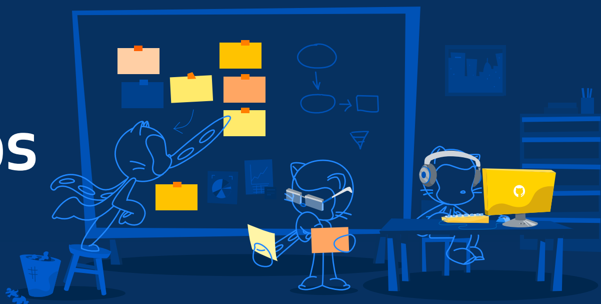
Teamwork powers DevOps

GitHub powers teams



GitHub helps more than two million organizations build better software together by centralizing discussions, automating tasks, and integrating with thousands of apps. Embraced by 31 million developers and counting, GitHub is where high-performing DevOps starts.

Get started with a free trial at enterprise.github.com/contact

Our on-premises and cloud solutions help enterprise teams:

Collaborate

Work across internal and external teams securely. GitHub Enterprise includes access to on-premises Enterprise Server as well as Enterprise Cloud—now with SOC 1, SOC 2, and ISAE 3000/3402 compliance.

Innovate

Bring the power of the world's largest open source community to developers at work, while keeping your most critical code behind the firewall with GitHub Connect.

Integrate

Build on GitHub and integrate with everything from legacy tools to cutting-edge apps, unifying your DevOps toolchain so you can keep things simple as you grow.

Work fast. Work secure. Work together.

Start a free trial

To find out more about GitHub Enterprise visit github.com/enterprise or email us at sales@github.com

Collaborating in DevOps Culture

Better Software Through Better Relationships

Jennifer Davis and Ryn Daniels

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Collaborating in DevOps Culture

by Jennifer Davis and Ryn Daniels

Copyright © 2019 Jennifer Davis and Ryn Daniels. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Virginia Wilson
Development Editor: Nikki McDonald
Production Editor: Deborah Baker
Copyeditor: Octal Publishing, LLC
Proofreader: Rachel Head
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

July 2019: First Edition

Revision History for the First Edition

2019-07-19: First Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Collaborating in DevOps Culture*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the authors, and do not represent the publisher's views. While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and GitHub. See our statement of editorial independence.

978-1-492-05244-9

[LSI]

Table of Contents

Preface v

1. Foundations of Collaboration 1
- Empathy 1
- Trust 2
- Psychological Safety 4
- Communication 5
- Summary 6

2. Collaboration in Practice 9
- Collaborative Discovery 10
- Collaborative Development 19
- Collaborative Production 23
- Summary 33

3. Conclusion 35

A. Further Resources 37

Preface

This book is intended for established leaders and those who are on the path to leadership within their organizations. It focuses on effective collaboration—one of the foundational elements of sustainable devops cultures described in our book, *Effective DevOps* (O'Reilly)—to guide decision makers and influencers through their devops transformations.

In writing this book, we have two goals: to give you an understanding of the foundations of effective collaboration—trust, empathy, and psychological safety—and to show you how to weave these principles into your company culture. Individuals who feel empowered and motivated to adopt the processes, practices, and tools necessary for healthy working relationships can help drive innovation. The efforts you make to promote collaboration within your company and to reward the individuals who embody collaboration principles will result in empowered employees, more productive teams, and a respectful workplace that people want to contribute to and continually improve.

Defining Devops

There is no single definition of "devops." Most agree that devops involves elements of people, processes, and technology organized in a way to help teams work well together to improve software creation and delivery. Many people think about devops as specific tools like Docker or Kubernetes, but devops is not about tools alone. Neither is it solely about specific practices like continuous deployment (CD) or continuous integration (CI). What makes tools and practices

"devops" is how they are used, not just the characteristics of those tools or practices.

We define devops as a way of thinking and a way of working. Specifically, it is a framework for sharing stories and developing empathy, enabling people and teams to practice their crafts in effective and lasting ways. It is part of the cultural weave of values, norms, knowledge, technology, tools, and practices that shape how we work and why.

"Devops," "devops," or "DevOps"?

We have had many discussions over the capitalization (or lack thereof) of the term "devops." A simple online poll showed overwhelming support for "DevOps." We also found a varying focus on the "Dev" and "Ops" within different organizations. This has led to the creation of terms like "DevSecOps" and "DevQAOps," as "DevOps" implies exclusively "Dev" and "Ops."

Ultimately, this is why we've chosen to style this word as "devops"—it reflects the original Twitter hashtag used to connect people wanting to help change conversations from "us versus them" to "enabling the business" with sustainable work practices that focus on people.

Successful projects require input, effort, insight, and collaboration from people across the organization; the problems inherent in your organization may not be limited to just developers and operations teams. We have deliberately chosen to use lowercase "devops" throughout the text of this book to reflect our view that this is an inclusive movement, not an exclusive one.

Collaboration in the Context of Practicing Devops

Collaboration is the practice of multiple individuals working together, building toward a specific outcome. Effective collaboration within the enterprise isn't like a school project gone awry in which one person dominates everyone else's efforts, ignoring other people's inputs and opinions in pursuit of their own good grade. It also isn't about some people picking up the slack for others who have checked out. Instead, it's about people coming together from different perspectives, and everyone providing input so that you can come to a

shared understanding with a collective team perspective. Collaboration is about building trust, empathy, and team psychological safety in an environment that often requires you to work with many people for short periods of time to get work done.

Software development and operations teams working together is a core part of the devops movement.[1] Before one team can successfully work with another team that has a different focus, the individuals on both teams need to be able to work with each other. Teams that don't work well on an individual or intrateam level have little hope of working well at the interteam level.

In this book, you'll learn about the building blocks of collaborative relationships and how to promote them in the workplace and apply them to your teams. We offer tips on improving communication within and between teams to strengthen your working relationships. And finally, we show you how to apply collaboration principles to different stages of the software development life cycle. This is not an exhaustive set of recommendations; our aim is to give you ideas and advice that you can begin applying in your organization today to improve team productivity, increase the frequency and quality of software deliveries, and focus more time on innovation.

1 As seen in the O'Reilly Case Study on Continuous Deployment by John Allspaw and Paul Hammond.

CHAPTER 1

Foundations of Collaboration

With the number of hours that we spend together working, it's critical to build durable and long-lasting relationships with colleagues. This facilitates an understanding of the work an organization needs to do, and helps individuals plan for lifelong careers in the industry. The bedrocks of good, collaborative relationships are trust, empathy, and psychological safety. In this chapter, we briefly define each of these and explain how to foster them at work. We end with tips for how to promote effective, collaborative communication.

Empathy

Empathy is that ability that helps us understand how another person feels and imagine what they might be thinking, and improves our emotional connection with others. Here are the most common and effective methods for increasing empathy that can be applied in the workplace:

Listen

Active listening—really concentrating on what someone is communicating, not just passively hearing what they're saying or planning our response—builds our empathy and gives others room to express their message.

