O'REILLY®

Effective DevOps

BUILDING A CULTURE OF COLLABORATION, AFFINITY, AND TOOLING AT SCALE

Jennifer Davis & Katherine Daniels

Effective DevOps

Building a Culture of Collaboration, Affinity, and Tooling at Scale

Jennifer Davis & Katherine Daniels

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Effective DevOps
by Jennifer Davis and Katherine Daniels

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://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com .

Editor: Brian Anderson
Production Editor: FILL IN PRODUCTION EDITOR
Copyeditor: FILL IN COPYEDITOR
Proofreader: FILL IN PROOFREADER
Indexer: FILL IN INDEXER
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

February 2016: First Edition

Revision History for the First Early Release Edition
2015-05-05: First Release
2015-07-22: Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491926307* for release details.

978-1-491-92630-7

[FILL IN]

Table of Contents

CHAPTER 1

Introduction

<section data-type="sect1"> <aside data-type="sidebar"> <h5>Early Release Edition</h5>

This book is a work in progress – new chapters will be added as they are written. We welcome feedback – if you spot any errors or would like to suggest improvements, please let us know. </aside>

<h1>Who This Book Is For</h1>

This book is aimed primarily at managers and individual contributors in leadership roles who see friction within their organizations and are looking for concrete, actionable steps they can take towards implementing or improving a devops culture in their work environment. However, individual contributors of all levels who want practical suggestions for easing some of the pain points they face will find actionable takeaways.

The audience is made up of a mix of professional roles, as devops is a professional and cultural movement that stresses the iterative efforts to break down information silos, monitor relationships and repair when misunderstandings arise between teams in an organization. Many may be leaders within their organizations who have worked closely with developers or operations engineers.

The book covers a wide range of devops skills and theory, including an introduction to the basic ideas and concepts. It is assumed that you will have heard of the term devops and perhaps have a rudimentary understanding of devops, tools and processes used in the field.

By the end of Effective Devops, our hope is that you will have a solid understanding of what having a devops culture means practically for your organization, how to encourage effective collaboration to help indivdiual contributors from different back-

grounds and teams deal with different goals and working styles to work together productively, how to help teams collaborate to maximize value between them while increasing employee satisfaction and balancing conflicting organizational goals, and how to choose tools and workflows for your organization that complement your organization. </section>

<section data-type="sect1"> <h1>How this Book Is Organized</h1>

This book is broken down into several parts, starting with an introductory chapter and then covering each of the pillars of devops.

 Chapter I, What is Devops Chapter II, Collaboration: Individuals Working Together Chapter III, Hiring: Choosing Individuals Chapter IV, Affinity: From Individuals to Teams Chapter V, Tools: Choosing and Using Them Chapter VI, Scale: Scaling Everything UP </section>

<section data-type="sect1"> <h1>Conventions Used in This Book</h1> </section>

<section data-type="sect1"> <h1>Using Code Examples</h1> </section>

<section data-type="sect1"> <h1>How to Contact Us</h1>

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://effectivedevops.net. *

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*. </section>

<section data-type="sect1"> <h1>Acknowledgements</h1>

Effective Devops would not have been possible without the help and guidance of many friends, colleagues, and family members. </section> </section>

CHAPTER 2
What is Devops?

What is devops? Some might define it as a software development method, while others might think that it is a set of tools and technologies such as configuration management and continuous delivery. We would argue instead that devops is a cultural movement that seeks to improve both software development and the professional lives of the people involved in the field. In order to fully understand what we mean when we're talking about devops, it is necessary to understand not only what the concept means and how it is used today, but also the history of how it came to be.

No cultural movement exists in a vacuum. Technology is part of culture. We are born, accept what is and introduce new cultural aspects as we live. The way that our overall culture influences technology, and technology influences culture shapes the fabric of how we live our lives.

Pre-1880, walking was the primary mode of transportation. Cities were compact with residences and workplaces intermingled. Streets were narrow and inconsistently paved. As automobiles were introduced, cities made decisions about the organization and infrastructure either in favor of automobiles or existing pedestrians. These days, some cities forgo infrastructure planning that factors in pedestrians at all creating busy roadways with no safe way to travel by foot. Walking shaped the city and how we worked, and the arrival of new technology changed the landscape accordingly.

Devops is part of the cultural weave that shapes how we work and why. While devops does involve certain tools and technologies, an equally important part of our culture is our values, norms, and knowledge. Examining how people work, the technologies that we use, how technology influeces how we work, and how people influence technology can help us make intentional decisions about the landscape of our industry.

This chapter will delve into the evolution of software engineering as it pertains to the history of devops, define the terms and ideas closest to the movement, and address

some common misconceptions that people often have about the subject. Unlike most histories on the topic, which often start slightly before the first DevOpsDays conference in 2009, we will go back further so the reader can gain a richer understanding of the ideas and principles that have shaped our industry over time, and how that continued evolution has changed and grown, making devops not only necessary but inevitable.

The History of Devops

In this section we will examine the history of the industry and how the recurring patterns and ideas have shaped the devops movement.

Developer as Operator

In the beginning, the developer was the operator. Early programmers were actually known as human computers. Jean Bartik, one of the original programmers of the Electronic Numeric Integrator and Computer (ENIAC) learned how to program the machine by reviewing hardware and logic diagrams of the device. Programming the machine and its 18,000 vacuum tubes meant setting dials and changing out cable connections across 40 control panels. At the time, the focus was on the hardware engineering and not on the software that Bartik, Kathleen Antonelli, Frances Holberton, Marlyn Meltzer, Frances Spence, and Ruth Teitelbaum built on the ENIAC. These women felt the pain to manage the systems as they had to replace fuses and cables. There were literal bugs in the system.

The Advent of Software Engineering

> "Coming up with new ideas was an adventure. Dedication and commitment were a given. Mutual respect was across the board. Because software was a mystery, a black box, upper management gave us total freedom and trust. We had to find a way and we did. Looking back, we were the luckiest people in the world; there was no choice but to be pioneers; no time to be beginners.
>
> —Margaret Hamilton

In 1961, President John F. Kennedy set the challenge that within the decade that the United States should land a man on the moon, and return him safely to Earth. With a deadline and no employees with the necessary skills, the National Aeronautics and Space Administration (NASA) needed to find someone to write the onboard flight software needed to accomplish this task. NASA enlisted Margaret Hamilton, a mathematician at Massachusetts Institute of Technology (MIT) to lead writing this critical software. In her pursuit of writing this complex software, Hamilton is credited for coining the term "software engineering". She also created the concept of **priority displays**, software that would alert astronauts to update configurations in realtime. She instituted a set of requirement gathering that included:

- debugging all individual components
- testing individual components prior to assembly
- integration testing

Space flight was not the only area software was becoming critical. As hardware became more available, people became more concerned about the impending complexity of software that did not follow standards across other engineering disciplines. The growth rate of systems and the emerging dependence on these systems was alarming. In 1967, discussions were held by the NATO Science Committee comprising of scientists across countries and industries to do an assessment of software engineering. A Study Group on Computer Science was formed in the fall of 1967. The goal was to focus attention on the problems of software. They planned a conference inviting 50 experts from all areas of industry. The 3 working groups of this conference were the Design of Software, Production of Software, and Service of Software. The goal was an effort to define, describe, and set in motion solving the problems with software engineering.

At the NATO Software Engineering Conference of 1968, key problems with software engineering were identified including:

- defining and measuring success
- building complex systems requiring large investment and unknown feasibility
- producing systems on schedule and to specification
- economic pressures on manufacturers to build specific products.

In 1969, the software that Hamilton's team wrote overrode a manual command that could have led to the Eagle not landing on the moon. The freedom and trust that the management gave to the team of engineers working on the onboard flight software and the mutual respect between team members, led to software that facilitated one of mankind's leaps in technology as Neil Armstrong stepped on the moon.

The interactions between management and engineering were studied further by Diane Vaughan, an American sociologist who did extensive research on the technical and cultural aspects that led to the space shuttle Challenger disaster in 1986. In January of that year, the space shuttle orbiter Challenger broke apart 73 seconds into its flight, killing its 7 crew members. While others did much to evaluate the technical reasons for the disaster, Vaughan was interested in the human side of things.

When investigating whether or not there was any misconduct during the months leading up to Challenger's launch, Vaughan found out that all of the managers involved had been complying with NASA's requirements throughout. These requirements included rules about how to make decisions about the technical risks of the equipment involved with space flight. Vaughan said, "We discovered that they could

set-up the rules that conformed to the basic engineering principles that allowed them to accept more and more risk. So they established a social normalization of the deviance, meaning once they accepted the first technical anomaly, they continued to accept more and more with each launch. It was not deviant to them. In their view, they were conforming to engineering and organizational principles. That was the big discovery. I concluded it was mistake, not misconduct."

In addition to this idea of the normalization of deviance, Vaughan has done research on early warning signs that point to issues later on down the road. Early signs are often considered in the air traffic control industry, where people and organizations are trained to identify these early warning signs to try and avoid having small mistakes turn into large catastrophes. Looking at the ideas of failure and risk from a human, sociological point of view in addition to a technical one was something that would be of great value in many industries going forward.

The Age of the Operating System

In 1979, Usenet was started by university students- Tom Truscott, Jim Ellis, and Steve Bellovin. It started out as a simple shell script that would automatically call different computers, search for changes in files on those computers, and copy changes from one computer to another using UUCP (Unix-to-Unix copy, a suite of programs allowing for file transfer and remote command execution between computers). To improve performance it was then rewritten in C. Ellis gave a talk on the "Invitation to a General Access UNIX network" at an academic Unix users group known as Usenix. This was one of the first ways to communicate and share knowledge across organizations with computers and grew rapidly in use.

While this tool started to facilitate the sharing of knowledge across universities and corporations, this was also a time where how companies were run was considered part of the secret sauce. Talking about solving problems outside of the company was not done, because knowledge of both the problems and the solutions was viewed as part of the competitive advantage. There was an intentional cultural drive for competitors to work inefficiently. This stymied a great deal of collaboration and limited the effectiveness of the communication channels that were available. This cultural siloization reflected in how companies grew with complexity.

As systems complexity grew, this led to the need for specialization of skills and role proliferation. This saw the formation of the system administrators specializing in systems management and minimizing costs of systems, and the software engineers specializing in creation of new products and features to address the new needs. Other more specialized groups were siloed off as well with the NOC (network operations center), QA, security, databases, and storage all becoming separate areas of concern.

In 1985, the [National Science Foundation Network](*http://www.nsf.gov/about/history/nsf0050/internet/launch.htm*) (NSFNET) was founded to promote advanced networking in the United States.

This created the institutional Tower of Babel with the different silos all speaking different languages due to differing concerns. Along with this siloization, the specific pains of software and the hardware that it ran on was also separated. No longer were developers exposed to the late night pages of down systems from being on-call, or the anger expressed by unsatisfied users. Additionally, programming's trend towards higher level languages meant that development became more abstracted, moving further and further away from the hardware and the systems engineers of the past.

In an effort to be proactive and prevent service outages, system administrators would document the set of steps required to do regular operations manually. System administrators borrowed the ideas of "root cause analysis" from total quality management (TQM). This led in part to additional rigor and attitudes towards minimizing risk. Lack of transparency and change management became the entropy demon that needed to be defeated.

The Beginnings of a Global Community

As interconnected networks allowed programmers and IT practitioners to begin sharing their ideas online, people began looking for ways to share their ideas in person as well. User groups, where practitioners and users of various technologies could meet to discuss their fields, began to grow in number and popularity. One of the biggest worldwide user groups was DECUS, the Digital Equipment Computer Users' Society, which was founded in 1961 with members consisting mostly of programmers who wrote code for or maintained DEC computer equipment.

The US chapter of DECUS ran a variety of technical conferences and local user groups (LUGs) throughout the United States, while chapters in other countries were doing the same globally. These conferences and events began to publish the papers and ideas presented at them in the form of DECUS proceedings, which were made available to members as a way of sharing information and growing the total knowledge of the community as well as the interconnectedness of its members. A similar community specifically for system administrators was found with the Unix Users Group (founded in 1975 and known today as USENIX) and its special interest group, the System Administrators Group (known later as SAGE and today as LISA). Separately, NSFNET "Regional-Tech" meetings evolved into the North American Network Operators' Group (NANOG), a community specfically for network administrators to increase collaboration to make the Internet better.

Contrary to the focus on knowledge sharing that was a primary feature of these local and global user groups, there was at the same time a great deal of secrecy in technology companies regarding their practices. The 1987 film Wall Street exemplified these

ideals, with the character of Gordon Gekko famously saying, "Greed, for lack of a better word, was good". Companies, in their quests for their own financial and material successes, kept their processes as closely-guarded secrets.

Companies kept their business practices and technical best practices to themselves, because if their competitors had inefficient practices, that meant their own relative success would be more likely. Employees were strongly discouraged or even explicitly forbidden from sharing knowledge at industry conferences to try and maintain this sort of competitive advantage. This is in stark contrast to more recent developments, where communities and conferences are growing around knowledge sharing and cross-collaboration between companies.

Trade Secrets and Proprietary Information

Information that is not generally known to the public that is sufficiently secret to confer economic or business advantage is considered a trade secret. Information a company possesses, owns or holds exclusive rights to is considered proprietary. Software, processes, methods, salary structure, organizational structure, customer lists are examples of items that can be considered a company's proprietary information. For example, proprietary software is software for which the source code is generally not available to end users. All trade secrets are proprietary; not all proprietary information is secret.

In addition to the changes in culture in the industry, commoditization and the costs of knowledge and technology impact what companies keep secret within their organizations.

The Age of Applications and the Web

In the late 90s, with the ease that new web applications could be created, people at companies that needed to be able to grow and change quickly to reflect the rapidly changing market had a problem they needed to overcome.

It was a time of angst and frustration. Endemic in system administration, we had the culture of "No" and "It's critical to preserve stability". In 1992, Simon Travaglia started posting a series on Usenet called The Bastard Operator From Hell(BOFH) that described a rogue sysadmin who would take out his frustration and anger on the users of the system. Toxic operations environments led to individuals viewing the rogue sysadmin as a hero and emulating behaviors.

In development, we had a culture of "It's critical to get these changes out" and "I don't want to know how to do that because I'll get stuck doing it". In some environments this led to developers risking systems by finding unofficial ways to work around the processes in place. This led to additional massive cleanups which further solidified

the idea that change is extremely risky. For the singletons in either group that tried to make change to the overall processes, they'd find themselves stuck in the mire of becoming the subject matter expert, locked into the positions of support that became critical to maintain.

In 2001, an invite went out to folks interested and active in the Extreme Programming (XP) community and others within the field. XP was a form of Agile development that was designed to be more responsive to changing requirements than previous development software methodologies, known for short release cycles, extensive testing, and pair programming. Seventeen software engineers got together in Snowbird Utah to discuss software development. They summarized their shared common values to capture the adaptiveness and response to change that they wanted to see in development with an explicit emphasis on human factors. This Agile Manifesto was the rallying cry that started the agile movement.

In 2004, Alistair Cockburn, a software developer who was one of the co-authors of the Agile Manifesto, described Crystal Clear a software development methodology for small teams based off of 10 years of research with successful teams. It described 3 common properties frequent delivery of usable code, reflective improvement, and **osmotic communication** between developers. Crystal also described 3 priorities safety, efficiency, and habitability.

This movement continued in software development for several years, and later began to have its influence felt elsewhere. In 2006, Marcel Wegermann wrote an essay on how to take the principles of Crystal Clear, Scrum, and Agile and applying them to the field of system administration. Along with Agile and Scrum, it contained several ideas that Wegermann argued could be applied to system administration as well as software development, wanting to bring newer and better practices to the field. In addition to giving a lightning talk on the subject where he suggested ideas such as version control for the Linux operating system's `/etc` directory, pair system administration, and operational retrospectives, he also started the Agile System Administration mailing list.

As web technology continued to grow and evolve, the ways that people communicated and collaborated online did too. Twitter, an online social networking service was introduced to the world not even a year later in 2006. At first it seemed very much like a tool for people wanting to share information in an abbreviated format, for short attention spans or for celebrities to reach out to fans. In 2007, however, the South by Southwest Interactive conference saw the use of Twitter skyrocket as Twitter placed screens in the hallways streaming twitter messages. Twitter quickly became a way for ad-hoc communities to be formed across the globe. For conferences, it was a way to get additional value out of the multi-track systems and connect with like minded individuals. The hallway track, a phrase often used to describe the interactions and conversations that take place in the hallways of conferences, had expanded from the

physical world to the web where anyone could discover and participate in these ad-hoc interactions.

Agile Infrastructure

At the Agile 2008 conference in Toronto, Andrew Shafer, a former software developer who was starting to take a great interest in IT concerns, proposed an Agile Infrastructure session. At the same conference, Patrick Debois spoke on incorporating scrum into operations "Agile Operations and Infrastructure: How Infra-gile are You?". Patrick worked with development and operations teams on a project to test data center migration. One day he would be working on agile development with the developers and the next day he would be firefighting with the operations team leading to a lot of context switching; the switching from one process or task to another. Context switching for humans cost anywhere from 5 to 30 minutes in lost productivity per context switch. Observing that he wasn't the only one interested in Agile system administration, Patrick contacted Andrew out of band to discuss agile system administration.

Around the same time, individual companies were beginning to not only make great strides towards processes that allowed them to keep up with the increasingly rapid changes of the internet, but were also beginning to share some of their stories publicly through communities that were building up around popular conferences like the [O'Reilly Velocity Conference](*http://velocityconf.com/*).

One such company was Flickr, a popular community site for photographers. After being purchased by Yahoo in 2005, Flickr needed to move all of the services and data from Canada to the United States. John Allspaw, a web operations enthusiast who had worked in systems operations for years, had joined the company as the Flickr Operations Engineering Manager to help with its scaling and now was charged with this massive migration. At the time, Flickr hosted over 3 billion photos with 40000 photos being served per second. Paul Hammond joined the Flickr Development team in 2007, and became the Flickr Engineering manager in 2008 heading the development org in collaboration with Allspaw.

Hammond and Allspaw co-presented at Velocity Santa Clara 2009 "10+ Deploys per Day," highlighting the revolutionary change that allowed the team to move rapidly. They didn't do this by setting out to break down silos or start a big professional and cultural movement. They were able to collaborate a great deal in their work at Flickr, which was in contrast to Allspaw's previous experiences at Friendster, where emotions and pressures ran high and there was little in the way of inter-team collaboration. The opportunities to work together that presented themselves were something that both managers took advantage of. Neither of them woke up one day and decided that things needed a big change, but rather they recognized the little pieces of working together that made things work well. They took note of these little things that they did together, which ended up becoming much bigger cultural changes.

The Beginning of DevopsDays

> "Don't just say *no*, you aren't respecting other people's problems… #velocityconf #devops #workingtogether"
>
> —Andrew Clay Shafer (@littleidea)

This tweet, from Andrew Shafer on June 23 2009, caused Patrick Debois to lament that even though he was watching remotely, he was unable to attend that year's Velocity conference in person. Paul Nasrat, at the time a lead systems integrator at the Guardian, tweeted in reply, "Why not organize your own Velocity event in Belgium". Inspired Patrick did just that, creating a local conference that would allow for developers, system administrators, toolsmiths and others in those fields to come together. In October of that year, the first DevopsDays conference took place in Ghent. Two weeks later, he wrote:

> "I'll be honest, for the past few years, when I went to some of the Agile conferences, it felt like preaching in the desert. I was kinda giving up, maybe the idea was too crazy: developers and ops working together. But now, oh boy, the fire is really spreading."
>
> —Patrick Debois

That first DevopsDays event ignited the powder keg of unmet needs, people separated in silos frustrated with the status quo identified with devops and a way of describing the work they felt they were already doing. The conferences grew and spread as individuals started up new DevOpsDays across the world. With the availability of the real-time communication platform of twitter, the hallway track never ended and #devops took on a life of its own.

As we reflect on history, we see the trend of a focus on the outcome. Many have seen the "10+ deploys a day" presentation from John Allspaw and Paul Hammond and taken away from it that the importance was the 10+ deploys per day, the quantity of deployments in a day. The title "10+ deploys a day" was the hook which pulled people in to see the presentation. The content of the talk how they achieved something that seemed impossible and not a simple metric. You can't declare "doing devops successfully" simply because you are "doing 10 deploys a day". This fixation on an outcome increases the stress to the human who is already stressed out because workflow of the organization doesn't allow for the adventurous dedication and commitment enabled with freedom and trust to be happy and productive humans building new leaps for mankind. As we will show in later chapters, focusing on the processes is a better way of thinking, because devops is about how we do things, and why, not just what we end up doing with those things.

Foundational Devops Terminology and Concepts

In order to effectively discuss the tools and techniques required to implement devops, it is necessary to define some of the major terms and concepts surrounding it. When

we take a look at the history of both software development and operations, we notice several broad concepts there that should be examined in order to understand the evolution of devops. Some of these terms are not foundational concepts of devops itself, but anti-patterns that devops tried to overcome. This section will discuss those concepts as well as ideas related to devops as it exists today and end with a definition of devops itself.

Waterfall

The waterfall methodology or model is a software development process with an emphasis on a sequential progression from one stage of the process to the next adapted from hardware engineering. One of the driving forces behind this model was the idea that bugs were easier to fix the earlier in the development process they were discovered, and so sought to ensure that each stage of the process would be completely finished before any work was started in the next. The original stages were requirements specification, design, implementation, integration, testing, installation, and maintenance, and progress was visualized as flowing from one stage to another.

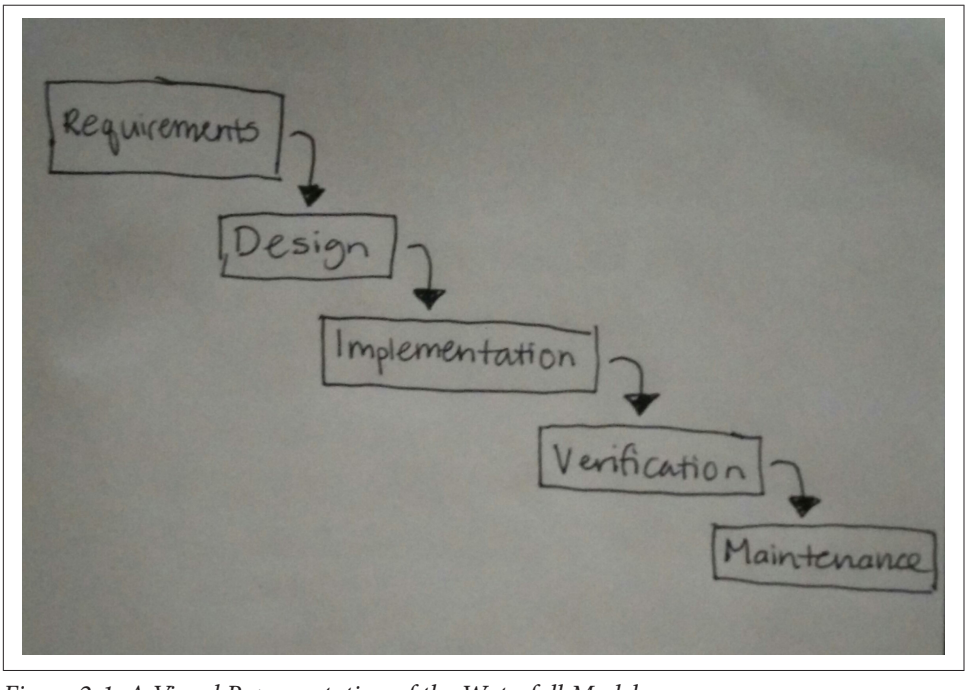

Figure 2-1. A Visual Representation of the Waterfall Model

Software development under the waterfall model tended to be very highly structured, with a large amount of time being spent in the requirements and design phases, with the idea that if both of those were completed correctly to begin with it would cut

down on the number of mistakes that would be found later. Part of the appeal of this stemmed from the high costs of delivering and changing software that was distributed on CD-ROMs or floppy disks to be installed by hand. Since fixing a bug on such software would require, for example, manufacturing and distributing another CD-ROM with a software patch, it was much more cost-effective to spend the time to get the design and specifications right up front.

The waterfall method makes sense in these cases where the cost of delivering software is high or for projects with requirements unlikely to change. As with every methodology, the waterfall method has its pros and cons.

Some advantages of using the waterfall method is its emphasis on documentation, discrete understandable phases and milestones. Documentation alleviates knowledge lost when individuals leave or new employees join a project. With understandable discrete phases, project members can finish a phase of the project and pass on to other members at the significant milestones.

Project manager Mary Lotz has argued that gathering and specifying all requirements in advance of any work being done is often the most difficult part of any software development project. Customers often don't know their exact requirements, at least not enough to make sure that they are 100% completely specified, and requirements will often change over time. As software is developed, limitations may be discovered that cause requirements or deadlines to change.

Projects that require more flexibility might benefit from examining some more iterative or agile methodologies as well. As we will examine further in later chapters, a key part of devops is being able to assess and evaluate different tools and processes to find the most effective one for your environment, but it isn't so rigidly defined as to prohibit any methodologies, even older ones such as waterfall.

Extreme Programming

- Communication
- Feedback
- Simplicity
- Courage
- Respect

Lean

The idea of lean originally stemmed from lean manufacturing, which was a system for eliminating waste within a manufacturing process. The Toyota Production System of the 1990s, also called "Just In Time" production is perhaps the best-known exam-

ple of lean manufacturing, with the main goals of the process being to eliminate waste and design out inconsistency. Lean systems focus on the parts of the system that add value by eliminating waste everywhere else, whether that be over-production of some parts, defective products that have to be re-built, or time spent waiting on some other part of the system. Stemming from this are the concepts of lean IT and lean software development, which apply these same concepts to software engineering and IT operations. Waste to be eliminated in these areas can include unnecessary software features, communication delays, slow application response times, or overbearing bureaucratic processes.

ITIL

ITIL, formerly known as Information Technology Infrastructure Library, is a set of practices defined for managing IT services. It is published as a series of five volumes which describe its processes, procedures, tasks, and checklists, and is used to demonstrate compliance as well as measure improvement towards that end. ITIL grew out of a trend that saw the growing number of IT organizations in the 1980s using an increasingly diverse set of practices. The British Central Computer and Telecommunications Agency developed a set of recommendations as a way to try to standardize these practices. First published in 1989, the books and practices have grown over the years, with the five core sections in the most recent (2011) version being service strategy, service design, service transition, service operation, and continual service improvement.

IT analyst and consultant Stephen Mann notes that while there are many benefits that come with ITIL's standardization and while there are over 1.5 million ITIL certified people worldwide, it has some areas where practitioners again might want to put additional focus. Mann noted that ITIL is often more on the side of being reactive rather than proactive, so we suggest organizations who have been using ITIL take note of ways that they can try to add more proactive planning and customer focus to their practices, as we will cover in later chapters.

Agile

Started with the writing of the Agile Manifesto in 2001, agile is the name given to a group of software development methodologies that are designed to be more lightweight and flexible than previous methods such as waterfall. The developers that created the Manifesto wrote:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over Processes and tools
>
> **Working software** over Comprehensive documentation
>
> **Customer collaboration** over Contract negotiation

> **Responding to change** over Following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.
>
> —The Agile Manifesto

Agile methodologies included processes such as Scrum, Extreme Programming, and Feature-Driven Development. While this section is not intended to be a complete lesson on the history of software development, it is important to note that these new methods placed a heavy emphasis on collaboration, flexibility, and the end result of working software, ideas which are closely related to the core tenets of devops. In their 2011 paper, "Overview and Guidance on Agile Development in Large Organization," authors Barlow et al noted that many large organizations find agile practices to be too flexible or extreme, but the ideas behind them are continuing to evolve along with the field of software development.

Is Devops Just Agile?

