become boring status updates

On the flip side, *bad* retrospectives or heartbeat meetings start to feel like a waste of time. They become rote, and more like status updates, as opposed to really stepping back and doing some fresh thinking together. This becomes a vicious cycle; there's less and less value, so people start to question their purpose. Eventually someone says: "Should we just cancel these? We have too many meetings already."

Some common pitfalls:

- **Not enough time.** Everyone hates meetings, so it's easy to make heartbeat meetings too short to do real retrospectives. Or to just skip the retrospective piece al-

together. But, this should be the *one hour* every week or month you invest to save *dozens* or *hundreds* of misspent hours going forward!

- **Not enough trust.** People are afraid to say what they really think in front of colleagues or leaders. Or it becomes a defensive exercise to prove that everyone is "busy." Busy is the new bored.
- **Bad or no strategy.** When the strategy is bad or the goals are unclear, retrospectives can just end up exposing that fact over and over again. In a healthy project, that's good! It surfaces something you can fix. In an unhealthy one, it just repeatedly pokes the elephant in the room.
- **Agile without agile.** Every organization says it wants to be "agile" nowadays, but most don't mean it. You can't "do agile" without retrospectives, or some ritual for reprioritizing. It's like doing archery without the arrows.
- **Hopeless over-capacity.** Many organizations have no shared view of the work they've committed to doing. Consequently, they're hopelessly over-committed. They're drowning in work they'll never really get done, and have no meaningful way to prioritize. Working in heartbeats and doing real retrospectives can help, but only if they start to whittle down workloads. Otherwise, they just remind everyone how screwed you all are—and that's not fun.

Matt Thompson is a 2017 Mozilla Fellow and an Agile trainer for non-profit organizations.

Chapter discussion and review

• Does your team perform regular "health checks" and "heartbeats"? If not, do you think your team would benefit from these? Why or why not? If so, what do you think are the most important benefits of conducting them?

• Matt writes that "a big reason people love working in heartbeats is that it makes work more *mindful*." Do you ever wish you and your team had more time to reflect on goals, lessons, and insights? What can you do to cultivate a culture of mindfulness on your team or in your organization?

• Have you ever experienced what Matt calls a "bad" retrospective? What made it ineffective? How can you help your team avoid bad retrospectives?

The benefits of tracking issues publicly

Chad Whitacre

A public issue tracker is a vital communication tool for an open organization, because there's no better way to be transparent and inclusive than to conduct your work in public channels. So let's explore some best practices for using an issue tracker in an open organization.

Before we start, though, let's define what we mean by "issue tracker." In simplest terms, an issue tracker is *a shared to-do list.* Think of scribbling a quick list of errands to run: buy bread, mail package, drop off library books, etc. As you drive around town, crossing each item off your list feels good. Now scale that up to the work you have to do in your organization, and add in a healthy dose of software-enabled collaboration. You've got an issue tracker!

Whether you use GitHub or another option, such as Bitbucket, GitLab, or Trello, an issue tracker is the right tool for the task of coordinating with your colleagues. It is also crucial for converting outsiders *into* colleagues, one of the hallmarks of an open organization. How does that work? I'm glad you asked!

Best practices for using an issue tracker

The following best practices for using a public issue tracker to convert outsiders into colleagues are based on our experience at Gratipay over the past five years. We help companies and others pay for open source, and we love collabo-

rating with our community using our issue trackers. Here's what we've learned.

0. Prioritize privacy. It may seem like an odd place to start, talking about *privacy* in a post about *public* issue trackers. But we must remember that openness is not an end in itself,[50] and that genuine and true openness is only ever built on a solid foundation of safety and consent. Never post information publicly that customers or other third parties have given you privately, unless you explicitly ask them and they explicitly agree to it. Adopt a policy and train your people. Here is Gratipay's policy for reference.[51] Okay! Now that we're clear on that, let's proceed.

1. Default to deciding in public. If you make decisions in private, you're losing out on several benefits of running an open organization, such as surfacing diverse ideas, recruiting motivated talent, and realizing greater accountability. Even if your full-time employees are the only ones using your public issue tracker at first, do it anyway. Avoid the temptation to treat your public issue tracker as a second-class citizen. If you have a conversation in the office, post a summary on the public issue tracker; give your community time to respond before finalizing the decision. This is the first step towards using your issue tracker to unlock the power of open for your organization. If it's not in the issue tracker, it didn't happen!

2. Cross-link to other tools. It's no secret that many of us love Internet Relay chat (IRC), Slack, or some other instant messaging technology. Or perhaps your organization already uses Trello, but you'd like to start using GitHub as well. No

50 https://opensource.com/open-organization/16/9/openness-means-to-what-end

51 http://inside.gratipay.com/howto/seek-consent

problem! It's easy to drop a link to a Trello card in a GitHub issue, and vice versa. Cross-linking ensures that an outsider who stumbles upon one or the other will be able to discover the additional context they need to fully understand an issue. For chat services, you may need to configure public logging in order to maintain the connection (privacy note: when you do so, be sure to advertise the fact in your channel description). That said, you should treat conversations in private Slack or other private channels just as if they were face-to-face conversations in the office. In other words, be sure to summarize the conversation on the public issue tracker. See above: Whether offline or online, if it's not in the issue tracker, it didn't happen!

3. Drive conversations to your tracker. Social media is great for getting lots of feedback quickly, and especially for discovering problems, but it's not the place to solve them. Issue trackers make room for deeper conversations and root-cause analysis. More importantly, they are optimized for getting stuff done rather than for infinite scrolling. Clicking that "Close" button when you've resolved an issue feels really good! Now that you have a public issue tracker as your primary venue for work, you can start inviting outsiders that engage with you on social media to pursue the conversation further in the tracker. Something as simple as, "Thanks for the feedback! Sounds similar to (link to public issue)?" can go a long way towards communicating to outsiders that your organization has nothing to hide, and welcomes their engagement.

4. Set up a "meta" tracker. Starting out, your issue tracker will be naturally focused on your *product*. When you're ready to take open to the next level, consider setting up an issue tracker about your *organization* itself. At Gratipay, we're willing to discuss just about any aspect of our organization, from our budget to our legal structure to our company name, in a public

issue tracker we call "Inside Gratipay." Yes, this can get a little chaotic at times—renaming the organization was a particularly fierce bikeshed!—but for us the benefits in terms of community engagement are worth it.

5. Use your meta tracker for onboarding. Once you have a meta issue tracker, a new onboarding process suggests itself. Invite potential colleagues to create their own onboarding ticket. If they've never used your particular issue tracker before, this will be a great chance for them to learn. Registering an account and filing an issue should be pretty easy (if it's not, consider switching tools!). This will create an early success event for your new colleague, as well as the beginnings of a sense of shared ownership and having a place within the organization. There are no dumb questions, of course, but this is *especially* true in people's onboarding tickets. This is your new colleagues' place to ask any and all questions as they familiarize themselves with how your organization works. Of course, you'll want to make sure that you respond quickly to their questions, to keep them engaged and help them integrate into your organization. This is also a great way to document the access permissions you end up granting this person. Crucially, this can start to happen before they're even hired.[52]

6. Radar projects. Most issue trackers include some way to organize and prioritize tasks. GitHub, for example, has milestones and projects. These are generally intended to align work priorities across members of your organization. At Gratipay, we've found it helpful to also use these tools to allow collaborators to own and organize their individual work priorities. We've found this to offer a different value than assigning issues to particular individuals (another facility issue trackers generally

52 https://opensource.com/open-organization/16/5/employees-let-them-hire-themselves

provide). I may care about an issue that someone else is actively working on, or I may be potentially interested in starting something but happy to let someone else claim it first. Having my own project space to organize my view of the organization's work is a powerful way to communicate with my colleagues about "what's on my radar."

7. Use bots to automate tasks. Eventually, you may find that certain tasks keep popping up again and again. That's a sign that automation can streamline your workflow. At Gratipay, we built a bot to help us with certain recurring tasks.[53] Admittedly, this is a somewhat advanced use case. If you reach this point, you will be far along in the process of using a public issue tracker to open up your organization!

Those are some of the practices we've found most helpful at Gratipay in using our issue tracker to "engage participative communities both inside and out," as Jim Whitehurst puts it in *The Open Organization*. But we're are always learning.

Chad Whitacre is the founder of Gratipay, an open organization with a mission to cultivate an economy of gratitude, generosity, and love. Gratipay offers pay-what-you-want payments and take-what-you-want payouts for open organizations.

53 https://github.com/gratipay/bot

Chapter discussion and review

• Does your team use a public issue tracker to make its goals and activities visible to the rest of the organization? To parties outside the organization? If not, what might change if you began tracking issues this way?

• Think about an item on your personal to-do list. How might making that item public impact the way you work on it?

• Are you able to "default to deciding in public," as Chad puts it? Why or why not? What barriers prevent you from doing this? Could you (or should you) overcome them?

• What's the relationship between transparency and accountability? How might your team's sense of accountability change if members adopted public issue trackers?

Three essential skills for fostering productive debate in your IT team

Rebecca Fernandez

Passionate debate fuels many open source communities and open organizations. Open and productive debate helps us refine and improve our ideas—and it ensures that everyone understands why a particular solution or idea is chosen.

Yet this kind of debate seems to be the exception rather than the rule among IT organizations. That's a shame, because open and candid conversations lead to better and more innovative solutions.

So let's take a look at three ways that you can foster productive debate within your IT team.

1. Lead by example

When you share an idea or a proposal, invite others' feedback. Ask them questions that invite productive dissent, and open your own mind to different views.

Examples of helpful questions include:

- If this idea didn't work out, what would be the most likely reason?
- If you were going to make one change to this idea, what would it be, and why?
- What challenges do you think we might run into, if we went this route?

- What are some things I'm not thinking about, but should be?

2. Resist the urge to immediately defend ideas

If your organization's culture isn't known for its tolerance of conflict, you will need to work hard to create a safe space for disagreement.

When someone is brave enough to criticize an idea, respond with curiosity and a desire to fully understand their perspective, rather than jumping to defend your own.