Ask questions

Cultivating curiosity and asking questions can help us challenge our assumptions and biases. Example questions to ask yourself: *What context could I be missing from that decision? What uncon-*

scious biases might I have that are affecting my opinions on this? Example questions to ask others: *What led you to choose X over Y? Could you tell me a bit more about your thought process on this?*

Appreciate individual differences

There are significant benefits to be gained from diverse teams in terms of creativity, problem solving, and productivity, though our differences can at times lead to short-term interpersonal conflicts. We can teach ourselves to appreciate these differences. Each of us has a different cultural background with unique experiences that inform our choices of how and why we work. Working with, genuinely listening to, and imagining ourselves as the various people that we work with can help us break down our biases, both conscious and unconscious.

Trust

Trust is the belief that those we depend on will meet the expectations we have of them. Building trust can increase the resiliency of a team and increase the likelihood that the team will accomplish its goals. Without trust, individuals can be protective of their projects or areas of responsibility, often to the detriment of their health or the team's overall productivity. Here are a few things to keep in mind when building trust within teams:

Swift trust

In devops we sometimes need to practice trust with people with whom we don't have a past working relationship. Swift trust, first explored by professor of organizational behavior Dr. Debra Meyerson,[1] occurs in short-term or short-lived teams in which trust is assumed from the beginning. A developer could join a new team with low trust and preconceived expectations of how operations generally works with development, or they could choose to adopt a default of trust and then verify its appropriateness based on the other team members' behaviors and words. The latter is an example of swift trust.

1 Meyerson, Debra, Karl E. Weick, and Roderick M. Kramer. *Swift Trust and Temporary Groups*. Thousand Oaks, CA: Sage Publications, 1996.

Trust, but verify

In this approach, you start from a place of cautious trust, sharing responsibilities; then you strengthen that trust as you verify the behaviors and actions of the individual(s) with whom you're working. This goes hand-in-hand with swift trust, and works best for short-lived teams because it is focused on desired outcomes rather than relationships. Someone who waits until trust has already been "earned" before granting access to resources or responsibilities (e.g., sharing commit access to a project they've been working on) might find themselves with a chicken-or-egg problem: how can someone earn trust if they're not given opportunities to earn trust because they aren't yet trusted?

Self-disclosure

Being open enough to share things about ourselves can encourage feelings of trust and intimacy between people and increase cooperative and collaborative attitudes. Be aware of the dangers inherent in power differentials between leadership and individual contributors. Guilting folks into sharing more than they are able or encouraging folks to trust or assume best intent without recognizing these differences can effectively derail the efforts being made. (You can read more about this in a thought-provoking article on *The Bias*.[2]) There is a balance that we must achieve when practicing self-disclosure in the workplace: not enough disclosure can cause others to feel uncertain, but too much or inappropriate disclosure can damage trust and credibility. (Camille Fournier has written about problems that can affect trust,[3] and you can read about collaborative strategies and tools to nurture trust besides manager READMEs in a blog post by Lara Hogan.[4])

Perception of fairness

Employees need to trust that they are being treated fairly. When people understand the requirements of their role and the process for pay increases and promotions, they are less likely to feel

2 Annalee. "How 'Good Intent' Undermines Diversity and Inclusion." *The Bias*, September 27, 2017. *http://bit.ly/2XwRzxW*.

3 Fournier, Camille. "I Hate Manager READMEs." *Medium*, November 21, 2018. *http://bit.ly/2XOBgkn*.

4 Hogan, Lara. "Tools for Introspection." [Personal blog.] n.d. *http://bit.ly/2Jn4JK1*.

unfairly overlooked. Developing formalized roles, job levels, and pay scales and providing reasonable transparency in these areas can go a long way toward improving employee satisfaction. Another key to a perception of fairness is sharing risk. If two teams are sharing work or resources but only one will be negatively affected if their project fails, the team with more risk might be distrustful of the team with less risk.

Psychological Safety

Psychological safety (a term coined and defined by Harvard Business School professor Amy Edmondson) is "a belief that one will not be punished or humiliated for speaking up with ideas, questions, concerns, or mistakes."[5] It is the feeling that an environment or team is safe enough for people to open up and take risks. It is what allows people to show vulnerability, to admit that they don't know something, to ask a question to which they might worry that they "should" know the answer, and to make mistakes and talk about those mistakes openly rather than trying to hide them.

NOTE

A 2015 internal study at Google[6] analyzed more than 180 Google teams, trying to establish what makes teams more or less effective. The researchers determined that psychological safety is one of the five key foundations of a high-performing team—and the most important.[7]

What does a team without psychological safety look like? People might spend hours struggling with a technical problem because they don't feel comfortable asking for help, or might waste time solving the wrong problem because they didn't understand a key point at a meeting and didn't want to admit it. Here are a few tips for building psychological safety in your devops teams:

5 Edmondson, Amy. "Building a Psychologically Safe Workplace." TEDx, May 4, 2014. *http://bit.ly/2Lhrmlc*.

6 Rozovsky, Julia. "The Five Keys to a Successful Google Team." re:Work (Google), November 17, 2015. *http://bit.ly/2Xz2VGx*.

7 "Tool: Foster Psychological Safety." re:Work (Google), n.d. *http://bit.ly/2LSWwPF*.

Encourage an "ask" culture
: Having achieved a secure position and social capital within your company, you have the privilege of asking questions. Even if you have all the answers, one of the greatest gifts you can give to new employees is to pretend that you don't, and ask questions on topics that need clarification or are critical to understand. Model this asking in visible ways during group meetings or in Slack channels.

Find shared connections
: Look for ways to interact with your team members beyond work to find shared connections. This doesn't need to be forced, as with icebreaker questions or team-building exercises. For distributed teams, getting regular video or voice time with colleagues or having a dedicated #chatter Slack channel to have general nonwork discussions can help.

Solicit feedback
: Ask people for feedback so that they know their opinions are valued. Providing explicit opportunities to share can help build an environment in which people view one another as collaborators, not competitors.

> **NOTE**
>
> Things like shared coffee breaks, shared lunches long enough to both eat and talk, and opt-in activities for people with common interests can go a long way toward building strong communities. Pay extra attention to how you can do this for remote employees—distributed teams do require extra work to develop and maintain healthy communication practices. Tools such as Donut can be paired with Slack to facilitate remote pairing, and a variety of video chat tools can help remote teams get face-to-face time.

Communication

Effective communication allows people to build shared understanding and find common goals, as opposed to working only in competition with one another. Aside from simply answering a question or telling somebody what to work on next, there are many different reasons why we communicate with one another—to increase understanding, assert influence, give recognition, and build community. Whatever the motivation, communication is an essential part of col-

laboration. Here are a few ways to improve communication and foster collaborative working relationships:

Incentivize collaborative communication
: Try publicly thanking people who bring an issue to someone's attention. People want to be recognized for their work and their contributions, so recognition of good communication is important to sustaining those practices.

Be wary of an interrupting culture
: Pay attention in meetings, and count how often people interrupt one another. Frequent interruptions require that time be spent repeating things that were said while someone else was talking, and can cause people to speak loudly simply to be heard. An interrupting culture tends toward competitive, rather than collaborative, communication. Cutting down on interrupting not only increases understanding, but also helps people feel that they are being heard, increasing trust and empathy.