Devops shares many common characteristics with the Agile movement, especially with the focus on individuals, interactions, and collaboration. You might wonder if devops is just "rebranded" Agile. While devops has certainly grown around Agile principles, it is a separate cultural movement steeped in history of software engineering with a broader reach that is inclusive of more than just developers. Devops extends Agile ideas and applies them to an entire organization, not only the development process. As we will see in detail in later chapters, devops has cultural implications far beyond what was seen with Agile and a focus that is broader than speed of delivery.

Community of Practice and Community of Interest

Communities of practice are groups of people who share the same role or concern and meet regularly to improve how they perform in an organization. Every role within an organization has the opportunity to form a community of practice, so there could be one community for developers, one for QA and testing engineers, and another for scrum masters. Communities of practice could also form around specific tools or languages, but in either case they are not restricted to people from any one project or team. These communities tend to work best when they are not mandated by management, but rather allowed to grow and change organically. Community activity may ebb and flow over time as roles and projects do. It is important to note that communities of practice are restricted to those people who are actively participating in the role that the community is focused on so that learning and discussion will come from peoples' real-world knowledge and experience.

A community of interest is similar to a community of practice, but instead of being limited to practitioners only, tends to be made up of people who are interested in the management, governance, and communication of the teams involved in an organization. They might take responsibility for overseeing or creating communities of practice, or discuss other higher-level issues that don't have as much effect on the day-to-day, real-world issues that practitioners are discussing. Some communities use the term in a different way, meaning a community of interest to be anyone who is interested in discussing a particular topic, team, or technology even if they don't practice it themselves. Both communities of practice and communities of interest are intended to be cross-functional, with emphasis being placed on learning and common goals.

It should be noted that while these communities of practice and interest are often seen in Agile organizations, they are not restricted to it. As an example, Linux user groups are communities of practice, often local, where active practitioners or users of Linux operating systems will discuss topics relevant to their work, such as dealing with Linux security issues or how to improve the performance of databases running on Linux systems. A community of interest might take the form of a Meetup group - a Python community of interest would probably include both professional Python programmers and people interested in playing around with or learning the language.

Blame Culture

A blame culture, or blameful culture, is one that tends toward blaming and punishing people when mistakes are made, either at an individual or organizational level. In this kind of culture, a root cause analysis as part of a post-mortem or retrospective is generally misapplied with the search for one thing that ultimately caused a failure or incident. If this analysis happens to point towards a person's actions as being the "root cause", that person will be blamed, reprimanded or even fired for their role in the incident. This sort of culture often arises from one that must answer to external auditors, or where there is some top-down mandate to improve performance according to some set of metrics.

Heavily siloed environments that lack an appreciation for transparency are fertile ground for blame culture. If management is set on finding one person or group of people to blame for each incident that occurs, in order to get rid of that "bad apple", individual contributors will be motivated to try to shift blame away from themselves and their own teams onto somebody else. While this sort of self-preservation is understandable in such an environment, it doesn't lend itself well to a culture of openness and collaboration. More than likely, people will begin withholding information about incidents, especially with regards to their own actions, in an effort to keep themselves from being blamed. Outside of incident response, a culture of blame that calls people out as a way of trying to improve performance (such as which developers introduced the most bugs into the codebase, or which IT technician closed the fewest tickets) will contribute to an atmosphere of hostility between coworkers as everyone

tries to avoid being called out. When people are too focused on simply avoiding having a finger pointed at them, they can't be focused as much on learning.

Silos

A departmental or organizational silo is a term describing the mentality of teams that do not share their knowledge with other teams in the same company. Instead of having common goals or responsibilities, siloed teams have very distinct and segregated roles. Combined with a blameful culture this can lead to information hoarding as a form of job security ("If I'm the only one who knows how to do X, they won't be able to get rid of me"), difficulty or slowness completing work that involves multiple teams, and decreases in morale as teams or silos start to see each other as adversaries. Often in a siloed environment you will find different teams using completely different tools or processes to complete similar tasks, people having to go several levels up the managerial chain of command in order to get resources or information from people on another team, and a fair amount of "passing the buck", moving blame or responsibility to another team.

The issues that can come from organizational silos take time, effort, and cultural change to break down and fix. Having software developers and system administrators or operations engineers be siloed, and trying to fix the issues in the software development process that came from that, was a big part of the root of the devops movement, but it's important to note that those are not the only silos that can be present in an organization. Cross-functional teams, discussed later, are often touted as being the anti-silos, but these are not the only two options, and in fact just because a team comprises only one function does not necessarily make it a silo. Silos come from a lack of communication and collaboration between teams, not just from a separation of duties.

The Old View and the New View

In his 2006 book, "The Field Guide to Understanding Human Error," professor of human factors Sidney Dekker laid out two ways that organizations usually approach issues. His "old view" describes a mindset in which human error is seen as something that causes systems to fail and needs to be eliminated, the idea being that mistakes are only made by "bad apples" who need to be rooted out and dealt with. This view is very often found in blameful cultures as it assumes that errors are caused by malice or incompetence and the individuals responsible must be blamed and shamed (or simply fired). In contrast to this is his "new view" which says that human errors are structural rather than personal - rather than making mistakes due to incompetence, people make the choices and take the actions that make the most sense to them at the time based on the circumstances they find themselves in, and that people should be educated and complex systems considered holistically when looking to minimize or respond to issues.

Root Cause Analysis

Root Cause Analysis(RCA) is a method to identify contributing and root causes of events or near-miss/close calls and the actions adequate to prevent recurrence. It's an iterative process that is continued until all organizational factors have been identified or until data is exhausted.

Organizational factors are any entity that exerts control over the system at any stage in its life cycle including but not limited to design, development, testing, maintenance, operation, and retirement.

One method of identifying root causes is the 5-Whys. This method employs asking "why" until the root causes are identified. It requires that the individuals answering "why" have sufficient data to answer the question appropriately.

A second and more systematic approach is to create a Ishikawa Diagram. Developed by Kaoru Ishikawa in 1968, this causal diagram gives a way to visualize and group causes into major categories to identify sources of variation, relationships among sources, and provide insight into process behaviors.

Often RCA is associated with determining a single root cause. Tools that provide event management often only allow a single assignment of responsibility. This limits the usefulness of root cause analysis as it focuses attention on the direct causes rather than the additional elements that may be contributing factors.

Human Error

Human error is a term often used as the root cause in a root cause analysis, being the idea that a human being made a mistake that directly caused a failure. With this often comes the implication that a different person would not have made such a mistake, which is commonly seen in a blame culture when somebody has to be reprimanded for their role in an incident. Again, this is an overly simplistic view, and is used prematurely as the stopping point for an investigation. It tends to assume that human mistakes are made due to simple negligence, fatigue, or incompetence, neglecting to investigate the myriad factors that contributed to the person making the decision or taking the action they did. In a blameful culture, discussion stops with the finding that a specific person made a mistake, with the focus often being on who made the mistake and the end result that it caused. In a blameless culture or a learning organization, a human error is seen as a jumping off point rather than an ending one, starting a discussion on the context surrounding the decision and why it made sense at the time.

Blamelessness

Blamelessness is a concept that arose in contrast to the idea of blame cultures discussed previously. Though it had been discussed for years previously by Sidney Dek-

ker and others, this idea was really brought to prominence with John Allspaw's post on blameless post-mortems, with the idea that incident retrospectives would be more effective if they focused on learning rather than punishment. A culture of blamelessness exists not as a way of letting people off the hook, but to ensure that people feel comfortable coming forward with details of an incident, even if their actions directly contributed to a negative outcome, because it is only with all the details of how something happened can learning begin to occur. We have to remember that the point of a post-mortem or retrospective after an incident is to prevent the same thing from happening in the future, and only if you accept the bad apples theory of Dekker's "old view" does it make sense to focus on the identification, blame, and removal of those bad apples. In the New View, with a focus on identifying many contributing factors and learning from each of them, blamelessness helps foster an environment where those factors can be brought to light.

Organizations used to the idea of blaming and punishing people for making mistakes, up to and often including firing them, might wonder if there is an exception to blamelessness for repeated mistakes. They might ask, if someone makes the same mistakes over and over again, doesn't that indicate an inability or an unwillingness to learn? The "new view" doesn't work this way. In the new view, we are supposed to examine all the circumstances surrounding an incident and how they contributed to the actions of the people involved so that the system as a whole (including the human operators) can be improved. If people keep making the same mistakes, how thorough or effective were the debriefings that followed these incidents? Was anything changed to improve the safety of the system as a whole? The philosophy of blamelessness still applies - people should not be blamed for mistakes that stemmed from systemic or organizational failures, especially if the organization could not or would not make changes that would help protect against those failures.

Retrospective

A retrospective is a discussion of a project that takes place after it has been completed, where topics such as what went well and what could be improved in future projects are considered. Retrospectives usually take place on a regular (if not necessarily frequent) basis, either after fixed periods of time have elapsed (every quarter, for example) or at the end of projects. A big goal is local learning - how can the successes and failures of this project be applied to similar projects in the future. Retrospective styles may vary, but usually include topics of discussion such as:

- *What Happened?* What was the scope of the project and what ended up being completed.
- *What Went Well?* Ways in which the project succeeded, features that the team is especially proud of, what should be used in future projects.

- *What Went Poorly?* Things that went wrong, bugs that were encountered, deadlines that were missed, things to be avoided in future projects.

Organizational Learning

> A learning organization is one that learns continuously and transforms itself . . . . Learning is a continuous, strategically used process — integrated with and running parallel to work.
>
> —Karen E. Watkins and Victoria J. Marsick, *Partners for Learning*

Organizational learning is the process of collecting, growing, and sharing the body of knowledge that an organization has. A learning organization is one that has made their learning more deliberate, setting it as a specific goal and taking actionable steps to increase their collective learning over time. Organizational learning as a goal is part of what separates blameful cultures from blameless ones, as blameful cultures are often much more focused on punishment than on learning, whereas a blameless or learning organization takes value from the experiences it has and looks for lessons learned and knowledge to be taken away, even from negative experiences. Learning can happen at many different levels, including individual and group as well as organization, but organizational learning has higher impact to companies as a whole, and companies who practice organizational learning are often more competitive than those who don't.

Post-Mortem

Unlike the planned, regular nature of a retrospective, a post-mortem occurs after an incident or outage, for cases where the outcome of an event was surprising to those involved and at least one failure of the system or organization was revealed. Whereas retrospectives occur at the end of projects and are planned in advance, post-mortems are unexpected before the event they are discussing. Here the goal is organizational learning and they benefit from having a systemic and consistent approach that often include topics such as:

- *What Happened?* A timeline of the incident from start to finish, often including communication or system error logs.
- *Debrief* Every person involved in the incident gives their perspective on the incident, including their thinking during the events.
- *Remediation Items* Things that should be changed to increase system safety and avoid repeats of this type of incident.

In the devops community, there is a big emphasis placed on having post-mortems and retrospectives be *blameless*. While it is certainly possible to have a blameful post-

mortem that looks for the person or people "responsible" for an incident in order to call them out, that runs counter to the focus on learning that is present in the devops movement.

Devops: Adding it All Up

> The danger for a movement that regards itself as new is that it may try to embrace everything that is not old.
>
> —Naturalistic Decision Making

After defining all of these related terms, it is tempting to try to tie them all together into one simple definition of devops. However, if it were that easy, there wouldn't have been nearly as much debate over the past five years as to what devops is and isn't. This book is not a prescription for the One True Way of doing devops. We don't offer you devops in a box, devops-as-a-service, or tell you that you are Doing Devops Wrong. What this book offers is a collection of ideas and approaches for improving individual collaboration, team and organizational affinity, and tool usage throughout a company or organization, and discuss how these concepts operate at different sizes and scales. Every organization is unique, and so while there is no one-size-fits-all way of doing devops, these common themes can be applied in different ways to every organization that wants to improve both the quality of their products and the efficiency and well-being of their employees.

While the term devops itself is a portmanteau of "development" and "operations", the core concepts of the devops movement apply much more broadly than just the development and operations teams. Companies building products or services are much more complex than simply those who write the software and those who maintain it in production. Like any complex system with many interdependencies it must be treated differently than the comparatively simple single "system engineer" that the industry began with. To be successful, a business needs to have many other teams and skills involved, including QA, security, network and database specialists, and even support, sales, and marketing, and all of those must work well together to be competitive in today's fast-paced environment.

Sometimes, in order to get around the issue of defining devops and get people talking about concepts and principles, they will use an exaggerated example of so-called "bad" behaviors as a way of focusing on the "good" behaviors that they see as being "devops". In order to talk about effective inter-team collaboration, someone might use a cartoonish example of a company that creates a devops team that serves only to act as go-betweens for the development and operations teams. It's an extreme example (though there are almost certainly places where it's actually happened) but it serves to get people talking about something more meaningful and applicable than a definition.

In the field of cognitive science, a folk model is a word or phrase that is used as an abstraction for more concrete ideas, and often substituted for those things, with the folk model being easier to understand than the concept really being discussed. An example of this is the term "situational awareness" which is often used as a stand-in for more specific ideas like perception and short-term memory. Folk models are not necessarily bad, but they can be problematic when different groups of people use the same term to refer to different underlying concepts. We would argue that in many ways, devops has become a folk model. Different people use it to mean many different things, which can cause miscommunication to occur - people will often spend more time arguing over what "devops" means, what folk model they are using for it, than they spend focusing on the ideas that they really want to discuss. For this reason, providing one definition of "devops" can be distracting away from what we want to talk about instead.

At the end of this chapter we will discuss our five pillars of devops as well as what the rest of this book will cover, and it is those core concepts that are what this book is designed to talk about, not the particular definition or folk model we will be using.

Common Devops Misconceptions

To clarify even more what we mean when we discuss devops in this book, we will clear up what are some common misconceptions about what devops is.

Devops only involves developers and system administrators.

While the name might indicate that it involves only developers and operations, and though the DevopsDays conference tagline is "the conference that brings development and operations together", the concepts and ideas of devops can and should be expanded to include others as necessary. There is no one definitive list of which teams or individuals should be involved or how, just as there is no one-size-fits-all way to "do devops".

Ideas that help development and operations teams communicate better and work more efficiently together can be applied throughout a company. Any software development organization should be considering aspects of the product life cycle including security, QA, and support in order to be most effective. In later chapters we will discuss considerations for involving these other teams into an effective devops environment.

Devops is a team.

Some people will argue very strongly against the creation of a designated "devops team". There are several valid reasons for this argument. Simply creating a team called devops, or renaming an existing team to devops as a way to check off an item on a

checklist, is neither necessary nor sufficient for creating a devops culture. If your organization is in a state where the development and operations teams cannot communicate with each other, an additional team adds the potential for more communication issues. Underlying communication issues needs to be addressed for any substantial and lasting change to stick.

Creating a separate team as an environment to kickstart new processes and communication strategies can be effective if it is seen as a greenfield project. In large companies generally this is a useful short term strategy to kickoff meaningful change and usually results in blending the team members back into designated role teams as time progresses.

In a startup environment having a single team that encompasses both functions can work as it allows for the team to embrace the responsibility and mission of the service as a collaborative unit rather than burning out a single individual on-call. Management will still need to facilitate clear roles and responsibilities to ensure that as the company grows the team can scale out as required.

This book will cover different team organizational options and inter-team communication and coordination strategies, but ultimately it's important to remember that there is no one right or wrong way of doing devops, and if having a devops team genuinely works for you, there's no reason to change it.

Devops is a job title.

Probably no job title in the past five years has been as controversial as that of the devops engineer. The job title has been described in various ways, including a system administrator who also knows how to write code, a developer who knows the basics of system administration, or some mythical 10X engineer (said to be ten times as productive as other engineers, though this is difficult to measure and often used figuratively) who can be a full-time system administrator and full-time developer for only the cost of one salary without any loss in the quality of their work. In addition to being totally unrealistic, the concept of a devops engineer doesn't scale well. At a very early-stage startup, it might be necessary to have the developers being the same people deploying the code and maintaining the infrastructure, which makes more sense in a cloud-based infrastructure like many startups have. As a company matures and grows, however, it makes sense to have people become more specialized in their job roles. Neither does it make much sense to have a director of devops or some other position that puts one person in charge of devops. Devops is at its core a cultural and professional movement, and its ideas and principles need to be used by everyone in order to be effective.

Devops is only relevant to web startups.

It is easy to see why devops makes sense for web-based companies - because the movement helps break down barriers that can impede development, it can do a great deal to speed up software development and deployment. This is especially good for web-based products where the need to move fast is much greater than with other forms of software - if a web company's processes are so slow that it takes a matter of weeks to fix a typo, chances are they aren't going to do very well - but this in no way means that it isn't relevant to other types of companies as well. Probably no company has ever complained that their processes were too efficient, and improving communication and empathy among teams and individuals is something that any company can benefit from. And while it might be easier to iterate on team structures and processes at a small startup with only a few people, making these sorts of changes is very possible even in the enterprise. The chapter on scale especially will discuss how devops concepts can be applied at larger organizations.

Devops is about the tools.

While tools are valuable, devops does not mandate or require any particular tools. This misconception is a big contributing factor to the idea that devops is only for startups, as large enterprise companies are less able to switch to the newest and shiniest technologies at the drop of a hat. Devops, being a cultural movement, is technology-agnostic. The principles that will be discussed in this book don't require any particular set of tools, instead being able to be applied to any technology stack. There is a fair amount of overlap between companies who practice devops and those who use containers or cloud providers, but that doesn't mean that those particular technologies are required - there are certainly companies successfully implementing devops while running on bare metal, and the chapter on tools will discuss how to choose and implement tools in a way that complements devops principles most effectively.

You need a devops certification.

Devops is a cultural movement - how do you certify culture? There is no 60 minute exam that can certify how effectively you communicate with other people, how well teams in your company work together, how your organization learns, or anything else like that. Certifications in technology only make sense in the cases of very specific technologies that require a high level of expertise to use, such as individual brands of networking equipment. Since devops doesn't have any one required technology or one-size-fits-all solutions, makes very little sense to try and write a certification exam for it. Exams do well at testing knowledge where there are simple questions with obviously right or wrong answers, and because what works best for one company won't necessarily be optimal for any other, there's no way to write such questions that

would be universally answerable the same way for devops. Any devops certification is more likely to be a money-making opportunity for whoever is running it than it is to provide any value to those who have been certified.

Devops means doing all the work with half the people.

There are some people under the impression that devops is a way to get both a software developer and a system administrator in one person - and with one person's salary. Not only is this perception incorrect, it is often harmful. At a time when too many startups are offering perks such as 3 meals a day in the office and on-site laundry as a way of encouraging workers to spend even more time in the office and too many "rockstar" or "10X" engineers are working 60-80 hours a week, misconceptions that drive people further away from work-life balance and more towards overwork are not what our industry needs. During very early stages, it is true that a startup can benefit from having developers who understand enough about operations to handle deployments as well, especially with cloud providers and other "as a service"s to handle a lot of operational heavy lifting. Once an organization gets past the point where every single employee must wear multiple hats out of sheer necessity, expecting one person to fill two full-time roles is asking for burnout. Devops doesn't save money by cutting the number of engineers your company needs in half. Rather, it allows organizations to increase the quality and efficiency of their work, reducing the number and duration of outages, shortening development times, and improving both individual and team effectiveness.

There is one "right way" (or "wrong way") to do devops.

Early adopters of devops practices and principles, especially those well-known for this in the industry such as Netflix and Etsy, are often regarded as "unicorns" who have cornered the marked on the "right" way to do devops. Other companies, eager to get the benefits of a devops culture, will sometimes try to emulate their practices. The term cargo cult, when used metaphorically, is used to describe the practice of emulating behaviors without fully understanding the reasoning or circumstances behind them, and isn't something to be encouraged. Just because a company who is successfully doing devops such as Netflix or Etsy does something doesn't mean that that is the "right way" of doing devops. Instead, devops not only encourages but also requires critical thinking about processes, tools, and practices - being a learning organization requires questioning and iterating on processes, not accepting things as the "one true way" or the way that things have always been done.

One should also beware of people saying that anyone who isn't following their example is doing devops "the wrong way." It bears repeating that while there are valid criticisms of devops teams or devops engineers, there are also documented cases of companies and people who make those terms work for them. Devops is a cultural movement, having core tenets and principles rather than strict definitions, and

because it isn't as rigidly defined as something like ITIL, it doesn't make a lot of sense to say that if something like a devops team is working in a particular instance that that's the wrong way of doing things. The companies who are doing devops most successfully are comfortable learning and iterating to find what tools and processes are most effective for them.

It will take X weeks/months to implement devops.

If some sort of management buy-in is required for an organizational transformation such as those involved in devops, one of the questions asked of the transformation is likely to be how long it will take. The problem with this question is that it assumes that devops is a fixed or easily definable or measurable state, and once that state is reached then the work is done. In reality, devops is an ongoing state - it is the journey, not the destination. Some parts of it will have a fixed end point - such as setting up a configuration management system and making sure that all the company's servers are being managed by it - but the ongoing maintenance and development of configuration management will continue. Because so much of devops is cultural, it is harder to predict how long some of those changes will take - how long will it take people to break old siloed habits and replace them with new collaborative ones? The following chapters each contain a section on measuring for success, illustrating ways that progress and effectiveness can be measured, but effectiveness doesn't mean being done - devops is ongoing.

Devops is about automation.

With many practitioners focusing on things like configuration management and continuous integration, some people see devops as just a way to automate traditional system administrators out of their jobs. Alternatively, people may see the focus on automation of some things as meaning that everything that can possibly be automated should be. Neither of these is true. Devops is a movement that wants to improve how people work together. If there are repetitive tasks that could be automated to free up a human from having to do them, that automation helps that person work more efficiently. Some cases like this are fairly obvious gains - automating server builds saves hours per server that a system administrator can then spend on more interesting work. But if more time is spent trying to automate something than would be saved by having it be automated, that's not improving anyone's workflow anymore.

There has been a great deal of discussion about the role of automation in any environment and the way that human factors affect what and how we choose to automate. We will go into this topic in more depth in the Tools chapter later in the book, but for this chapter we will suffice it to say that devops does not mean a simplistic, all-or-nothing view of automation.

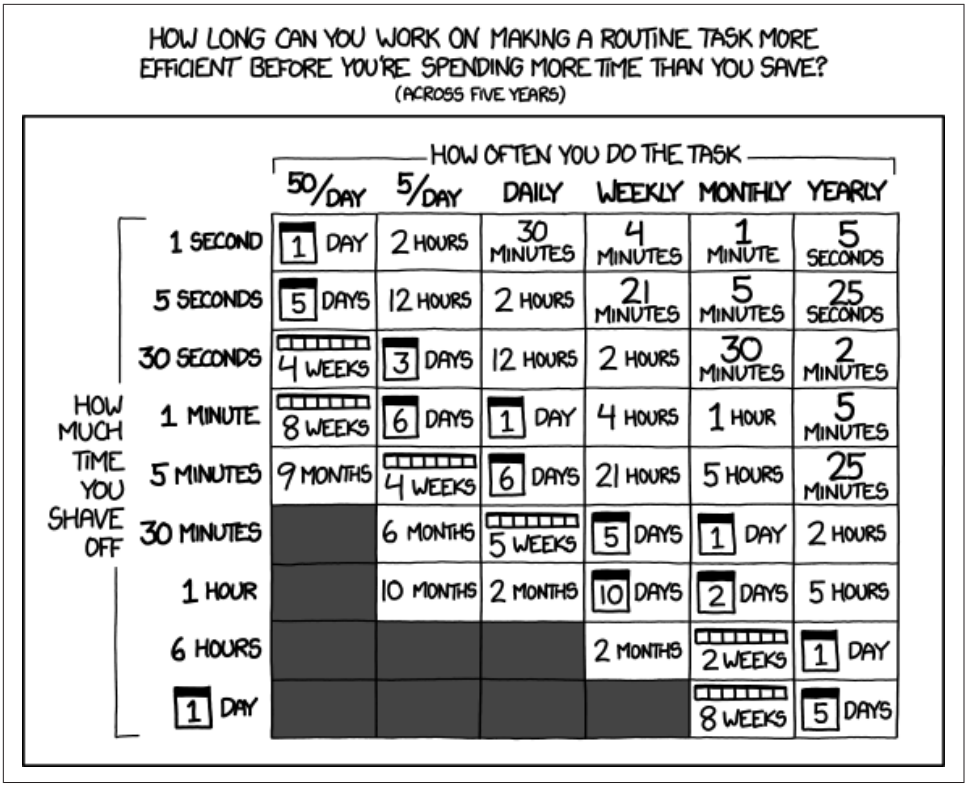

Figure 2-2. A Comic from XKCD on Time Spent versus Time Saved, https://xkcd.com/1205/

Devops is a fad.

Because devops is not a particular technology, tool, or methodology, it is unlikely to become obsoleted or replaced. Ultimately, time will tell if devops fades into obscurity a footnote in the historical recounting of workplace organization. A movement about improving organizational effectiveness as well as individual employee happiness seems very unlikely to be merely a passing trend. While it may seem similar to ITIL, Lean, or Agile in ways, and one might wonder if its popularity will start to wane as those did (even if they are still in use in some organizations today), but the primary different is that things like ITIL or Agile have strict definitions, and those definitions rarely, if ever, change. Devops, on the other hand, is a movement defined by ideas, not a strict definition. It is the continuing conversations and evolutions of processes and ideas, and that evolution and growth that carried it so strongly through its first five years will likely prove to be its staying power for years to come.

There has been some discussion in the devops community recently as to whether or not devops has lost its direction. Critics of the movement say that it is too defined by

negative spaces, by people saying what devops isn't rather than what it is (or not providing a concise definition for it at all). They also claim that devops isn't unique, that it is merely a rebranding of ideas that have come before it, and that it will be abandoned as soon as the next name or trend comes along. While it is true that several of the driving ideas behind the devops movement have indeed been around for some time, the zeitgeist of devops as something more than the sum of its parts is something new and different. People have certainly argued against functional silos before, suggested learning organizations, advocated for humane systems or advocated for automation and measurement. The devops movement is the first to combine all of these ideas, and to do so with measurable success. This book will cover those ideas and show how to harness and leverage them in ways that can continue to grow and evolve with your company, just like the movement itself.

The Current State of Devops

It is inspiring to see how far the devops movement has come in the six years since Patrick Debois held the first DevOpsDays in Belgium. The 2014 State of Devops Report published by Puppet Labs has findings that show that companies who are doing devops are outperforming those who aren't, finally showing numerically what many people have already suspected - that an emphasis on having teams and individuals work together effectively is better for business than silos full of engineers who don't exactly play well with others. High performing devops organizations deploy code more frequently, have fewer failures, recover from those failures faster, and have happier employees.

The number of DevopsDays conferences have increased from 1 in 2009 to 18 all over the world in 2014. Not only that, but each year brings DevopsDays events happening in new locations worldwide - this is not a phenomenon that is limited to places that are considered technical hubs with concentrations of tech employees like Silicon Valley or New York. There are dozens of local Meetup groups with thousands of members in even more locations around the globe, not to mention the conversations about the topic happening daily on Twitter. Bigger conferences such as O'Reilly's Velocity include tracks on devops and cultural changes in general in their programs.

The Devops Compact

Consider, as an example of the importance of communication and understanding, two friends who want to leave a party at the same time. One of them, the General, isn't familiar with the neighborhood they're currently in, and asks her friend George, who is familiar with it, if she can follow him as she drives until she gets back into a part of town that she's comfortable with. George agrees, they decide what intersection will be the point where the General feels comfortable navigating on her own, and they leave the party in this two-car convoy.

This convoy requires a compact between these two people. They have to set up and maintain a *shared mutual understanding* that they are going to complete this convoy together. This requires a lot of communication in advance to work out where the *boundaries* of this compact are. How fast is the General comfortable driving, so George knows what speed he should maintain as she follows him? What should they do if the General falls behind due to traffic or a red light? What if one of them gets a flat tire? Where will the convoy end - at the General's house, or at some intersection that she's familiar with? All of these things must be decided and communicated in advance, so that both people are on the same page and that shared mutual understanding is present.