A good technique is to repeat their concerns using your own words, and then ask whether you've understood them correctly.[54] After you reach clarity on their perspective, respond respectfully to any points of disagreement.

Here again, you want to encourage continued dialog with a good follow-up prompt, such as: "So that's how I see it. But what are your thoughts?"

You might also need to mediate between more vocal and quiet team members. A good technique is to encourage the more vocal team member to express their ideas first, and to jot down a summary of their key points. Read that back to them, and ask for confirmation that you've captured it correctly.

Then turn to a quiet team member and say, "Ok, now I'd like to hear your thoughts. Which parts of that do you agree with, and where do you see things differently?"

If the more vocal team member interrupts, keep your focus on the quiet team member and say firmly, "Hold on, I want to hear the rest of what _____ has to say first. Please, continue."

54 See Lauri Apple's chapter in this volume.

3. Call people up, not out

In almost any passionate conversation—particularly in organizations where people are inexperienced with productive conflict—at some point, one person will step out of line and start to make things feel personal.

There's often a moment where things start to turn ugly, and you will be tempted to respond by "calling them out" on their poor behavior.

The key to returning the debate to a productive place is to respond as soon as you see this happening—and not to call them *out*, but to instead call them *up*.

Your goal is to model good debate and respond in a way that compels everyone to elevate their behavior, rather than escalate it. Typically, this means ignoring attempts to provoke an angry response. Instead, respond in the way that reminds everyone that you are all working together toward a shared purpose. Demonstrate by your response that you believe everyone in the conversation is a reasonable, rational, decent person.

Focus on the essence of what they've said—even if you have to dig and guess a little to figure out what that is—and be unfailingly polite and reasonable as you invite productive dialog.

You might say something like: "So, what I think I'm hearing is that you're really worried about this, and you're frustrated because it seems like nobody's listening. Or maybe you're concerned that we're missing the significance of it. Is that about right?"

Or perhaps: "It sounds like you've given this a lot of thought, and you're frustrated that we're asking what seem like obvious questions. Would you be willing to start at the beginning and walk us through the basics, so we all feel confident that we're understanding your proposal? We could hold our questions until the end, if that would help."

If the person is very upset, anticipate that it will take a few minutes of patient attempts to de-escalate before they can respond in a helpful way. In most cases, they will come around, and ultimately your team's trust and respect for each other will grow as a result.

Ultimately, you want to help your team make the connection between these productive debates and the better outcomes they drive. In your project retrospectives and your team meetings, point out how everyone's willingness to engage in uncomfortable conversations helped you deliver a great solution, and thank them for their contribution to that.

Rebecca Fernandez is a principal employment branding and communications specialist at Red Hat, as well as the maintainer of the Open Decision Framework.

Chapter discussion and review

• Rebecca argues that passionate debate "seems to be the exception rather than the rule among IT organizations." Do you agree or disagree? Is passionate debate common on your team or in your IT organization? Why or why not? Should (or could) that change?

• What strategies for engaging team members who might not always speak up in meetings have you found successful? What can you do to collect critical feedback from everyone in your organization?

• Rebecca advocates "calling people up" rather than "calling people out." What does this mean to you? What's the difference? Why is that difference important?

• "Ultimately," writes Rebecca, "you want to help your team make the connection between these productive debates and the better outcomes they drive." Do you think your team does this effectively? If not, what can you do to change that?

Mastering feedback loops

Jimmy Sjölund

In most situations, from getting clothing advice to seeking peer review of the next scientific discovery, we harness the help of people around us in order to discuss and analyze potential next steps. Hardly anyone thinks up a perfect solution right off the bat; it's an iterative process full of trials and errors, adjustments, and new experiments.

And it's a process we can always improve. This chapter offers some advice for doing just that.

What are feedback loops?

Feedback loops are supposed to be great and solve all sorts of problems. So what are they, exactly?

Remember when you were a child and you drew your first picture of a cat? You proudly showed it to your parents, and they suggested you put a tail on it. You went back to the drawing board (literally) and added the tail, showed them the result, and then they put the drawing up on the fridge.

That was an early feedback loop for you: A process that fed into itself, like the snake eating its own tail.

You might be familiar with another early feedback loop called the "Deming Cycle":

Plan – Do – Check – Act

W. Edwards Deming later updated this to:

Plan – Do – Study – *Act*

. . . which I agree is a better description.

Similar cycles or processes are The OODA Loop (which Jim Whitehurst discusses in *The Open* Organization), The Shewhart Cycle, Six Sigma (*Define – Measure – Analyze – Improve – Control*), and The Lean Startup (*Build – Measure – Learn*).

Common to all these is a scientific approach to working in an iterative mode: try something, learn from it, and adapt your work accordingly when moving forward. In other words:

> *Practice doesn't make perfect. Practice makes permanent. Feedback makes perfect.*

Why are feedback loops important?

Shorter feedback loops (that is, loops that take less time between the moment you try something and the moment you learn about its effects or outcomes) allow you to fix or improve work quickly and derive additional value faster. Performing a small fix, receiving feedback on it, and trying again should not be a tremendous burden—as it would be if you'd been working on something for a long time and find out you'll have start over. In other words, you might be running in the wrong direction, but if you receive feedback early you won't have such a long way to backtrack when starting again.

Various agile software development methods consider short feedback loops important for *being* agile and producing the *right result* in the shortest amount of time.

Teams can arrange and work through these feedback loops via techniques like "pair" or "mob programming," daily standup meetings, and sprints. When working via pair or mob

programming, for instance, developers experience feedback loops directly between the people working together on a task. In daily standup meetings, they get feedback from coworkers. After a sprint, they would preferably receive feedback from the customer. In all these cases, the feedback people receive helps them improve their work before beginning an additional step in a process.

This way of working is possible even outside of software development, but here I want to specifically focus on how it can enhance IT organizations. I'll cover a few general terms, which apply to all departments, teams, and manners of work.

How can we enhance feedback loops in IT organizations?

In IT organizations, being transparent is a prerequisite to giving and receiving feedback. People can be transparent about both their ongoing and their planned work.

For example, I've had positive experiences using kanban boards with operations teams. When the board is visible to all stakeholders, managers, and other teams, everyone receives feedback on the current status of work items and current priorities. People also have the opportunity to receive spontaneous feedback from someone looking at the board and noticing something that, for instance, another team might also be working on, or is not important anymore, or (even better) something very important that's missing from the board but should definitely be on it.

I recommend searching for feedback as early as possible—right from the start, if you can. I'll often outline an assignment and run that by a manager and key stakeholders. It's the best way to find out if I have understood the task properly and helps me set *their* expectations for what I'll deliver next.

This all sounds simple enough. So why is this often so hard to put into place? What are some of the most common blockers, and what can you do to improve or facilitate better feedback?

For feedback loops to work well, you need to have an open climate, one where people feel safe sharing their thoughts. Incidentally, paying attention to feedback loops can also help you improve the current climate in your organization. That's the beauty of a feedback loop. As it feeds into itself, if the climate and culture around you is not open by default, then you can change this by being more open yourself, offering feedback, making sure you get feedback from others, internalizing that feedback, and improving. Little by little, you and the organization around you will improve, too. As Mahatma Gandhi put it: *Be the change that you wish to see in the world*.

Remember: Keep your feedback loops short. You should seek feedback before investing too much time or money into something. Then changing things isn't so difficult, should you need to do it.

You'll also want to make sure you have opportunities to receive valuable feedback from people who usually are not comfortable sharing their thoughts openly. Often, people will solicit feedback at a demo or a meeting—meaning they do it *in person* and receive it when people *speak up*. That is perhaps not the best form of communication for everyone, and sticking to that single feedback environment might cause you to miss great insights you'd otherwise want to know about. One way to improve this could be to send out a post-meeting email to everyone involved, reminding them to send their feedback directly to you (preferably within a set time frame). Some people prefer to formulate their ideas in their own time and through a medium that

suits them better (rather than speaking up in front of everyone!).

The best kind of feedback you can receive is that which comes directly from an actual user or customer—but what do you do when you're working with infrastructure several layers *away* from the customers?

In the best of worlds, customer feedback would trickle down to all involved areas and teams, but we all know that this is difficult and usually doesn't happen in real life. Depending on your products or services, you could arrange workshops together with the customer and include people from all layers of the organization. I have done this with great results. Discussions and ideas that would never have popped up otherwise suddenly appear and action plans get put in place.

If you have regular meetings with your customers, you could invite people from other parts of the organization as guests every once in a while. In my experience, all parties have appreciated this kind of initiative.

In other organizations, one might consider the surrounding internal teams and departments to be customers and facilitate feedback loops with them. They could, in turn, get feedback from *their* customers. I recommend that you try to set this up in all teams, as you will need good feedback not only from your external customers but also from your partners and peers as well.

Last, but not least: Remember that feedback loops doesn't necessarily have to involve *human* interactions. For instance, you should receive valuable feedback from your monitoring systems and incident management tools. Automated feedback can make you aware of slower response times that might indicate an underlying problem with a new release, re-occuring minor incidents could be the result of soon-to-be faulty hardware that

would cause a major outage, and statistics showing a growing user base of your services should trigger a plan to scale up the environment in time before it suffers from performance issues.

Some feedback on feedback

In the end, a team's ability to feedback loops boils down to two factors: communication and transparency.

When you're open about your progress and willing to accept others' insights, you can more quickly adapt and create the best outcomes. For a long time, working in silos until you're ready to reveal your results has been the norm; however, that's changing as we discover the advantages of involving more people and ideas into the design and execution processes.

The world is changing more rapidly than it ever has, and adapting quickly has never been more crucial. Ignoring the feedback loops occurring all around you could cause your solution or idea to arrive too late—or to chase the wrong problem altogether. Feedback, on the other hand, makes perfect.

Jimmy Sjölund is a senior IT service manager and innovation coach at Telia Company. He's an open source evangelist working in organization development and exploring agile and lean workflows. He's also a visualization enthusiast.