Leaders can wait to speak
: Senior engineers or leaders within the organization can practice waiting for others to speak before sharing their own opinions. People won't always feel comfortable voicing disagreement to someone with more organizational clout than them. This is especially true of people newer to an organization, whose fresh eyes often bring a new perspective and valuable insights.

Practice communicating as "remote by default"
: This means using remote-friendly methods of communication like email and group chat for as much communication as possible and as the first choice of method, not a last resort. If an employee posts a question in the team's chat room instead of walking over to a colleague's desk, employees on distributed teams can learn and participate in discussions that they wouldn't otherwise be part of.

Summary

A foundation of trust, empathy, and psychological safety is required to establish a collaborative culture and foster devops within your organization. People in environments that are higher-trust and more supportive overall tend to be better at communicating effectively. An organization in which people actively look for ways to support and

help one another throughout the software and product development life cycles is an organization built on collaboration. Without team psychological safety, people won't take the risk of asking the questions critical to clarifying goals and objectives. In Chapter 2, we look at practical ways to apply collaboration principles in the software development life cycle.

CHAPTER 2
Collaboration in Practice

Although devops is a cultural movement and not something that can be created or purchased with specific tools, collaboration principles can be put into practice throughout the software development life cycle that can help engineering effectiveness in concrete ways.

The development pipeline is the progression of stages a product goes through, from design to delivery. No one pipeline describes every development environment, but many environments do share common patterns.

Generally, product development progresses in stages. Regardless of what methodology a team chooses to follow, people will establish a set of stages or gates to qualify whether software is ready to progress within the pipeline. In modern environments, these stages could look something like the model shown in Figure 2-1.

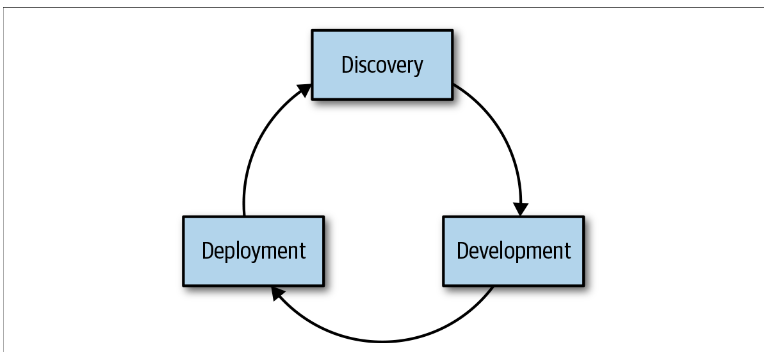


Figure 2-1. Stages in the software development life cycle

Let's look at each stage more closely:

1. In the first stage, we have discovery. Collaborative discovery could look like a team discussing a minimum set of requirements to be delivered within a set period of time, like a week or two weeks, depending on the frequency of delivering a working feature.
2. In the second stage, we have development. Collaborative development could involve pairing on code and test development, code reviews, and work segmented into small pieces that can be accomplished within the set period of time and with the work visualized on a project board. The code is submitted for review, and additional tests are run against it. Testing might be manual or automated and done from different perspectives, depending on the particular testing goal (e.g., performance, security, compliance, or functionality of the product). If the tests are successful, the code is merged into the main branch. An artifact can be built from the merged code at this point.
3. In the third stage of our example development pipeline, we have deployment. Deployment occurs after the code passes its tests. We might have a manual or scheduled gate that promotes the artifact into our production environment or releases it to the customer.

Within your environment, you can have many more stages, and every stage has opportunities for individuals to collaborate. This chapter is full of actionable advice on how to collaborate more effectively at different stages. Let's begin with discovery.

Collaborative Discovery

The collaborative process begins long before engineers start writing code in their editors of choice. How do we figure out what we should work on right now to further business value? Between maintaining the current product and triaging incoming requests, prioritizing work over time can be challenging—especially with the weight of expectations coming in from other teams within the organization. We need to continuously maintain an array of different relationships within the environment to ensure the successful delivery of the desired outcomes.

In the discovery phase of a product within an organization, processes and activities include design, requirements gathering, architecture review, and project planning. These processes might be defined differently across industries, and sometimes even across departments within a single organization. The degree to which these processes have been formalized will vary as well.

> **NOTE**
>
> Organizations of any size can have "islands"—separate parts of the company with less-than-ideal communication with other departments. The larger the organization, the more opportunity there is for these areas to form. A group within an organization that comes up with a product or project and makes all of the decisions on it without consulting other stakeholders can signal the presence of an island within the organization. If you recognize islands in your environment, more focus and time might be needed to build bridges to facilitate change.

Roles and Responsibilities

Many larger organizations will have a product team or department that is responsible for coming up with a strategy, roadmap, or other similar documents for products and product features. Product teams focus on user experience (UX) with the product, growing the user base, and prioritization of new and modified features for competitive advantage. Having a clear strategy is key to creating a successful product—this strategy will later inform what gets built and with what constraints. An overly vague strategy such as "we want to be the best product in our field" offers insufficient guidance and leads to individuals building in gaps that can result in misalignment.

Product strategy and design alone are no guarantees of success. An idea without implementation and execution is just an idea. Your product team should not work within a silo, where they come up with product requirements and "throw them over the wall" to engineering to implement, just as developers should not throw software over the wall for operations to maintain. It is essential to cultivate and maintain a collaborative relationship between all parts of the organization.

Product and Engineering Collaboration

Some strategies that can help collaboration between different departments include the following:

- *Share tools between departments.* If a product team uses Jira and engineers track work in GitHub projects and neither group has access or visibility to the other's tools, that cuts down on shared context, with materials "hidden" from the other department because of the different tools used. If sharing tools doesn't work, ensure transparency by making sure that adequate training is available so that the different teams can easily use each other's products.
- *Encourage pathways for communication between groups.* Representatives from engineering should have invites to regular product planning meetings, and a product representative should have invites to engineering status updates.
- *Define and share the process for making decisions and acceptable forms of criticism.* As mentioned earlier, it's vital that everyone who wishes to has the opportunity to share their thoughts about decisions in the manner that will best convey the information they have.
- *Make sure that the people who do the work are included in defining schedules.* Design and engineering are responsible for the implementation of their work and will have valuable input as to whether a timeline is feasible. Product has critical information about customers' needs and the impact on business value. When everyone participates in creating and signing off on product plans before they are considered final, this eliminates the surprises that harm relationships between teams when impossible deadlines are missed.

Just as collaboration between different parts of engineering is vital, maintaining communication and trust between engineering and other departments in the organization is necessary for a healthy organization.

NOTE

Every organization defines team structures and processes that require different strategies for collaboration. Making processes *explicit* rather than *implicit* makes them easier to reason about, to communicate to new members of teams, and to change if necessary. Document processes and keep the documentation up to date!

Requirements Throughout the Organization

Requirements gathering typically comes in the form of thinking about user requirements. In an organization with Agile-based practices, this might look like user stories: statements of the form *As a user, I want to <some goal> so that <some reason or motivation>*. User stories are brief descriptions of desired functionality from the perspective of a user. A single story describes the desired experience for a user in a shared language that both business and technology experts can understand. Telling different stories about our users helps us validate the work that we are doing in language that everyone on the team can understand. These stories help us plan the work associated with our project and guide our work, but they don't specify any implementation.