George is going to have to make some changes to his normal driving behavior during this convoy. He'll have to be less aggressive about going through intersections when the light is yellow, to avoid having the General get left behind when it turns red. He'll have to keep an eye out in his rear view mirror to make sure she's still following him. He'll have to be extra diligent about using turn signals to communicate his plans clearly. And if something goes wrong and the General makes a wrong turn, he'll have to double back to find her again, which will delay him getting home. And because these things, like traffic, can't be predicted in advance, they'll both have to figure it out as they go, dynamically adjusting their own short-term goals and behavior in order to achieve this shared goal.

We view devops as a similar compact.

Instead of people, we have teams working together. Instead of two friends both trying to get home from a party, the shared goal is creating and delivering software. In a siloed, non-devops environment, the lack of a shared understanding would be like the General trying to follow George back to familiar territory without letting him know she is doing this - it might end up working, but without communication of intentions the odds are stacked against it. Devops is a compact that different teams will work together, will communicate their intentions and the issues that they run into, and dynamically adjust in order to work towards the shared organizational goals.

Just like George and the General might run into traffic or car troubles, an organization working on a software product will certainly run into issues or roadblocks along the way. But with the shared understanding that everyone is still a part of the compact, everything else turns into repair. We repair our misunderstandings about who would be working on a particular feature or when something would get done. We repair bugs that affect our understanding of how the software is supposed to behave. We repair processes and their documentation when things don't go the way we expect in production.

We're going to take this idea of a devops compact and show how both the technological and cultural aspects of devops are ways of developing and maintaining this shared mutual understanding.

What's Next in this Book

Now that the historical context for devops has been explained, its major terms and concepts defined as we will be using them in this book, and some of the most common misconceptions have been addressed, it will be easier to dive into greater detail in further chapters. While this book will not tell you one "true" way to do devops (as no such thing exists, and anyone who says otherwise is selling something), it will cover tested and practical examples of how the concepts we explain can be put into practice.

We present you the five pillars of effective devops:

- Collaboration
- Hiring
- Affinity
- Tools
- Scaling

Each pillar is covered in a separate section which will include a detailed case study. We have worked to pull examples from industry covering a diverse set of companies from web startups to large enterprises. While it isn't strictly necessary to read the chapters in order - a reader who has some immediate decisions to make regarding the tools their team or organization is using is welcome to start with that chapter, for example. - it is recommended that they all eventually be read as it is the combination and harmony of these five pillars that truly make devops effective.

CHAPTER 3

Collaboration: Individuals Working Together

Introduction

While one of the guiding principles that originally helped shape the devops movement was enabled software development and operations teams to work more effectively together, we believe it is best to start off by discussing what helps people work better together at an individual level. After all, teams are made up of individuals, and if a team cannot work well within itself, on an individual or intra-team level, there is very little hope of that team being able to overcome those issues and work well on a higher, inter-team level. This chapter will look into the human factors that can make or break positive individual collaboration, discuss strategies for effective communication and collaborative work, and address common misconceptions and issues that can arise in these areas.

We'll start out by looking at people as individuals, and understanding the different factors that motivate people and how they work. Following up with a discussion on individual growth and development, and how work environments can affect this, we will then move into the different types of negotiation styles that can occur. Understanding these different styles will lead into a deeper dive into the collaborative style, how communication can be best used to achieve this, and the importance of trust and empathy among teams.

Individual Differences and Backgrounds

When looking at collaboration between individual members of the same team, we are assuming that the team as a whole has shared goals. We will discuss what differentiates a team from other groupings of individuals in the upcoming chapter, Affinity,

but for the sake of this chapter we will assume shared work goals among team members. This being the case, one of the biggest sources of strife for team members trying to work together come in the form of different personal goals, backgrounds, or working styles.

Goals

Although members of the same team will generally have the same overall professional goals, usually communicated to them by their manager, they will likely have different personal goals when it comes to what they want out of their jobs.

- For some people, their current position is an important stepping stone in their career progression, while others may think of it as "just a job", something they are doing while they consider a career change, pursue side projects, or support their families. The former, being more invested in this particular position, are likely to put more consideration and effort into it, and might feel resentment towards the latter, who they might perceive as not being "team players" or not pulling their weight.
- Many people want to learn and grow their skills, but the specifics of this can vary from person to person. More junior people are often eager to get as many new experiences and learning opportunities as possible. More senior people may be looking for specific projects or leadership roles, while some engineers are looking for environments where they can experiment freely with whatever new bleeding-edge technology strikes their fancy. Depending on the goals and working styles of the rest of the team, these different learning goals might find themselves at odds.
- Some people might be focused more on their specific work while others might place value on growing their networks or more community-focused activities such as mentoring or speaking at industry conferences. To the latter, heads-down coding-only sorts of engineers might be perceived as aloof, or not interested enough in the bigger picture, while the other way around might see things as not contributing enough to "real" work. Clarifying team and company expectations around these different types of contributions can go a long way towards minimizing resentment for these reasons.

Backgrounds

As we'll discuss in more depth in the Hiring chapter, there are great benefits to be gained from diverse teams in terms of creativity, problem solving, and productivity, but these can certainly lead to short-term interpersonal conflicts among people from very different backgrounds, either personally or professionally.

Professional Backgrounds

One of the biggest differences in professional backgrounds is the size of companies that people have worked at previously. Especially in the startup world, there is a strong preference towards hiring and working with people with previous startup experience. This makes sense to some extent - especially in early-stage startups, success can be more likely when key individuals have had successful startup experiences before - but beware of being overly biased towards people who have worked mostly (or exclusively) at larger organizations. Enterprise experience does not disqualify someone from being able to work well at a much smaller company.

There certainly will be cultural differences and expectations to overcome with someone who is moving to a company of a different size in either direction. Focus should instead be placed on the new viewpoints and benefits that different backgrounds can provide, and how well people are able to learn to contribute in their current environments. Ideally, team members should all be learning from each other. In this example, startup experience may be more directly relevant to a startup team, but in a collaborative atmosphere, the contributions and ideas from someone with more enterprise experience would be considered for how well they can be applied to benefit a startup team, not dismissed out of hand.

Technical versus non-technical is another area of difference that can cause friction between people. This can take place in a company-wide context, where perhaps engineers are seen as being more valuable to the company and teams like support, sales, and marketing treated like second-class citizens. If these feelings are mirrored at all by management, such as at an early-stage startup where all or many of the cofounders are engineers themselves, this can cause a serious loss of morale among these non-engineering employees. People need to feel that their work is appreciated, and if they sense that the company as a whole doesn't value the contributions of them or their team, they may start looking for a job at another company that does.

This is not limited to non-technical roles, of course. In many traditional software development shops, IT and related roles (system and network administrators, operations engineers, and database administrators to name the most common ones) are often treated the same way. Ops being seen primarily as a cost center, or being something that was really only ackowledged when something went wrong and there was some sort of outage, or being viewed by other teams as barriers or gatekeepers, was at least a part of what spurred the beginnings of the devops movement in the first place.

Even among engineers, there can be differences in peoples' backgrounds. It used to be that software engineers almost exclusively had technical backgrounds, whether that be a degree (or more) in a field like computer science or computer engineering or a lifelong history of working with computers. It's easy to look at someone who started tinkering with their parents' computers as a young child and taught themself to program soon after that as a "natural" engineer.

These days, we are seeing come into existence many more coding bootcamps, short programs (usually between 3 and 6 months in length) designed to quickly and effectively teach people the skills required to get a job as a software developer. These are a way of making tech jobs available to people who are changing careers but don't have the money or time to spend on a traditional four year degree program. With some bootcamps designed specifically to provide safe learning spaces for underrepresented groups in tech such as women or people of color, they can be a great resource for companies looking to improve the diversity of their engineering staff as well. However, there is still some bias in some places towards people with "traditional" engineering backgrounds. As we'll discuss more in the hiring chapter, it is important to be aware of these biases, whether conscious or unconscious, when growing and maintaining a successful team.

One final place where professional backgrounds might cause friction between team members is job level or experience. When looking to hire, most teams express a preference towards hiring more experienced people or "senior" engineers, with the thinking that a more senior person will be quicker to get up to speed and start contributing to the team. However, there is a limited number of senior engineers - far fewer than the number of companies looking to hire them, it seems. In addition to simply getting more years of technical experience, more junior employees need guidance and coaching to help grow them into senior ones. When looking at the people on your team, it's important to consider how effective they are at teaching or mentoring in addition to just their technical skills.

Types of Mentoring

The most traditional type of mentoring is **senior-junior**, where a senior engineer mentors a junior one, usually in a more organized capacity as part of some formal mentoring program. This is good for leveraging the expertise of more senior team members to help grow the skills of the junior ones. This works best when the senior employees have enough communication skills, teaching ability, and patience to help other people truly learn (an impatient person will just grab the keyboard away and do it themselves). In the best case, the questions from the junior employee can help the senior one think through things they took for granted before, to question whether a solution is the best one rather than just "the way we've always done it".

Senior-senior mentoring is less common, where two senior-level employees mentor each other. There can be a lot of deep knowledge sharing in these types of cases, but if both people have been senior for a while at the same company, they might lose the questioning and new perspectives that can come from having a fresh set of eyes looking at things.

Finally, **junior-junior** mentoring happens when two junior-level employees work to help each other learn. This might happen on rapidly growing teams, where either there was no senior engineer to be part of the process or any senior staff were too

busy. Having someone else to learn with can enable both people to learn more quickly than they might on their own, but without any experienced people to steer them towards good practices or help when they get stuck, this can also result in some less-than-ideal outcomes.

Personal Backgrounds

Often when people think about increasing the diversity of their team, what they are looking for is a wider range of personal backgrounds. Including aspects such as gender, sexuality, race, class, and education level, diverse personal backgrounds can increase the strength of an engineering, product-focused, or customer-support-driven organization by having a greater number of experiences and points of view. As we will discuss further in later chapters, there are many benefits to having an increasingly diverse workforce, both in individual organizations but also in the industry as a whole, but as people learn to work well with people who are different from them, there can be increased friction.

If a team has previously consisted entirely of white heterosexual men, working with women, LGBT people, or people of color might require some adjustments of peoples' behavior. Similarly, if a team used to consist of just young, single individuals, bringing on team members with family repsonsibilites will likely highlight places where work-life balance needs to be addressed. It can be very beneficial to make sure you have an HR department that understands diversity-related concerns, and that both individual contributors and managers are able and encouraged to take unconscious bias training.

The point of these sorts of initiatives is not to have companies become "political correctness" police, but rather to foster and ensure an environment where every employee feels safe and included. Without personal safety, there is unlikely to be trust between employees, and without trust there will not be the empathy and honesty that are necessary for truly effective collaboration. Personal backgrounds also tend to contribute to power differentials that can affect or even prevent negotiation.

Working Styles

> It is hardly possible to overrate the value, in the present low state of human improvement, of placing human beings in contact with persons dissimilar to themselves, and with modes of thought and action unlike those with which they are familiar. […] Such communication has always been […] one of the primary sources of progress.
>
> —John Stuart Mill

Both personal and professional backgrounds can affect how people collaborate, but even unrelated to those areas there is a great variety of working styles that people can have. These styles can be described as a group of different axes or spectra:

Table 3-1. Different Working Styles

Introvert	Extrovert
Asker	Guesser
Starter	Finisher
Analytical Thinker	Lateral Thinker
Purist	Pragmatist
Night Owl	Early Bird

This table shows some different ways in which peoples' working styles can differ. Many of these aspects are more spectra than fixed binaries, where instead of being 100% on one side or the other, people will fall somewhere along the scale, including in the middle. Many of these categories were described by Laura Thompson, Director of Cloud Services Engineering and Operations at Mozilla, in a talk she gave at the Monitorama conference in Portland, Oregon in 2015. We'll start our discussion of working styles by taking a look at these axes in more detail.

- **Introvert vs Extrovert**: Introversion and extroversion are often misunderstood to be whether someone is shy or outgoing, but more accurately it describes where people draw their energy from and how they "recharge their batteries", so to speak. Introverts recover energy by being alone, or in small well-known intimate groups, while extroverts recharge by being around and interacting with people. This is not to say that introverts are quiet or don't like people, but simply that they find it more draining than extroverts do. This may make extroverts more likely to enjoy group projects or organizational roles where they get to interact with many people, while introverts may prefer a cubicle or office where they can work quietly than they would an open-office floor plan.
- **Asker vs Guesser**: Ask versus guess culture came from an internet forum post written in 2007 talking about different ways that people approach asking things from others. Askers feel that it alright to ask for most things, with the understanding that they may well get "no" for an answer, while guessers tend to read more into situations and avoid asking for things unless they're fairly certain that the answer will be "yes". The issues that can arise when these two sorts of people interact is that askers might find guessers to be too passive and not direct enough, while guessers often find askers to be presumptuous. This is one area where we find that clarifying and documenting how your team members are

expected to communicate can, if not necessarily overcome years of habit from one culture or the other, at least ensure that people are on the same page.

- **Starter vs Finisher**: Starters are people who love coming up with new ideas and getting them off the ground - they are energized by the process of beginning a new project. They might love experimenting with new technologies, refactoring existing code in fairly substantial ways, and looking for new greenfield projects to take on, but after something is 80 or 90 percent of the way there, they lose interest in the details that would take it to 100%. Finishers, on the other hand, like tying up all the loose ends, fixing any remaining issues in a project, and generally hate to leave things feeling less than 100% complete. Often finishers can be found on operations team, where they get to focus themselves on the final touches that make things operationally ready. Starters will likely get bored if asked to do a great deal of finisher work, while finishers might feel overwhelmed and not know where to start if they find themselves asked to be starters.
- **Analytical Thinker vs Lateral Thinker**: Analytical thinkers have the ability to focus on facts and evidence, dissecting complex things into simpler pieces, eliminating extraneous information or invalid alternatives. They tend toward being organized and interested in the details, especially in working out how to execute on something and what will or will not work. Lateral thinkers have the ability to find information more indirectly, finding the missing elements, examining issues from multiple perspectives, and eliminating sterotypical patterns of thought.

- **Purist vs Pragmatist**: Very similar to analytical vs lateral thinkers is the distinction between purists and pragmatists, especially when thinking about engineering problems. A purist wants to use the absolute best technology to solve a problem, and if that perfect technology doesn't exist, they will want to create their own. Purists are much less comfortable with things that require workarounds or making compromises around their engineering principles. Pragmatists instead are much more focused on practicality, weighing the cost of trying to create an ideal solution versus working with the realities of their current environments and constraints. Pragmatists will think about how to operationalize something and get it working in their actual production environment, rather than the purist approach of focusing on a technology in an of itself.
- **Night Owl vs Early Bird**: Finally, people differ in their working habits based on when they find themselves being most productive. In the simplest sense, people differ between night owls who are more productive in the evening hours versus early birds who will likely get into the office and start working before anyone else. People might also differ in how much background noise they can handle without getting too distracted or how long they need to work before taking a break. While making sure that people are able to participate in necessary meet-

ings, allowing engineers to work flexible hours to ensure that they can be individually more prductive is definitely something to consider.

For several of these axes that we just described, most teams are benefitted by having a mix of types represented in the people on them. For example, a team that consists only of starters, while great at coming up with product ideas and getting them off the ground, maybe in some form of minimum viable product, is likely to find that they run into stability and reliability issues without the input of finishers who want to focus on operational details. A team of only purists might find themselves never actually shipping anything because it never meets their exacting standards without some pragmatists to focus on what they can actually accomplish by their next deadline.

If you're looking to increase the effectiveness of your team, a good step can be having individuals assess themselves to figure out where they fall on each of these different axes. From there, you might find some changes that need to be made in terms of work assignments. This is a useful exercise if you notice that someone's productivity doesn't seem to be in line with where it is expected to be and there aren't any extenuating factors that would explain this - it could simply be the case that a starter has been given assignments better suited to a finisher, and different tasks more suited to their style would make them much happier and more productive. When arranging for work assignments on larger projects as well, knowing who falls where on each axis can make sure your project has a good balance between starters and finishers, critical and creative thinkers, and purists and pragmatists.

Of course, with different working and collaborative styles, there are certainly opportunities to run into disagreements. The biggest place that we've seen this occur on teams is with purists versus pragmatists, where planning meetings for projects tend to get derailed with long debates about the "ideal" solution versus the practical one (or, with several purists, even longer discussions about which ideal solution is the most ideal). A solid understanding of what the deadlines and other requirements are for each project can help to keep these sorts of things in check. We'll also touch more on some of these issues in the upcoming section on communication.

It's important to create and maintain a work environment that can be supportive of people across these various spectra of styles. Keep an eye out for office policies that unnecessarily favor some over others, for example, requiring people to be in at 8:00 sharp in the mornings even if there are no meetings that would necessitate that (or not allowing people to attend meetings remotely), or a loud open-office floor plan that has no areas where people can work when they need quiet, distraction-free focus.

Individual Growth

Key to collaboration between individuals is creating a work environment where those individuals are encouraged and enabled to grow. Growth is certainly an area where teams can turn competitive rather than collaborative - in the wrong environment, individual growth becomes a zero-sum game that necessarily pits people against each other. In this section we'll discuss both individual and group factors that can affect peoples' growth as employees.

The Right Mindset

Research has shown that the mindset that people have about their own abilities and more specifically where those abilities come from has a significant impact on how people learn and grow. Dr. Carol Dweck, and professor and researcher of social and developmental psychology, described this in terms of two different mindsets. With a *fixed mindset*, people believe that their talents and abilities are innate, fixed traits - either they are naturally good at something or they aren't, and that state is seen as immutable. In a *growth mindset*, talents and abilities are seen as things that can be learned and improved with effort and practice. These mindsets can impact how people work, how they approach challenges, and how they deal with failure.

A fixed mindset, believing that skills and traits are fixed and static, makes people feel that they have to constantly prove themselves to others. If a person thinks that they are either smart or not smart, and that that is set in stone, they will obviously want to prove to themselves and the people around them that they fall into the smart category. While one might think this would do a good job of encouraging people to work their hardest, this isn't necessarily the case.

Someone with a fixed mindset views failures of any kind as proof that the individual is inherently not smart, not talented, not good enough, or some other negative state. In order to avoid failures and those feelings of inadequacy, people with a fixed mindset may stay away from situations in which they might fail, which means they are less likely to, for example, work on projects where they would have to learn new skills. People avoid uncertainty as a way of avoiding failure and disapproval. This means that people with fixed mindsets are less likely to pick up new skills on the job, which over time might make them less hireable which they will ironically view also as a fixed trait. They also tend to focus a great deal on comparing themselves with their peers - a very competitive mindset - to confirm their beliefs about their traits.

Growth mindsets, on the other hand, lend themselves much more to individual learning and learning environments. Someone with a growth mindset believes that their intelligence and skills can grow and change over time - if they are currently not knowledgeable about a particular area, they believe that with enough time, effort, teaching, and practice, they can become knowledgeable about it. This isn't to say that

everyone has the potential to be the next Albert Einstein or Marie Curie, but rather takes a more practical view that while not every skill can be mastered or perfected, nearly every skill can at least be improved.

In this case, challenges are viewed as learning opportunities, ways to gain new skills and knowledge or practice and level up existing ones. Without the fear of failure that can be hard to overcome in a fixed mindset, more risks can be taken and more growth can happen. Failure is viewed not as a sign of an inherent personal flaw, but simply as something that happens during the learning process.

Blamelessness and Learning Organizations

This view of failure applies at the organizational level as well as the individual one. Consider the blameful culture that we introduced in the What is Devops chapter. A blameful culture, when dealing with a failure, looks for the individual(s) who they believe caused it so they can be removed, either from the project or the organization. This is often because they view failure in a fixed way - if someone made a mistake, it is seen as being because they were not good enough or not smart enough and because that is viewed as immutable, they don't give the person chances to improve. The organization as a whole tends to stagnate in this way. Focus is placed not on dealing with failure well and learning from it, but rather on avoiding it altogether.

A blameless view of failure works so well in part because it adopts a growth mindset, acknowledging that mistakes happen but operating under the assumption that both people and organizations are capable of learning, growing, and improving. The team might not currently be good at something, but it *can get better* so people are looking for ways to get better, ways to learn, and ways to improve. This focus on learning, education, and self-improvement produces smarter and more robust individuals and teams.

The Role of Feedback

Dweck's years of research found that the nature of the feedback people received was a key factor in whether they developed a fixed or a growth mindset. If someone does well at something and the praise they receive is, "Good job, you're so smart," the emphasis on smart pushes them towards a fixed mindset, making them less likely to take on challenging tasks or anything that might call that smartness into question. If, on the other hand, someone is praised by saying, "Good job, you worked so hard on that," they will associate their successes with the effort that they put into something, not an innate quality, making them more likely to take on challenges and try again after setbacks in the future.

The original studies in the area of feedback and mindsets were conducted with school-age children, but the idea that the type of feedback people receive can shape the mindset they have certainly applies to adults as well. A mindset might originally

form during the childhood years, but even a fixed mindset is not a fixed thing - someone who learned a fixed mindset as a child has the potential to develop a more learning-focused growth mindset as an adult.

This is very important when considering employee growth and performance. People with fixed mindsets tend to pay attention only to feedback that relates directly to their present abilities, tuning out feedback that speaks to how they could improve in the future. Growth mindset individuals, on the other hand, were very attentive to any feedback that could help them do better, being focused on learning and improving themselves rather than on their current state.

Keep both of these things in mind during employee reviews and feedback periods. When people are giving feedback, either as a manager or as an individual contributor, they should be emphasizing peoples' efforts, actions, and the work and thought that people put into things, focusing on what people *can do* rather than what they *are* and thus guiding people towards a growth mindset. This is the case for both positive and more negative feedback. Consider these examples:

"George is clearly an intelligent person - he intuitively understands the way that distributed systems behave and interact. He isn't very good with people though, and isn't the kind of person that others go to when they need help."

"The General has clearly put a great deal of work into understanding the distributed systems that she works with, and that effort shows in her deep knowledge of how these systems behave and interact. I'd love to see her work more on how she comes across to other people, so other people can learn from her experiences."

How are these two employees likely to react when receiving these pieces of feedback? Though the big pictures of each are the same (good at distributed systems, not so good at people skills), the details of how this feedback is framed and given make a world of difference. George's feedback contains fixed mindset phrases - "an intelligent person", "intuitively understands", "isn't the kind of person" - that imply that these are unchangeable facts about him. The General's feedback is put differently - "a great deal of work", "that effort shows", "see her work more on" - these phrases focus on her work and actions, what she has *done* in the past and should *do* in the future, not how she innately *is* and this will lead to or reinforce a growth mindset.

Reviews and Rankings

The goals of giving employees feedback are two-fold. First, feedback in the form of things like performance reviews is designed to let people know how they are doing so they can grow as individuals, level up their own skills and work to fill any gaps in their knowledge or skill set. Aside from benefit to the individual, there is also benefit to the organization by way of figuring out which people are performing better and contributing more. The rationale for this is that if there are some people who aren't

doing as well as their peers or are consistently failing to improve, the organization would be better off without them.

There have been multiple books dedicated entirely to managing employee performance, so we will not go into a great amount of detail ourselves here. We will touch on a few considerations that we've found to have a greater amount of impact on both organizational and individual levels.

Frequency of Feedback: As recently as 2011, it was found that 51% of companies do annual performance reviews while 41% do them semi-anually. However, more and more companies are beginning to realize that feedback and reviews can be much more impactful if given more frequently, *if* the feedback itself is helpful to those receiving it. Obviously if feedback provides no new or actionable information there will be no benefit to getting it more ofen. However, for feedback that is useful and actionable, greater frequency does lead to greater benefit for both individuals and organizations.

If someone isn't on the right track with something that they're doing, waiting up to a year for their next annual review isn't good for anyone involved. They will likely go through this time thinking they are doing well, leading to a nasty surprise come review time, and the psychology of getting feedback shows that people generally don't react well to negative surprises like this - this is knows as amygdala hijacking and causes an emotional response rather than an intellectual one, making people less likely to fully understand and be able to act on feedback they are being given. Habits that have been going on for longer are harder to break, so it makes more sense to try to nip a bad work habit in the bud rather than letting it continue.

Smaller, shorter feedback cycles mean that adjustments are smaller and thus easier to make. This is a big driving factor behind teams moving away from the waterfall model for software developments towards more agile practices and why continuous delivery works so well. Annual performance reviews are similar to waterfall in that the delay in getting feedback on how things are going can have a negative impact on how things go overall, so move towards the more agile idea of continuous feedback.

Ranking System: Especially in larger organizations, various ranking systems are often used to categorize or classify employee performance. One of the biggest changes in recent years is the move away from stack ranking, also referred to as forced ranking or forced distribution. Popularized by then-CEO of GE Jack Welch in the 1980s, the underlying beliefs of this practice is that the *top 20* percent of the workforce is the most productive, and the middle 70 percent work adequately. The remaining 10 percent should be fired - often referred to as "rank and yank". This ranking creates a drive for employees to avoid being in the 10 percent group.

When individuals in a system are forced to compete through comparing accomplishments with others this leads to a fixed mindset,increasing the challenge of effective

communication. In this manner the effects of the system can be readily seen as clear, transparent communication is not perceived as valuable to the individual. Sharing information can impact your rewards, career advancement, even whether you have a job. Studies have shown that stack ranking actually hurts performance instead of helping it, but luckily there has been a marked decline in organizations using it in recent years.

Having some system of ranking or categorizing performance isn't necessarily bad, just the idea of forcing performance into preset quotas. Having some kind of formalized system for measuring performance can be helpful, especially if combined with useful and frequent feedback to the employees, as it can provide clear steps forward for people who want to improve and grow their careers. Many startups, looking to move away from ranking systems that they feel are too corporate, do away with rankings and reviews entirely. However, in the chaos and change that categorizes these early-stage companies, lack of feedback can be detrimental to individuals. Additionally, without any kind of formal procedures or guidelines, it is easy for favoritism to come into play, intentionally or unintentionally.

Looking at these factors, we can see how feedback and rankings around individual performance, rather than being something that impacts only one individual at a time, can actually have impacts on collaboration throughout teams and organizations. Turning performance reviews into a zero-sum game inhibits communication and collaboration as everyone is more focused on looking out only for themselves to protect their jobs instead of working towards creating value for the company as a whole, let alone what might provide the most benefit for customers. By focusing on comparing a person's performance with that of others, it unintentionally turns focus inwards so people only consider what will benefit themselves and keep them from being one of the "yanked" employees in a stack ranked organization.

Frequency and formality of feedback play a role in creating a collaborative environment as well. Some formality in the process is certainly a good thing, but consider the ease of how information will flow at one big yearly review versus a much smaller weekly catch-up kind of review. If someone is overwhelmed trying to take in and understand an entire year's worth of feedback, they are less likely to be able to provide any feedback in return, whether that be a self-assessment of their own performance or addressing how someone else (their manager, peers, or team as a whole) could be improved as well. If someone has a much smaller weekly or biweekly feedback session with their manager, they will get more practice both receiving feedback and giving it in return. This will lead to greater information sharing in both directions, not just from the top down, which creates a more collaborative environment overall.

Organizational Pressure

There are two main types of pressures that individuals will sometimes find themselves under in a workplace - individual and organizational. Individual pressure comes from within - self-motivation that drives people to work and to improve. What we're going to discuss here is organizational pressure, how the organization as a whole responds to pressures from events (usually unplanned events) and how those pressures and stresses impact people at an individual level.