Chapter discussion and review

• Does your team or organization effectively address the feedback loops at its disposal? Why or why not? Would a change to your team's culture have any effect on the way it approaches feedback?

• Jimmy suggests that "Practice doesn't make perfect. Practice makes permanent. Feedback makes perfect." What does this mean to you? Is it relevant to the work your team or organization is doing?

• What do you think is the most valuable form of feedback your team can receive while it's working on projects? Do you frequently receive this feedback? Why or why not? Can you refine your feedback processes or mechanisms in any way?

What to do when your open team has impostor syndrome

Laura Hilliger

Recently I facilitated a week of creative work with my colleagues on the Planet 4 project[55] at Greenpeace. One evening, when we came together in a closing circle after a day of intense creative work, I asked the participants to share how they were each feeling about the day. We allowed these reflections to manifest into conversation.

A concern surfaced: The task of creating a new ecosystem of sites for Greenpeace became, for a moment, completely overwhelming.

"Are we the right people to be doing this?" someone said.

That sentence hit me hard. It expressed the feeling that we shouldn't be in the position we're in.

It expressed a kind of impostor syndrome.

At Opensource.com, we've published several articles about impostor syndrome, which (as Nicole Engard notes in her piece,[56] according to the Caltech Counseling Center) "can be defined as a collection of feelings of inadequacy that persist even in face of information that indicates that the opposite is true."

---

55 http://wiki.greenpeace.org/Planet4

56 https://opensource.com/life/16/5/fruits-deeper-discussions-impostor-syndrome

In this particular case, however, multiple people were feeling impostor syndrome—not necessarily of themselves, but as part of their collective relationship to a project. This put a new spin on how open leaders might think about and ultimately address this phenomenon with their peers and teams.

How should we think about impostor syndrome when we aren't talking about how a single individual might be feeling inadequate? What if, instead of someone saying "I don't think I'm the right person for this team," an entire team is expressing, collectively, that "We are not the right people to be doing this"? What might make a group of people feel collectively inadequate?

Collective inadequacy?

Perhaps we can relate this collective inadequacy to a version of the Iron Triangle.[57]

In any project, there will be challenges that shift the sides of the Iron Triangle. If the project needs to be done ASAP (a *time* variable), the scope will need to stay small and resources will need to adequate. If the *scope* balloons, the project will take more time and require more resources. If *resources* (like people or money) are scarce, the timeline will lengthen and the scope may need to be reduced. This is how the Iron Triangle works.

Very rarely is a project perfectly scoped, perfectly timed, and perfectly resourced. The Iron Triangle isn't just a theoretical framework for running a project; it's also a mechanism for understanding how the project team might be *feeling*. The language we use to talk about the sides of this triangle are related to the emotional well-being of a team.

57 https://opensource.com/open-organization/17/2/new-perspective-meritocracy

Talking about expectations

If the expectations for (or "scope" of) the project feel unachievable, a group of people may begin to feel overwhelmed or inadequate. Often in the social justice realm, for example, we talk about "changing the world" and "mass mobilization" and "shifting cultures" or "changing the public perception." We tell people we are going to create a world where everyone is equal and diversity reigns supreme and openness will become the default setting. We write project manifestos and briefs that proclaim the ultimate mission of an organization, and we ascribe project goals to this mission with little regard for the contingencies a project will inevitably have. Our language indicates that we're intending to create a complicated, multifaceted, systemic shift.

These aren't SMART[58] goals; they're wild and unrealistic speculations. They overstate the impact a single project is going to have on a system. Of course, some projects can actually cause instant shock to a system—things that change the course of history immediately. But I can't think of one off the top of my head (even the lightbulb took time to become a pervasive technology). A variety of factors go into creating systemic change, and a single project isn't going have instantaneous effects on a larger system.

Even if the deliverables for a project are well defined, these lofty expressions can make it feel like the scope is bigger than it actually is. When developing briefs and project manifestos, or when talking to new hires or teammates about the project, try to think seriously about the words and phrases you're using. A description that misrepresents a project or product's scope might lead a future team to reel at its ambition.

58 https://www.projectsmart.co.uk/smart-goals.php

Resourcing language

Another place where ambition needs to match reality and feasibility is in the area of *resourcing*.

People and money are two basic types of resources. In both cases, the words we use to describe our resources might create feelings of inadequacy.

I'm definitely guilty of calling my peers and colleagues "rockstars," "brilliant," and "genius," and I'm not going to suggest that positive semantics should be left out of group discussions. However, we should consider constructions that may serve to reinforce unattainable expectations. Use consideration when speaking on behalf of a teammate or the team itself. Statements like "Oh, I'm sure we can do that; Amy is a rockstar" place an intimidating expectation on Amy that she hasn't had the opportunity to digest. Likewise, claims like "We'll pull an all-nighter" might create a tension point between the team and an individual who is unable to *pull* said all-nighter.

The same is true for how we talk about money. If you're holding a project's purse strings, and you actively remind your team that "a lot of money is on the table," then you're not helping. No one is striving to go over budget; it might be best to shield some or all of the team from budgetary concerns so that they can just do the work.

While a team might actively strive to create a positive working atmosphere through the language it uses to describe what it's doing, occasionally positive feedback from *outside* the core team is necessary for dispelling feelings of inadequacy that might be gathering. You can become a resource to your colleagues. When people feel like no one notices them working hard, feeling like what we do *matters* becomes more difficult. So be the person who spends an hour every week looking into another team's project, and tell that team something you like

about the work they're doing. Thank them for contributing to the greater good.

Talking time

Time is a human construct, and deadlines are mostly arbitrary. When leading a team, strive to be fluid and adaptable with your expectations for how long it takes to do anything, and use language that helps people feel adequate.

No team knows what events are going to throw the project off its timetable, and no one misses a deadline on purpose. Of course, we need to be accountable to one another and live up to our commitments, but we should be aware of how we talk about time in relationship to our projects.

Asking your team "When do you think you'll have that done by?" is a good example of a positive semantic relationship with time. This question is very different from "When will that be done?" or "We need this by date X." Asking your team to give an assumption of finishability instead of a commitment to it allows the team to see the deadline as slightly flexible, which it should be. This creates accountability, puts the power in the team's hands, yet doesn't create a potential pressure point. At the end of the day, every project can take an extra day.

There's also a difference between asking "Why isn't that done yet?" and "Is there something blocking you?" The lizard brain will interpret the first phrasing as threatening, as if you expected something long ago. When we're threatened, the brain tells our bodies to release chemicals and electrical signals that cause stress and tension. The second phrasing activates no such

threat because it uses neutralized language to ask about the status.[59]

Listen to yourself

We all know that what you say can directly impact someone's emotional well being. However, we don't often think about the nuances in the language we use at work. If a project team is showing signs of feeling collective inadequacy or Team Impostor Syndrome, listen to how project stakeholders are interacting. There might be some tiny semantic adjustments that can help the team know that they're the right people for the job.

Laura Hilliger is an artist, educator, writer and technologist. She's a multimedia designer and developer, a technical liaison, a project manager, and an Open Web hacktivist who is happiest in collaborative environments. She's an advocate for change and is currently working to help Greenpeace become a more open organization.

59 https://opensource.com/open-organization/16/8/managers-do-you-delegate-or-donate

Chapter discussion and review

• What is "impostor syndrome," and have you ever experienced it? Has your team?

• Laura cautions readers about using language that inadvertently perpetuates unreasonable expectations. Have you heard others using this kind of language to describe their projects or co-workers? Have you ever used it? How can you alter your language choices to make different kinds of impacts?

• Laura stresses the importance of fostering "a positive semantic relationship with time" in your team and organization. What does this mean? Do you think your team and organization have this kind of relationship to time?

When innovation trumps process

Allison Matlack

Traditionally, IT has been intensely process-oriented. I imagine that's because IT organizations are usually tasked with saving the world on a budget that's only big enough for them to keep the lights on and the water running. When your team bears that kind of weighty responsibility for organizational success or failure, following tried-and-true procedures is important. Doing things "the right way"—strictly according to defined processes—can be critical.

But as the industry trends toward DevOps—where IT is increasingly responsible for generating new business value, not just keeping those proverbial lights on—IT organizations need to reexamine their approaches to stakeholder and change management if they want to keep up. They must now balance d*oing things right* not only with *doing things fast* but also with *doing the right things*—whatever the "right things" may be as the organization moves forward.

So it's time for some serious reflection: Is your organization more focused on *doing things right* for the sake of process and consistency, *doing things fast* to meet arbitrary deadlines, or on *doing the right thing* for the customer? And what's the right balance of each of those things for you?

Defining your 'why'

At a recent conference where I gave a presentation about some of the positive effects of scaled, agile methodology on cross-team relationships and collaboration, someone asked me how to convince others to pay *more* attention to doing the right thing and *less* attention on the processes that defined how to do things right. He told me his team had upset others in the company by experimenting with aspects of agile methodology and starting to talk directly to their customers—that is, they released more frequently (*doing things fast*) and used feedback loops to iterate on the product so they could deliver what the customer wanted more quickly (*doing the right thing*). The rest of the company, he said, was more concerned about filling out forms than on delivering what the customer wanted when they wanted it (*doing things right*). The company had expressed no interest in changing that process, and leadership felt this team's speed was making the rest of the company look bad. The agile team, on the other hand, grew increasingly frustrated by processes that seemed to be in place for the sake of process.

I'm sure there's another side to this story, where someone has a good reason for every form and process. But this story made me think of the message Simon Sinek shares in his 2011 book, *Start With Why*: Before concentrating on *what* you do and *how* you do it, you should figure out *why* you're doing it in the first place. What's your goal? What's your purpose? The *how* and the *what* will fall into place as soon as you define your *why*.

In this example are two competing goals. The majority of the company is "doing things right" because that's the way they've always done it, and they'll achieve consistency, stability (less risk), and conservation of the top-down hierarchy. Becoming more agile and focusing on "doing the right thing"—even if it runs against those entrenched processes—might introduce risk

and shake up the status quo, but the results include improving development efficiency and delighting the client.