Within the discovery phase, we explore our assumptions about what we are building, ensuring that we get sufficient points of view. We need to be in alignment with the work as individuals and a team to ensure that we have the same goals in mind as we work. Through exploring user stories, we also can identify the crucial areas that will identify when our work is done, or our acceptance criteria. If we don't do this work and talk through the different personas and their stories, we can expend a great deal of effort and time on work that will ultimately end up being thrown away.

Product teams need to ensure that they understand the engineering requirements for bringing a new product or feature to fruition. Work also needs to be prioritized based on what's in progress and its value to the organization and the availability of engineers, especially any with specialized knowledge. This might mean asking questions like these:

- What else is engineering working on and what other projects or deadlines will affect their work capacity and availability?

- Will the new product or feature require substantial changes to existing code or infrastructure? If significant code refactoring, a new infrastructure component, or major architectural changes will be required, that can have a big impact on time frames.
- Does the team have the necessary expertise to develop and maintain a new feature already, or will it need to consider training or hiring? For example, if a product team wants to add support for an Android app after being iOS-only for a while, it is likely that either Android engineers will need to be hired or time given for on-staff engineers to come up to speed on a new platform.
- What is the overall stability of current products? An engineering organization that is spending significant time fighting fires will be less equipped to take on new projects.

Engineering is not the only department whose requirements need to be considered. Support teams play crucial roles in a company's success, but their demands are too often overlooked. You will want to ask similar questions of your support organization and its capabilities. Does it have the required expertise to support a new feature or product, to be able to answer user questions, and to help customers troubleshoot? What is the current average support volume?

As much as possible, it is important to maintain relationships between the design, product, engineering, and support departments to make sure everyone understands one another's needs and requirements so that teams can work together to help solve customer problems. Make sure silos don't build up between departments as you work to eliminate them between teams within a department.

Architecture and Planning

Whereas design and requirements gathering tend to be about the "what" and the "why" of the product or feature development, architecture and project planning address the "how." In general, implementation specifics and details should be left in the hands of people who have the necessary domain expertise to make those decisions (generally the engineers who will be doing the bulk of that work), but that doesn't mean there aren't opportunities to collaborate throughout this part of the process, as well.

One of the goals of architecture and project planning should be figuring out how to build and test the product in a sustainable manner. As we've discussed previously, it's important to avoid throwing work over a metaphorical wall. After a product or feature is built, what happens to it? Part of the engineering process is making sure it is understood how something will be run when it is in production in front of customers. How will it be monitored? What are the debugging and troubleshooting processes? How often will it be updated, and what will those maintenance workflows look like? Is the full life cycle of the product understood?

Defining the project

We need to set clear boundaries around projects and make sure that everyone involved has a shared understanding of what those boundaries are. Taking time to clearly define and document various aspects of a project's scope at the beginning can decrease the number of misunderstandings and the amount of wasted work throughout the project. Things you will want to define when planning a new product or feature include the following:

- What does "done" mean? When can a team say that it has finished working on one stage and move on to other work?
- Is it more important to ship something by a certain date with fewer features, or is it more important to ship something feature-complete and past the deadline?
- Who will make decisions when issues come up along the way? Are the stakeholders who worked on the original design or idea the same ones who will take responsibility for modifications?
- Who will own a new product or feature after it has been shipped? Who will be responsible for maintaining and operating it, and what will those responsibilities entail?

Again, these questions and definitions should likely involve people from multiple departments insofar as new products do not exist in a vacuum. If a particular engineering team will be responsible for maintaining a new product, service, or supporting infrastructure component, it is crucial that it has the opportunity to help clarify the scope of work. Involving key stakeholders like this from the beginning can help prevent unpleasant surprises down the line, especially when it might be too late to change direction quickly. You don't want

to find out after months of work developing something that you'll lack the bandwidth to properly support it after launch.

> **NOTE** Having templates for project proposals, design documents, decision records, or your organization's planning documents of choice in a public, shared location can help keep multiple groups on the same page. When teams share a project proposal template, collaboration becomes easier, as all teams will get the information that they need to plan or make decisions.

Planning and review processes

Having clearly defined processes in place (and documented where people can easily find them) is vital to collaborative efforts at scale. A process does not necessarily mean "too much bureaucracy" or that an organization is dealing with so much red tape that it is unable to move forward with anything. The right amount of process can make sure that different groups know what they need to provide to one another and how to best work to maximum collective benefit.

We recommend that organizations, especially at the enterprise level, implement architecture and operability review processes as part of their product planning and development cycles.

Architecture reviews are for when significant changes are planned to the architecture or underlying infrastructure of your products. Significant changes shouldn't come as a surprise because that can affect trust and how people work. People rewriting core parts of an application in their new favorite language and similar impactful changes are a sign that communication has severely broken down. Taking the time to think through the ramifications of such changes in advance is not "too much process" designed to gatekeep engineering progress; instead it is a sign of mature engineering practice.

> **NOTE** Check out John Allspaw's blog post on Etsy's architecture reviews[1] for more ideas about the kinds of elements that should go into your planning and review process.

---

1 Allspaw, John. "Multiple Perspectives on Technical Problems and Solutions." *Kitchen Soap*, August 12, 2017. *http://bit.ly/2L9j13f*.

Operability reviews should be considered throughout the planning process to help uncover issues with the architecture that might hinder managing and monitoring the application in production. These reviews ensure that everyone understands what it means to operate something in production, who will be responsible for it, how to troubleshoot and maintain it, and similar topics. The goal isn't arbitrary gatekeeping. By incorporating operability reviews earlier in the planning rather than right before production, we can reduce wasted work and incorporate changes into the design of the product to facilitate everyone being able to understand and operate the product better.

> **NOTE**
>
> In addition to strengthening interteam relationships within engineering, pay attention to the relationships that engineering has with other departments. Sales and product divisions often come up with ideas for new products and features, but they need to be certain that what they are suggesting is realistic based on the engineering constraints of the current team. For example, a sales team should not make specific promises for new features or releases without engaging the engineering teams as well as understanding the impact on any features, maintenance, or technical refresh in progress.

The specifics of who should participate will vary from company to company; however, it's important to be sure that you find ways to give interested parties the chance to take part in discussions without making the discussion processes overly long or demanding. Here are some ways to accomplish this:

- Have designated representatives from interested groups participate; for example, have one or two engineering managers or senior engineers take part in a planning meeting with the product team rather than entire development or operations teams. Document who will be participating in each step of the process and why.
- Share notes or recordings from discussions and planning meetings as widely as possible. Often, people will appreciate being able to know what's going on and this visibility can reassure them that their interests will be represented and that they don't need to actively participate themselves.

- Make it clear to whom people should talk if they have comments, questions, or other concerns throughout the discussion processes.
- Provide templates for discussions when possible and give relevant groups the opportunity to collaborate on these templates. For example, if there are a few questions that engineering always asks when a product group proposes a new feature, make those questions part of the product discussion template up front. One of the goals here is to minimize surprises.
- When asking for feedback from different teams, be as specific as possible about where in the process you are. Is this the beginning of an idea that product has been brainstorming, or is there a big-customer deal riding on this feature that needs to be implemented by a specific date?