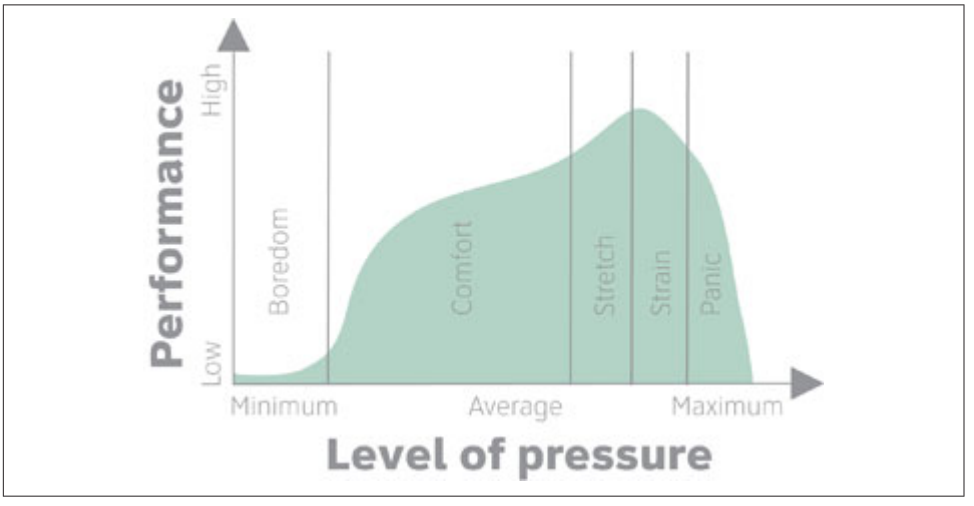

Figure 3-1. Organizational Pressure Curve

This relationship between organizational pressure and organizational and individual performance can be seen here. Too little pressure and the result is boredom, as there is no motivation to perform - even internal motivation is unlikely to hold up if the organization as a whole doesn't seem to care. As pressure increased to average levels, so does performance increase as well - but this is only true up to a point. Short periods of higher than average pressure will see short-term performance gains, but as the pressure continues to increase, either in intesity or in duration, performance starts to decrease again. At the very upper edge of the pressure curve is panic, where there is too much going on for people to be able to respond rationally or effectively and performance is severely negatively impacted.

Recent research has identified six main factors of organizational pressure and how these factors can impact performance. Taking a look into these factors, we'll see how teams can make changes in ways that will improve individual collaboration as well as organizational performance.

- **Workload**: How much work is expected of employees, and if they have the time and resources necessary to complete their expected work, is a big factor of organ-

izational pressure. As we mentioned in the What is Devops chapter, devops is not a way to get twice the work out of the same number of employees - while job descriptions might start to blur and startup employees might need to wear multiple hats, you should *not* be expecting one person to do the work of two full-time people. Getting feedback from people on how they are feeling about their workload, especially when done frequently (again, yearly reviews are not the way to go here), can go a long way towards assessing if a team's workload needs some redistribution or if it is necessary to hire more people.

- **Monotonous Work**: Monotonous work can quickly drain employee motivation, especially among more senior-level people, or if there is unnecessary competition among employees for who gets to do non-monotonous work. Some repetitious work is to be expected for junior employees who need the repetition to cement newly learned skills. Especially if the work is something that could be automated, leaving little reason why it should be done by hand, monotony is something that should be addressed. We'll address automation more in the Tools chapter, but also consider additional training or employee rotations to keep people engaged and learning. Monotonous work is more often seen in the boredom part of the pressure curve.
- **Career Development**: People who feel that they have a clear way to progress at their current job, seeing a benefit to their hard work, are more likely to feel satisfied by the effort that is expected from them. Conversely, a company that lacks career development options, whether that be a startup so small that it hasn't yet bothered to put together things like job levels, skills matrices, or requirements for promotion or a company that is perceived to play favorites when it comes to who gets promotions and raises, will likely see more turnover and less productivity as organizational pressure increases.
- **Work Relationships**: Having supportive and even friendly work relationships can go a long way towards improving morale and productivity. This becomes especially true at higher levels of organizational pressure - people who feel supported by the people around them, who are able to turn to others for help as opposed to it being everyone out for themself, or even people who are simply able to vent a little bit to an understanding ear over a coffee break are happier and more productive. Make sure people are given opportunity and encouragement to get to know each other - one company found that syncronizing peoples' coffee breaks across different teams so they were able to take breaks together increased their profits by $15 million.
- **Conflicting Goals**: When employees are given conflicting goals, it makes it more difficult to know where they should be focusing their attention, as well as having the effect of increasing their workload as they try to go in multiple directions at once. Keep an eye out for other people, whether that be managers or individual contributors, trying to manage (or micromanage) employees. This is another

area where having more frequent feedback sessions can be beneficial - someone who is able to ask for clarification of their roles or goals at a weekly session will spend much less time being frustrated than someone who can only do so semi-anually.

- **Compensation**: Finally, although research has shown that money is only a motivator up to a certain point, a serious mismatch between organizational pressure (especially workload) and compensation can harm morale and productivity. People want to feel that the company understands and appreciates the effort that they are putting in. Here as well if there is no clear process for negotiating compensation or raises and bonuses appear to be given out unfairly, issues are likely to arise.

Again, there are many other books and papers that go into even greater detail on management strategies for both individual and team performance. Take a look at the Further Reading section at the end of this chapter for recommendations on where to look if you want to learn more about these topics.

Superstars and Superflocks

With the rise in popularity of concepts like "rockstar developer" and "10X engineer", many companies and hiring managers are trying to hire those elusive "superstars", offering ridiculous compensation packages or the ability to work with whatever programming language or other tool strikes their fancy. This might actually do more harm than good, however.

Evolutionary biologist William Muir of Purdue University performed an experiment on flocks of chickens, trying to find out how to make chickens more productive in terms of their egg production. A regular flock of regular chickens, left to its own devices for six generations, ended up increasing its productivity. Muir also created a "superflock" from the most productive chickens, and with each generation selected only the most productive chickens to breed the next. Instead of being even more productive, this superflock ended up with all but three members dead. The "superchickens" were only more productive at the expense of the productivity of others.

As it turns out, these same principles apply just as well to humans in the workplace. A study at MIT that looked at the productivity and creative problem solving skills found that the most productive and creative teams were not the ones created from all "superstar" engineers. Intelligence and raw engineering talent wasn't at all a good predictor of the best teams. Rather they found that the best teams had higher social sensitivity (better known as empathy), gave each other close to equal time to speak, and had more women in them. It was unclear if having more women helped bring about the higher empathy and more equal speaking time since women are often socialized to be empathetic, listen more, and interrupt less, but it was clear that the increased empathy and communication were deciding factors in team productivity.

This is an idea known as *social capital*, or the value of the social networks and interactions that people have. Social capital works through means such as greater information flow, reciprocity and helpfulness, and interdependency and trust. Compare this to a team that is focused around a superstar employee - help and information are likely to only flow in one direction, there is no interdependency, and likely very little trust. Social capital is something that takes time to develop, and whose benefits become increasingly apparent as time progresses.

To get the productive teams and organizations we want, we need to stop focusing on these "superstar" employees that erode trust and social capital, and instead focus on growing empathy among our existing teams. Collboration, helpfulness, and communication are all things that help people bring out the best in each other, and being able to bring out the best in others, rather that competitively only focusing on the best in yourself, is what takes people and teams from good to great.

Negotiation Styles

In today's workplaces, with increasing demands for both increasing product performance and reducing costs, individuals are more and more likely to find themselves with competing demands for where they should focus their time and attention or what their goals are. The conflicts that will arise from these demands will need to be resolved somehow - there are several different ways this can be done, which we'll look at here in terms of negotiation styles.

Territory and "Personal" Space

Edward Hall proposed four different zones of space; intimate, personal, social, and public in his book *The Hidden Dimension*.[1] Intimate space is specific to our closest relationships; family and close friends. Personal space is specific to the casual acquaintances, friends and work associates. Social space is specific to space that is comfortable for social interactions with acquaintances and strangers. Public space is specific to space that is perceived as impersonal.

When companies press for constricting space by doubling up offices and creating small shared cubicules they start to encroach into personal and social spaces adding to the tension between individuals. This tension can add to any conflicts created by differences in goals and motivations.

Competition occurs naturally where people are coexisting or sharing the same space. It's not necessarily a negative thing, but the lack of sharable goals means that it doesn't

1 Hall, Edward T. The Hidden Dimension. Garden City, N.Y.: Doubleday, 1966. Print.

lend itself well towards accomplishing shared goals. If people on a team are focused on competing without any kind of cooperation or collaboration, results for the team as a whole will be less optimal with increased time to market, decreased innovation, and decreased morale.

Accomodation is a second style of negotiation of individuals within a team that involves one individual helping another with the goal of building a better relationship. If George accommodates the General, the General benefits at George's cost.

Avoidance is a third style of negotiation of individuals within a team that involves individuals avoiding the problem. While this style doesn't increase conflict it doesn't generally lead to improvement. If George avoids resolving conflict with the General, neither benefits. Avoidance dynamics can be observed in multiple ways:

- High tension between individuals.
- Lack of depth in communication and resolution.
- Rarely set goals or direct no's.
- Unfullfilled commitments and missed deadlines.

Power Differentials

We can't have a discussion on conflict and negotiation without making note of differences in power. Power differentials can happen because of a variety of reasons. This can be as simple as the power structures built into the workplace, where managers have more power than their reports or senior engineers having more power than junior ones. However, they can also take more subtle forms, with members of underrepresented groups in the tech industry such as women, people of color, or LGBTQ people having less power than members of more represented or dominant groups.

These power differentials can have substantial effects on negotiation styles between people. On the one hand, people with less power might avoid any kind of conflict, hoping to avoid being co-opted or having to make compromises they don't want to make, knowing that they have less power and are likely to be the only ones "compromising", which isn't really a compromise at all. Alternatively, the high-power side might also avoid conflict or negotiation because they see no need to - if they can impose their will or solution on others, who have no choice but to accept it, they don't have much incentive to negotiate. It's important to keep these power differentials in mind when considering negotiation styles.

Compromise is a fourth style of negotiation of individuals within a team that involves individuals helping one another by each giving up something of value to get something of value. If George and the General compromise, then they both benefit but at a

cost to each. It should be noted that when only one side is giving something up, this is accomodation and not a true compromise.

Collaboration is the fifth style of negotiation of individuals within a team working with others to achieve shared goals, often taking the form of knowledge sharing, learning, and building consensus among the individuals involved. For a team to best work towards its goals, its members must be able to work together towards goals that are shared between all of the individuals involved. In this chapter, we'll see that teams with more individual collaboration are more productive as a whole, as well as being better from the points of view of the people in them - and since lower turnover is usually better for team morale and productivity, this is a positively reinforcing cycle.

From Competition to Collaboration

One of the key factors to getting individuals to work well together is modifying the style of negotiation in the workplace from competition to one of collaboration.

What makes a team a collaborative environment as opposed to a competitive one? Since competition occurs when resources that multiple people want are scarce enough that people have to worry about not getting any or enough of the resources they want or need, a team that acknowleges these factors and helps individuals to mitigate them is more likely to be a collaborative one overall. A good manager can be the key to having reports who work well together and trust each other rather than being pitted against each other. This involves a great deal of understanding different working styles and improving communication. These skills can be incredibly valuable for individual contributors to understand themselves.

Communication

A large part of fostering collaboration as the primary negotiation style among a team or workplace ultimately comes down to communication. We first saw this idea in action when we introduced the devops compact - without effective communication, neither the shared goals, strategies taken to reach them, or contingency plans would have been anywhere near as likely to succeed.

Why Communicate

Aside from simply answering a question or telling somebody what to work on next, there are many different reasons why we, as people, communicate with each other. Five key reasons are understanding, influence, recognition, and building community. In this section we'll take a look at these different reasons for communicating before moving onto the topics of what and how we communicate.

Understanding

A large part of communication is designed to increase understanding - this could mean a clearer understanding of what someone expects from us, a deeper understanding of a technical topic, or anything in between. As discussed earlier, formal mentorship programs are a great way of increasing understanding, but even without this, there are plenty of opportunities to increase peoples' understanding in your environment successfully. An established community of practice meeting, whether it comes in the form of regular coffee talks, a team hackathon, or bug fix sessions, is a great opportunity to show new people the set of expectations that are implicit in the environment. This is a form of implicit understanding, picking up on ideas, norms, and social customs through observation, rather than the explicit understanding that comes from a mentoring session or a formal lecture.

Embracing a *learning culture* and encouraging social engagement around knowledge sharing provides appropriate contextual clues towards understanding that are not generally present through self-learning on a subject. Rather than examining our environments and trying to encapsulate everything into checklists or other documents we need to recognize the importance of community building in environments.

When a single individual becomes responsible for large amounts of systems and processes, that individual distils a large amount of knowledge through situational awareness and application of learning. Without active dissemination of this knowledge to others, you build islands of knowledge that are vulnerable to external events in your organization. Having just a couple people who understand a given topic also increases pressure on those individuals ("No, George can't go on vacation, he's the only one who can fix the database!") which can increase stress and the likelihood of burnout. Communicating to share and spread understanding is a great way to grow the skills and increase the robustness of your organiation.

Many times, understanding includes an aspect of historical perspective. Given the complex systems that we work with and the organic way they grow and evolve over time, it is not always obvious to someone new to a team or project why things are the way they are. This sort of context is greatly important to being able to fully comprehend and contribute to something. This is especially true for operations teams who are tasked with deciding if something is anamolous or not - was this alert a false alarm or is there an actual issue that needs to be investigated? Being able to communicate the historical contexts allows new or more junior team members to grow and develop their knowledge and understanding much more quickly than they would otherwise.

Influence

Communication can also be designed to influence people. The most common example of this, in a work context, is trying to get someone "on your side" or to come

around to your point of view when there is a disagreement about how to do something. If George wants to use a nosql data store for the team's upcoming project but The General thinks MySQL would be a better fit, they will both try to *influence* the people around them, whether that be their peers or their manager.

There are different methods of influence, some that are more positive or collaborative than others. Certainly one can influence others by interrupting anyone who disagrees with them, by being the one who argues the loudest and the longest, or by using some sort of power or coersion. None of these lend themselves well to a healthy or empathetic team dynamic - while influence may have been achieved, everyone else is likely to feel resentful. As we'll discuss more later, the most effective way to influence others is to find enough common ground that not only will they do what you want, but they will actually *want* what you want as well.

Recognition

Giving recognition is another common reason that people communicate. Giving recognition can improve morale, since people obviously want to feel that their work and accomplishments are noticed and appreciated, it can enhance cooperation between employees as they see each other more as both generous and contributive, and it can help to reinforce behaviors at work that you would like to see more of. Recognition usually has two parts to it - the identification or realization of something that should be recognized, and the actual communication of that sentiment.

Identification of opportunities for recognition is a skill that takes time, since if you aren't in the right mindset - for example, being in a negative mood, being stressed out due to a heavy workload, or being in a team environment that it incredibly competitive and "everyone for themself", it will be harder to realize times when praise or recognition would be appropriate. Communicating the recognition is another skill. Some people feel less comfortable praising others, especially if there hasn't been much recognition in the workplace previously. Recognizing people publicly might feel more uncomfortable than doing so privately, people might feel more recognized if praised publically, such as in a team meeting.

Building Community

Finally, communication can be used to build communities. As the previously-mentioned MIT study showed, teams with greater empathy and more equal communication are more creative and more productive, and building community goes hand in hand with these things. Teams where people regularly talk about things outside of strictly work-related matters have higher levels of trust and empathy, are able to be more productive and handle stressful times better as a group. People often interact better on an individual level when they are able to see each other as complete individuals, not just email addresses or entries in the company's staff directory.

There should not be an expectation that employees will become best friends outside of work, and there is a fine line between getting to know someone as a person and getting too invasive or personal - some people are more willing to share personal parts of their lives than others, and that's fine. The key isn't to force community-building interpersonal communication, but to create opportunities for it, gently encourage it, and then allow it to happen naturally. Building relationships and building community both take time; neither happen overnight and neither can be forced. Something 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 towards building strong communities.

What we Communicate

Understanding why we communicate leads to understanding *what* we communicate. There are many different things that people might communicate to each other in the course of their work, and different tools or media will be more or less effective for these different types.

The contents of our communications can take on many different forms. Sometimes we are asking questions to which we require a response, while some questions might be putting out feelers and just trying to gauge the general feel of something. We might be brainstorming and trying to quickly gather as many ideas for something as possible. We might be having a discussion and trying to reach a consensus or make a decision on something significant. We might be sending out information that requires no response, but some of it might be just of casual interest while some might be mandatory for everyone in its audience to read and understand. Some communications might make more sense when spoken or heard aloud, some might be better written down, and some make more sense conveyed visually.

These different types of communications can and do happen over multiple types of media. If we're asking for information, we could send out a Google Form for people to fill out, send an email to a mailing list with a subject of "Please Respond," or we could ask our Twitter followers in 140 characters. Different qualities of our communications determine which media will be more appropriate than others. Communication can be urgent (needs a response right away) or non-urgent (needs a response whenever is convenient for the recipient), and it can be casual (doesn't really matter if it gets seen) or important (definitely needs to get seen and understood).

With these different ideas of what we communicate in mind, let's take a look at how the content affects the tools we choose for various communications.

How we Communicate

The methods for communication we choose will, if you're trying to be most effective communication, depend on the content, urgency, and importance of your communi-

cations. In addition, you'll want to consider what kind of audience you're trying to reach, and how much context and investment from the intended audience will be needed for the communication to be effective. We can consider how organized the communication needs to be, versus more free-form.

Here, we'll break down a (non-exhaustive) list of communication methods by a variety of factors.

Table 3-2. Different Communication Tools and Methods

	Urgency	Audience Reach	Investment	Context Required	Organization
Email	Low	High	Medium	High	Medium
Impromptu in-person (or video)	High	Low	Medium	Low	Low
Chat	Medium	Medium	Low	High	Low
Meeting	Very Low	High	High	Low	High
Twitter	Low	Medium	Low	High	Low
Github Pull Request	Low	Medium	Medium	Medium	Medium
Post-it Notes	Very Low	Medium	Low	High	Low
PagerDuty Pages	High	High	High	Medium	Low
Nagios Alerts	Medium	High	High	Medium	Low
Books or Blog posts	Very Low	Low	Medium	Medium	High
Pictures, Graphs, and Gifs	Low	Low	Low	High	Low

Let's look at what the columns in this table indicate in more detail.

- **Immediacy** refers to how quickly communication can be established - walking up to someone in-person has high immediacy because you can tap them on the shoulder and interrupt them, whereas email has low immediacy because you can't control how often other people check their email. Meetings can have very low immediacy because scheduling around the availability of people and meeting places can be very time-consuming indeed.
- **Audience reach** is how well a medium allows you to reach all of your intended audience, so while an email to an individual has a pretty good chance of being seen by who you want, chat messages will likely only be seen by people who hap-

pen to be online (or in a given channel) at the time, depending on what kind of offline messaging and alerting options your chat solution has.

- **Investment** describes how much time and effort is required for people to participate with a given form of communication. Meetings are one of the highest in terms of investment as people have to take time out of their other work and either go someplace in or dial in remotely to participate. An email, book, or blog post requires a medium amount of investment in terms of finding the time to read it thoroughly, but something like chat or twitter is low investment.
- **Context** is how much context is required for a given communication medium, or how likely misunderstandings are to occur without it. Twitter, chat, and email require high context, because of how easy it is to misinterpret phrasing or tone - in general the shorter the text communication is, the more likely misunderstandings are to occur because of context being lost. In-person (or video) forms of communication are much lower context because people can see body language, hear voice and tone, and quickly bring up and resolve misunderstandings.
- **Organization** refers to how organized the thoughts or ideas ought to be in a particular medium. Meetings are high organization because they really should have an agenda so peoples' time isn't wasted. Email is medium because people can choose to organize their thoughts quite a bit before sending, while chat and Twitter are low due to their often rapid nature and short form.

Communication and Context

A lot of how we communicate is also impacted by the context of our communications. This is not just the amount of context that various communication media and methods can provide as described above but also the situations and circumstances in which communication can take place.

Regular communication as part of everyday work is likely to be very different from the communication that takes place during an emergency, such as a site outage or other operational issue. While shared jokes, internet memes, and funny cat pictures might be a good way to build cameraderie and trust during normal work, they can be an unwelcome distraction during an ongoing issue. Companies that rely heavily on chat would do well to create a separate chat room or channel for these situations such as a "war room" which is reserved for straightforward, on-topic communication only.

Context can also informed by the power differentials discussed earlier. Talking with or sending an email to someone higher up in the organizational hierarchy will likely have a different tone to communications directed at peers or people further down in the org chart. Keeping in mind the effect that these differentials can have on negotiation and collaboration can be a good way of improving the effectiveness of interpersonal communication. A comment that might seem clearly lighthearted when sent to

a peer might take on a very different tone when sent to someone with less power in the professional relationship.

This can happen in more subtle ways as well, as there are other power differentials that exist outside of a company's org chart. This is especially true when considering gender diversity in the workplace and addressing the unconscious biases that exist there. Studies have shown that when women use the same language as men in terms of how they phrase things, they are perceived as more "harsh", "abrasive", or "aggressive", in a negative light, while men are praised for how "straightforward" and "take-charge" they are. On the other side of things, women are often judged negatively for softening their language by apologizing or using hedging words like "just". If they interrupt as much as their male colleagues they are often deemed "unlikeable", but without interrupting, depending on the office culture, it can be hard to ever get your opinions heard. These sorts of contexts can have an enormous impact on how successful our communications are and should be kept in mind.

Finally, whether or not team members are located together can also be very impactful on communication. If a company is just getting started with allowing remote workers, or simply hasn't given their remote employees much focus or attention, remote communication and collaboration can take a serious hit. If the majority of work-related decisions happen in person, usually outside of a formal meeting context, remote employees might find themselves missing out of valuable information.

One way of dealing with this is the practice of communicating "as remote by default". This means using the remote-friendly methods of communication, most commonly email and group chat, for as much communication as possible and as the first choice of method, not a last resort. If, instead of walking over to someone's desk to ask a question but asking instead in a team's chat room, remote employees are given a chance to learn and participate in these discussions that they wouldn't have before - this leads to more information and better visibility for the entire team. Having a searchable record of communications for future reference can be very valuable as well.

Trust and Empathy

Effective communication, in addition to more commonly thought of use cases like distributing information, is key to building trust and empathy between individuals, and a shared foundation of trust and empathy is what enables devops to really work. This goes back to the root of the devops compact that we introduced earlier in the book. In order to be able to assume that all parties are still on board and still working towards the same goals, we have to be able to trust them, and in order to really understand and get on board with these shared goals, we have to be able to empathize with each other.

Creating a compact involves being able to establish and communicate a shared vision or shared goals - the commonalities that will be shaping the big picture of what people are working on even as the details differ. Aside from increased empathy and a common focus, a shared vision should give individuals a much clearer picture that will help direct, inform, and guide autonomous action. Goals that are too vague or that don't seem relevant are harder to fully grasp and realize, which might not provide individuals with the motivation, context, or ability to choose effective courses of action.

Increased trust can go a long way towards increasing the resiliency of a team. Without trust, individuals can be very protective of their projects or areas of responsibility, often to the detriment of their own health or the team's overall productivity. Imagine, as an example, a team of system administrators so protective of their servers, and so distrustful of anyone else touching them, that they restrict any privileged server access at all to just their own team. If other teams don't have the ability to install necessary software or deploy code to these servers, this team is likely to become a bottleneck, a barrier that other teams will end up resenting or finding ways to work around. This is a typical example that comes to mind when people think of the drawbacks to heavily siloed environments.

That kind of negative impact can become even more apparent when it is a single person rather than a team being too closed off. If only one person knows about or has access to something, they become a single point of failure for that thing. If that thing breaks and that one person is sick or on vacation, the rest of the team might be dead in the water and unable to be productive until the one person can be reached, which will either lead to the rest of the team being blocked or that person never being able (or willing) to take time off. With increased trust, this knowledge and responsibility can be shared not only between people but between teams, increasing the resiliency of the organization as a whole.

Developing Empathy

Empathy, the ability to understand and share someone else's feelings, is a skill that can and should be learned and developed. Its benefits are becoming increasingly well-known, both in and out of the workplace. More empathetic individuals are less egocentric, less socially aggressive, and less likely to use stereotypes when considering others - they are also much more likely to compromise during debates or other disagreements rather than tending towards one of the other negotiation styles discussed earlier. Research has also shown that increased empathy is positively correlated with better job performance.

While a great deal of empathy is developed (or not) in childhood, there are many ways that empathy can be learned as an adult as well. We'll take a look at a few of the most common and effective methods and how they can be applied in the workplace.

Listening

Listening is important for building empathy in general, but it can be even more beneficial during disagreements or other heated discussions. Too often when we are disagreeing with someone we are merely waiting for our own next chance to start talking and planning what we are going to say, rather than really listening and trying to understand where the other person is coming from. Instead, try slowing down and forcing yourself to actually listen, and instead of interrupting, wait until they are actually finished speaking before considering your own response.

Active listening is another good skill to consider here. This involves reflecting what you think the other person just said, paraphrasing or summarizing to make sure that what they heard and understood was the same as the original meaning or intention - this makes sure that both parties are on the same page and are actually talking about the same thing. Paying attention to non-verbal cues, such as tone of voice, rate of speech, body language, and facial expressions is also a key part of listening. Because these non-verbal cues aren't conveyed via text, making sure you have a good video (or at the bare minimum, audio) setup is key if you have any remote employees.

Asking Questions

After listening, asking questions can be a great way to build understanding and clarify meaning, often as part of active listening. In addition to asking questions of others, we can also ask questions of or to ourselves. Cultivating curiosity about others, whether they be strangers or the other people on our team, is a habit that can expand empathy by helping us understand where other people are coming from. Questions could take the form of "Could you clarify what you meant when you said X?" or something more hypothetical, such as "Where do I think that person on the train is heading right now? What might they be looking at on their phone?", or self-facing, "What unconscious biases might I have that are impacting my opinions on this?" Combined with then listening to the answers we get, either from ourselves or others, asking questions can be a very powerful tool for building empathy.

Imagining Other Perspectives

Going beyond asking hypothetical questions about what other people might be thinking, doing, or feeling, we get to trying to imagine ourselves in other peoples' shoes. It's one thing to say that we should assume good intentions, but we can go further and ask ourselves, how might this person be feeling right now when I disagree with them? What good intentions can I identify that this person holds? What might their positive motivations be, and how do they impact the disagreement or discussion that we are having? What valid arguements might they have against my point of view?

A research study conducted at Harvard University showed that simply being given a description of someone else didn't do much to increase empathy, but being given

more specific information about a disagreement and the other person's point of view, such as information about the person's thoughts, or overhearing the details of a discussion or conversation enabled them to step into someone else's shoes much more easily, making them more likely to compromise and negotiate.

Appreciating Individual Differences

In addition to imagining what other peoples' thoughts, opinions, and motivations might be, we can teach ourselves to appreciate those differences as another way of cultivating empathy. Working with, genuinely listening to, and imagining ourselves as the various people that we work with can go a long way towards breaking down biases, both conscious and unconscious. Consider the different working styles described above and how they complement each other - both starters and finishers, purists and pragmatists, can combine their skills to bring projects to fruition. Then take this understanding and appreciation and apply it to the other personal and professional backgrounds you might find - appreciating the benefits of differing perspectives can go a long way towards appreciating diversity and building empathy among different people.

Developing Trust

Trust and empathy go hand in hand - as one grows, so often does the other, and visa versa. There are different strategies that can be used specifically for developing trust, however, and both are needed to foster an environment that can be really and reliably collaborative. One of the differentiating factors between a group and a team is the presence of trust (though we'll discuss more differences in the Affinity chapter).