Which do you think will lead to increased business and customer loyalty? Which defines *why* the company is in business and inspires people? (Hint: It's not filling out forms.)

Leading by example

Once you define your *why*, figuring out *how* to do the right thing for your customers becomes easier. In our example here, the group focused on delighting the client has the right idea. Building brand loyalty and market mindshare are difficult if you're not willing to take some risks and be open to doing things differently. The world is moving too fast for us to spend months planning and years implementing; no one is going to wait for us. We have to iterate quickly and be prepared to change directions a few times along the way if we can hope to continually deliver what our customers want when they want it.

I would encourage the frustrated team in this scenario to document the business value they've seen as a result of implementing an agile methodology (increased revenue is a great motivator for everyone's boss' boss' boss). They can connect *what* they're doing to that overall reason *why* the company is in business. They could perform a retrospective to identify exactly which processes slowed them down or made delivering what the customer wanted more difficult, then work with other departments to find ways of streamlining the path to delivery. Or, if all that fails, they could try to reduce their dependencies on other teams to minimize the disruption.

Change can be uncomfortable. It can take a long time. But oftentimes, small teams like these can make the biggest impacts, leading by example. As the team learns more about why certain processes are in place and how to work more efficiently

with the procedures they have to follow, hopefully the rest of the organization will begin to see the value of focusing on doing the right thing rather than doing things right. And as the organization begins to identify which processes are necessary and which can be changed to leave room for more flexibility, both the customer and the organization win.

Allison Matlack has been a member of the Red Hat Customer Portal team since 2011. She's been an Open Organization Ambassador since 2016, helping others find ways to put open principles into practice.

Chapter discussion and review

• What's the difference between "doing things right now," "doing things right," and "doing the right things," as you see it? Does your team seem to favor any one of these three approaches to work?

• "Before concentrating on *what* you do and *how* you do it," Allison says, "you should figure out *why* you're doing it in the first place. What's your goal? What's your purpose?" Does your team "ask 'why'" before it begins collective work? Should it?

• Allison argues that "change can be uncomfortable. It can take a long time. But oftentimes, small teams like these can make the biggest impacts, leading by example." In what ways can your team act as an example to others? What do you hope others can learn from watching you work together? And what can you learn from other teams in your organization?

Better IT culture via the Socratic method

Lauri Apple

When it comes to "most valuable tools for untying mental knots and figuring things out," two items appear at the top of my list.

The first is this clip from Benny Hill about what happens when we make assumptions.[60] I saw it on (the opposite of a flat-screen) TV as a child years ago, and still reflect on it several times weekly. Its message—operating by assumptions is unlikely to end well, so don't—hasn't failed me yet.

The second is the Socratic method, which has helped me reinforce the open organization values of transparency, collaboration, and sharing in my work and related extracurricular activities[61]—making work more fun and rewarding as a result.

As Benny Hill might point out, reinforcing those open values when we're *making assumptions* rather than *operating on a foundation of facts and concrete information* can be difficult. While we might intend to be open, fiction will motivate our actions if we're operating on assumptions. The Socratic method helps us to create a foundation for being collaborative, challenging, and open—*truthfully*.

---

60 https://www.youtube.com/watch?v=R6jaKkE0RsI

61 https://opensource.com/open-organization/16/5/appreciating-full-power-open

What the Socratic method is

The University of Chicago Law School, where Barack Obama taught constitutional law until making a slight career change, describes the Socratic method as an inquiry practice based on "asking continual questions until a contradiction was exposed, thus proving the fallacy of the initial assumption."[62] A catchier description, offered by this quick how-to for using the method with children,[63] is "clarify, synthesize, restate."

Here's how it works: Typically, a "protagonist" presents some scenario or question for an audience (a group or individual) to consider. A series of ensuing questions then points toward gaining a clearer understanding of the issue at hand—its subtleties, potential angles, logic, and possibilities.

We have Greek philosopher Socrates to thank for this exercise. He left no known writings (talk about taking the *Agile Manifesto*'s call for "working software over comprehensive documentation" to an extreme!), but did impress his students enough so that they carried forth his legacy. (Eventually he was executed for "corrupting the young," so he clearly made an impression on authorities, too.) Today, you'll find the method at work in law school classrooms (it was a go-to tool for my professors), though usually not delivered in much of an "open organization" way. Luckily, those of us working in open or agile organizations can apply the Socratic method without sneers, jeers, and grade point average pressures.

62 http://www.law.uchicago.edu/prospectives/lifeofthemind/socraticmethod

63 https://www.youtube.com/watch?v=_CPLu3qCbSU

Why use Socrates' method

Even in agile and open organizations, we can become vulnerable to entrenched thinking that closes us off to new opportunities and ideas. Some members of our team will be louder than others and dominate decision-making. Some practices will become guidelines or standards by habit, even if they're not the best practices. Myths will emerge over time: "It's how we've always done it," "we need permission," "Rockstar Ninja Dev-man doesn't like it, so we shouldn't," or "I can't." In the race to finish projects, we might overlook our open values and rely too heavily on tools to get the job done. We're human, and we're fallible.

The Socratic method offers us a way to stay faithful to open organization values. As agile trainer Scott Duncan shared with the Scrum Master Toolbox podcast,[64] it emphasizes "asking people what they think about things rather than telling them what they ought to think." In this way, it keeps us from going into prescriptive mode. Instead, people and teams have the space and opportunity to draw conclusions by themselves, based on their own thoughts and motivations. They own the content of the conversation, while the person playing questioner/protagonist makes notes, listens carefully to what's being said, draws connections, and points out contradictions.

For team or project managers, the Socratic method provides an opportunity to act as a servant-leader; clarity of purpose is the goal. If you're striving for self-managed, autonomous teams, then this framework allows you to help others

64 http://scrum-master-toolbox.org/2016/07/podcast/scott-duncan-explains-how-to-use-the-socratic-method-to-enable-team-development/

examine their thought and work patterns to find answers they've derived themselves.

Collaboration—meetings, brainstorming sessions, retrospectives—becomes less emotional and much more driven by consensus based on facts.[65] Rockstar Ninja Dev-Man—who might be brilliant with code, but not much of a team player or objective thinker—will be humbled when faced with his own fallacies and contradictions. Socratic questioning might uncover that the least-experienced member of the team offers the most practical, well-reasoned solution, so it also levels the playing field.

Socratic techniques also increase transparency by knocking down myths, ghosts, and assumptions that separate teams from the "real issues," or prevent them from achieving goals. For example, if you work in a large and complex software development organization like I do, you might hear teams talk about being "blocked" by another team. Team A will complain that, before moving forward, Team B has to make some change to a repository; until then, it's nothing but cricket noises, sighs, maybe another round at the ping pong table. Resigned to having someone else fix their issue, Team A develops an attitude of learned helplessness. With some communication and coordination, they could InnerSource[66] with Team B to get their change made faster, or apply Socratic methods themselves with Team B to uncover that, hey, maybe everyone waiting around for each other is not the most effective delivery method.

Socratic thinking points us toward *how* we might solve problems when we feel stuck looking at an issue as either possi-

65 http://blog.sandglaz.com/using-socratic-method-improves-collaboration/

66 https://paypal.github.io/InnerSourceCommons/

ble or impossible. For example, when challenging their own blockers, Carbon Five used the Socratic method to assess those "blocked" development tasks and find alternative paths forward.[67] They discovered many of their designated "blockers" were, as Wikipedia notes, "concepts that seem to lack any concrete definition." In many cases, "we can't" is a concept that lacks concrete definition; "we can't" . . . *because why*? According to whose rules? Are there any rules? Or are we setting up our own invisible walls and imaginary authority figures who will stop us from trying to solve this problem? Challenging perceptions with questions like these will help you to better assess whatever obstacles you're facing, judge whether those things are real or not, then tackle them differently.

And lastly, using the Socratic method can help people become more comfortable with failure.[68] In Germany, where I live and work, the belief that failure is a big deal/sin/source of shame is still common enough for people to talk about it. My company has been countering this by creating opportunities to talk about failures as learning experiences, and integrating Site Reliability Engineering's "no blame" stance toward incidents. But adapting to changes, even positive ones, takes time. One-on-one's are the best approach for reducing the potential shame-damage for the person who's fearful of failure; in a personal conversation, the risks and visibility are low.

That's where the Socratic method shines.

67 http://blog.carbonfive.com/2011/01/14/the-socratic-method-and-agile-why-we-should-question-everything/

68 See Gordon Haff's chapter in this volume.

How I Socratic method'ed

Recently I used the Socratic method with several developer-colleagues on my team. We're still in something of a "storming" phase, but heading toward norming. It's a great bunch of guys: Productive, accountable, humble, and experienced—software craftsmen all the way.

We've needed to work on our communication flow: How much, when, by whom, and to whom. We've talked about communication in retrospectives, planning meetings, team autonomy health checks, and elsewhere, but communication is a broad topic that relies heavily upon people's personalities and comfort levels. Perception of what's possible is also a strong influence: In my experience, many devs are better communicators than they give themselves credit for. With this in mind, I grabbed a whiteboard and a spare room and each dev joined me for an intense round of Socrates and mind-mapping.[69]

The first dev-volunteer and I started our conversation with a blank board and no expectations. Perfect. He's somewhat reserved, so my only agenda was getting him talking. He started sharing ideas and thoughts—which he always does, but in the context of a one-on-one his words seemed more completely his own. We explored some of the why's and how's underlying his ideas: Why are we not doing now some of the things we'd like to do? How could we make this new idea happen? What or who could stop us from doing it? And on and on, like this, until we had a full board and something of a "ladder of related concepts" related to communication blockers, outcomes, and aspirations. The concepts we'd covered ranged from emotional to tactical: The word "fun" at the top, signifying what's possible, and an unhappy-face (what happens when we don't ask for help, or what

69 See Justin Holmes' chapter in this volume.

we need to avoid taking on too much of one thing we don't like) at the bottom, in a kind of whiteboard-dungeon.