Design artifacts

Having concrete, searchable artifacts that come out of processes such as project planning, brainstorming, and design sessions can be incredibly helpful, not only for building up long-term organizational memory, but also for making these processes more collaborative. In larger organizations or ones that are distributed across multiple time zones, it can become more challenging to get all the necessary stakeholders or participants in one place at the same time.

Look for ways in which your teams can produce concrete artifacts that other teams and individuals can read and contribute to asynchronously. For example:

- Templates for discussions, meeting formats, and similar could be kept in Google Docs, allowing groups who are planning an instance of a discussion to make their own copy of the document as needed.
- If your organization has an internal documentation repository (whether this is Google Docs, Dropbox, an internal collaborative wiki, or similar), make sure you have a standard location where all templates and other artifacts should be kept.
- Whenever possible, try to have conversations in media that are designed to be searchable and asynchronous: use email or GitHub issues rather than Slack or other chat software.

Collaborative Development

Collaborative development begins with version control. Trying to write code together in an environment without source control and a shared integration process is a recipe for frustration and increased friction. In the development phase, teams collaborate to write code and tests, review code,[2,3] and integrate code.[4] In this section, we focus on test-driven development and continuous integration, although collaboration isn't limited to these specific practices.

Increased Transparency

Often in enterprise environments, there are lingering issues with transparency, especially around how open teams are with their code and work practices. This can hinder collaboration and result in wasted work. The first step in increasing transparency is adopting one version control tool across the organization. The second step might be adopting practices common in open source communities —an approach known as *innersource*. With innersource, most code isn't locked down to just the developers working on it. Everyone is encouraged to contribute. Access controls are placed in integrating code, which allows teams to balance ownership while increasing the opportunity for different perspectives and contributors.

Because projects are worked on in the open, teams can start their projects using existing software within the organization. Progress isn't limited, and individuals can fork code as necessary; they don't need to start from scratch every time.

Another benefit of having work in the open is that it allows for increased transparency around the decision process regarding what gets worked on and why. This builds trust and alignment across teams. Even when teams are not cross-functional, this allows security, database engineers, and operations to collaborate directly with developers and testers.

2 Cleek, Billie. "How to Conduct Effective Code Reviews." *The Digital Ocean Blog*, March 28, 2018. *https://do.co/2LdU86q*.

3 Lum, Tracy. "How to Give and Get Better Code Reviews." *Medium/Hacker Noon*, July 3, 2018. *http://bit.ly/2XJWfAC*.

4 McMinn, Keavy. "How to Write the Perfect Pull Request." *The GitHub Blog*, January 21, 2015. *http://bit.ly/2xHxDxG*.

Test-Driven Development

Often, the collaborative nature of establishing acceptance criteria during the discovery phase leads to the first tests that drive test-driven development (TDD). Formally, TDD refers to a specific software development methodology involving a red, green, refactor development cycle:

1. Write a new test. This is done before writing any code that is part of a new feature. At this point, the new test should fail because the feature it tests hasn't been written yet. This is the "red" stage.
2. Write code for the new feature. At this stage the focus is on writing code that will allow the test to pass, and no more. This is the "green" stage.
3. The tests should now be run and they should all pass—both the new test for the new code and all existing tests (there should be no degradations).
4. If necessary, code can be refactored, optimized, or otherwise cleaned up at this point, taking care not to cause any test degradations in the process. This is the "refactor" stage.

Although it is not necessary to strictly follow TDD, the practice of writing tests *as code is written* throughout the development process rather than all at once at the end is widely considered to be a good strategy. Testing should not be an afterthought, and waiting until the very end of a development cycle to see whether new code has caused any regressions can end up wasting lots of development and testing time.

When developers are pairing or otherwise collaborating on writing code, tests should also be written and run. Consider strategies that include the following:

- Verify that there are ways to run tests locally, such as on developer laptops, before code needs to be merged upstream. Ensure that your testing environment allows for developers to run subsets of the full tests that exist—such as testing just the class or module that they are working on. The quicker it is for engineers to run tests, the more likely it is that they will actually test their changes! Quality testing can improve the quality of code sub-

mitted back to the shared repository, improving the affective trust individuals have in one another.

- Document your testing patterns internally. If you use a particular testing framework or library, confirm that engineers know what that is and what conventions they should be following when they write their tests.
- Testing shouldn't be solely for people with the job title of software developer. If your site reliability engineers, system administrators, or operations engineers are writing code, that code should have tests as well. Even making sure that bash scripts are run through a linter before they are committed and deployed can help over time.
- Not all tests can or should be automated. It's an important skill to develop exploratory testing.

Continuous Integration

Continuous integration (CI) is the practice of merging new changes to code into the main or master branch continuously (or as often as is reasonably possible) rather than waiting until the very end of a long development cycle to do so. In modern development environments, that might mean dozens of merges into the master branch per day. In larger organizations CI practices may differ from team to team. This is one area in which teams should standardize on the process so that ad hoc teams can be formed and quickly follow the processes that have been established.

CI is key to being able to work quickly, effectively, and collaboratively. When engineers are working, they will test, commit, and merge small units of work as they are completed—such as adding a new method and a test for that method—rather than waiting until an entire new customer-facing feature is completed. Other developers can then pull down those changes and merge them into their own development branches, making sure that everyone is working on the most up-to-date version of the codebase possible.

This prevents the problem of "integration hell," where dozens or hundreds (or even thousands) of changes are integrated all at once at the end of a project, and only *then* do people run integration tests. With hundreds of changes being implemented at a time, it's difficult to pinpoint which change broke something (or even to tell that

something is broken at all!). On the other hand, if there was only one small commit between the tests passing and the tests starting to fail, it's trivial to identify which commit contained that breaking change.

CI allows developers, testers, and operations staff to collaborate on minimizing risk while incorporating change through reducing the size of each change.

Collaborative CI practices

Following are some improvements that you can apply within your organization to level up the collaborative practices in CI:

- *Use a single CI platform.* Whether it's a cloud-based CI service or self-hosted on-premises, everyone using the same tool can greatly improve collaboration between teams. When different teams have different tools for CI, it makes it harder for individuals to use tools effectively across teams. It's important to reduce complexity as much as possible with the tools that we use to build software, especially when the services we are building are complex themselves (for example, a service that is composed of many microservices). This way, when a problem exists in one part of the service, we can quickly see the state rather than trying to figure out the status across different tools.
- *Define infrastructure as code (IaC).* Reduce differences between local, test, and production environments. Infrastructure configuration should have testing.
- *Convert manual tests to scripted tests whenever possible.* Automate scripted tests. When code is committed and pushed back to the shared repository, tests should be triggered for that repository to run automatically. Don't rely on people remembering to run tests when computers can remember for them.
- *Eliminate friction of style choices by implementing automated linting on integration.* This depersonalizes style choices and improves the quality of code through discovery of potential logic flaws.
- *Invest time in refactoring or removing flaky tests.* Most tests are code, and should be maintained like code to minimize friction. If tests are slowing down feature development and release or if there is fear of modifying the test suite to incorporate new tests,

you will likely see slow progress toward building features and implementing bug fixes, which impedes your ability to build customer value.