Swift trust is a form of trust that occurs in short-term or short-lived groups or organizations, where trust is assumed to be present at first and then verified as time goes on. First explored by professor of organizational behavior Dr. Debra Meyerson, it is often used in quick-starting groups or teams that lack the time necessary to develop trust the way that normally occurs naturally in longer-term relationships (here meaning any relationship between individuals, not specifically or at all limited to romantic relationships). Because time is limited, time members will initially assume trustworthiness and then verify and adjust that trust later based on others' actions.

Research has showed for some time that self-disclosure is one of the hallmarks of trusting relationships - being open enough to share things about ourselves can increase feelings of trust and intimacy between people, as well as increase cooperative and collaborative attitudes. Of course, there is a balance that must be achieved when practicing self-disclosure in the workplace. Not enough disclosure and suspicions may grow, wondering what someone is hiding and if they can be trusted, but too much or the wrong kind of disclosure, including inappropriate admissions or what

might sound like betrayal of someone else's confidence can damage trust and credibility as well.

The trust-but-verify model can be used when dealing with sharing professional responsibilities as well. Someone who waits until trust has already been "earned" before doing something like sharing access to a project they've been working on might find themselves with a chicken-and-egg problem: how can someone earn trust if they're not given opportunities to earn that trust because they aren't yet trusted? Instead, people should be encouraged to share responsibilities with trust given first and followed up by verification. This is also true for sharing power and decision-making ability in addition to things like project work and responsibilities.

Trust plays a role when considering the human side of growing an organization in addition to the technical one. As we'll see more when discussing employee retention in the hiring chapter, the perception of fairness is very important to employee satisfaction, meaning employees need to be able to trust that they are being treated fairly. One thing that can help in this regard is developing formalized roles, job levels, and pay scales, as well as providing a reasonable amount of transparency in these areas. This can help people to understand what the requirements of their role are or processes for pay increases and promotions, which can notably decrease the feeling that they are being unfairly passed over or wondering why someone else got a promotion, feelings that can understandably decrease trust.

Sharing risks in addition to sharing responsibilities and resources is key as well. If two teams are sharing work or resources on a project, but only one of them will be negatively impacted if something goes wrong with it, the team with the risk might be distrustful of the team without. In addition, this can lead to a power differential in favor of the team without (or with less) risk and the problems associated with that.

CHAPTER 4

Hiring: Choosing Individuals

Introduction and Audience

This chapter is aimed at those who want to examine their current hiring practices, optimize the hiring and interview process, and measure and iteratively improve hiring in their environment. It will be beneficial to managers, hiring or otherwise, who have hiring needs. We will cover identifying your hiring goals, interviewing techniques, hiring for diversity, and retention of employees - all key aspects of growing, transforming and maintaining an effective organization.

Determining your Hiring Needs

The first step in hiring is determining what kinds of candidates you are looking for. There are quite a few factors that you'll want to consider in this step. With tight time constraints with the hunt for talent, or limited window for open job requisitions in a tightly constrained budget focused team, it can be tempting to spend little time on analyzing needs and move ahead quickly to try to source and interview candidates. The more time you spend finding out what you need to complement the team you have, the more successful your searches are likely to be.

Position and Skills

The skills you're looking for might seem relatively straightforward - if you're running a Ruby on Rails application, it might seem obvious that you need to hire experienced Ruby on Rails developers. There is more to having a person be a good fit for a position than the particular technologies they've used before, however.

The position you're looking to fill, and the reason it needs to be filled, can have a good deal of influence on what skills will be a good fit. If you are looking for someone

to fill a very specific role for a very particular project, then it might make sense to focus on someone who already has the specific skillset you need. For example, if your company has started processing payments and you need to be PCI compliant by an upcoming quarterly deadline, then it would likely be better to find someone who is well versed in those specific compliance issues already rather than training someone who isn't.

Without these kinds of time pressures, you can search more broadly for candidates who will fit your team in a more cultural sense. Even if you need a specific skillset, such as an operations team who has outgrown cloud-based architectures and needs to start hiring data center technicians and network engineers, with a reasonable amount of time you can be more flexible as to how you get those skillsets. If someone on your existing team has an interest, could they fill that position with proper training? What about a more junior person who is looking to grow their skills?

Even when you're looking for someone to work with a particular technology, without a time crunch, direct experience with it isn't always necessary. In our Ruby on Rails example, if you hire someone who has worked with Rails before but is a poor culture fit or lacks critical thinking and troubleshooting skills, what will happen if you find it necessary to switch technologies in the future, or add a new one to your stack? On the other hand, if you find an excellent Python developer who has great people and learning skills, her ability to pick up new things will be a much better benefit in the long run.

A Balance of Skills

Recall the axes of Collaborative Styles that we introduced in the Collaboration chapter. When thinking about the requirements for a position, you'll probably want to keep these axes in mind. Take a look at where your current team members fall on the various axes, and keep an eye out for any where the team is starting to become unbalanced. While it probably doesn't matter so much if you have more night owls than early birds, if you find that your team doesn't have a good balance between starters and finishers, and purists and pragmatists, that can lead to issues with overall team productivity and quality. If you find that you need to hire one or the other to address this imbalance, keep that in mind throughout the interview process.

Timeframe

As we touched on in the previous sub-section, there are occasionally circumstances when time pressures require faster hiring decisions than you might otherwise want to make. Outside of externally mandated deadlines, time questions often revolve around how quickly new team members will be able to get up to speed and make meaningful contributions to the team. People often shy away from hiring more junior candidates

because they worry that it will take too much time for them to get "real" work done, or that more senior team members will have to spend too much of their time training and mentoring the new people.

When considering time, you should also take into account a candidate's fit within the team. If someone has expressed a strong preference for working at small startups and yours is growing rapidly, will you have to replace them in six months when your company has gotten bigger than one they want to work at? If you hire someone who is technically very experienced but who nobody else can stand to work with, how much time will be spent replacing either them or other team members who leave because of that person? Also, as we'll discuss further in the sourcing section, not being willing to invest in training and growing more junior candidates will likely lead to a more homogeneous team, which is something you'll want to consider as well.

Budget and Resources

Your budget and resources will obviously have an effect on the number of people you can hire, as well as often their seniority. If you can't or won't pay close to a competitive market rate for the area you're hiring in, you will find far fewer candidates willing to accept your offers. This is an area where having distributed teams can be a benefit - a competitive salary in Denver, Colorado will be less than what is market rate in San Francisco, California, because of the incredible difference in cost-of-living expenses, and opening up positions to more locations will provide a much wider range of candidates.

Smaller startups, especially ones without the benefits of millions and millions of dollars in Silicon Valley venture capital funding, likely won't be able to offer the same salary (or stock options or insurance options) as a bigger, more established company. What you can do to make up for that comes in the form of cultural benefits. Can you offer flexible hours, remote work, vacations and parental leave, or just a culture that strongly discourages working more than 40 hours a week? Can you send employees to training, let them speak at industry conferences, give them a platform to publish technical blog posts, or let employees spend some of their time working on projects they find interesting outside of their normal responsibilities?

It's important to be as competitive as is financially possible when considering candidates' compensation, but most people also have non-monetary goals with their career including making a postive impact on the organization, helping solve social or environmental challenges, or working with a diverse group of people. Providing a more creative environment that provides outlets for these other goals can be the differentiating element that makes a company stand out from the competition.

Sourcing

Once you've determined what you are looking for in a candidate, the next big question is where to find candidates.

Job boards are a good place to start because they tend to be frequented by people who are actively looking for new opportunities. Generic job boards such as Monster and Indeed get postings in front of more eyes, but industry-specific job boards tend to get higher quality candidates in terms of matching up with tech-specific skill sets. StackOverflow Careers is one of the most well-known technology job boards, and companies that post to it can share their score on the Joel Test, a 12-question test created by software engineer, writer, and CEO Joel Spolsky that is designed to help companies create productive environments for software developers to work.

The Joel Test

The Joel Test's questions, which can be found online, are a good starting point for evaluating your company in terms of how good a place to work it is for software developers. It is only a starting place however, and it doesn't necessarily apply to every company. It should also be noted that there are many more teams equally important to companies than development teams, and you should regularly be assessing how productive your workplace is for those teams as well.

A new kind of site gaining in popularity somewhat resembles a dating site - both companies and individual job seekers create profiles, can search for profiles based on criteria they select, and can choose what kinds of communication they want to get based on other profiles. Some sites, such as Whitetruffle, have profiles that are anonymous until there is a mutual opt-in. Once a company and a candidate have both indicated that they're interested in each other, they are given each other's details and the ability to contact each other. Because of the opt-in requirements, sites like this are often a good way to cut down on candidates who aren't really interested.

Recruiters are often popular with companies and hiring managers because they cut down on the amount of time that managers or individual contributors have to spend on sourcing instead of their other responsibilities. There are internal recruiters, who are full employees of the company they represent and find candidates for only that company, as well as external recruiters who are working for many companies at the same time, usually on some kind of commission-based structure.

For individuals, recruiters are often seen as a nuisance through repetitively bulk emailing, incomplete form letters, or inappropriately categorized job requirements based on current position. We'll go over some detailed examples in this chapter, but

keep in mind that a recruiter can do a great deal of damage to your company's reputation among potential candidates nullifying the potential time-saving benefits.

An alternative to reaching out to people who haven't had any contact with your company or given any indication that they are looking for a new job is to find people organically. When your current employees go to local meetups or tech conferences, they will be meeting people and making connections, often genuinely getting to know people as opposed to cold-calling (or cold-emailing) them. If these people ever become open to new opportunities, they'll often reach out to these sorts of connections and acquaintances first, before looking on job boards or contacting recruiters themselves. The reasoning behind this is that there is a greater amount of trust with a personal connection, with people who can answer questions more candidly and vouch for the companies and other employees. Candidates want to know that they'll be happy with whichever job they take, and that information is much more believable coming from an individual contributor than from a recruiter who is getting paid to talk up the company.

Diversity

Many companies and teams these days are looking to grow and hire diverse teams. Although there are many resources available that can go into much more depth than we have space for here, we'll briefly discuss the benefits and axes of diversity and how teams can source and hire a wider range of people.

Benefits of Diversity

The tech industry tends to be very homogeneous in terms of the people in it, consisting mostly of heterosexual, cisgender, white men, in proportions much higher than they are found in the general population. There have been many studies that show that diverse, heterogeneous teams tend to outperform homogeneous ones.

> "Strength lies in differences, not in similarities."
>
> —Stephen Covey

Diversity is critical for innovation, with the differing ideas, perspectives, and viewpoints that come from different backgrounds being a crucial part in developing new ideas. Diverse teams will be able to develop products that reach a wider customer base due to these differing experiences. The more closely different groups or individuals work together, the more creatively stimulated people tend to be.

A 2006 study from Dr. Samuel R. Sommers, director of the Diversity & Intergroup Relations Lab at Tufts University, showed that racially diverse groups performed better than all-White groups. Heterogeneous groups exchange a wider range of information and discuss more topics than homogeneous ones. Additionally, White people individually performed better in mixed groups than in all-White ones. Similar studies

have shown that gender has the same effect - mixed gender groups out-perform groups of all men on individual and group levels.

These benefits are even more pronounced when groups are working on tasks that require creativity, where divergent thinking can be of benefit, or when they are required to interact with non-group members. In practice, this means that teams that interact with customers will benefit from increased diversity (especially if your employees are not also users of your product), leading to increased customer satisfaction. A 2000 study from researcher and management professor Orlando C. Richard showed that cultural diversity in the workforce led to increased company performance during periods of company and business growth.

Although diverse groups do lead to increased performance at individual, team, and company levels, the drawback can be that they can also caused increases in interpersonal conflict, which often leads to lower morale. It makes sense that differing viewpoints and opinions can lead to disagreements and conflicts, so it can be beneficial to make sure employees know how to handle conflicts most effectively. It's important to make sure that disagreements are handled well, not just by whoever screams the loudest winning, because teams that can survive these interpersonal conflicts show even more performance gains in the long term.

Having employees who know how to resolve disagreements without damaging their work relationships is always a good thing. Thinking back to the devops compact, we have to know that we are ultimately working towards the same goal, and act with the understanding that we still share that goal in spite of disagreements.

Axes of Diversity and Intersectionality

Many diversity initiatives in tech start by recognizing a lack of women in the workplace. Rectifying the gender disparity is necessary, but not sufficient. There are many different axes that diversity can take. These include:

- Gender and gender presentation
- Race and ethnicity
- National origin
- Sexual orientation
- Age
- Veteran status
- Disability

Increasing diversity on any one of these axes is important, but only one axis does not mean your company is truly diverse, nor that it is a safe place to work for a wide range of people. Intersectionality is defined as the study of intersections between dif-

ferent forms of oppression or discrimination, and how these different forms of oppression are interconnected. The term was first coined by legal scholar Kimberlé Crenshaw and is an important consideration when thinking about diversity at your company.

Diversity, like any other devops practice, is not a simple thing that can be implemented once and then checked off a list of things to do. It's an iterative process that must be monitored and measured. Your reasons for considering diversity matter, and how successful your efforts are will depend on them. Both initiatives are ones that should be undertaken out of a genuine concern for improving the lives of all people in your company and the community as a whole.

Unconscious Bias

People often think of sexism, racism, or any other "-ism" as an overt thing like one might see when watching Mad Men. Those sorts of biases are certainly things to be fought against, but unconscious biases can be even more insidious in their subtlety. Unconscious biases are shaped from our environments and the times and cultures we live in, and we often don't notice their presence. These are ingrained thought patterns that cause us to assume that men are more qualified than women with the same qualifications, for example, without even realizing that we're doing it. The best way to fight against unconscious biases is to become (and stay) aware of them, which is why companies like Google and others are starting to offer unconscious bias training for their employees.

Hiring Considerations

There are things to be considered throughout the hiring process when trying to improve the diversity of your workforce. We will touch on additional ideas in the interviewing and retention sections as well, but here are a few things to keep in mind when sourcing candidates.

- Keep an eye out for language that can be exclusionary, such as overtly masculine or militarized phrasing as well as overtly sexist, racist, or homophobic remarks in both job postings and communications from recruiters. External recruiters especially should be given as much guidance as possible on the tone and culture that your company is trying to convey. More specific examples of this will be given in this chapter's case study.
- Be aware of those unconscious biases - even when we mean well, we often inadvertently assume that a resume is better when it has a name that sounds like it belongs to a white man's on it. When possible, have everyone involved in the sourcing and hiring processes take unconscious bias training, and keep personally identifying information out of the sourcing process for as long as possible.

- There are recruiters and consultants who specialize in creating diverse teams. If you are struggling with finding as many diverse candidates as you'd like, it might well be worth bringing in a professional who has more experience in the area to help.
- While some teams are fond of having candidates submit "homework" as part of the screening process, keep in mind that this might put members of marginalized groups at a disadvantage, whether that be women with family responsibilities or people who don't have the time or inclination to do free work for a company.

Inclusivity

It doesn't matter much if you manage to interview and hire diverse employees if you can't retain them. Along with diversity, companies need to start inclusivity initiatives to ensure that individuals in the minority experience belongingness and encouragement to retain the uniqueness within the work group.

A common narrative we see in the case of a small startup who has hired their first woman in their engineering team. From the rest of the team's perspective, this is great - they're being more diverse, they now have an internal resource to help them avoid any potentially sexist mistakes! From the woman's point of view, however, it doesn't necessarily feel like an inclusive environment at all.

Many teams of men, upon hiring their first woman, will do things like say "Hey guys," when entering the office or a meeting, then look at the woman, wonder if she might feel excluded by that, and follow up with an awkward "…and girls (or gals)." This is well-intentioned, but many women said such additions make them feel more awkward and excluded rather than included. On top of the fact that we shouldn't be addressing adult women as "girls", drawing attention to peoples' differences from the rest of the group does not make for a feeling of inclusion. There is often a cost to members of minority groups in this kind of environment as they are often expected to take on a great deal of diversity-related work on top of their regular work responsibilities. They are asked to review job postings to make sure they are free of racist and sexist language, to represent the company and give it a diverse face at industry or recruiting events. They are often asked to represent all members of whatever group(s) they happen to be a part of - no woman is a spokesperson for all women. One woman cannot give insight into what all women everywhere are thinking or feeling.

Stereotype Threat

Stereotype threat is what happens when people find themselves in a position when they are at risk of confirming a negative stereotype about themselves and the group they are a part of. It has been shown in over 300 different studies to decrease individuals' performance, especially when they expect discriminations based on their group

membership or identity. For example, take the stereotype that women are worse at math than men. Women who are exposed to this stereotype will perform more poorly on math exams than those who aren't, as well as displaying more stress responses such as elevated heart rate and increased cortisol levels. Long-term exposure to stereotype threat can have the same negative long-term effects of mental and physical health that chronic stress does.

Studies have shown that a sense of belonging within a group can help to mitigate stereotype threat. If people are welcomed into the larger group or environment, if they feel that they are genuinely included, they are less influenced by the negative stereotypes that might lower their performance (and health).

There are many things that can be done to make sure your work environment is as inclusive as possible. We'll discuss more of these points in the Interviewing, Onboarding, and Retention sections later in this chapter, but here are some things to keep in mind when sourcing candidates:

- Make sure that any recruiters you use are aware of your diversity and inclusion goals.
- Send employees to unconscious bias training or an Ally Skills Workshop.
- Lead by example, and call out problematic language or behavior - don't leave that to be the sole responsibility of minority team members
- Organize employee resource groups. Establish places to address the needs of the diverse individuals, including community building, networking, and support. These groups facilitate resilience for the individuals mitigating some of the costs of being different to the majority.
- Audit your work environment. Determine how accessible key elements of the environment are to differently-abled employees beyond the government mandated requirements.
- If you ask people to do the work of reviewing your job postings to be more inclusive, compensate them for that work.
- Pay attention to if your language or behavior is racist or sexist (or homophobic or transphobic) all the time, not just when people from those groups might be present - preemtively create an inclusive atmosphere.

We will touch on these ideas and how you can make them part of your interviewing and retention processes later in the chapter. Overall, keep in mind that you get out of sourcing what you put into it - it's not enough to just throw together a job posting and hope that diverse, quality candidates will materialize out of thin air.

Interviewing

Interviewing is about more than just weeding out people who don't have enough technical skills to do a job well. It is a tricky process that not only assesses skills but a person's fit within a team as well. The goal of each interview should be to determine if the team and the candidate are well suited to work together, and it's important to remember that this should go both ways. It's not only about whether a candidate would be a good addition to the team, but also about whether or not the team would be good for the candidate.

Before the Interview

Preparation is just as important for interviewers as for interviewees. Being unprepared for an interview is a waste of time on both sides of the table - not something you want for your team or an impression you want to give out to future candidates. The first thing to do is to figure out the hiring criteria for the interview you are doing. Are you looking for a particular skill set for a specific project? Are you trying to fill a gap in your existing team, or replacing a person who left? Do you need to have someone senior, or are you willing to train and mentor a more junior candidate? If there are any hard requirements that would absolutely disqualify a candidate, figure those out well in advance (and be sure you're communicating these to candidates as well).

Once you know what you're interviewing for, people need to know how to interview well. This is a skill that takes practice, so it can often be helpful to have two people interview together, one more experienced and one less, so the more junior person can learn from the other's experience. In addition to unconscious bias training (those pesky things can and do affect us during interviews as well as just when looking at resumes) training people how to interview well can be a great benefit, especially if you are going to be hiring a large number of people any time soon.

Finally, figure out what the logistics of the interview will be. When should the candidate arrive, and who will greet them when they do? Who will they be talking to? The schedule should be determined and communicated to the candidate as far in advances as possible, especially for interviews that last the better part of a day. Interviewing is often nerve-wracking enough without having to worry about logistics. Setting expectations for the position and the interview process makes it a better experience on both sides. Be sure to plan for beverages, snacks, meals and restroom breaks as well based on the length of the interview.

What questions will people be asking? If multiple people will be interviewing the candidate in sequence, you probably don't want each of them to be asking the exact same questions. Of course it can be helpful to get different peoples' perspectives on how good a candidate's answers were, but since pairing can accomplish the same thing, you want to make sure you're not wasting peoples' times going over the same ques-

tions. A good strategy is to compare the criteria you have for the position and the knowledge areas of people on your team, matching them up as best you can to determine who will be asking about what.

There are a few things to keep in mind when interviewing a diverse range of candidates. If at all possible, make sure that people have a chance to interview with people who are like them. This can be tough when just starting with a diversity initiative, but even if they're on another team, a candidate of color would likely appreciate the chance to interview another person of color, to be able to ask them about their experiences at the company. Find out before the interview if your candidates have any schedule restrictions that would make a full-day interview difficult, such as transportation and childcare needs, and make sure that your interview spaces are accessible for more than just able-bodied people.

During the Interview

Interviews are challenging to every organization. We all value our time, and getting a good sense for if a candidate is a good fit for the team and visa versa usually requires a great deal of it.

A lot of discussion has happened around the "best" interview questions, especially for software engineering candidates. Books have been written on the sorts of questions that companies like Google are known for using. *How Would You Move Mt. Fuji* is one such example of this, coaching candidates on how to think through and answer questions such as that or similar ones like "how many piano tuners are there in New York City". The problem with questions like these or brainteaser questions is that they are very poor predictors of how employees will actually perform on the job.

Indeed, any question that seems to exist only to make the interviewer feel smart and the interviewee feel either stupid or nervous should be avoided. There are plenty of ways to assess how much a candidate knows without purposely exhausting them or making them feel stupid. Interviews can be a very good indicator of company culture, whether that is deliberate or not. An interviewer who continually interrupts a candidate is likely to do the same as a coworker, and an interview that is adversarial in nature indicates a culture that is probably aggressive and adversarial as well. Overly hostile interviews tend to self-select for more aggressive candidates, which can drive away people such as women who are socialized for, or simply prefer, much less aggressive styles of communication.

Aim instead for an interview that is conversational in style. Your goal is to find out if a candidate is suited to join your team, but also to get them to want to join. Conversations that feel more informal than a six-hour marathon whiteboarding session often do much better job of that. And like puzzlers and brainteasers, whiteboard coding of algorithms doesn't correlate strongly with job performance either - if you do feel the

need to get a demonstration of skills, having a candidate pair with a current team member on some actual work is a much better way of doing that.

As an interviewer assessing a candidate's skill, your goal should be to probe and question until you determine where the boundaries of their knowledge are. Think of it as finding the edges of a potato, or some other mostly smooth but somewhat irregular object. The edges of this object will be uneven, with some places that are lower than others - gaps in knowledge - and some that are higher - areas of deeper knowledge or expertise. You are trying to figure out where those edges are, what areas do they know more and less about, to get as complete and accurate picture of the candidate as possible.

For this reason, it can often be beneficial to have a wide range of people interview a candidate, to find as many of the edges as possible. Keep in mind that these include "soft" skills as well as "hard" or technical ones. Many companies have people from other teams or other areas interview candidates to see how people interact with other disciplines, such as having a developer candidate interview with operations and security, or possibly even people in less technical roles, as an engineer who can't treat less technical people with respect will not be a good addition to your team. When thinking about soft skills, it is good to specify in advance what things like "personality" and "culture fit" mean to your team. Too often, people use the term "culture fit" to describe "someone I'd like to go out for drinks with" and this tends to lead to increasingly homogeneous teams. Many jobs in past were based on who you knew, or the so-called airport test of "would I want to be stuck in an airport with this person". Having several different people interview a candidate can help alleviate this to a degree, as you will likely get more points of feedback to consider, especially if reviewers cannot see others' feedback before they have given their own (to help avoid what they say being colored by what others have said). Remember that you are not hiring people to be your friends, but rather people whose skills, technical and otherwise, will benefit your team and your company.

After the Interview

After the interview you will want to get feedback from all the interviews as to what they thought of the candidate. It's better to do this sooner rather than later so the experience will be fresh in peoples' minds. As mentioned previously, it's also best to keep other reviewers' feedback hidden from reviewers until they have submitted their own to avoid coloring peoples' opinions with those of others.

It helps to have specific criteria for reviewing determined in advance. If you have a standard set of technical questions you ask, you might come up with a scale based on how correct and complete their answer was. Otherwise, having a system where people can enter feedback in a consistent format will be helpful. The more consistent the

feedback gathering process is, the more likely it is that people will be able to agree on the outcomes.

Once all the feedback has been gathered, it is helpful to get all the interviewers together to have them discuss their opinions - again, sooner is better than later with this. You want to find out if people are inclined or disinclined to hire the candidate, how strongly those opinions are held, and why. When a decision has been reached, and an offer made an accepted, you can move on to considerations of onboarding and retention.

Onboarding

They say you never get a second chance to make a first impression. Cliché, to be sure, but a new employee's first impressions of your company and team can be hard to shake. Especially for junior people who have less industry experience, the first few days can make or break the beginning of a relationship with a new team member.

As much of the logistical work as possible should be done before the new person arrives, both to minimize stress on them and so that they can start being productive sooner. This can include:

- Adding them to whatever HR, benefits and payroll systems, so there's less time they have to wait before they're getting paid and receiving health insurance. (No, jobs aren't only about the money, but it's hard to focus on doing your best work when you're wondering if you're going to be able to eat and pay rent. Big tech areas like New York and San Francisco are incredibly expensive, and people who are just entering tech or who have been underpaid in previous tech positions might not have been able to build up a big buffer in savings. It's not all about the money - until you have to worry about the money.)
- Purchasing and configuring a computer for them. If you don't have any automated provisioning for personal computers, at the bare minimum make sure the computer is purchased and ready to be set up on the employee's first day. Without even a computer, there is very little a person can do to even start trying to get going - unless all your team's documentation happens to be printed out (and up to date)! If they need other hardware, such as a company phone or portable wireless hotspot device for people who will be on-call, get that purchased as soon as possible as well.
- Setting up their email and calendar accounts, and adding them to any mailing lists, shared calendars, and other groups they need to be a part of. If you don't have a list of these shared accounts, start one. Nobody wants to realize a few months down the line that they forgot to add the new team member to the team's calendar.

- Creating accounts on any other systems they will need access to. This will depend on their job function and what systems your team uses. Common examples include instant messaging, ticketing, time management, and documentation systems or sites. Engineers will need access to any source control and configuration management systems, as well as server login credentials. This is another place where creating a list of accounts to create (or even better, automating this process) can be very beneficial to smooth the process for future hires, especially if you plan to do a lot of hiring in the near future.

Setting expectations is an important part of the onboarding process as well. You should have communicated your general expectations for the position clearly during the interview process, but it can be good to reiterate and clarify those, especially if a fair amount of time has passed since the interview, as can be the case with new college hires who often interview several months before graduation.

Giving new employees some work to get started on should be done as soon as possible - maybe after a couple days of getting set up with HR and getting computers and accounts all squared away. New employees want to show what they can do and learn, and current team members are likely interested in how well the new person can start contributing, especially if the team is overworked and bringing on new people to lighten everyone's load. Unless you happen to have a very clear idea of an employees skill level, it is beneficial to have potential work at a few different levels so they can find out where they're comfortable.

Most teams have some sort of backlog in whatever ticketing or work tracking system they use. A week or so before a new employee starts, existing team members can pick a few of these tickets that seem appropriate and assign them to (or just earmark them for) the new person. Be sure to set expectations around these as well. If your team does ticketing a certain way, such as having a policy that no tickets are to be left unclaimed for more than a day, be sure to communicate these policies. It's usually helpful to have a current team member who is very good at using the system take some time to walk the new person through everything.

Make sure the new person knows what work is expected of them in their first couple weeks, and even more importantly, who they can turn to if they have questions or get stuck on anything. Don't leave a new employee sitting alone, feeling blocked and not knowing where to turn! Anyone who feels stuck and abandoned like this is unlikely to stick around of be happy for long. Make sure there's at least one person on the team who can be a dedicated resource during these first couple weeks - yes, it will be time out of their normal workday, but that's vastly preferable to going through the entire interview and hiring process only to lose someone because they didn't feel like they could be part of the team.