I brought in the key concepts I covered with the first dev to subsequent one-on-one meetings, and there I asked what those concepts meant to new participants. Each developer brought his own views and defined them in different ways, which helped me to become better acquainted with their thought practices and motivations. Some concepts drew immediate responses that we then investigated, while other concepts seemed unfamiliar. Narratives formed; I used the whiteboard to point out related steps or ideas, ask them if they saw the connections or not, and drew lines (including dotted ones) to point out relationships or contradictions.

For example, one of the concepts I scribbled on the board was called "taking one for the team," which I explained as sacrificing oneself at a level that engenders burnout, boredom, isolation, or frustration. I intended to probe deeper into what happens when we don't say "no," enforce boundaries, and ask each other to help us. Would the devs draw the connections between this concept and our efforts to communicate more effectively? They did, while bringing their own personal values and beliefs into the discussion.

While my colleagues shared their thoughts, I detached and aimed to show no emotion. Instead, I listened and wrote things they said on the whiteboard. As the thoughts flowed, I treated every word as the truth in that moment—until I heard something vague or possibly contradictory. Then I'd prod a bit further, ask for clarification, and synthesize points. The whiteboard provided a record for contemplating and deriving conclusions. If I saw or heard a possible connection or contradiction, I'd point it out, but in the form of a question: "You said

this here, but you've also said this opposite thing there. How do you reconcile the two?"

Each conversation took about an hour and a half, and each was different from the others because each developer set the direction himself. Had I set the agenda beyond introducing a few simple concepts, the day might have turned tedious and repetitive. Instead, I focused on staying calm, neutral, and observant. This was incredibly fun, and at the end of the last conversation I felt energized (as an extrovert predictably would). It was a rigorous, open way to get to know my colleagues better and collaborate for solving our communication issues.

At the end of every conversation, I asked the guys for feedback. Maybe they were being polite, but the response was favorable. I don't attribute their demeanor to anything magical I did, however, but to the discovery and exploration made possible by the Socratic method.

Closing tips

Even if you're applying the Socratic method in a private chat, you still have to be mindful of your own part in the discussion and the signals you're communicating. Thinking back to our great teacher, Benny Hill: Don't assume that your team is limited in its abilities or has negative intentions.[70] Becoming emotional about people's responses is another way to get knocked off the path to enlightenment; you're undermining your power and distracting from the goal if you do. Focus on your partners: Pay attention to words, pauses (silence communicates loads of information), gestures, and tone.

If you're planning to use the Socratic method yourself, remember:

70 See Jonas Rosland's chapter in this volume.

- Don't assume that you have all of the answers. Leave room for yourself to be surprised and informed (after all, you're also a student).
- Stay neutral and supportive, so the other person or team feels like they can be wrong or experiment with their thoughts without being judged or reprimanded.
- Keep in mind that not everyone in your team or organization will be ready to undertake this method of inquiry (my colleagues were generous enough to be open to it; that we all shared both a great level of respect for each other and a commitment to the team helped as well). If your entire team isn't ready to try this approach, "pilot" it with one or two willing team members.
- Promise everyone you'll stay neutral, but first be confident you can uphold this promise. Guaranteeing—and then reneging on your promise of—a safe environment can be harmful to the exercise. As part of this bargain, be sure that your participants know that gossip or unconstructive criticism are off-limits.
- Reiterate that your goal is to gain clarity and awareness, to help the team or organization strengthen its foundation for future decision-making or actions.
- Time pressures might inhibit you from finding willing partners, and even one long conversation full of inspiring "aha!" moments won't bring about total enlightenment. Much of what you're unearthing in a Socratic dialogue involves someone's thought patterns, which are shaped over lifetimes and become habits. You might have to arrange regular, brief one-on-one meetings focused on a single question at a time, before the awareness sticks.

But, hey—don't take my word for all this. Try it for yourself. All I know is that I know nothing.[71]

Lauri Apple develops and evangelizes Zalando's open source efforts. She's also a producer/Agile project manager for the company's core search engineering team and co-leads Zalando's InnerSource initiative. She's based in Berlin.

71 https://en.wikipedia.org/wiki/I_know_that_I_know_nothing

Chapter discussion and review

• Have you ever used (or participated in) the Socratic method as part of team culture-building? If so, have you found the exercise useful? Why or why not?

• Lauri argues that "while we might intend to be open, fiction will motivate our actions if we're operating on assumptions." What are the assumptions that guide daily life and work among you and your teammates? Have you ever taken time to unpack and examine those assumptions?

• How often should teams reflect on the assumptions underpinning the work they do? Does your team currently have plans to do this?

• Lauri notes that "the Socratic method provides an opportunity to act as a servant-leader." What do you think this means? Have you ever played this role? Does anyone else on your team assume this role? What are its benefits?

Forming and onboarding an agile team

Jen Krieger and Hina Popal

There are several schools of thought on how to form and onboard an Agile team, and we've tried them all to see which one works best. What we've learned is simple: No single, easy solution works for all teams, because teams are made of people and people are different! We've written this chapter for the "easy path" team—the team that has the perfect product, the perfect vision, and the perfect preloaded list of things to work on. We hope it will help inspire you on how to approach onboarding your Agile team.

Some cautionary advice: Agile is never easy. Onboarding a new team can be a full-contact sport. This process can work better when you have a professional guiding the effort, but we've also learned that it is just as attainable without those people—when the team truly embraces the foundations of the Agile Manifesto and believes in their product. What follows is the "secret sauce" influencing the Red Hat Product & Technologies Agile Practice approach to the onboarding experience.

While every onboarding strategy is different, most unfold in a similar "three phase" architecture. So we've organized this chapter to mirror that process. We'll discuss assessment (Phase 1), kickstarting (Phase 2), and inspection (Phase 3).

1. Assessment

When beginning with a new initiative, you will want to observe current conditions and assess what the organizational structure will allow you to do. You need to:

- understand the environment of your organization, team, and work
- figure out what you can leverage
- be aware of rules you have to abide by

Identifying the answers to these items is a precondition for drafting a plan that you can execute in the existing environment. This can be an overwhelming exercise, but (in true Agile fashion) we're going to break it down into digestible, prioritized chunks.

Assess the Organization. First, you want to look at the overall state of the organization, evaluate how everything is structured, and ask the following types of questions:

- Are departments isolated within their respective disciplines?
- Is collaboration between teams and individuals encouraged?
- Is there a mix of roles in different departments or do teams consist of one role?
- How do different departments communicate with each other?

Agile approaches emphasize multidisciplinary teams, so knowing if the organization's structure actually encourages cross-functionality is critical. In organizations that tend to emphasize *individual* skills, people aren't used to working together for a common goal in a fast-paced environment. Imagine having a butcher, a pastry chef, a raw vegan, and a home cook working

together to open a restaurant in six months—without ever having worked together before. That could be a disaster!

Additionally, you'll want to identify the key players in your organization, how they feel about adapting to an agile approach, and what they think it is. You want know if they're in favor of setting up an Agile environment, or if they think that Agile is just another way to get what you need out of your staff faster.

Using this information to help structure your team will be crucial to grow a good reputation and also grow the presence of Agile in your organization. This will help you avoid hearing "I told you so" when you explain why the team hasn't cured cancer after the first sprint.

These are factors people often overlook before they begin—but when assessed beforehand, they can be game-changers for your team in terms of setting and managing expectations.

Assess the team. The next part of the assessment—and the most important—concerns people.

A general health check is important for benchmarking where the team currently stands and determining their overall mindset. You'll want to know how they feel about the process. What will happen when you want to define the workflow together? Will they push back on the idea of having a structured environment, or will they thrive on having a set of rules to follow?

You'll also want to know how long they've been working together. Is there going be a level of "New Kid" syndrome—where new members feel left out surrounded by others with established relationships—or are team members already comfortable with one another?

Another important point of assessment is the personalities of your team members. Examine these in order to accommodate their needs. Everyone on the team will bring value to the

process, but you'll want to know who will be outspoken and who will shy away.

Assess the work. Now that you've evaluated the team, it's time to figure out what it needs to deliver.

At the end of the day, we create teams to develop products. Knowing if the team's product is going to make or break the organization, or if failures can be absorbed without causing too much harm, is important. You should also understand how well-defined the vision is—something with a scope that changes every week will probably not be something your team can deliver in the next six months. Knowing that up front will help set everyone's expectations.

You'll collect this information just for yourself, so you can make decisions later in the process. However, you'll also want to be sure you understand the context of the organization and the personalities of your team members before implementing any new processes.

2. Kickstart the Team

Creating the right environment for team success isn't always easy. We've seen two general approaches. The first: Start the team with a specific structure and framework to follow, then allow them to modify over time. The second: Allow the team to determine a starting point and organically develop their process as time goes on. Both can work, but your choice will largely depend on what you learned during assessment.

For example, if you have a team working on a product release that will very visibly impact the bottom line and they have a tight deadline, it might be better overall if you started with a specific structure. Early successes can help the team (and their management) gain confidence that the change in behavior isn't going to cause undue harm to the business.

The first thing you'll want to accomplish when kickstarting a team is getting team members together for an initial kickoff meeting. We encourage covering the following topics.

What is Agile? In this section, we typically cover only a few small points. We want new teams to focus on understanding the feedback they receive (whether from daily meetings to discuss where things stand, from test feedback on code they are integrating, or from a product release). The key point here is that the team understands *where* and *when* they receive feedback.

We also want new teams to understand and expect certain types of behavior from themselves and others early in the formation of a team. Generally, three key concepts apply here. They're rooted in the foundation of Agile, as well as kanban:

- If it takes you more than two emails to resolve a conversation, pick up the phone (or video conference!). In our experience, conversations held face-to-face always resolve faster than over a text-based form of communication.
- Keep any agreed-upon method for tracking work updated and visually in front of the team. This is especially important in the case of distributed teams. We want teams to visualize the work they have in their overall work system.
- As David Anderson puts it: "Stop starting, start finishing." We encourage team members to limit their work in progress.