Collaborative Production

In the deployment phase, we deploy the product with manual or scheduled gates and monitor and observe the product in use. Remember, work doesn't end after the deploy button has been pushed. The moment that code is live in production, you must monitor it to verify that it is running properly, and you will need to observe and debug it if something goes wrong. In this section, we focus especially on what happens after code is in production: on-call responsibilities, retrospectives, and organizational learning.

Continuous Delivery and Deployment

Continuous delivery and continuous deployment are often confused, in part because they are both referred to as "CD." It's important that everyone is on the same page when it comes to understanding what is being implemented when folks talk about "an initiative to implement CD." Let's take a closer look at the two processes:

Continuous delivery
: Changes to software are integrated and tested automatically such that the resulting artifact is ready to be deployed. There might be a manual gate that promotes the software to the production environment. This is also the more appropriate process when the artifact is being delivered to a customer who identifies when they want the software to be installed.

Continuous deployment
: Changes to software are integrated and tested automatically and the resulting artifact is promoted into production automatically. This is often viewed as the next level of maturity from continuous delivery.

Whether continuous delivery or continuous deployment, CD allows the entire team to collaborate on seeing code in use as quickly. Loss of knowledge is minimized, as features and bug fixes were worked on recently, rather than months previously. Automation of the build, verification, and deployment steps allows the team

to improve its confidence in the change process, which affects how folks handle bug fixes, outages, and other incidents as well.

Which version of CD makes the most sense for you will depend on the details of your environment, applications, and deployment processes, but it is recommended that code be continuously integrated and as easy to deploy quickly as possible. Processes that allow for rapid code deployment also provide rapid breakfixes and outage resolution.

Collaborative Deploy Processes

Making deployments collaborative doesn't mean that one person watches while another person runs the deployment. Rather, it is about making certain that there is visibility before, during, and after the process. Here are a few tips for increasing visibility:

- Chat is a tool that a team can use to encourage a culture of collaboration. It's also useful in the deployment process. You might choose to have a fully chat-based process, using chatbots in #deploy channels so that anyone in the channel can see what is being deployed, and when. If you don't do that, you should consider having chat integration, such as a chatbot that announces deploys that were performed by other deployment tools, again so that people know what is changing.
- Make sure that your deploys are visible in your monitoring and observability tools. Being able to see a visual marker in a Graphite graph that shows when a deploy was made would help provide context if there were a marked change in behavior of the application.
- For high-risk deploys, buddy systems in which individual team members pair up during the deploy process allows for individuals to separate out the different aspects of the deployment, seeing the impact versus the execution. This can reduce the time to discovery of issues and provide the necessary support to resolve them, with one person being responsible for communicating out necessary changes and impact while the other rolls back the upgrade or deploys a fix.

  Although "graph watching" should generally not be required for every deploy (after all, that's what automated monitoring and alerting tools are for), there are certainly times when people will

feel better knowing that someone is keeping an eye on the most important metrics, such as when flipping the switch at the end of a big migration project.

The goal with collaborative deployment is informing individuals of expected change to minimize surprises. Surprises erode trust and can affect the practice of empathy.

Creating Sustainable On-Call Environments

Nearly every organization today has some sort of on-call requirements, in which engineers are expected to be available to respond to issues, usually in their off-hours. Historically, on-call was a very siloed practice and only system administrators had on-call responsibilities. This led to resentment and adversarial relationships when people felt they were being treated unfairly, and to burnout, health issues, and poorer work outcomes as long-term sleep deprivation made its effects known. Empathy allows us to break these patterns and create on-call environments that are humane and sustainable.

Humane Staffing and Resources

A driving factor of devops was the adverse effects that historical software development practices had on those operating the software or servers it ran on. Practices such as CI and IaC have improved this, but there are still other considerations.

Availability and maintenance

When we run websites that our users expect to always be available, the question of when to perform maintenance that requires any kind of downtime or will affect even a subset of services often comes up. From a strictly user-facing perspective, it would make sense to perform maintenance at a time that would affect the fewest users. So, a company in the United States whose highest traffic times are during US daytime hours might want to perform maintenance during the late night or early morning in US time zones, but that wouldn't necessarily be the case for a company with a primary user base in Asia.

However, that might not be ideal from the perspective of those performing the maintenance. This is not simply a matter of people being unwilling to stay up late or get up early; it's a matter of how alert, responsive, and effective people can be if they aren't suffi-

ciently rested. From an operator's point of view, they should be doing critical maintenance operations when they are most alert and awake. If that strictly isn't possible based on the needs of the users, such as when a maintenance task would take too long and the financial losses would be too great, there are still ways to mitigate the costs to the maintainers.

Employees should be compensated fairly for the off-hours work they need to do. If it is expected as a regular part of their work, that expectation should be made clear in the job description so people can assess whether the position is the right fit for them. Allow and encourage people to take care of themselves and their health—for example, by having them take off the day after they perform late-night maintenance. Make sure that, if at all possible, maintenance tasks are spread out enough or the team is large enough that people have sufficient time to recover between these off-hours shifts. Depending on your circumstances, covering transportation or meal costs for these events would do well, too. If job roles change and someone ends up getting off-hours work added to their responsibilities, their compensation should be adjusted to match.

Work–life balance

It's important to keep work–life balance in mind when planning your headcount for the upcoming year. Capacity planning is just as important for people as it is for servers. If people on your teams will be required to regularly work evenings and weekends in addition to weekdays in order to meet expectations, that is a recipe for poor work, poor morale, and burnout. Devops is about creating sustainable work practices, and how individuals are expected to approach their work–life balance is a key part of that.

Even though many jobs in operations-related fields have traditionally required off-hours work—either for maintenance, on-call, or nonstandard shifts to provide 24/7 coverage—it is important to keep in mind that these kinds of requirements can be unintentionally biased against people with substantial responsibilities outside of work. Young, single people without children (or high-maintenance pets) will find it much easier to devote their off-hours to work than people with partners, children, or other family responsibilities. People with longer commutes or health considerations will also be more adversely affected by these kinds of requirements. Part of growing and maintaining a diverse and inclusive team requires taking these

things into account and considering how your job requirements can be adjusted to be more inclusive.

Team size considerations

Having sufficient people responsible for rapid responses to alerts and incidents, either by participating in an on-call pager rotation or having multiple shifts of people working throughout the day, is another consideration. This is one area in which larger companies often have it much easier—a larger company is more likely to have multiple teams in different global offices, making a "follow-the-sun" rotation relatively straightforward to implement.

In this kind of setup, multiple teams (often three) that are distributed around the world will each work during their normal daytime working hours, being physically far enough apart that the end of one shift will coincide with the beginning of the next shift's hours. This enables the teams to collectively provide around-the-clock coverage, without requiring people to work during their local nights.

Even a mid-sized company is likely to have a full team of operations engineers or system administrators who can participate in an on-call rotation, but at smaller or younger companies, this is unlikely to be the case. Although you might not think you have enough operations work for a full operations team, you should avoid having only one person be responsible for on-call duties. This will likely mean sharing on-call responsibilities among as many people as necessary to give individuals a chance to recover and catch up on sleep.