Managers should set up recurring 1:1 meetings with their new reports as soon as possible. Depending on how frequently you meet with your existing team members, you

may want to increase the frequency for new ones until they've settled in a bit. Check in regularly to make sure that they don't have any outstanding questions and aren't being blocked on anything, that they aren't having problems with any of the systems or other team members. Make sure there isn't any confusion as to how to do something or what they should be working on.

When considering diversity, think about the social groups and activities that exist that employees can participate in. Are employees allowed or encouraged to start groups, such as a women-in-engineering group or an LGBT group, as a way of connecting with their coworkers and creating safe and supportive spaces? Do people have access to leaders or mentors who are similar to them? If you have several more junior women in engineering, but no senior women engineers or women in engineering leadership, for example, those junior employees may wonder if there is room for them to grow at your company.

In terms of making sure an environment is as inclusive as possible, also think about what office activities, especially social or "extra-curricular" ones, are opt-in versus opt-out. An opt-in activity is one that employees are not a part of unless they choose to join, whereas an opt-out activity is one that all employees are expected to participate in by default unless they explicitly opt out of doing so.

While it may seem like opt-in creates a barrier to entry that might discourage people from joining, opt-out has the issue of requiring people to draw attention to the fact that they don't want to do something, or provide specific reasons for doing so. For example, many companies view drinking alcohol as the default after-work activity, and startups especially are prone to peer pressure when it comes to drinking, where not going to a bar after work labels someone as different or "not a team player". A new employee is likely to feel uncomfortable having to explain reasons for opting out of this to new coworkers they don't know very well, which is then likely to make them feel excluded. In this example, an office kitchenette stocked with a variety of alcoholic and non-alcoholic drinks with no mandatory times when "everyone" is expected to partake, is a much more inclusive setup.

In general, an office environment that is as inclusive and safe as possible will be one where people feel like they are becoming part of the team more quickly. It is quite a bit of work to get a new team member up to speed, but the more efficient the onboarding process, the more quickly new team members will be able to contribute.

Retention

In the competitive environment of today's tech industry, keeping employees around is of increasing importance to employers. Employee retention affects not only team productivity, but morale as well, as frequently losing coworkers can cause additional stress to the remaining employees as well as hinting at larger problems with the team

or the company. If many employees are leaving for reasons like getting greatly increased salaries or because they're worried about the direction the company is going in, that often doesn't bode well for those who stay. While some reasons employees leave, such as family circumstances causing them to move away from a position that doesn't support remote work, aren't controllable, there are many factors of retention that are. This section will examine those factors.

Compensation

Money isn't everything, and more and more frequently people are choosing healthy work environments and companies they feel connected to over simply a larger paycheck. Even so, people want to feel like they're being paid competitively. A recent study found that employees who stay at companies longer than 2 years end up making significantly less money - 50% less over only 10 years - over the course of their careers. Conventional wisdom, especially among individual contributors, says that the best way to get a substantial raise is to change jobs and negotiate a higher starting salary from a new company. On average, employees staying at a company can expect around a 3% raise, which comes out to be effectively more like 1% when taking into account the 2% rate of inflation. Changing jobs, however, they can expect an increase of between 10 and 30 percent - even at a company you love, over time that kind of disparity is hard to look away from.

To help combat this, start by making sure that you are paying competitively from the beginning. Employers are often tempted to offer the lowest starting salary they can get away with to help their bottom line. This often disproportionately affects members of minority groups who tend to make notably less on the dollar than white men. Offering salaries and benefits that are in line with industry averages can help attract these people who many have been significantly underpaid at previous positions. Transparency about the salary negotiation process, pay bands (if your company uses them) and other issues related to compensation can help with retention. People not only want to feel like they are being compensated competitively but also that they are being treated fairly.

One thing that can help, especially in retaining employees who tend to be penalized for negotiating salary rather than rewarded for it, is transparency around the process for raises. Having a clearly defined and publicly (within the company) documented process is key. A process that relies on people asking or only happens when managers think of it is far more prone to unconscious biases than one that happens on a regular schedule with clearly defined parameters. Having pay bands rather than simply relying on managers' judgments can reduce these kinds of biases as well. Make sure that everyone, managers and individual contributors alike, is aware of the process around raises (and yearly bonuses, if your company has them) as well as who to talk to if they have a concern with the process.

Non-Monetary Benefits

Being paid competitively and fairly is important, but once employees are compensated well enough that they are able to enjoy a good standard of living and save money for the future without having to worry about whether or not their rent check will bounce (especially in rental markets like New York and San Francisco), non-monetary compensation can often be more valuable to them than additional salary increases. For smaller or less mature companies that might not be able to offer the same salaries that more heavily-funded or more profitable ones can, these kinds of perks can be a good way to attract and retain talent.

It is important to note that when we talk about perks, we are not talking about things like beer fridges and ping-pong tables in the office. Those kinds of things tend to contribute towards an atmosphere more resembling a frat house than a professional office space, and can be a deterrent towards people who feel uncomfortable in such an environment, whether that be women, non-drinkers, or people who simply don't want to play table sports at work. Meals, especially ones that are healthy and include options for a variety of dietary restrictions, can be a plus (pizza tends not to be vegan, lactose-free, gluten-free, or especially healthy) but be wary of offering breakfasts and especially dinners as perks, as those meals that fall near the ends of the day tend to indicate a culture where people are expected to arrive early and work late on a regular basis.

Benefits you might consider include:

- Remote opportunities: Whether this is for attracting a wider range of candidates than you might be requiring them to be local or retaining employees whose life has taken them away from one of your company's offices (moving away to have children, to be near parents, or to avoid paying the ever-increasing New York and San Francisco rents are popular reasons), offering the ability to work remotely can be a big benefit.
- Educational opportunities: These might take the form of instructors brought onsite, sending employees to conferences or training seminars so they can either learn new skills or improve existing ones, or going as far as to provide tuition or textbook reimbursements for people who are seeking continuing education in fields relevant to their profession. Personal and professional development are important to people, so providing opportunities (as well as time) to pursue these opportunities can be a great benefit.
- Flexible work hours: Unless there is a legitimate reason for requiring people to work specific hours, a little flexibility can go a long way. This, like remote work, shows trust in the employees and teams as well as respecting the lives and responsibilities that people have outside of work. Flexible hours can allow people to pursue hobbies, avoid rush-hour commutes, or take care of family and house-

hold responsibilities while getting their work done at times that are convenient for both them and the rest of their team.

- Work-life balance: Continually working 50-80 hour weeks actually has a negative impact on overall productivity, not a positive one. Make sure that employees are allowed and encouraged to come into and leave the office at reasonable times, and to not spend their time at home working or checking email constantly as well. One of the best ways to do this is to provide good examples from management - employees who see their supervisor emailing them at 10 pm or 5 am may feel pressure to respond that late or early, whether that pressure is intentional or not. If employees are on-call, providing extra time off for each on-call shift can be a great benefit also.
- Vacation: Make sure that your company provides vacation time and that people are actually using it. Outside of whatever holidays are required by the laws in your country, try to avoid mandating when vacation days should be used. For example, providing 10 days of vacation but requiring that 8 of them be taken during the weeks around Christmas and New Years leaves only 2 vacation days for the other 50 weeks of the year - far from ideal (especially considering not everyone celebrates Christmas). For companies considering unlimited vacation policies, consider this article by CEO of the TravisCI continuous integration company Mathias Meyer on how those can be problematic.

Overall, employees should not feel like their company is trying to shortchange them when it comes to monetary or non-monetary benefits. People who feel that they are treated fairly, taken care of, and have a process for addressing any questions or complaints they might have are generally happy employees.

Growth Opportunities

Before money and work-life balance, the number one reason that people leave jobs is due to a lack of opportunities for advancement. Nobody goes into a job expecting or hoping that it will be a dead-end one. People want opportunities to grow their skills and to demonstrate that growth, whether that means more independence, more choice in the projects that they take on, being trusted with bigger projects, and leadership opportunities.

Keep in mind that leadership doesn't just mean management. Some companies have the management track as the only one with clearly defined job levels and the only way that employees can advance in their career, but many technical ICs have no desire to go into management. Leadership for these people can mean leading projects and expanding the impact that their contributions can have in an organization. Make sure that you have a way for non-managers to advance as well. Ideally this would mean creating a technical track with clearly defined levels of individual contributor growth, as well as a management track.

Having a clearly defined process for growth and promotion is critical for management and individual contributor tracks alike. Make sure that there are definitions of job levels available to employees, with enough detail that people can see clearly what they need to do to move from one level to the next. The process for promotion should be clearly and publicly defined as well. A "process" that consists only of people getting metaphorically tapped on the shoulder by management with no more visibility than that can be incredibly frustrating for employees who don't get chosen, as well as being rife with opportunities for unconscious bias to appear. All employees deserve the opportunity to grow, not just those who happen to be friends with their boss or the CTO.

Going along with this, make sure that employees have chances to explore different areas of interest at the company. If a software developer, after several years, wants to explore an interest in operations or security, for example, they will likely go to another company to explore that if their current one doesn't afford them any way to do so. While mangers generally shouldn't try to poach employees from other teams, they should be open to the possibilities that peoples' interests and career goals will change and try to work with that whenever possible. Some companies do what they call "senior rotations" where employees get to take a few weeks to work on teams other than their own once they've been with the company long enough. Whether in their current job area or another one, growth throughout the entire course of their career is very important to people, so providing opportunities for that is necessary if you want them to stick around.

Workload

In general, people are looking for workloads that are challenging but doable. Continuing off of what we were saying about growth opportunities, challenging work that allows people to test themselves and grow their skills is important to their senses of satisfaction about their jobs.

Too much challenge can lead to problems. People might feel that their company or their manager expects too much of them and isn't willing to give them the time, support, or resources they need to get it done, or it might seem like their manager is out of touch with reality and doesn't understand how much work they can actually accomplish. It might indicate a team that has taken on too much or has an uneven distribution of work, where some employees are relatively relaxed while others are overworked. Whatever, the reasons, long-term overworking of people (as opposed to a one-off period of crunch time) can have very negative impacts.

Overworked employees might just end up leaving the company for a different job that doesn't require so much of them. Perhaps even less desirable is the possibility that they will stay, but be suffering from burnout. It is important for managers to regularly check in with the team as a whole and with individual employees to make sure that

they don't have an unrealistic amount of work they are trying to take on. Make sure that employees are taking vacations as needed as well - if a team or person has just finished a period of extra work, such as towards the end of a project, encourage them to take some time off, and make sure that everyone is taking at least one good-sized vacation (or even staycation) per year to avoid burnout.

Burnout

Burnout is a term that refers to long-term exhaustion and lack of interest in work and often in activities outside of work as well. Symptoms of burnout are very similar to those of clinical depression, and has even been called a form of depression in some recent studies. People suffering from burnout may start isolating themselves from others, pay less attention to their own personal needs, have problems sleeping, and have feelings of indifference, helplessness, and hopelessness. It often arises from prolonged periods of stress and overwork, which are far too common in the tech industry, especially in Silicon Valley startups that idealize the "heroic" hacker or "rockstar" developer. Mental health is as important as physical health, if not more so, and taking care to avoid burnout should be a priority for every team and every company.

Many tech companies have some kind of on-call position - people whose responsibilities include carrying a phone or pager and responding to incidents outside of standard working hours. It is very important to make sure that on-call is not placing undue stress on people. If at all possible, make sure that the on-call responsibilities are shared between at least two people - this is really the bare minimum. Having only one person on-call 24/7/365 is pretty much asking for that person to get burnt out. No one person should be forced to give up *all* their nights and weekends in perpetuity. Ideally, on-call rotations would be shared between several people, possibly with several rotations as your company grows, to give every individual adequate time to catch up on sleep and relax in between their shifts.

If people are part of an on-call rotation, make sure they are compensated for it. Some companies increase the salary of employees who have ongoing on-call responsibilities, some pay extra for each hour that someone carries a pager or for each off-hours incident they have to respond to, some provide extra vacation hours or days for each on-call shift. If on-call responsibilities are part of a job from the beginning, make sure that the extent of these responsibilities is clarified up front, so employees know what they are signing up for and can negotiate their compensation accordingly. If these responsibilities are added after someone has started, make sure there is an opportunity for them to discuss the details and compensation with their manager.

On the other side of things, make sure that employees are not underworked and unchallenged. Challenges, which include new projects, learning new technologies or tools, taking on bigger projects or having more responsibility, are key to making sure

employees are growing and staying engaged with their work. Contrary to what some managers fear, most people do not want to have lots of downtime where they will play games or waste time on the internet out of boredom - they want to be engaged and learning. Employees who seem to be less engaged might not have enough to do, or the work that they do have might not interest them. If they have been working on the exact same kind of work for a long time, either because they haven't been given anything new to challenge them or because they've been assigned to a project that is taking a very long time to complete, they might well be getting bored. Again, regular check-ins with your team are important to keep in touch with them and where they're at.

Alignment with Working Styles

In the Collaboration chapter we addressed several different working or collaborative styles, including starters versus finishers and purists versus pragmatists. If the work that people are regularly being given doesn't align well with the type of work they most prefer to be doing, that can make their workload feel higher than it is, and disengagement with the work itself is likely to decrease a person's happiness and productivity.

Culture and Atmosphere

Culture and atmosphere are both things that sound broad, vague, and hard to usefully define at first. Yet many reasons that people might start feeling unhappy are hard to define at first. In this subsection we'll examine a few of the less tangible factors that can cause people to stay at or leave positions.

"Cultural Fit"

Cultural fit is a term that is so vague it can be problematic, often used by people making hiring decisions who seek, either consciously or not, to keep a certain amount of homogeneity within their team, perhaps hiring people who went to the same school, like the same sports, or participated in the same fraternity as them. This is a misuse of the idea of cultural fit, as it takes very superficial ideas of culture and uses them to create an atmosphere of exclusivity. Culture is often defined as the ideas, customs, and social behavior of a people or society, and when we look deeper into those areas we can see how culture in this way can make or break someone's desire to stay with a company.

Ideas can mean a lot of different things when talking about a company or a team. Most broadly, an idea for the company is its value proposition, what it is selling, how it has chosen to make money. Some companies value is essentially in advertising, or selling user data to advertisers. As mentioned previously, individual's motivations and

concerns extend beyond the paycheck to caring about what the organization does and how it achieves it. If someone feels that their company's values conflict too much with their own personal ones, or they simply don't have any interest in what the company is doing, there is a force driving the individual away from the company regardless of the company's success.

This can be something to try and address during the candidate sourcing and interviewing phases, but these ideas are often ones that can change over time, perhaps as business practices also change. Someone might feel that there was value alignment when they started, only to see that change, or realized that what was a good fit on paper didn't match up with how those values were played out in reality.

Ideas might also be interpreted to mean what is considered valuable in a given organization or team. As a typical example, in many organizations, especially before the idea of devops rose to popularity, operations or IT work was often very undervalued. IT was seen as nothing but a cost center. Investment in IT was seen as something to be minimized at all costs, as IT was thought to provide very little to no value to the company.

Similarly, one might feel devalued by their team or manager. If a few team members are very buddy-buddy with their supervisor, and if that friendliness leads the supervisor to listen to them more, the other team members can easily feel that their contributions aren't valuable to the team.

Anyone whose ideas are different from the majority might quickly feel this way as well. We've touched previously on considerations for helping to increase the diversity of a team or organization, but it's very important to consider the impact that this can have on people who aren't "the norm". A lone woman on a team of men, or a person of color in an otherwise all-white team, especially on a team that primarily handles disagreements by way of who yells the loudest, might very well feel that their contributions are neither heard nor valued.

Customs are the traditional or widely accepted ways of behaving, speaking, or doing things. A lot of things about a workplace can be viewed as customs in this light, including:

- How work is assigned, and who is responsible for assigning it
- How members of a team or the same level within the company communicate with each other
- How managers communicate news to their reports
- When people arrive at and leave the office
- Technical processes for doing work
- How promotions, raises, and bonuses are awarded

One of the problems with customs is that they can be hard to recognize as just one way of doing things and not the only way, because once people are used to them they have a tendency to fade into the background. Often it takes a new pair of eyes and a fresh perspective to point out that there might be a better way of doing something.

It is important to recognize and value these insights - "we've always done it this way" is not a sufficient reason for continuing to do something. Refusal to change or to even consider new ideas is how teams and companies stagnate, often leading to them getting passed over by their competitors. It's human nature to fear change, to reject the unfamiliar, but we should recognize this tendency in ourselves and actively work against it to make sure that we're hearing, considering, and choosing the best ideas, not just the ones we're most comfortable with.

Company customs regarding promotions, raises, and bonuses merit special consideration when trying to improve diversity. It bears reiterating that even though we might not want or notice them, unconscious biases can easily creep in here. If these sorts of things are awarded solely at a manager's discretion without people applying, being encouraged to apply, or something like having all employees of a certain role or rank being considered, unconscious biases can (and often do) come into play.

Social behaviors, the last major part of culture, cover a wide range of things in how people interact. Pay attention to the way people communicate - do more "senior" employees talk down to or talk over those who are lower in the ranks than them, or are all ideas treated with respect regardless of who they came from? Do people tend to interrupt each other in meetings, or do they wait until others have finished speaking? Is this true among only peers, or with management as well? When people have disagreements, how are they resolved - by calm discussions, by consensus, or by people simply yelling at and over each other until all but one party has given up out of frustration? How do decisions get made?

Social behaviors also include ones that we might more often think of when we hear the word "social". How do teams get to know each other or bond? There are many benefits to better knowing the people you work with, including greater empathy and more effective communication, but there are more or less effective ways to improve camaraderie. More corporate environments might opt for some awkward ice breakers and trust falls, while startups might tend towards a trip to the nearest bar. The most effective might be something in between, soliciting input from all team members and trying to find something that is agreeable to everyone. Be aware, however, that some people might not feel comfortable giving their opinions publicly. Someone who has struggled with alcohol abuse, for example, would likely be hesitant to explain in from of their coworkers why they don't want to go on a work-sponsored bar crawl. Make sure to give people ample opportunity to give their opinions in a private, safe manner.

Similarly, the ways that people socialize in the office can be very telling as to the overall social behaviors. People will generally notice who frequently goes to lunch or cof-

fee with their team's manager, especially if this is not a privilege afforded equally to everyone on the team. (While it is true, especially at startups that are often founded by people who previously knew each other, that people might become friends at work, it is important to avoid unconscious biases and blatant favoritism in the office.)

Many offices, especially smaller ones, develop activities that often take place after hours or during breaks. Some might have cold-brew coffee or beer on tap, today's equivalent of the water cooler. Some might have ping-pong or foosball tables that employees can use at the end of the day or to blow off some steam. Increasingly common in environments dominated by younger male employees are things such as remote controlled helicopters or Nerf guns. These things are not universally bad, but they can be exclusionary of those who don't like such things. Someone who doesn't care for kids' toys in a work environment is likely to be pretty upset by getting hit in the head with a poorly aimed Nerf dart in the middle of trying to get work done.

Keep in mind that people might not feel comfortable speaking up against things that happen in the name of "fun" - nobody wants to come across as being anti-fun. It's hard to be the only one to speak up against something that's been a longstanding part of the culture, and if someone is a minority on the team they might feel even less comfortable coming forward. Making sure that people are comfortable speaking up about their opinions, as well as paying attention to what kinds of activities are common around the office, can go a long way towards promoting a culture that everyone feels like they are a part of.

Problems with Management

Many articles have been written around the idea that people often don't leave jobs, they leave their bosses. There can be myriad reasons why people don't like working for a manager, but some of the most common include:

- Distribution of work: Feeling like work is not distributed equally among team members, leaving some to be overworked while some are underworked, or that certain people always get the best projects, is a common source of frustration.
- Lack of (or too much) attention: A manager who gives each of their reports only a cursory 1:1 every quarter is likely to be much less in touch with reports' goals, successes, problems, and frustration than one who makes time monthly or even weekly. On the other side of things, people who feel like they are micromanaged are likely to be unhappy with that situation as well.
- Poor communication: This works both ways. A boss who doesn't listen (or doesn't seem to) is a common frustration, but the manager's communication skills are equally important. Someone who doesn't pass on important information to their reports in a timely manner, who forgets critical details, or who can't get a point across in an effective or timely manner isn't an effective communicator.

- Fairness: A very common frustration with employees is the feeling that their boss is treating them unfairly. Maybe some members of the team get all the good projects, get bigger raises and bonuses, or are first in line for promotion. Transparency around team and company processes can help alleviate this, as can unconscious bias training for managers.
- Lack of guidance: In the same way that employees value career growth opportunities as something that will keep them at a company, they also look for a manager who will actively help facilitate this growth. A manager who seems uninterested, unable, or unwilling to help their reports progress in their careers is a manager that those reports are often eager to get away from.

It can be difficult, as a manager, to consider that you might be the reason some of your employees are unhappy, but it is a very important thing to think about. We mentioned earlier that often times, management is the only way to progress up the career ladder at some companies, and at startups people often get moved into management due to organizational need, not necessarily out of desire or skill. We recommend that all managers, new and experienced, get regular management training. Management is a very different career path from engineering, requiring a very different set of skills, and it is in everyone's best interests if managers are regularly making sure they're doing the best job they can be.

Having discussed several considerations for sourcing, interviewing, and retention as they relate to hiring, we'll now take a look at how these ideas can be applied in the wild.

Case Studies

For this Chapter's case studies we spoke with two people involved in hiring in different capacities: a director of devops at a private e-commerce and crowdsourcing company, and a technology lead at a private digital marketing and design firm.

The Director, who started his career as a developer, joined a large e-commerce company that develops software for online stores as well as retail POS systems, and built their operations team from the ground up. He later built from the ground-up the operations team at a digital media and publishing company, as well. At the company, he oversees production operations and corporate IT from the technical side as well as helping to grow both those departments.

Looking for an opportunity for a director-level position as well as to find an environment with plenty of room for learning and growth, the director was drawn to the company due to its diversity and culture. The company's senior management were over half women, including the CTO and CEO, which is much more diverse than most Silicon Valley companies. It currently has around 125 employees, with about 30 of those in engineering. In terms of their operations, they run on about 50 servers on

a combination of physical and cloud infrastructure, with autoscaling (increasing or decreasing the number of servers running in a cloud infrastructure based on some metric such as server CPU load) of up to hundreds of job workers as load necessitates. To them, doing devops effectively means having operations engineers working closely with developers to build beautiful automated systems that help the business achieve its goals. This involves teaching developers about operations work and automation in particular, then working closely with them across the life cycles of the systems they build.

The director leads the devops team at the company, which currently consists of four engineers besides himself, ranging in experience from a junior engineer in his first devops role to a former director making his way back into being an individual contributor. All of these were hires that the director made. Their process involves approval from the VP of Engineering for the headcount, followed by sharing the job posting they create on Twitter and the Github and StackOverflow job boards. Like many companies, they haven't had a great deal of success with recruiters and have found these industry-specific job boards and individual contact to be much more effective at finding the kinds of candidates they need.

The interview process starts with two phone screens - one by a current team member, one by the director himself - and passing the phone screens leads to a full interview which involves two engineers, an engineering director, a manager on the business side of things, and finally the VP of engineering. Some interviews are more technically focused, while others focus on getting to know the person. For the director's team, this means finding out what they liked and disliked about their previous positions, what they're looking for, and what excites them. He notes that he tries to find out if candidates have any strong opinions (on text editors, or SQL versus nosql, or their favorite Linux distribution) and whether or not those opinions are so rigidly held as to be completely inflexible. They have found that people who refuse to change their minds about things tend not to be people who fit in well to their team environment. Interviewing with people outside of their direct team, such as with the business manager, is useful for finding out if candidates work well with other teams, especially non-engineering teams.

He realized that their hiring process needed some improvement when it was pointed out that their job posting was glorifying *hero culture*, where "heroic" behaviors such as working long hours, and single-handed troubleshooting and "firefighting" to keep services up and running are looked upon as desirable. The problems with this kind of culture are that it is unhealthy, with long hours and working weekends leading to burnout as well as physical and often mental health issues, and that it tends to attract people who are more interested in being "heroes" for their own recognition or gain than they are in working effectively as part of a team. Though the glorification of hero culture in the job posting was inadvertent, it was affecting the types of candidates the director was getting for the positions he was trying to fill.

Examples of these kinds of job descriptions include:

- Asking for candidates to "give 110%" or "go above and beyond" - phrases like this tend to be indicative of teams with little to no work-life balance. In addition to being unhealthy, these kinds of requirements are biased towards single men and away from women or any other candidates that have family responsibilities (or simply want to get home at a reasonable hour most nights)
- Describing a team that "works hard and plays harder". You are hiring employees, not friends, and expecting your employees to spend their non-work hours at work social events, especially when those events tend to be heavily focused on alcohol, is off-putting to many candidates, especially those who aren't young men.
- "Awesomeness" or some other vague quality. A term like this is vague enough to be close to meaningless, and attracts the sort of people who think they exude "awesomeness", which often goes hand in hand with egotism and unwillingness to learn or listen, excluding groups like women and people of color who tend to suffer from impostor syndrome. This includes to asking for candidates who self-identify as "rockstars", "ninjas", "wizards", and the like as ways of describing their skills..
- Homework or otherwise requiring applicants to prove their knowledge is another thing that shows a lack of respect for employees' time and something else that tends to be biased towards people with fewer responsibilities outside of work

These and similar requirements that emphasize hero culture tend to lead to workplaces where employees are often losing sleep, which itself leads to degradation of creativity, productivity, and empathy, as well as eventual job dissatisfaction, loss of self-confidence, and burnout.

Using recruiters, especially third-party ones, can also lead to issues, as recruiters who don't share your values won't necessarily represent your company well, and may be incredibly off-putting or even offensive to candidates. The following examples are not from the director or the company, but are an example of the kinds of things to especially watch out for in either job descriptions or messages from recruiters (internal or external).

Lack of Effort or Attention to Detail

If an email begins with "Dear %%FIRSTNAME%%, we are looking for someone to fill a %%JOBTITLE%% position," that's a clear sign of someone who copied and pasted from a template and couldn't even be bothered to glance over their email for these very obvious mistakes before hitting send. Even copying a potential candidate's skills from their LinkedIn profile isn't always a sure bet, especially without proofreading - sending an email to someone asking to discuss their "experience with back-end devel-

opment and drinking" is a clear indication that someone either didn't double-check their work or is hiring for a very unhealthy company culture. Copying and pasting the same form letter to every member of a team is a lack of effort that will not go unnoticed by those teammates - word about companies who use these tactics gets around.

In the same vein, asking a potential candidate to recommend some other people if they aren't interested in the position comes across as being incredibly lazy. These days, nearly every company is actively hiring - if they knew any qualified candidates they would be talking to them themselves, not doing free recruiting work for someone else's company. People are often willing to give recommendations, but to their friends or colleagues they know and trust, not to complete strangers who bother them with repeated solicited emails.

Exclusionary or Unprofessional Language

Look out for language in job postings or recruiting emails that are likely to alienate candidates. Exceptionally masculine language - "crushing code", "rockstars", and "are you a [TECH] weapon" are likely to be off-putting to people who don't fall into a stereotypically masculine mold. Even worse than this is overtly sexist or homophobic language. A job description that lists "making it rain on them hoes" and "partying with rockstars" as job perks and "be totally gay for code" as a requirement is wildly inappropriate in a professional context, pointing to a company that is likely to be hostile to women or LGBT people. Saying that you're looking for a "nice dude" who is "under 30" does the same thing, and specifying gender and age is often illegal depending on the country you're hiring in.