What are we here to achieve? All team members need to understand the *vision behind* and the *direction of* the work you're asking them to achieve. Kickstarting a team should always include an overview from the business or a stakeholder that reviews these concepts and then also describes how the

team and organization should measure the success of their work. The only rule of thumb here is that everyone *must* understand the vision, so limit the use of buzzwords. Success measurements always should include some form of quantitative measurement.

How are we going to achieve it? This is the section of the kickoff where you discuss the team norms at play in terms of meetings, cadence, timing, and similar issues. You'll want to discuss a starting "Definition of Done" (e.g. what does "done" mean for the work the team is doing?), discuss the initial expected workflow for how work should be completed (do we work on anything we want, or is the work prioritized?), and set the foundation for future learning (when do we follow up with additional sessions to discuss other concepts?). Know that you won't be able to achieve complete understanding of the process you are intending to use in a kickoff meeting. Generally, people learn by doing.

Who is part of the team? This may be last on the list, but it is the most critical. Team members are human, and they'll often fall back into old habits. Setting expectations with everyone, describing early on what their roles are and what you expect from them, is critical to the overall success of the team. We've observed that inviting team members to collaborate on what they *think* their role should be helps significantly in obtaining the level of engagement you want from your team.

During this entire kickoff process, we like to frequently encourage open questions and answers while also acknowledging that people in the meeting should feel fine with their discomfort about speaking up. What you're trying to convey is that you are open to change, open to ideas—and generally want the team to achieve success together. In order to do that, their opinions must count!

3. Inspection

As the Agile Manifesto says: "Simplicity—the art of maximizing work that is not done—is essential." This principle embodies the final stage of the onboarding process: inspection and adaptation. Inspecting and adapting are the core functions of an effective Agile team. They'll also take the most time and attention in the early days of the team's engagement. If you aren't talking about what's hindering the team, then you don't have the opportunity to improve. Feedback loops help obtain an optimal environment for teams to eliminate waste in their process.

The following are examples of Agile processes that incorporate feedback loops:

- Code Review
- Continuous Integration
- Continuous Delivery
- Retrospective
- Sprint Review

Inspect your actions. Inspecting and adapting focuses on acknowledging your actions and determining what to do next. It is a decision gate in which the team identifies (inspects) an action and determines a) if it's beneficial or b) if it should be modified (adapted).

An explicit feedback loop common to many approaches is the "retrospective."[72] Often a team will discuss (inspect) what went well, what could have gone better, and what they've learned from the process. From there they decide what they can do in the near future to improve their process (adapt).

72 See Matt Thompson's chapter in this volume.

Adapting to benefit. Here are the benefits we've observed when working with teams who have gotten the gist of the "inspect and adapt" concept:

- They learned to tackle what they could fix rather than talk about the things outside of their control. This helped improved morale and general happiness.
- They learned to use metrics to identify when their process wasn't working. They reduced the time between their feedback loops and could show the improvement using data.
- They learned that it's ok to be friends and have fun—that work doesn't always have to be stressful. This helped the team gel.
- They learned that a retrospective wasn't a punishment meeting, but a chance to address the things that aren't working.

Conclusion

We've covered a lot in this chapter on Agile team formation and onboarding. This process is a messy one! You can say the same thing many times to many different people and have everyone implement it in different ways. The important thing to remember is that the benefits the team can experience by adapting to an Agile approach far outweigh the mess. Additionally, most people skip Phase 1 (Assessment) and never give Phase 3 (Inspection) a second thought. If you try only one thing with your teams, our recommendation would be this: Always identify the ways the team receives feedback and ensure they meet regularly to determine how to improve those feedback loops.[73] If the

73 See Jimmy Sjölund's chapter in this volume.

team does take that process seriously, the overall end result will likely be the same—it will just take them longer to get there.

Jen Krieger is Chief Agile Architect at Red Hat. Most of her 20+ year career has been in software development representing many roles throughout the Waterfall and Agile lifecycles. At Red Hat, she led a department-wide DevOps movement focusing on CI/CD best practices. Most recently, she worked with the Project Atomic and OpenShift teams.

Hina Popal is an Agile Practitioner at Red Hat. She began in the public sector doing government contracting work while pursuing her passion for Agile as a way to avoid bottlenecks in a world full of bureaucracy. After a few years, Hina jumped to a fast-paced environment and is tackling the world of open source by working with the Project Atomic and OpenShift teams at Red Hat.

Chapter discussion and review

• Jen and Hina outline a three-step process for forming and onboarding agile teams. "Most people skip Phase 1 (Assessment) and never give Phase 3 (Inspection) a second thought," they say. Why do you think this is the case?

• Have you ever been part of a newly formed, agile-focused team? What was the experience like, especially at the beginning? What would you change about your own oboarding process?

• What does "agile" mean to you? What the benefits of being agile? What are the drawbacks? Does your team consider itself an agile team? Why or why not?

A formula for running an accountable IT organization

Stephen Gold

At CVS Health we have a framework that we've formed and embedded inside the IT organization that drives our culture and outcomes. It's what I call the ACT framework, A-C-T, and it stands for accountability, collaboration, and tenacity. These three cultural and behavioral ingredients, combined with the right technology and the right process, make the recipe for running an effective IT organization.

It all starts with accountability. Ultimately, as a CIO, what I'm looking to cultivate is an "accountable organization" as opposed to "an organization that is held accountable." This is an important nuance that is perhaps best illustrated in a real world example.

When you watch a basketball game and a foul is called (assuming the call was accurate), there are three typical player reactions. There is the player who commits the foul and doesn't respond with any reaction or emotion. It's kind of an inert, passive response. Then there are two extreme reactions on either end of the accountability spectrum. The worst kind of reaction is the player who takes out his mouthpiece and starts screaming at the referee, aggressively arguing, "That wasn't a foul!" This kind of behavior is the furthest thing from being accountable. On the opposite end of the accountability spectrum is the player who

raises his hand and simply says, "That foul is on me." This is a person who is accountable for his actions.

In all of these scenarios, a foul was in fact committed. The referee is going to hold the person accountable, regardless of how they respond. Similarly in technology, we make mistakes or issues arise that team members know we will be held accountable for. But when this happens, I don't want our people to let it roll off their backs with no reaction, or worse, try to defer blame or point fingers. I want to foster, culturally and behaviorally, a team that is accountable for our actions and willing to own our outcomes.

Getting to the root of accountability

When you are working with accountable people, the first thing they will do when an issue arises is become passionately and unwaveringly committed to understanding the root cause of the issue – not just fixing the symptoms, but rather treating the illness. We use the "Five Whys" methodology, which states that you have to ask "why" five times in order to find out the real truth, the real root cause, of an issue.

Let's use a fictitious example that has nothing to do with technology. Say somebody calls into work: "Joe, I can't come to work today." In a typical work environment, that might be the end of the discussion. But in a truly accountable workplace, it's just the beginning. Let's use the "Five Whys" to find out the root cause of the issue:

"*Joe, I can't come to work today.*"

1. "OK, just out of curiosity why not? Is everything OK"?

"*My car won't start.*"

2. "Why won't your car start?"

"*The battery died*."

3. "Why did the battery die?"

"*Because my alternator belt snapped*."

4. "Why did your alternator belt snap?"

"*Because it was five years old*."

5. "Why was your alternator belt five years old?"

"*Because, I did not follow the prescribed maintenance schedule for the vehicle.*"

Now we've gotten to the root cause. It's a lot easier to blame the car or the battery, but at the end of the day, our actions or inactions drive outcomes. Obviously, people in technology aren't talking about a car battery or alternator belt. They're talking about hardware or they're talking about software. But, ultimately, just like the example above, the discussion is not about a router, for example, it's about the person behind the router. We know that routers fail, plain and simple. So now that we know routers fail, we need to explain why we don't have two of them; we need to explain why they're not configured for 100 percent failover. This has nothing to do with the router; this has everything to do with how we architect and engineer these systems.

The "Five Whys" is a great acid test to see whether we are acting with accountability. When a system crashes, or there is a defect, or a project is late, or whatever the issue—ask why, and keep asking why until the answer to the question begins with the words "I" or "we." When the answer begins with "I" or "we," then you know you have an accountable organization.

Collaboration and tenacity round out the framework

Collaboration and tenacity are the two other essential elements of the ACT framework. Collaboration is all about working with our customers on solutions. The process shouldn't be a series of handoffs between departments; we need to work together to solve problems.

When things are done serially, requirements are thrown over the wall and then we do a design and we throw the design back over the wall, and they sign off on it and we go off and develop it, and back and forth. That's really not the most effective or efficient process. Instead, we try to embed ourselves inside the business and work with our customers to achieve the best outcomes.

Tenacity, to me, is like Superman laying over the tracks. As you evolve organizations, there are always going to be gaps. The gaps could be technological, they could be process gaps, and they could be people gaps. What I try to cultivate and aggressively reward are the behaviors that overcome those gaps.

A tenacious organization is one that is dedicated to keeping our commitments to our customers at all costs. I've come to realize that the better you get at people, process, and technology, the less you need to depend on the tenacious nature of this ACT formula. But especially in times of transformation, you need a tenacious organization to carry you through any obstacles that may be standing in the way to success.

Stephen Gold is Executive Vice President and Chief Information Officer for CVS Health. In this role since July 2012, Gold is the company's senior technology executive and has responsibility for all information systems and technology operations, including information technology strategy, application development, technology infrastructure, and business and technology operations.

Chapter discussion and review

• How would you describe the difference between an "accountable organization" and "an organization that is held accountable"? Why is that distinction important when trying to run an accountable IT organization?

• Think of a recent technology problem your IT organization faced. Now apply the "Five Whys" methodology to find the root cause of the issue. Does this exercise lead to new insights? How can the "Five Whys" help your IT organization become more accountable in the future?

• In what other ways can you cultivate and reward collaboration and tenacity in your IT organization?