> **NOTE**
>
> Even in the short term, sleep deprivation can lead to difficulty concentrating or performing well, irritability or anxiety, and an increased risk of high blood pressure or heart attack—and these effects compound when sleep deprivation is sustained long term.
>
> Health in general, and burnout specifically, is a very real consideration in understanding the overall health of your organization. Prioritizing the short-term financial or material gains for your company over the long-term health of the people within it will lead to long-term losses.

Patterns for Sustainable and Collaborative On-Call

In this section, we share specific practices to build up a collaborative and sustainable on-call rotation. It's critical to avoid having any single individual or group feel like they are shouldering an unfair burden, because that can affect the relationships we want to build between individuals and across the teams that support a specific product or feature.

On-call onboarding

Right from the get-go, it is important to introduce on-call responsibilities in a compassionate and psychologically safe way. If you are bringing new team members into an existing rotation, let them know that they aren't expected to be able to respond to every alert themselves right away. Create an environment in which they are able to ask questions.

New people can be incredibly valuable in situations like on-call where alert fatigue can build up and people can get used to the way things have been. If you create a safe environment, you can get new team members asking questions like "Why do we have this alert when we never seem to respond to it?" or "Why isn't there a graph for X?" and myriad other questions that can illuminate ways to improve your monitoring, observability, and alerting for everyone.

It's a good idea to maintain living documentation about your on-call process, including things like how to get any required equipment, any tools/systems/dashboards that a person might need access to, and what the expectations are around response times. Ideally, a new person in the rotation will be able to get all the information they need before they take the pager for the first time, but if you don't have such a document, new members can help build one up as they learn the ropes. If, rather than adding new team members to an existing on-call rotation, you are creating an on-call rotation for the first time, you can use the idea of an onboarding document to help you define the parameters of your on-call service.

On-call buddies

If you are bringing new people into an on-call rotation or spinning up a new rotation from scratch, you might want to take advantage of a buddy system. For existing rotations, this can be a form of specific mentorship in which someone familiar with the processes and

responsibilities can share their knowledge with someone new. This can decrease stress for new people by giving them backup (here's the specific person you escalate to first if you run into an alert that you don't know how to handle) and give them an explicit place to ask questions.

Defining ownership

With any on-call rotation, it's important to explicitly define roles and responsibilities. You don't want multiple people responding to the same alert and potentially stepping on each other's progress!

If your organization is one in which each team has enough people to sustain its own individual rotation, you can begin by saying that each team is on call for its own products and services. However, it becomes more complicated when teams are too small to do this sustainably, such as in teams with fewer than four people. It might not be sustainable or humane to be on call 24/7 for 1 week out of every 4 (as opposed to every 8 or 10), especially for people with more responsibilities outside of work.

It's important to think about what experience and expertise is required across a specific service. Ideally, we don't have individuals who are domain experts because we can end up burning them out by always needing them to be available for specific alerts.

Measuring On-Call Impact

There are a few ways to measure the impact of supporting specific services and determining how many people are needed to sustain a humane on-call rotation that encourages collaborative practices. Identify which services alert outside of working hours and the associated costs and values. What is the impact of the service not being available to customers? What is the impact on the on-call engineer? How long does the service take to repair? How quickly are individuals expected to respond to alerts, and what happens if they miss an alert? There should be a fair amount of analysis of alerts in general, including identifying flaky alerts and eliminating alerts that are not actionable. It might be possible for a service to move to an unowned state with no on-call support, or one that is non-alertable outside of business hours. This should be explicitly documented so that it's not a surprise to anyone (including any executives who have to answer to angry customers!).

Another metric to uncover is how often specific individuals are required for specific alerts. This is a signal that there is a single point of failure within the system and time should be spent leveling up other members of the team to understand the problem.

Creating a supportive environment

It can be incredibly stressful to feel the weight of responsibility of a critical service without the support of others when on call. In the spirit of the site-reliability engineering mentioned earlier, it's important to create a culture in which people actively and gladly support one another. You want to have an environment in which people's response to an escalating incident is "How can I help?" rather than "What did you do wrong?" Every engineer at some point will run into something that they can't handle on their own, and being able to ask for help is critical to encourage and sustain a psychologically safe team, which has an impact on all the other non-on-call work that we do.

Collaborative Retrospectives

Whether you call them retrospectives, postmortems, learning reviews, or something else, the medium-term processes of following up after incidents are crucial to creating an on-call experience that is sustainable rather than one that causes burnout. Without any sort of effective follow-up, incidents are likely to keep recurring, creating an unnecessary burden on the people responsible for responding to them. An effective retrospective involves a written timeline of the incident and a review meeting during which the meat of the discussion and learning happens.

Creating the incident timeline

The first step to having a retrospective, after the immediate response has been taken care of, is creating a timeline of what happened. This will involve pulling together sources such as alerts from PagerDuty, chat logs from relevant incident response or operations engineering Slack channels, and any other written context that is necessary to paint a clear picture of what happened when. Generally, the person who was the active on-call will lead in putting together the timeline, as most of the collaborative discussion will happen during the review.

Facilitating an incident review

The main part of an incident review is a facilitated meeting (often called the postmortem or retrospective, though the preparation of the timeline is an important step that should not be forgotten). The main parts of the meeting include the following:

- *Reviewing the timeline.* A skilled facilitator will ask questions designed to expand understanding of not only what happened, but why. Blamelessness is a key concept here. Assuming that people don't intentionally make mistakes, we want to understand why they took the actions they took, whether that be why they thought a particular change was safe to deploy or why they looked at a particular dashboard when troubleshooting. A successful timeline review will be more of a group discussion than a monolog by whoever was on call during the incident.
- *Calling out things that were surprising.* If an incident didn't have anything surprising or confusing happen during it, it would be called a planned maintenance. The things that surprised us are the things that we can learn from. These events can be discussed during the timeline review, but they should be noted and explicitly called out afterward, as they usually lead into...
- *Planning remediation items.* From the things that were unexpected, concrete next steps can be discussed. For example, if you were surprised that you didn't have a graph of error rates for a particular service, creating one might be a good remediation item. (Not every surprise needs its own remediation item, though.) It's important not to go into an incident review with remediation items already planned, using the review as a way to justify that work. There's also no need to create a certain number of items to make a review look more "productive." Keep in mind that in order for remediation items to be effective, you will need to prioritize that work. If people become used to ignoring remediation items, it weakens the entire incident review process and makes it likely that incidents will repeat.

Ideally, everyone who participated in the incident response should attend the review meeting for that incident. In the event that calendar availability or time zones prevent this from happening in a timely manner (within a week is best so that the timeline is still fresh in people's minds), it is better to have a smaller review sooner than it

is to wait weeks or months for calendars to align, because people can forget context and incidents can recur in the meantime.

> **NOTE**
>
> The process of facilitation for incident review or postmortem meetings is beyond the scope of this book. It's an important skill that you should not overlook to maximize the utility of the postmortem process, and this guide from Etsy[5] is a great place to start.