Misplaced Focus on Technology

Many engineers are excited by getting to use new technologies, so it makes sense that people might use those technologies to get people interested in their jobs. Too much focus on tech, especially without doing due diligence, can backfire, however. If you're asking for experience in a particular technology, do your research. Asking for 10 years of experience in a product that has only existed for 2 years makes it look like you don't know what you're talking about at all - probably not the impression you're trying to give.

Also be aware that candidates are more and more often interested in what a company is doing, not just the technologies. If your technology is interesting with good reason, feel free to mention it, but sending an email to a potential candidate that talks *only* about the tech without even mentioning what that technology is building and what the company does is a mistake. An engineering team that is always using the latest "hot" new bleeding-edge tools (and again, descriptions like "hot" and "sexy" are very likely to alienate large groups of potential candidates) isn't necessarily going to sell your team very well.

Linting Your Job Descriptions

Linting is a term in computer programming that describes programatically searching for suspicious, dangerous, or non-portable code constructs that are likely to cause issues. Engineers can "lint" their code to do this kind of analysis on it, checking for common errors or style problems, before they commit it to the main code repository.

A similar tool exists for analyzing job descriptions or postings, or recruiter emails, to check for some of the common issues that we have described here. You can use this tool yourself at joblint.org to catch some issues you might not be aware of, and to remind you of what you should check for in the future.

The company later revised their job postings and got rid of these "heroic" descriptions and instead highlights the teams cultural values, including work-life balance. For example, they note that they give every on-call engineer an extra day off after each 1-week on-call rotation, to help counteract the stress and sleep deprivation that is so often part of being on call. They also work to optimize away the unpleasant parts of working in operational areas by doing things like having a rotation for who is responsible for responding to walkup questions and other interruptions from other engineers, and using a ticketing system to track these requests.

Other examples of better job postings include:

- Mentioning general skills rather than specific technologies. Instead of saying you want someone with 2 years of Puppet experience, try advertising for concepts such as automation of repetitive tasks and configuration management. Also assess whether or not a specific number of years experience is actually required - in many cases it isn't and those kinds of hard requirements will leave out candidates who are actually qualified.
- Call out important cultural values. By culture, we're not talking about having a team who drinks beer and plays foosball together, but rather cultural values such as empathy, effective communication, getting rid of silos, and work-life balance. (Of course, don't mention values that your team doesn't actually have - lying about your culture to get hires will quickly be found out and word of that will spread.)
- Make sure your job descriptions are gender-neutral and free of aggressive terminology. You're looking for somebody who can write code, not "crush" code.
- Specifically call out your company's commitment to diversity if that's something you're working on. Mention perks that will be appealing to a wide range of applicants - instead of a beer fridge and ping-pong table, talk about a culture that

encourages people to leave work on-time, parental leave, and training opportunities.

More Diverse Hiring Resources

Model View Culture, an independent publication writing on issues of culture and diversity in tech, has a list of 25 tips for diverse hiring that is an excellent collection of resources for anyone looking to improve the diversity of their teams.

The new job posting has been working very well for the director and his team - they've had five successful hires, with only one person who didn't end up being a good fit. They've managed to take their infrastructure from one of entirely unmanaged snowflake servers to one that is completely automated, because of the people that they've hired. Their team worked to hire engineers who are fanatical about automation and testing, and then let them do their best work. This ended up with a complete overhaul of their automation and testing infrastructure, providing the entire engineering organization with simple, well-documented tools that everyone knew how to use and contribute to.

Moving now to the technology lead at the digital design agency, we meet someone who has been working in technology for nearly 15 years and across industries including higher education, finance, media, and advertising. To her, devops means leveraging the coding skills of developers with the operational knowledge of system administrators to provision and operate reliable computing systems at scale, something that is important regardless of the specific industry those systems are supporting.

As a team lead, her focus is on growing and improving her team, similar to the director previously described, but at over 500 people, her company is operating at a much larger scale than his. This necessitates different specific hiring practices, even though their general objectives are the same.

Because their organization is so large, they have staffed recruiters who work full-time in house to find qualified candidates. If a team lead or hiring manager isn't satisfied with the candidates they are getting from the recruiters, it is their responsibility to work with the specified recruiter to try and resolve those issues. The interview process consists of a phone interview, which usually lasts around 30 minutes and covers a candidate's work history, followed by a number of on-site interviews. The exact number depends on the team and the type of position a candidate is interviewing for. For example, an entry-level developer would only be requested to come to one on-site interview. The higher the position a candidate may be considered for, the more interviews a candidate may have.

Once the interview process has been completed and a hire has been made, every employee at this company is assigned a Career Developer. Career Developers act as mentors to the employees assigned to them, which is usually a number around four, though they have found over time that no Career Developer has been very effective with more than seven mentees, as they all have other work assignments and responsibilities as well. Their primary goal as a Career Developer is to help employees be successful at the company, in whatever form that takes. Each of them meets with their mentees individually at least once a month.

A type of feedback called *360 feedback* is used for company performance reviews. In this method, feedback is gathered from a variety of people in an employee's immediate work circle, often including their direct peers, their direct reports, and the person (or people) they report to. In this case study, 360 feedback is provided anonymously, and a Career Developer will receive a copy of all the feedback for their mentees. In this way they can help their mentees process that feedback, doing things like coming up with a performance improvement plan if there was significant negative feedback. This is a large part of how Career Developers help their mentees, in addition to things like career direction advice or assisting with perplexing technical issues.

In addition to career growth and mentorship, this company takes retention very seriously as a key part of their hiring strategy. They have realized that while monetary compensation is important, there are other ways of making employees feel valued aside from simply increasing their salaries.

In their weekly technology staff meetings, all team members are encouraged to talk about what they're working on and describe the contributions they've made to projects they are working on. Once a month, employees are asked to nominate one of their coworkers for what is called a *spot bonus*, which combines the interpersonal satisfaction that comes from peer respect and recognition with a monetary bonus.

Letting employees explore multiple interests and grow their careers in the directions they choose is an important retention strategy as well. The technology lead shared a story of a front-end developer on her team who was very valued on the team, but found himself wanting to do more than just JavaScript and CSS. He discussed this with his Career Developer, who brought it to the attention of the tech lead, and together the three of them came up with a way for the developer to start spending at least 25% of his time each week working on other creative technology projects. This slight modification to his responsibilities allowed him to grow his skill set, gave him the opportunity to work in an area of technology that excites him, let the company retain a talented (and much happier) developer.

These two case studies taken side by side show how even though companies have the same overall objectives (to hire and retain talented engineers), their sizes and specific situations led them to different hiring and retention techniques. In the next sections, we'll give examples of how to measure and troubleshoot your own hiring initiatives,

examining the common factors that you'll need to consider no matter where your company is at.

Measuring Success

There are many different metrics that can be used to assess the success of your hiring initiatives. Some people might be tempted to look simply at onboarding numbers: How many new people have you managed to hire? Or how many people have you made offers to? Or how many candidates have you found that you want to bring in for in-person interviews?

These numbers, while they might be easy to measure, don't really provide any kind of insight into the actual effectiveness of your hiring and retention strategies. As a first step, consider the recruitment funnel - similar to a customer acquisition funnel, an employee acquisition funnel can help identify where issues might be popping up in your process.

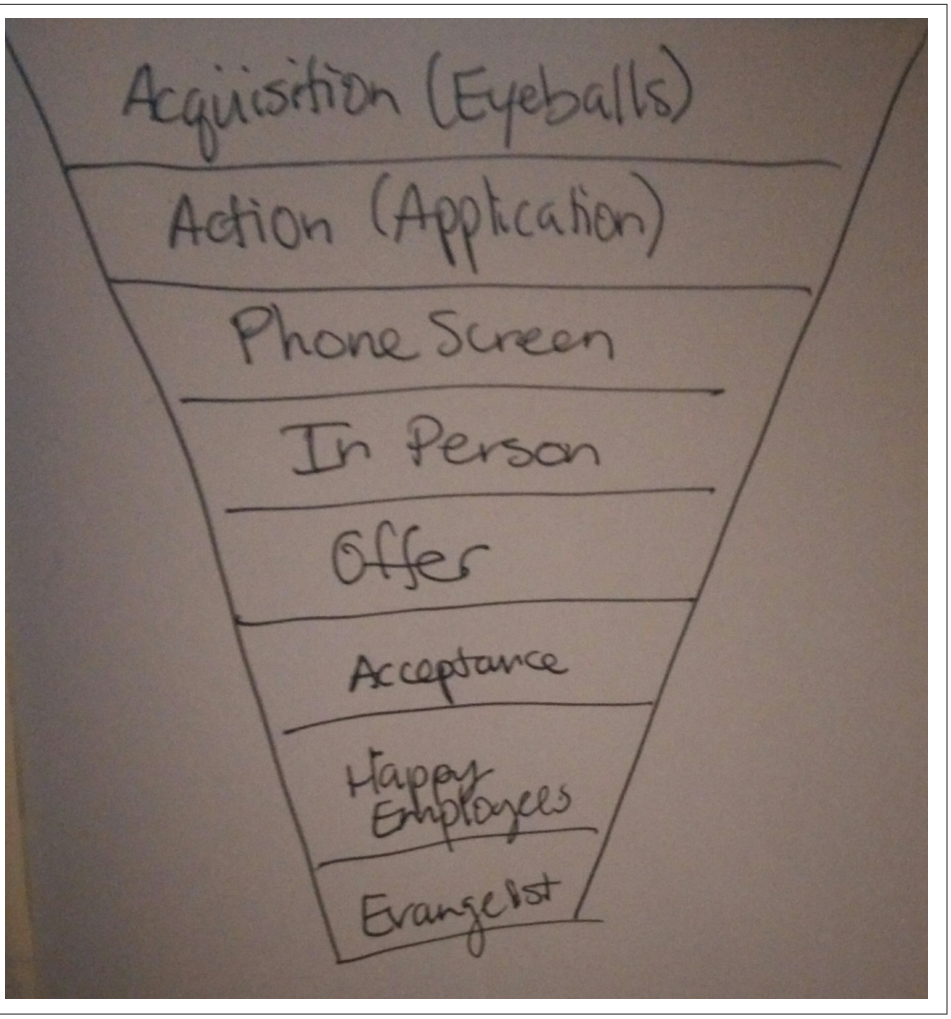

Figure 4-1. The Employee Acquisition Funnel

Like other acquisition funnels, it is expected that each phase be smaller than the previous one. If you make sure to measure your numbers at each of these phases over time, that can provide insight if you notice changes happening. Maybe a smaller number of applicants can be traced back to changes in the wording of a job description, or just a job posting that expired and needed to be renewed. Fewer people making it to an in-person interview could be due to a variety of factors, but it's hard to assess which ones unless you can know how the percentages are changing - it could be fewer people applying, or more people applying but proportionally fewer of them making it to each stage of the interview process. Without knowing which case you're

dealing with, it will be much more difficult to figure out what actions to take in response.

A decrease in the number of people who are applying for a position if the same number of people are viewing it could indicate that something in the job posting is driving away candidates - the other case study and the troubleshooting section of this chapter give examples of how the content and tone of your job descriptions can affect how they are viewed by potential candidates. Fewer candidates making it through phone or in-person interviews could indicate issues with either candidate screening or the interview processes themselves, and fewer offers being accepted relative to the number of offers being extended could very well mean that your compensation isn't competitive enough.

Other metrics that you should consider measuring when considering the effectiveness of your hiring include:

- **How many people that actually interview with 0% interest in the position?** Engineers are often encouraged to take interviews every so often (6 or 12 months is common) to make sure that they are keeping up with industry trends and making sure their interviewing skills are still sharp. Some people will interview with the goal of getting an offer that they can then leverage as a way of getting a raise from their current employer. While it can be very difficult to detect this kind of behavior, if you notice that very few of your offers are being accepted but believe they are competitive, you might consider if your screening processes aren't gauging interest well. Alternatively, there could be significant mismatches between the job as described and its reality.
- **How many years does the average employee stay?** While it is true that employees these days are less likely to remain with the same employer for as long as members of previous generations, where having more than two or maybe three employers over the course of one's career was incredibly uncommon, it can be very revealing to keep track of how long most of your employees stay. Significant changes in these numbers might be traceable back to events such as changes in leadership or company strategy, new management or budget cuts that affected training and compensation. Employee surveys can provide more detailed information, if people are willing and able to share that level of detail, but high-level retention numbers are important as well.
- **How do you measure the long term growth/education of your employees?** Many companies, especially smaller or younger startups, don't give much consideration to growing their employees, as they are much more focused on growing the company. With all the costs associated with onboarding new employees as opposed to retaining existing ones, it makes sense to pay attention to how you are going to grow and retain personnel. Measuring rates of promotions, training

budgets, and employee satisfaction with their professional growth opportunities can go a long way towards achieving this.

How do you know when you've made a good hire? Often, it takes time to be able to tell. People will have periods of adjustment as they get used to working together, especially in situations where you're actively trying to increase the diversity of your team. In operations teams, which tend to be even more male-dominated that software development, it may take some time for men to get used to working with a woman, especially if they've never worked with a female peer in a professional context before. When bringing in more junior employees, it is a given that it will take some time to get their technical skills up to speed as they learn their way around the environment.

When asked about his most successful hire, the director talked about a hire who had been a data center technician looking to make a move into a more involved operations role. The candidate expressed concern that his lack of operations experience and lack of college degree would be a problem, but he had an obvious enthusiasm for the subject and desire to learn and improve his skills that the director, like many people, valued more than simply the number of years experience on a resume. The candidate had also been discounting a great deal of experience he had working on side projects, including work as a package maintainer. He was offered a position that gave him a safe environment in which to experiment and learn, as well as colleagues who were willing to mentor and guide him, and has ended up making significant and valuable contributions to the team. This just goes to show how actual job requirements can end up being more flexible than you might think.

In general, things to look for when considering the success of your hiring initiatives include:

- **Alignment with your hiring goals**. The first section of this chapter introduced how to identify your hiring goals. If you were looking for a specific skill set for a particular project, was that project successful? If you were aiming to improve the diversity of your team, were you able to source, hire, and retain a more diverse set of candidates? The particulars of this will vary based on what your goals were, which is why defining them in the first place is so important.
- **Employee Retention and Satisfaction**. Hiring new employees doesn't do much good if you can't keep them in your organization. In fact, frequent employee turnover can be detrimental both in terms of productivity due to the time costs of onboarding new employees and taking over existing work from old ones, but also in that morale can suffer when people are leaving regularly. Having regular anonymous workplace surveys, where employees can give honest opinions about their work, the environment, and the company as a whole without fear of retribution, as well as exit interviews for employees who leave, will likely help uncover problem areas that need to be addressed.

- **Communication and Collaboration**. How well do new employees work with the existing teams? An engineer who is technically brilliant but that nobody can stand working with is unlikely to be a successful hire. If they don't end up leaving, they're likely to drive other employees to leave if their interpersonal behavior is problematic enough.
- **Productivity**. Perhaps the most common hiring goal is to get more work done as a company grows. If more work isn't getting done, it's worth digging in to try to find out why. If there is a skill set mismatch, that might indicate some changes that need to be made to the interview process. A new employee who is feeling blocked and struggling to contribute may be uncovering some cultural or process problems in the organization that need to be addressed.

Without defining beforehand what success looks like, it is difficult to measure it. Make sure that your hiring goals are specific and measurable as much as possible at the beginning of every hiring cycle, and you'll have a much better chance of meeting them.

Troubleshooting

What do you do when your hiring practices aren't meeting your needs. Chances are, you've probably run into at least some areas that you want to improve upon, even if it's just struggling to find enough candidates, a common complaint when there seem to be far more companies hiring devops skills than there are people looking to change jobs. In this section, we'll take a look at some common issues people run into with hiring and how to troubleshoot them.

We aren't getting enough candidates.

In today's competitive job market, finding candidates who are even open to talking about new positions can be difficult. If you work for a smaller company that isn't well-known in the industry, relying on having jobs posted on your company's website or word of mouth can be difficult. Some other ways to find candidates include:

Attend Local Meetup Groups

Meetup groups are becoming more and more popular, and not just in big metro areas that already have a well-developed tech scene. If you're looking for a particular skill-set, attending Meetup events in those areas can be a good way to make connections but also to get a feel for the community and the competition. Keep in mind that most Meetups frown upon recruiters who have no interest in the topics being discussed showing up with the sole purpose of trying to hire people. Sending developers to Meetup groups for their programming language or tool of choice can not only be a good learning experience for them but also a way of making genuine connections

with other people in the industry. You never know when someone you happened to meet locally might have the answer to a tough problem you've been working on, or six months down the line knows someone who happens to be looking for a job.

This can also work at larger industry conferences as well, though unless you live in an area that happens to have a lot of big tech conferences like the Bay Area, this is likely to be a more expensive strategy. Still, keeping a finger on the pulse of the industry is a great way to make sure your workplace is a competitive one.

Rethink Your Requirements

As we saw in the case study, your job posting might be hurting you without you even realizing it. If you're having problems finding candidates, have several people you trust to give you honest feedback look over your job descriptions. Look for things like, are you asking for a particular tool, programming language, or number of years of experience for a position that doesn't really need it? Are you asking more of a position than one person could reasonably be expected to do? Is the salary range you're offering for this position in line with the industry and the job requirements? (Remember, devops is not a way to get the work of two people for the price of one!)

Also consider whether you can take the time to train a more junior person for the position you need. With so many coding schools and bootcamps, as well as people who take other non-traditional paths into the tech industry, there are a lot of intelligent, qualified candidates out there, even if they don't have a 4-year degree or years of industry experience already.

Give Back to the Community

A spirit of generosity to the community, often demonstrated by speaking at conferences, writing public blog posts, and contributing to open-source projects, can be a great way of finding candidates. These sorts of things help to get your company's name out there, and demonstrate that it's a place to work where these sorts of activities are not only allowed but encouraged.

Expand Your Searches

Having employees putting feelers out at Meetups, conferences, user groups, and the like is great, but sometimes you might want to bring in a recruiter or tech sourcer, especially if you have a very particular role you need filled. The thing to keep in mind with using recruiters is that the recruiters you choose to work with reflect on your company. Most engineers are inundated with emails or LinkedIn requests from recruiters. Often times these recruiters haven't bothered to do even basic research, such as contacting a candidate for a Java position when that candidate doesn't even mention Java on their resume. Recruiters who try too hard to be cute or clever also aren't loved by candidates - asking if someone has gotten eaten by an alligator instead

of replying to your third unsolicited email is unlikely to get a positive response, and word gets around about companies or recruiters who use these kinds of tactics.

We aren't getting diverse candidates.

Similar to dealing with a shortage of candidates in general, not getting enough diversity in the candidates you do find can be very frustrating. In addition to the suggestions mentioned previously in the case study, you should have some people you trust read through your job postings.

It can often be helpful to have people of the sort you're trying to hire look over your job descriptions, but there are a couple potential issues with this. First, you may have a chicken-and-egg problem - how do you get diverse people to critique your postings if you can't manage to get any diverse candidates? Second, asking people in underprivileged groups to do diversity work for you is problematic if you are expecting that work for free. These people are on average already paid less than white men for doing the same work, so asking them to do additional work on top of that without compensation isn't helping anyone except you. There are diversity consultants out there who do this kind of work for a living; for the most part it would be better to bring in one of them.

In addition to off-putting job listings, a lack of diverse candidates might suggest other problems with your company culture. If your executive team and management are very homogeneous, you'll likely find it harder to find individual contributors who don't fit that same mold. People often times don't want to be the only different person on the team, and they can be hesitant to work for places where they don't see opportunities for growth and mentorship. A company known for overworking employees, for alcohol-laden social events, or a history of problematic behavior from employees, whether on company time or not, will turn people off from your organization as well.

These sorts of cultural problems require longer-term and more substantial fixes than simply changing the wording of a job description, but they are even more important.

Interviews are a waste of time for the team.

For technical skill mismatches, make sure your job description is clearly stating all necessary and desired skills. Don't list something as a "nice-to-have" if it's essential for the job you're hiring for, but also don't list something as necessary if you're fine with hires learning it on the job. If you're working with a recruiter, make sure they understand enough about what skills are needed to weed out any obvious mismatches.

Be aware of potential unconscious biases in sourcing candidates. It's easy for people to bring in their friends or people they know from the community because they're familiar, but that familiarity can cause them to move candidates ahead further than they might ordinarily progress (nobody wants to have to tell a friend that they aren't up to

snuff). People might also tend towards bringing in candidates who are similar to them, passing over candidates who might be more qualified but don't resemble the "typical" engineer they're used to. Unconscious bias training can go a long way towards getting rid of these kinds of issues.

Finally, make sure that everyone is on the same page regarding the interview process. If people are repeating questions that other interviewers have already asked, that very well might be a poor use of current employees' time. Having people asking irrelevant questions also isn't ideal. For example, if you're hiring for a position that requires no database knowledge, having the candidates be interviewed by a DBA who will only examine those skill areas isn't doing anyone involved any favors.

People aren't accepting our offers.

In our experience, we've found that the biggest reason that people don't accept job offers is compensation. As we've previously addressed, devops is not a way to get two employees' work for only one salary. The benefits that come from an effective devops culture will often eventually include cost savings, cutting down on salary costs should not be your primary reason for making these kinds of changes. If you are putting out offers that aren't being accepted, take a good look at whether or not your compensation packages are competitive.

This can include other factors aside from simply the annual salary numbers. Health coverage, including parental leave and transgender-inclusive healthcare, is very important. Vacation policies are important, especially for candidates with families. A healthy work-life balance can also be a factor. Asking somebody to "give 110 percent" and often work nights and weekends, or to be the only person on-call, is unlikely to make people want to join your team. Take a good look at your total compensation packages compared to what you're asking of your candidates and make sure that you have a good match.

Conclusion

There is no one magic bullet for hiring, no solution that will source amazing candidates out of thin air and make them fall in love with your company. Often times, problems with hiring can be indicative of deeper problems with a company's culture. It can be uncomfortable to address these kinds of issues - nobody likes to hear that they are doing things poorly, no matter the circumstances. But working to identify and address these issues will help not only your own organization but the industry as a whole.

CHAPTER 5

Tools: Selection and Implementation

This chapter is aimed at those who want to examine their current tools, optimize the process of selection and elimination, and measure and iteratively improve the tools in their environment.

We will cover the most common concerns and decisions that exist for different types of tools. It's important to note that while we may occasionally call out a feature of a specific product as being well-suited for a given need to illustrate the values that encourage collaboration, hiring and affinity, the authors are not paid to endorse any particular tools and strive to present an objective view of the current tool ecosystem.

Introduction and Audience

Generally the first place people look to implement change is through tooling. We have included this chapter towards the end of the book to emphasize the importance of examining **collaboration**, **hiring** and **affinity** early in your process. While tool choices tend to be easy wins, it can lead to obscuring issues in the interactions between team members as well as cultures across teams. This leads to invisible failure conditions as culture debt builds up.

One challenge to environments that use a waterfall methodology may be that the team is focused on discovering all parts of a tools strategy before implementing any single aspect. Some individuals may find themselves searching for tools that will solve all the issues present in the organization, speed up delivery of innovation and software, while providing value to customers. Looking for holistic solutions is essentially looking for the unicorn. Our industry is filled with very hard problems that are still in the process of being solved. If it is not obvious how to do something manually, there aren't going to be great solutions towards solving the problems programmatically.

Time and energy spent towards planning for all potential problems will lead to learning that there is no single solution.

Even if not tracking every single part of a tool ecosystem, teams experienced with certain types of failures may have trouble deciding on a single technology to implement in the stack due to fear of choosing the wrong technology. Deploying tools that can help visualize and track this contention for time, and wasted effort can help build an environment of continuous learning.

Tools are objects that exist in the world and are demonstrable improvements that individuals can show when management asks, "Are we succeeding at our devops initiatives?". Changes that impact culture outside of the technical landscape can be especially challenging when there is not executive buy-in towards cultural improvements, or when teams are deeply entrenched in their own current patterns.

Presentations, marketing, and certifications focus on tooling when talking about devops, illustrating the value in easy-to-consume graphs. Company newsletters, mainstream media, and conference booths will display lists and articles covering the "best" tools for a devops toolchain. How can you tell the difference between a company trying to sell a solution that could be effective in your environment versus a company trying to get on the trend of devops?

Some people de-emphasize the importance of tools, saying "devops is not about tools", and "anyone who is trying to sell a devops tool, toolchain, devops in a box, or anything similar is simply looking to make a quick buck off of people who don't know any better". How do you know where to start looking to introduce devops into your environment with any effectiveness, with the cacophony of competing opinions on whether or not tools are essential to devops?

In this chapter, we will help you take a critical look at your current toolset, optimize the tools you are using, and selectively choose new tools to complement your current environment while averting the disaster of cultural debt buildup. In the appendix, you will find a checklist that will help you start assessing the current state of your tools. This chapter will provide key strategies for examining the value of advertised tools to help you determine whether a tool is useful for your environment. It will help explain why some current toolchains that are outdated or ineffective may be worse than having no tooling at all.

If you're overwhelmed by how many tools are out there claiming to be a necessary part of a devops transformation and want to know which of them, if any, you actually need in your environment, this chapter is for you. People looking to understand which tools are most important, which to change or implement first, and how to choose between various competing tools will also find this chapter beneficial.

Why Tools Matter

Tools are inherent to our jobs, inherent to how we solve the problems we face each day. Our comfort level with the set of tools that are available to us, and our ability to adapt to new tools as they evolve and shape our thoughts and ideas. The availability of collective knowledge within the palm of your hand combined with the collaboration across organization and company boundaries through open source software is dramatically disrupting the status quo of work. Companies mired in managing infrastructure configuration management by hand with unknown numbers of divergent systems, unable to quickly change and respond to market demands will struggle against their counterparts who have managed to contain their complexity on one axis through infrastructure automation. While it is possible to manage servers by hand, or even artisinally crafted shell scripts, a proper configuration management tool is invaluable especially as your environment and team changes.

Even the best software developers will struggle if they are working in an environment without a version control system in place. Tools matter in that not having them, or using them incorrectly, can destroy the effectiveness of even the most intelligent and empathetic of engineers. The consideration you give to the tools you use in your organization will reflect in the overall organization's success. You'll find that what is a good tool for some teams might not be a good one for others. The strength of tools comes from how well they fit the needs of the the people or groups using them. If you don't need feature X, its presence won't be a selling point when considering which tool your organization should use. Especially in larger organizations with teams numbering in the dozens, finding one tool that meets the needs of every team will be increasingly difficult. You will have to strike a balance between deciding on one tool that will be used across the entire company consistently and allowing more freedom of choice among individual teams. There are benefits to both the consistency and manageability that comes from having only one tool in use in an organization, and also from allowing teams to pick specific tools that work best for then. We will discuss how to strike that balance in more detail in this chapter's case study.

Think back to the two main ideas covered in the previous chapters: individual collaboration and team affinity. If tools, or lack thereof, get in the way of individuals or teams working well together, your initiatives will not succeed. The cost of collaboration is high, investing in no or poor tools can raise the costs dramatically.

Humans have a long history of using tools in order to get jobs done more effectively. Moving from typewriters to word processors allows people to more easily make changes and correct mistakes. Going from punchcards and assembly languages to higher level languages lets us better understand the code that we're writing, and even more so lets us share that code with others and have them easily understand it. These tools were not invented as ends in and of themselves - they were all created to make specific jobs easier for the people using them, and that's an important thing to keep in

mind when choosing tools for use. These tools all allow us to collaborate more as software has moved from being written by one person and only read by that same person to being written by multiple people, multiple teams, and having to be understood and maintained by different teams, sometimes years later.

Often when we talk about tools, we talk only about the software side of things - which programming languages we write with in which IDEs, which text editor or shell we prefer, which configuration management solution and which chat program. It's important to keep in mind that tools might mean hardware as well. A smaller, lighter laptop is less physical strain when traveling to conferences or bringing a computer through a data center. Choosing a hardware RAID solution over a software RAID solution costs more money but offers feature like battery backups and easier maintenance. With so much being in the cloud or as a service these days it's easy to focus most of our attention on software, but it's important to keep hardware in mind as well.