Institutionalizing experimentation with impact mapping

Justin Holmes

Impact mapping is a technique for building shared understanding between leaders and project teams. Delivered in an engaging workshop format, impact mapping is the perfect way to initiate a work stream in a way that encourages innovation. Gojko Adzic first documented the technique in a 2011 brochure; it's an excellent guide for individuals who want to facilitate the workshop. This chapter aims to complement Adzic's original text with a guide for leaders who want to sponsor impact mapping initiatives but may not facilitate the workshops themselves. In particular, I'll provide a succinct overview of impact mapping as a practice, and then offer guidance on ways leaders can use impact mapping to establish experimentation as an expected behavior during project delivery.

What is impact mapping?

The simplest way to understand impact mapping is to unpack the phrase itself.

The term "impact" in this context refers to a human behavioral change, something affected by the delivery of a product feature or a process change. Impact mapping defines the value of any work effort in terms of its "impact" (not merely its "completion"). This idea comes to us from the design thinking

community, and has significant implications for the ways leaders incentivize risk-taking and therefore innovation (as I'll discuss in the next section).

The term "mapping" is derived from the concept of the "mind map," which participants build as part of the workshop. This special kind of mind map—also known as an "impact map"—is carefully constructed to surface the assumptions underlying a work effort. Specifically, impact mapping seeks to highlight all assumptions that:

1. a specific deliverable will lead to a specific behavioral change, and
2. a particular behavioral change will help the organization achieve its goal

True to its lean product development roots, impact mapping provides a framework for using metrics to translate these assumptions into testable hypotheses.

Given the high value of the outputs of impact mapping, you may be surprised to learn that facilitating the technique is actually fast and cheap. It requires no expensive tools or training,[74] so barriers to entry are low. If you'd like to use it in its simplest form, you can probably begin by reverse engineering a map without metrics from the project on which you are currently working. It will take you about 30 minutes at the whiteboard. With approximately four hours, a prepared facilitator can lead key stakeholders to an impact map with basic metrics for your next strategic initiative. More complete maps require an additional preparation phase to create comprehen-

74 A digital drawing tool that supports live editing by multiple users can be quite helpful, as sometimes the maps are too be big to fit on a reasonably sized whiteboard. I use *https://coggle.it/* and Adzic made *https://www.mindmup.com/* expressly for this purpose. It's also a good idea to purchase a copy (or several) of Adzic's original text, which is very affordable.

sive metrics. But these extra tasks are asynchronous, so completing an impact map won't require locking a team in a room for weeks at a time (instead you can schedule the process around busy stakeholders).

The result is an engaging, approachable, and high-value workshop that will guide your project from its beginning through its conclusion.

Making experimentation expected behavior

Many practitioners will leverage impact mapping to quickly and clearly connect their projects' deliverables to a value proposition. This is an especially useful application of the technique, given organizations' propensity for getting lost in their activities and forgetting *why* they are doing their work in the first place.[75] But for leaders willing to adapt their management strategies in order to institutionally foster innovation, impact mapping offers much more.

Much of modern management theory can trace its roots to Taylor's *Principles of Scientific Management*, which suggests that management should systematically design the *what* and *how* of all work in an organization.[76] Workers, then, faithfully execute that work. Anyone who has ever worked on a large "waterfall" IT project, with its long cascading chain of requirement handoffs, has experienced Taylorism. Scientific management (and, transitively, waterfall IT projects) are optimized to deliver predefined outputs; however, as this book discusses, innovative organizations are optimized for experimentation, where the outputs are unknown.

75 See Allison Matlack's chapter in this volume.

76 See Matt Micene's chapter in this volume.

So how do leaders optimize for experimentation and ensure that their organizations deliver necessary outcomes?

Economic decision rules offer one straightforward and proven approach. Documented by Donald Reinertsen,[77] this practice allows low levels of the organization to control the decision making process so long as the resulting decisions align with management's economic model. Because impact maps force organizations to measure value in terms of human behavioral changes (instead of merely the delivery of project scope), and because impact maps concretely tie work to the broader organizational mission, we can think of impact mapping as a structured approach to building economic decision rules. In this case, project teams feel empowered to decide which outputs to deliver and how to deliver them, but with the constraint that these outputs affect the behavioral changes agreed to when building the map. And remember: These behavioral changes are assumed to be effective proxies for achieving the project's stated goal, until experimentation shows otherwise.

In the context of an organization's IT culture specifically, the application of economic decision rules derived from impact mapping has two significant implications:

1. Project teams have incentive to experiment with low cost prototypes to validate that their approach will deliver the required outcomes early in the delivery process. This is opposed to the traditional IT project delivery model that focuses on delivering a negotiated list of requirements at all costs.

77 Donald Reinersten describes this idea in detail as principle 13, "The First Decision Rule Principle: Use decision rules to decentralize economic control," in *The Principles of Product Development Flow: Second Generation Lean Product Development*.

2. Managers have incentive to ensure their desired outcomes have well defined measurements to enable the team to be confident in the results of their experiments. This is opposed to the traditional approach, where managers focus on ensuring requirements have been properly defined and successfully handed over to the project team.

The result is a system of project management that allows the lowest levels of the organization to (as Gene Kim puts it earlier in this volume) "discover their way to greatness," but do so in a way that ensures leadership can still direct the organization towards success. Of course, impact mapping is not a panacea for creating an innovative IT department.[78] But impact mapping does provide leaders with a practical tool for making experimentation—and by extension innovation—the default approach to project delivery.

Justin Holmes is a passionate consultant who helps teams deliver better software products, faster. He is a fan of methods that lead delivery teams towards shared understanding and technologies that capture that understanding in software. Justin is active in the Behavior Driven Development, Domain Driven Design and Lean Product Development communities. You can also find him helping customers accelerate their innovative ideas in Red Hat's Open Innovation Labs.

78 It does not, for example, address the key questions about evaluation and failure that Jim Whitehurst raises elsewhere in this volume.

Chapter discussion and review

• Have you ever participated in an impact mapping session? How would you describe the experience to someone who never has?

• Justin argues that "impact maps force organizations to measure value in terms of human behavioral changes (instead of merely the delivery of project scope)." How do you and your team measure the value of your work? Are you utilizing the proper metrics when you do so? Can you think of more effective ways to demonstrate the business value of the work you're doing?

• Impact mapping, Justin writes, is one way to "incentivize risk." What does this mean to you? And do you agree? How does your team handle "risk" today, and should this approach change?

Assuming positive intent when working across teams

Jonas Rosland

When teams in the same organization—or even across organizational boundaries—start to collaborate, they will most likely realize that not all of their goals align. The IT team, for instance, might not have the same criteria for success as the sales team. Different teams have different benchmarks, even if the teams are part of a larger organization (as in the case of relationships between a developer team and an operations team).

But the teams all strive towards the same goal, which is to make the organization successful. And they rely on each other to accomplish that goal.

For this reason, learning more about *why* a team is working on a specific project, not just focusing on *how* they're involved and *what* the project is about, can help you foster better relationships across your organization. Focusing on why people act the way they do allows teams to better understand one another's goals and purposes, and helps everyone assume the best intentions from everyone involved in an effort.

Assuming positive intent when working across teams involves a few fundamentals, which I'll discuss in this chapter. First, it involves paying attention to your team's sense of identity. It also involves knowledge sharing across groups. And

finally, it involves potentially changing the way you reward behaviors.

"They just don't get it"

Teams typically form strong social group connections, which, on paper, might seem excellent. Teams with strong social connections share willingly and help each other. But a strong sense of group identity can also introduce the real possibility that the group will learn to see itself as *different from* and sometimes *better than* other groups. This may lead to an unhealthy tendency to turn away from those outside the group, which can result in teams not sharing vital information, or increased conflicts and a reduction in the quality of deliverables. Along the same lines, strong social ties can affect team merges, and organizations often encounter difficulties when working with new teams.

This disconnect between teams can lead to simple misunderstandings that are blown out of proportion, leading to frustration and anger towards other teams and their members. If you are part of this strong social group, you might hear statements such as "They just don't get it" and "Why don't they understand?" thrown around as morale-boosters and laughed off. But statements like these aren't helping either team succeed.

One way of dealing with this antisocial behavior is to create a new group dynamic by continuously rotating members or having them work very closely together on joint projects. By doing this you gain new viewpoints and mindsets between the different groups, slowly removing the barriers between them. Removing misunderstandings and friction between teams is imperative when developing a positive working environment across team boundaries. It will no longer be an "us versus them"

discussion, but rather a newfound focus on creating value together.

Sharing knowledge

When creating a new open source project or open sourcing a proprietary product, it's critical that the audience that you're trying to reach understands *why* they should use it or get involved. If your documentation only focuses on *how* it's built and *what* it does, the possible success of your project will most likely be limited.

The same is true in any open organization. But by using open discussion platforms, members of a specific project community can share knowledge with each other and encourage involvement by constantly receiving feedback and ideas. The community will amass information and know-how that people can share using different media to promote the project (YouTube videos, marketing material, stickers, logos, code, how-to blog posts, conference sessions, and much more). A common practice in open source projects and their respective communities is encouraging participation in a variety of forms, not just by writing code. Having a diverse set of people work together across teams where their different specialties are appreciated also means that the project will have greater impact. But on top of that, making sure the community is aware of new features, release dates, and upcoming changes ensures that they can easily communicate to others on how the project is progressing. When everyone is working with the same knowledge, people are more likely to assume they're working with the same *intentions*, too.

Changing behaviors

When working across teams and with new team members, it's important to recognize the individual strengths and weak-

nesses *within* the team and not just view the team as a whole, single unit. Making sure that everyone on the team learns how to deal with new tasks, both rudimentary and advanced, leads to the team working better together and accomplishing more. We can use methods first utilized in the early 1900s to ensure that team members enact desirable behaviors when they're using new tools and processes.

Research done by B.F. Skinner shows that positive and negative reinforcement shapes behavior. Because all actions has consequences, this means that if a consequence is positive there's an increased probability of that action being encouraged and repeated. Likewise, Skinner's experiments show that responses were better and faster when the consequence was positive, compared to when they were treated to a negative response.