Organizational Learning

Every organization has its own body of institutional knowledge—the things that the people in the organization collectively know and understand. One of the main goals of processes like incident reviews is to increase the body of organizational knowledge, to help the organization learn. Without organizational learning, the same issues are likely to happen over and over. This might look like a production incident that keeps happening in the same way because nobody was able to take steps toward fixing it, or one team struggling with issues that another team has already experienced and solved.

Psychological safety is very important in creating a learning environment. Consider a workplace in which employees are screamed at, fired, or otherwise punished for making mistakes or even for bringing issues to light so that they can be addressed. If people don't feel safe speaking up when they make a mistake, they might instead deliberately hide problems and certainly won't feel safe asking for help fixing them. This naturally does not help make the systems better-performing or more resilient.

The goal of on-call rotations and incident reviews is not to punish anyone or to add extra work: it is to facilitate organizational learning. When systems break or behave in unexpected ways, that provides opportunities to make them more robust, to improve the tools or documentation around them, to share knowledge among more people who need it. A healthy and sustainable on-call culture is continuously improving and learning, requiring everyone involved to view one another as fellow learners and collaborators, rather than having an adversarial mindset.

5 Allspaw, John. "Etsy's Debriefing Facilitation Guide for Blameless Postmortems." Etsy/Code as Craft, November 17, 2016. *http://bit.ly/2j6YEq2*.

Institutional memory

Institutional memory describes how well the organization as a whole remembers what it has learned from past experiences. In an organization with poor institutional memory, it is easy to end up in situations in which people know "this is the way we always do things" but nobody really remembers *why*. Poor institutional memory makes systems and organizations fragile because it is an understanding of historical contexts and trade-offs that helps build resiliency.

Building and maintaining an organization's institutional memory is a collaborative effort. A primary goal is to ensure that relevant knowledge is shared widely throughout the organization and not limited to one team or, even worse, one person. To help facilitate this learning, make certain that artifacts are stored in a manner that makes them accessible throughout the organization, searchable, and available in the long term. These artifacts can include the following:

- Timelines, meeting notes, and recordings from postmortems and retrospectives.
- Alert configurations. If possible, these should be in source control so that commit messages can explain why a given alert was added.
- Decision records, project proposals, design documents, and similar planning artifacts. Ideally, these will include discussions of alternatives that were considered and other contextual information that can help inform future engineers as to *why* a particular decision was made.

Summary

We've shared specific collaborative practices that enforce and support the bedrock of collaboration: trust, empathy, and psychological safety. These collaboration principles are not just theoretical; they are meant to be applied in the real world, and they work. Think through the different stages in your software development process, reviewing every step along the way. Identify where you are feeling the most painful interactions that are hindering your relationships and growth, and start applying the practices that you can incorporate into your work now. This is just a representative subset of prac-

tices to encourage collaboration; look for other opportunities to foster trust, empathy, and psychological safety within your teams.

CHAPTER 3

Conclusion

More than just a sea change around software development practices, the principles and ideas found in devops touch all parts of an organization and can be used even by large enterprises or government agencies. Collaboration at both individual and team levels is a key element of any devops transformation, but it is just one piece of the foundation.

No matter what the specifics of your organization's culture or journey might look like, the end goal is not to have some fixed number of deploys per day, to use a specific open source tool, or to do things simply because other organizations have been successful doing them. The end goal is to create and maintain a successful organization that solves a problem for your customers. Take the time to proactively define your goals and the values and ideas that you want to help you get there, regardless of your industry or size. Don't wait until you find that your implicit values have been defined for you and it feels too late to change them.

Devops is about invitations to be involved in the ongoing change process, gratitude for wins that occur in every team within the organization, and explicit rejection of bullying behaviors. As with a garden, it takes continued feeding, watering, and weeding to nurture the organization toward sustainable growth and business success. And just as buying a bouquet of precut flowers cannot be considered gardening, buying a tool that claims to be a "devops solution" isn't devops. It is the ongoing work to build and maintain a culture that makes devops truly effective.

We can all work to transform our organizations and the industry itself to be more productive, sustaining, and valuing. Please share your stories with us at *authors@effectivedevops.net*. We hope that this book inspires you to explore working and learning with others as one part of your effective devops strategy.

APPENDIX A
Further Resources

Here are some further resources for learning more about collaborative culture and devops:

- Friedman, Ron. “Schedule a 15-Minute Break Before You Burn Out.” *Harvard Business Review*, August 4, 2014. *http://bit.ly/2XBbThF*.
- Greaves, Karen, and Samantha Laing. *Collaboration Games from the Growing Agile Toolbox*. Victoria, BC: Leanpub/Growing Agile, 2014.
- Gulati, Ranjay, Franz Wohlgezogen, and Pavel Zhelyazkov. “The Two Facets of Collaboration: Cooperation and Coordination in Strategic Alliances.” *The Academy of Management Annals* 6, no. 1 (2012): 531–583. *http://bit.ly/facets-collab*.
- Heffernan, Margaret. “Why It’s Time to Forget the Pecking Order at Work.” TED-Women 2015, May 2015. *http://bit.ly/heffernan-pecking*.
- Hewlett, Sylvia Ann. “Sponsors Seen as Crucial for Women’s Career Advancement.” *New York Times*, April 13, 2013. *http://bit.ly/nyt-sponsorship*.
- O’Daniel, Michelle, and Alan H. Rosenstein. “Professional Communication and Team Collaboration.” In *Patient Safety and Quality: An Evidence-Based Handbook for Nurses*, edited by Ronda G. Hughes. Rockville, MD: Agency for Healthcare Research and Quality, US Department of Health and Human Services, 2008. *http://bit.ly/comm-collab*.

- Popova, Maria. “Fixed vs. Growth: The Two Basic Mindsets That Shape Our Lives.” BrainPickings.com (*http://brainpickings.com*), January 29, 2014. *http://bit.ly/fixed-vs-growth*.
- Preece, Jennifer. “Etiquette, Empathy and Trust in Communities of Practice: Stepping-Stones to Social Capital.” *Journal of Computer Science* 10, no. 3 (2004).
- Schawbel, Dan. “Sylvia Ann Hewlett: Find a Sponsor Instead of a Mentor.” Forbes.com (http://forbes.com), September 10, 2013. *http://bit.ly/hewlett-sponsor*.
- Silverman, Rachel Emma. “Yearly Reviews? Try Weekly.” *Wall Street Journal*, September 6, 2011. *http://bit.ly/wsj-reviews*.
- Stone, Douglas, and Sheila Heen. *Thanks for the Feedback*. New York: Viking, 2014.

About the Authors

Jennifer Davis is a senior cloud advocate at Microsoft. Previously, she was a principal site reliability engineer at RealSelf, developed cookbooks to simplify building and managing infrastructure at Chef, and built reliable service platforms at Yahoo! Jennifer is a core organizer of devopsdays and organizes the Silicon Valley event. She is the founder of CoffeeOps and coauthor of *Effective DevOps* (O'Reilly).

Ryn Daniels is a staff infrastructure engineer at Travis CI, where they solve interesting operational problems at scale. Previously, they were a senior operations engineer focusing on infrastructure deployment and monitoring. They have written and spoken about operations, organizational learning and postmortems, and engineering culture, and also coauthored *Effective DevOps* (O'Reilly).