This means that rather than waiting to congratulate new team members when they finished tasks, encouraging individuals often and repeatedly while they are progressing through tasks will teach them faster what path to take to complete specific tasks thanks to positive reinforcement.

Thinking about your teams this way you will create a more inclusive atmosphere by removing the notion of "rockstars": team members who always stay late, the ultimate troubleshooters, the ones with all the key information to any decision. Celebrating individuals who pull all-nighters can do more harm than incentivize. The presence of rockstar performers on a team may not necessarily lead to increased success for that team; it can actually have a negative impact on the performance of others.[79]

79 See Laura Hilliger's chapter in this volume.

Research from the Harvard Business School[80] shows that there are three mechanisms that may account for the decline in productivity within teams when certain individuals are seen and treated at rockstars:

1. A reduction in effort ("I don't see the point in trying")
2. Increased risk-taking ("I must do something amazing to be noticed")
3. Deterioration in cognitive processing ("I make more mistakes")

When talking about these star performers, it's critical to understand their role in the team. They're viewed as rockstars for a reason: They excel at what they set out to do. Don't undervalue them; instead, identify a different measure of their success. Celebrating valuable contributions such as customer interactions, education of others, and innovative ideas give teams a more varied view of what it means to be successful.

Creating an inclusive atmosphere by trusting everyone, not just star performers, to be capable of delivering the correct solutions for a certain set of problems can lead to more open and vibrant discussions, suggesting new and innovative ways of thinking. It also makes assuming the best-directed optimal solutions from everyone just a little easier.

As a result of assuming positive intent when working across teams, organizations and international borders, open source community members working together towards common goals can make projects very successful in their mission. This applies for a wide variety of projects such as the Linux kernel, the Go programming language—and this book.

80 http://www.hbs.edu/faculty/Publication%20Files/13-016_5a1d0819-eab1-48d9-8923-a1d83c98b7a7.pdf

Jonas Rosland is a community builder, open source advocate, blogger, and speaker at many open source focused events. As Open Source Community Manager at {code} by Dell EMC, he is responsible for the growth and prosperity of the {code} community.

Chapter discussion and review

• What does "assuming positive intent" mean? What does it look like in practice? What are its benefits? And what are some challenges or barriers to it?

• At several moments in his chapter, Jonas stresses the importance of understanding why an individual or team is performing a particular task. Do you think you have an adequate sense of why your colleagues and teammates do the work they do—not just what they do or how they do it? What gaps in your knowledge would you like to fill?

• Does your team have "rockstars," and, if so, does it tend to let them influence decisions in ways that other team members can't? Does your team benefit from "rockstar culture," or should it work to change this culture?

Appendix

The Open Organization Definition

Preamble

Openness is becoming increasingly central to the ways groups and teams of all sizes are working together to achieve shared goals. And today, the most forward-thinking organizations—whatever their missions—are embracing openness as a necessary orientation toward success. They've seen that openness can lead to:

- **Greater agility**, as members are more capable of working toward goals in unison and with shared vision;
- **Faster innovation**, as ideas from both inside and outside the organization receive more equitable consideration and rapid experimentation, and;
- **Increased engagement**, as members clearly see connections between their particular activities and an organization's overarching values, mission, and spirit.

But openness is fluid. Openness is multifaceted. Openness is contested.

While every organization is different—and therefore every example of an open organization is unique—we believe these five characteristics serve as the basic conditions for openness in most contexts:

- Transparency
- Inclusivity
- Adaptability
- Collaboration
- Community

Characteristics of an open organization

Open organizations take many shapes. Their sizes, compositions, and missions vary. But the following five characteristics are the hallmarks of any open organization.

In practice, every open organization likely exemplifies each one of these characteristics differently, and to a greater or lesser extent. Moreover, some organizations that don't consider themselves open organizations might nevertheless embrace a few of them. But truly open organizations embody them all—and they connect them in powerful and productive ways.

That fact makes explaining any one of the characteristics difficult without reference to the others.

Transparency

In open organizations, transparency reigns. As much as possible (and advisable) under applicable laws, open organizations work to make their data and other materials easily accessible to both internal and external participants; they are open for any member to review them when necessary (see also *inclusivity*). Decisions are transparent to the extent that everyone affected by them understands the processes and arguments that led to them; they are open to assessment (see also *collaboration*). Work is transparent to the extent that anyone can monitor and assess a project's progress throughout its development; it is open to observation and potential revision if necessary (see also *adaptability*). In open organizations, transparency looks like:

- Everyone working on a project or initiative has access to all pertinent materials by default.
- People willingly disclose their work, invite participation on projects before those projects are complete and/or

"final," and respond positively to request for additional details.

- People affected by decisions can access and review the processes and arguments that lead to those decisions, and they can comment on and respond to them.
- Leaders encourage others to tell stories about both their failures and their successes without fear of repercussion; associates are forthcoming about both.
- People value both success and failures for the lessons they provide.
- Goals are public and explicit, and people working on projects clearly indicate roles and responsibilities to enhance accountability.

Inclusivity

Open organizations are inclusive. They not only welcome diverse points of view but also implement specific mechanisms for inviting multiple perspectives into dialog wherever and whenever possible. Interested parties and newcomers can begin assisting the organization without seeking express permission from each of its stakeholders (see also *collaboration*). Rules and protocols for participation are clear (see also *transparency*) and operate according to vetted and common standards. In open organizations, inclusivity looks like:

- Technical channels and social norms for encouraging diverse points of view are well-established and obvious.
- Protocols and procedures for participation are clear, widely available, and acknowledged, allowing for constructive inclusion of diverse perspectives.

- The organization features multiple channels and/or methods for receiving feedback in order to accommodate people's preferences.
- Leaders regularly assess and respond to feedback they receive, and cultivate a culture that encourages frequent dialog regarding this feedback.
- Leaders are conscious of voices not present in dialog and actively seek to include or incorporate them.
- People feel a duty to voice opinions on issues relevant to their work or about which they are passionate.
- People work transparently and share materials via common standards and/or agreed-upon platforms that do not prevent others from accessing or modifying them.

Adaptability

Open organizations are flexible and resilient organizations. Organizational policies and technical apparatuses ensure that both positive and negative feedback loops have a genuine and material effect on organizational operation; participants can control and potentially alter the conditions under which they work. They report frequently and thoroughly on the outcomes of their endeavors (see also *transparency*) and suggest adjustments to collective action based on assessments of these outcomes. In this way, open organizations are fundamentally oriented toward continuous engagement and learning.

In open organizations, adaptability looks like:

- Feedback mechanisms are accessible both to members of the organization and to outside members, who can offer suggestions.

- Feedback mechanisms allow and encourage peers to assist one another without managerial oversight, if necessary.
- Leaders work to ensure that feedback loops genuinely and materially impact the ways people in the organization operate.
- Processes for collective problem solving, collaborative decision making, and continuous learning are in place, and the organization rewards both personal and team learning to reinforce a growth mindset.
- People tend to understand the context for the changes they're making or experiencing.
- People are not afraid to make mistakes, yet projects and teams are comfortable adapting their pre-existing work to project-specific contexts in order to avoid repeated failures.

Collaboration

Open organizations are communal. Shared values and purpose guide participation in open organizations, and these values—more so than arbitrary geographical locations or hierarchical positions—help determine the organization's boundaries and conditions of participation. Core values are clear, but also subject to continual revision and critique, and are instrumental in defining conditions for an organization's success or failure (see also *adaptability*). In open organizations, collaboration looks like:

- People tend to believe that working together produces better results.

- People tend to begin work collaboratively, rather than "add collaboration" after they've each completed individual components of work.
- People tend to engage partners outside their immediate teams when undertaking new projects.
- Work produced collaboratively is easily available internally for others to build upon.
- Work produced collaboratively is available externally for creators outside the organization to use in potentially unforeseen ways.
- People can discover, provide feedback on, and join work in progress easily—and are welcomed to do so.

Community

Open organizations are communal. Shared values and purpose guide participation in open organizations, and these values—more so than arbitrary geographical locations or hierarchical positions—help determine the organization's boundaries and conditions of participation. Core values are clear, but also subject to continual revision and critique, and are instrumental in defining conditions for an organization's success or failure (see also *adaptability*). In open organizations, community looks like:

- Shared values and principles that inform decision-making and assessment processes are clear and obvious to members.
- People feel equipped and empowered to make meaningful contributions to collaborative work.
- Leaders mentor others and demonstrate strong accountability to the group by modeling shared values and principles.

- People have a common language and work together to ensure that ideas do not get "lost in translation," and they are comfortable sharing their knowledge and stories to further the group's work.

Version 2.0
April 2017
The Open Organization Ambassadors at Opensource.com
github.com/open-organization-ambassadors/open-org-definition

Learn More

Additional resources

The *Open Organization* mailing list

Our community of writers, practitioners, and ambassadors regularly exchange resources and discuss the future of work, management, and leadership. Chime in at www.redhat.com/mailman/listinfo/openorg-list.

The "Open Organization Highlights" newsletter

Get open organization stories sent directly to your inbox. Visit opensource.com/open-organization to sign up.

Discussion guides

Want to start your own *Open Organization* book club? Download free discussion guides for help getting started. Just visit opensource.com/open-organization/resources/.

#OpenOrgChat

Our community enjoys gathering on Twitter to discuss open organizations. Find the hashtag #OpenOrgChat, check the schedule at opensource.com/open-organization/resources/twitter-chats, and make your voice heard.

Get involved

Share this book

We've licensed this book with a Creative Commons license, so you're free to share a copy with anyone who might benefit from learning more about the ways open source values are changing organizations today. See the copyright statement for more detail.

Tell your story

Every week, Opensource.com publishes stories about the ways open principles are changing the way we work, manage, and lead. You can read them at opensource.com/open-organization. Do you have a story to tell? Please consider submitting it to us at opensource.com/story.

Join the community

Are you passionate about using open source ideas to enhance organizational life? You might be eligible for the Open Organization Ambassadors program (read more at opensource.com/resources/open-organization-ambassadors-program). Share your knowledge and your experience—and join us at github.com/open-organization-ambassadors.