Not all tools are created equal - this is something that applies to both hardware and software. If they were, we wouldn't need to write this chapter - you could just pick whichever tool was simply the cheapest, or the one whose logo you liked the best. Even among tools that nearly everyone agrees are key like configuration management or source control, some are better suited to collaboration than others.

Tools can be a way of giving back to the community. Chances are, the problems that you are having aren't unique to your organization, or at least not so unique that you wouldn't be able to share some of your tooling with other companies. Sharing and open-sourcing the tools you've written keeps other companies from having to reinvent the wheel.

But what about companies who worry that open sourcing their tooling will take away their competitive advantage? If your company's success depends on your tooling, that should be part of your business model, so that you're making money from either the software or, as many companies do these days, the support for that software. If you use any open source software, you owe it to the open source community to give back in some way - nobody likes a person who only takes and never gives anything back. If every company was selfish and never open-sourced anything, the industry as a whole would be less innovative because people would have to spend their time writing their own version of tools that already exist, solving problems that have already been solved by other companies, which is time that can't then be spent on things that actually bring value to the company's bottom line. Technology gives us the ability to make significant positive changes to the world, and we should be able to focus on that instead of writing some tool that already exists somewhere else.

Contributing to open source is an excellent reflection of company intentions. Open sourcing software within companies encourages teams to contribute to each other's projects rather than reinventing the wheel, and it exposes people, both individual

contributors and managers, to the benefits of open source collaboration. Contributing to open source and using open source often go hand in hand as well. Teams that are used to the open source community are more likely to look for open source solutions that already exist rather than writing their own. As an example, Yahoo went through five homegrown configuration management solutions prior to finally adopting Chef. How much time and energy went into developing those internal solutions that could have been better spent on their core products and services?

Many companies look towards the well-known companies in the devops space and their open source contributions, such as Netflix and Etsy, and feel compelled to start writing and open sourcing many of their own tools as well. Despite the benefits of open source contribution, balance is key here - too far in the other direction and you end up with Not Invented Here (NIH) syndrome. This is a term used to describe companies who, on principle, refuse to use third party tools because they originated outside the company. There can be various reasons for this behavior, with the most common probably being competition, where companies don't want to acknowledge solutions created by competitors, let alone use them, or fear of software that is external and unknown. Maybe they don't trust that other people could write code as well as they could themselves, or maybe they'd rather write something than figure out how to read and use someone else's code. Some people just like the challenge of creating software that they haven't before, or trying a project written in a new language.

There can be valid concerns about using a third party solution. If it is something that will be integrated into an existing software project, that project's license might have to be updated or change to match the license of the new external component. There is also the possibility that the software's maintainer will abandon it, no longer providing bug fixes, security updates, or support, or that future development will introduce breaking changes. Companies might also not want to be tied to a particular vendor for a variety of reasons. Still, there are serious drawbacks to a case of Not Invented Here. Unless you have a team of security experts, any cryptographic software you write is likely to contain bugs and thus security vulnerabilities. A company not in the business of networking is unlikely to get any benefits from writing their own DNS server - any effort they can produce is unlikely to be better than BIND, and certainly won't be worth the time spent in development, maintenance, and troubleshooting a piece of software that nobody else has any experience with.

Tools can and will enforce behavior, which will affect your culture. It is critical to examine tools when examining the behaviors and cultures, and this is where the true importance of your tooling choices lies.

As we discussed in Chapter 1, a lot of working effectively comes down to developing shared mutual understandings and negotiation around the inevitable miscommunication and misunderstandings that come when trying to navigate multiple goals at once. Tools can help to communicate with each other. Tools can help work out

boundaries. We also said that, given the knowledge that both parties are still in the compact, everything else comes down to repair, and repair operations are best served with tools. It is the strengths of the tools we choose that will become our greatest strengths when trying to work together effectively.

Availability of data, empowered to inform and guide decisions. Tools shape the questions we ask, the information readily available to us, how we analyze the information, and build from it. Tools change the dynamics of how the company communicates point to point to vast interconnected networks.

Many organizations cling to homogeneous standardization of laptops with specific tools to ensure reliability and security. It is important to balance these concerns with allowing for flexibility that will let people be more productive at an individual level. We'd also like to note that firewalls that prevent collaboration on social tools like Twitter and Facebook, eliminate a source of knowledge and collaboration from users of services. As we'll discuss more in the Collaboration and Hiring chapters, there are better ways to ensure employee motivation and productivity than policing their behavior the way that one would for young children.

Standardization of tools can also help create stable bridges from old to new as the technologies being used at a company change. If there is a consistent process for evaluating and choosing new tools, as well as retiring old ones, organizations will be more likely to decide upon a tool that meets the most peoples' needs, make sure that any necessary features that were present in an old tool are also features of a new one, and ensure that employees are properly trained to be able to effectively use a new piece of hardware or software. Without this kind of bridging, employees might be more resistant to new tools or technologies simply because the change was too abrupt or the transition wasn't handled well.

Choosing specific tools within our tool categories matters because they help shape and define the way we work. For example, choosing git as your version control system encourages pull requests in a way that cannot happen with something like subversion, because while git allows for repositories to be forked, subversion uses only one repository for everybody. The same behavior can ultimately be accomplished in subversion, but it is much more complex to do so, and because of that overhead many people will choose not to go that route if they even know it exists. So while both of these version control systems technically allow for this same kind of collaborative behavior, only git encourages it, and this will be seen in the work patterns of developers working with each system.

Why Tools Don't Matter

There have been differing opinions on whether or not tools matter, and how much so if they do, over the course of the devops movement. Saying that tools don't matter

developed in response to too many vendors jumping on the bandwagon of calling everything "devops" (regardless of whether or not that label was accurate) trying to sell their products. By saying that tools don't matter, people wanted to convey the idea that tooling is not a sufficient condition for a devops culture to exist. Tools do not fix broken cultures, they can serve to expose and exacerbate existing conditions.

Your company might fail if you can't figure out how to use configuration management at all to the point where your beautiful and unique snowflake servers are constantly causing lots of downtime and thus lost revenue, but if you are using configuration management properly, the choice to use Puppet or Chef (or Ansible or Salt or even CFEngine or some new CM system that hasn't even been written yet) doesn't matter if you can use it to do what you need to do. While there are technical differences between those different tools, what really matters is if the tools have the features that your particular organization needs to solve their problems.

Conway's Law is the idea that software tends to end up being developed in ways that mirror the structures and organization of the teams that developed it. A corollary to this might be that teams tend to end up choosing and using tools in ways that mimic their original structure and communication patterns. Two teams that don't communicate with each other aren't going to start doing so just because the company started using Slack as their new chat system. But since tools shape behavior, having tools that reduce the friction required to communicate with other teams makes it more likely that communication will start to take place. If a company doesn't even have any chat software, or if what they use has technical limitations that prevent inter-team communication, it will be much harder for communication to happen.

Conway's Law

Conway's Law is a saying named after Melvin Conway, a computer programmer who stated that "organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations", or, more simply stated, software components will tend to communicate in the same way that the teams who designed and created them communicate with each other. This means that in order for two software components to work together if they are each being designed and implemented by a separate team, those two teams must be able to communicate as well. Conversely, teams that do not communicate well, such as in a heavily siloed environment, will tend to create products that don't work together well either.

Sometimes, the way tools are used is of greater importance than the specific tool chosen. Take a ticketing or bug-tracking system, for example. If every team decides that tools don't matter and picks a specific ticketing system that complements their working style, changes are high that intra-team issues will arise down the line that

impact cross-functional teams as well as increasing the overall communication that management have to be aware of to distil flow and current status. Individuals will either have more accounts to manage, or lack the visibility into another teams work. This lack of visibility is one of the problems that often plagues siloed organizations. Siloization can lead to duplicated effort, a lack of clarity or detail as to what is actually being worked on (or whether or not work is actually getting done at all), and distrust between teams.

This same principle can be applied not only to ticketing systems but also infrastructure automation, chat systems, deployment tools, and any tool that is used by multiple teams within an organization. While it is important to figure out everyone's requirements and try to meet as many of them as possible, it is unlikely that 100% of your people will be 100% happy with any tool at all - compromise is pretty much guaranteed. At some point, continuing to argue and debate over which tool to use doesn't gain anything, and may end up causing hostility as arguments continue to occur in addition to all the time lost. It's tempting to say just pick one tool and stick with it.

In order to steer clear of the kinds of logistical nightmares that can arise from having each team using a different ticketing system, for example, in most cases it makes sense to have everyone end up using the same tools across the company. Even if there isn't 100% satisfaction, this kind of standardization will cut down on support or licensing costs as well as making things easier on whatever help desk staff that has to support these tools.

There are of course exceptions to this rule. If a team needs to be somewhat isolated for whatever reason, there isn't necessarily cause to force them to use the same tooling as everyone else if there is a reasonable need for them to do otherwise. PCI compliance, for example, requires a very strict separation of duties, so that a team doing PCI work is likely to have separate computers running on a separate network from the rest of their organization. In a case like this, since they are already somewhat segregated from the rest of their environment, they could conceivably use different tooling without having a detrimental effect on the organization as a whole. These are decisions that you will need to make on a case-by-case basis.

Even though there are so many commonalities, each team and each company is going to have unique needs and experiences. In the case studies in this chapter, we will look at how two companies approach their tooling selection and implementation decisions. Despite their many differences, common practices emerge showcasing how different devops can be even with similar tooling. The case study will help illuminate points for consideration that you can use in your own organization.

With those caveats in mind, arguments about which particular tool to use among choices that all fulfill your requirements don't make sense. This chapter isn't going to tell you that X is the One True Y Tool for Devops, because there is no such thing.

That would be the same thing as declaring *ed*.footnote[ed is a line editor for Unix. At one point, it was the default editor for systems and it's terseness made it very difficult albeit powerful to use in automation.] the true victor in the editor wars. As tools do heavily influence behavior, give serious consideration when evaluating your environment, assessing the cultural and technical landscape, and collaboratively defining the goals and visions of the team or organization to make the right choice. Keep in mind that this is an ongoing process that requires continual re-evaluation.

Tool Ecosystem Overview

Configuration Management

Started in the 1950s by the United States Department of Defense as a technical management discipline, configuration management (CM) has been adopted in many industries. Configuration management is the process of establishing and maintaining the consistency of something's performance, functional and physical attributes throughout its life-cycle. This includes the policies, processes, documentation, and tools required to implement this system of consistent performance, functionality, and attributes. Specifically within the software engineering industry, ITIL, IEEE, ISO, and SEI have all proposed a standard for Configuration Management. As with the term "devops" this has led to some confusion in the industry about a common definition for configuration management.

To ensure a standard common place from which to build from for the audience of this book, we define configuration management as the process of identifying, managing, monitoring, and auditing a product through its entire life including the processes, documentation, people, tools, software, and systems. Often infrastructure automation is conflated with configuration management which creates a divide with other disciplines usage of CM.

Version Control

Version control records changes to files or sets of files stored within the system. This can be source code, assets, and other documents that may be part of a software development project. Developers commit changes in groups called commits or revisions. Each revision along with metadata such as who made the change and when, is stored within the software depending on the version control implementation.

Version control systems can be categorized by the process of how the metadata and revisions are stored: local, centralized or distributed. While the inner workings of version control systems are beyond the scope of this introduction, understanding the overall concept of version control and the limitations of different models as they

influence the collaboration and affinity of team members is important in understanding impact on software development.

Local vs Centralized vs Distributed

Local version control is managed through saving patch sets of differences in files on a single node local to where the files are stored.

Centralized version control is managed through a single remote server containing all versioned files. Clients check out files from this centralized location.

Distributed version control is managed through entire repositories being replicated across different nodes.

Having the ability to commit, compare, merge, and restore past revisions to objects to the repository allows for a richer cooperation and collaboration within and between teams. It minimizes risks by establishing a way to revert objects in production to previous versions.

Infrastructure Automation

Infrastructure automation is creating systems that reduce the burden on people to manage services and increase the quality, accuracy and precision of a service to the consumers of a service.

Example 5-1. Convergence versus Congruence

Convergence is defined as the process of arriving to a desired end state based on calculating the route from the current starting point. If there is a failure, another round of execution will still strive to reach the desired end state by re-calculating the route without rolling back changes and starting from the beginning.

Congruence is starting from a blank slate and following a sequence of steps to massage a system into a desired state.

System Provisioning

Once companies had to plan, buy, and provision hardware in data centers. Now, companies have the option to invest in cloud infrastructure with the option towards on-demand computing where they purchase only what they need and scale up and down as necessary.

Infrastructure automation allows the definition and control of how a given system is set up to be described in code, from the system settings to the programs that are installed and running to user management and network configuration. Benefits of

describing infrastructure as code include repeatable, consistent, documented, and resilient processes that withstand some amount of failure. This frees up time, improves efficiency of staff, allows for more flexibility, and risk measurement. It also increases the degree of confidence that individuals have in the machine setup and deployment being identical reducing the amount of time spent debugging problems based on system differences.

Provisioning automation is an extension to infrastructure automation allowing companies to define infrastructure in terms of the clusters of dependent systems required to define their infrastructure and not just single nodes. It allows individuals to specify how they want a group of servers to be provisioned once, and then to automatically use that specification as many times as they want later. Often, server hardware manufacturers such as HP and Dell will provide a provisioning tool that will only work with their brand of hardware, but there are also open-source solutions available.

Different Linux distributions will often provide operating system-specific tooling as well. As an example, Cobbler and Kickstart can be used to automate the provisioning of systems running Red Hat Enterprise Linux or CentOS. Operations staff can write "Kickstart files" that can specify hard disk partitioning, network configuration, which software packages to install, and more.

Provisioning automation tools can even be made to install and set up infrastructure configuration. In this way, the creation and management of servers can be automated nearly completely from end to end, reducing the amount of time required to do the repetitive tasks of clicking through tedious installation screens. Computers are much better suited to these tasks than humans are, reducing errors and ensuring consistency in delivered product.

Some infrastructure automation tools have been extended to provide this functionality. Chef Provisioning which is included in the Chef Developer Kit lets you define the whole cluster of systems including hardware, network and software.

Hardware Lifecycle Management

Every company has to deal with hardware lifecycle management in one way or another - though the advent of the cloud and infrastructure or platform services has cut down on the amount of attention this requires to some extent. The hardware life cycle begins with planning and purchasing (or leasing), continues with installation, maintenance, and repair, and ends with trading in, returning, or recycling hardware that has reached the end of its life.

Provisioning automation tools greatly cut down on the amount of manual work that must be done during hardware installation and after hardware maintenance. Without it, installing new hardware meant manual configuration by system administrators or data center technicians, which, like all repetitious work, tends to be error-prone as

previously mentioned. Automating the provisioning steps makes bringing up new hardware more reliable in addition to being much faster.

This same automation can aid in repair and maintenance. If a server must be taken out of production to do something like replace a bad memory module or upgrade a hard drive, automation of provisioning and server state can ensure that it gets put back into production in the same state, with the same configuration, as it was when it was taken out, or that if a server needs to be completely rebuilt it can be done so easily. If a particular process needs to be followed for maintenance or decommissioning of hardware, this can be automated as well, freeing up system administrator resources to focus on more intensive work.

Continuous Integration

Continuous integration (CI) is the process of integrating new code written by developers with a mainline or "master" branch frequently throughout the day. This is in contrast to having developers working on independent feature branches for weeks and months at a time, only merging their code back to the master branch when it was completely finished - long periods of time in between merges mean that much more has been changed, increasing the likelihood of some of those changes being breaking ones. With bigger changesets, it is much more difficult to isolate and identify what caused something to break. With small, frequently merged changesets, finding the specific change that caused a regression is much easier. The goal is to avoid these kinds of integration problems that come from large, infrequent merges.

In order to make sure that the integrations were successful, CI systems will usually run a series of tests automatically upon merging in of new changes. When these changes are committed and merged, the tests automatically start running to avoid the overhead of people having to remember to do so - the more overhead an activity requires, the less likely it is that it will get done, especially when people are in a hurry. The outcome of these tests is often visualized, where "green" means the tests passed and the newly integrated build is said to be clean, and failing or "red" tests means the build is broken and needs to be fixed. With this kind of workflow, problems can be identified and fixed much more quickly.

Test and Build Automation

Test and build systems manage the testing, workflow and build process that qualifies and builds releases. Test and build automation automates these processes eliminating the manual steps required. Just as with infrastructure as code, this hands the tasks to the computers to handle.

Back in the days of the first computers and compilers, programs were rarely contained in more than one source file. As programs began to grow in size and complexity, developers started to split them out into multiple source files. Standard libraries of

code made available to users of a given programming language added to the complexity. With so many different source files needing to be compiled together correctly to get the final program executables, it became necessary to automate the build processes.

Build automation tools today usually specify both how the software is to be built (what steps need to be done and in what order) and what dependencies (what other software needs to be present in order for the build to succeed). Some tools are best suited to projects in specific programming languages, such as Apache's Maven and Ant which, while they can technically be used with other projects are most often used with Java projects. Others, such as Hudson or Jenkins, can be used more broadly with a wider range of projects.

These tools usually fall into one of three use cases. On-demand automation is run by users, often on the command line, at the users' discretion. For example, a developer might run a Make script by hand during local development to make sure she can build the software locally before checking it into version control. Scheduled automation is automation that runs on a predefined schedule, such as a nightly build. Nightly builds are created every night, usually at times when nobody is working so that no new changes are taking place while the software is building (though this is becoming less doable as teams get more globally distributed). Finally, triggered automation happens as specified events happening, such as a continuous integration server that kicks of a new build every time a commit is checked into the code.

Test, Monitor, or Diagnostic

Words can be problematic and distracting. Often tests, monitors, and diagnostics are conflated causing more churn within and between teams. In order to work together, teams need to establish clear boundaries by establishing a common vocabulary to encode information. This encourages mutual understanding without limiting any individual team member or requiring everyone to know every single detail.

During Sysadvent 2014, Yvonne Lam identified a set of questions a team should ask to build this shared context around tests, monitors, and diagnostics.

1. Where is it going to run?
2. When is it going to run?
3. How often will it run?
4. Who is going to consume the result?
5. What is that entity going to do with the result?

Lam further enumerated a set of definitions that could be applied to clarify the differences.

Tests run against non-production systems and qualify the system or software readiness. A test generally runs when something changes.

Monitors run against pre-production and production systems on a schedule. A monitor generally runs frequently or is triggered by an event.

Diagnostics run against production systems on demand due to an event.

Continuous Delivery

Continuous delivery is the process of releasing new software frequently through the use of automated testing and continuous integration.

Application Deployment

Application deployment is the process of planning, maintaining, and executing on the delivery of a release of software to the compute resources required.

In the general sense, the craft of application deployment needs to consider the aspect of changes underneath the system. Having infrastructure automation building the compute, operating system and dependencies required to run a specific application minimizes the impact of inconsistencies impacting the software.

Depending on the application type, different engineering concerns may be important. For example, databases may have strict guarantees in terms of consistency. If a transaction occurs, it must reflect in the data. Application deployment is a critical aspect to engineering quality software.

Continuous Deployment

Continuous deployment is the process of deploying change to production through the engineering of application deployment that has defined tests and validations to minimize risk.

The faster software changes make it into production, the quicker individuals see their work in effect. Visibility to work impact increases job satisfaction, and overall happiness with work leading to higher performance.

It also leads to opportunities to learn quickly. If something is fundamentally wrong with the design or feature, the context of work is more recent and change can occur.

Continuous deployment also gets the product out to the customer faster.

Metrics

Metrics are the collection of qualitative and quantitative measurements. Generally they are compared against some benchmark or established standard, tracked for ana-

lytics, or for historical purposes. Often metric are siloed within functional organizations which can impact choosing the right direction with product development.

Metrics are one of the key parts of monitoring - data can be gathered and stored for nearly any part of even the most complex web software, and different teams can have different metrics that they keep track of and use in their work. Statsd and Graphite are very commonly used and a powerful combination for tracking, storing, and viewing metrics.

There is a community driven effort to define the set of system and application metrics that should be collected grouped by protocol, service, and application on github in the metrics-catalog repo.

Logging

Logging is the generation, filtering, recording, and analysis of events that occur on a system due to operating system or software messages. When tracking down the source of a software issue, one of the first things that engineers often do is to check the logs for any relevant error messages. Logs can be treasure troves full of useful information, and with storage getting cheaper and cheaper, just about any log you might want can be saved and stored for later use. Logs can come from the applications you develop, from third-party tools you use, even from the operating system. As there is no standardization across software for logging, it can be difficult to categorize and qualify events within logs to identify patterns of concern.

A single system generate hundreds of lines of logs a day. In modern environments that have thousands of applications running on hundreds or thousands of servers, the sheer volume of log data can be overwhelming - it is no longer a simple matter of searching through one log file. Much work has gone into developing applications that handle the storage and searching of logs.

Monitoring

Monitoring is a large topic that can be split into multiple facets events and analytics. The methods of information collection include metrics and logs. Monitoring includes gathering basic system-level metrics such as if a server is up or down, how much memory and CPU are being used, and how full each disk is, or it might be higher-level application monitoring which can range from how many user requests a web server is handling, how many items are queued in a queuing system, how long a given web page takes to load, and what are the longest-running queries going into a database. While once solely the domain of system and network administrators, as software grows more complex and teams collaborate more, people are beginning to realize that it's a core reflection of product health.

One of the benefits of using configuration management is that it can be used to help automate the process of setting up monitoring of new systems or services. Monitoring only works when it's configured to monitor the right things - you don't want to realize only after an incident has occurred that monitoring hadn't been set up properly yet. Once you have determined what you need to be monitoring, configuration management can be used to automatically monitor new hosts or services as they are deployed or provisioned. Additionally, many configuration management systems have monitoring built in to them that can provide information such as when each host checked in, to gain additional confidence that systems are configured as expected.

In today's environment, with more and more systems being connected to the internet and the increased visibility that comes with incidents that are highly publicized, companies are starting to care an increasing amount about security. While the topic of how to secure your environment is far beyond the scope of this book, we will say that in addition to checking on the health of your systems and applications, you should consider monitoring the security of your environment as well. This could mean anything from tracking numbers of failed login attempts to setting up a sophisticated intrusion detection system, but either way it is not something that should be forgotten or left until the last minute.

In general, monitoring is the process of tracking the current state of your systems and environment, usually with the goal of checking whether or not they meet some predefined conditions of what the desired state is. Often monitoring, alerting, and testing are conflated. This leads to confusion around understanding what we are trying to accomplish or build. As mentioned above, monitoring usually runs on a predefined schedule while tests are run in response to changes. Alerts are automated communications sent to humans about the results of a test or a monitor.

Alerting

Monitoring and alerting are important not only from a performance perspective, but also as a way of trying to make sure that you find out about potential issues before they become actual issues for your customers. When the United States' HealthCare.gov site was first launched in October 2013, they originally had no monitoring or alerting to let them know whether or not the website, which had been two years in the making, was up or not. As discussed by United States Digital Service administrator Mikey Dickerson in several industry talks he's given, his team was reduced to watching new sources such as CNN to report on whether or not the site was having issues as their original form of monitoring during the site's first few months of automation. While it is not a panacea, a well-considered alerting strategy could have cut down on some of the embarrassment that came from having those issues be so public.

When reasoning about alerting, there are several factors that need to be considered:

- **Impact**: Not all systems have the same impact - something that is widespread, affecting multiple systems or a large group of customers, has a much higher impact than something that affects only a small subset of systems or people. Some incidents aren't customer facing at all, or might affect systems that have enough redundancy so as to not have much impact that way. To avoid alert fatigue, as we will discuss in more detail later, alerting should be restricted to incidents that have the most impact.
- **Urgency**: Similar to impact, not all issues are equally urgent. An urgent issue is one that requires a fast (or sometimes immediate) response. For example, your site being completely down such that you are currently losing money or customers is much more urgent than a purely informational blog site being unreachable. Different stakeholders will likely have different opinions about what is urgent, so its important to consider all the stakeholders when configuring your monitoring and alerting.
- **Interested Party**: Primarily, the parties interested in an incident are those affected by it - this could be your customers (or a subset of them) or groups of employees in the case of internal service incidents. Interested parties could also be taken to mean who is responsible for responding to an incident. For example, if only DBAs can deal with a particular kind of database issue, it would make sense to alert them, rather than alerting an operations team whose only action would be to call the DBAs.
- **Resources**: What resources are required to respond to a given incident or alert, and what is the availability of those resources? Is there enough human coverage to make sure that multiple incidents can be responded to, or do you have only one on-call person that is a single point of failure without backup? Does your organization have the resources to function without a given service, piece of hardware, or individual? These are all things to consider when setting up your alerting.
- **Cost**: There is cost associated with monitoring and alerting, from the cost of a monitoring service and solution, to storage space for historical monitoring or alert data, to the cost of sending out alerts to humans, not to mention the costs associated with responding to incidents from the point of view of the person handling the issue to the cost to your company if a given service is unavailable.

In general, alerting is the process of creating events from the data that is gathered through monitoring. We will now take a look at events in more detail.

Events

Event management is the element of monitoring that is concerned with existing knowledge around impacts to systems and services. For 24x7 services this generally

reflects the need for real-time information about the status of all the different components of infrastructure. A system is configured to monitor a specific metric or log based on a defined event and signal or alert if a threshold is crossed or an alert condition has been met.

With much software development now being done on web software that is expected to be available 24/7, more consideration is being given to handling alerts that occur when engineers are at home instead of in the office. One way of dealing with this is to set up as much automated handling of events as possible.

Many alerting and monitoring systems have built-in ways to automatically respond to a given event. The Nagios monitoring system, for example, has "event handlers" that can be configured for different alert conditions. These handers can do a variety of things, from automatically restarting a service that had crashed to creating a ticket for a technician to replace a failed hard disk. Automated event handlers can cut down on the amount of work that Operations staff have to do (and likely the number of times they get woken up off-hours as well) but they are not without their risks. It's important to make sure that your failure conditions are clearly defined, that the event handler process is understood well enough to be automated, and that there are necessary safeguards in place to keep the automation from causing more problems than it solves.

No alerting system is 100% accurate 100% of the time. A **false positive** event is when an event was generated when there wasn't actually an issue. If your events generate alerts such as pages that are designed to wake people up off-hours to deal with them, a false positive will disturb someone's sleep unnecessarily. On the flip side, a **false negative** is when an incident occurred without generating an event for it, which could lead to a longer time before the issue was detected and resolved. There are costs to both false positives and false negatives, and which one is better or worse to risk will depend on your specific issues and environment.

Over time, you will want to tune and adjust your monitoring and alerting as you learn more about the true impact of your issues and events. We recommend monitoring the trends of your alerting, including information such as whether or not any action was taken for each event, how many of your alerts were actionable overall, and how many of them occurred off-hours.

Alert design, or how to create alerts that convey information in the most efficient way for humans to interpret, is a big topic in alerting these days. Etsy developed their OpsWeekly tool to allow for this kind of tracking as well as categorization of alerts by type of alert and component. Keeping track of alert trends, and performing analytics on the alert data can make a big difference in improving the effectiveness of alerts, as

well as improving the health and happiness of the humans whose job it is to respond to them.

Alert Fatigue

Just as with other first-hand responders, in the field experience leads to implicit knowledge of what alerts are noise. It may be hard to generalize an automated mechanism that handles all cases clearly, but it is important to keep working to improve the efficiency of the alerting system. Alert fatigue, or desensitization to alerts (usually false positives) can lead to slower response to actual issues as well as contributing to burnout.

Environments change, something that was a problem before may not be a problem now due to change in function in software or complexity grows and old way of solving the problem doesn't work anymore. Humans can change to deal with the issue quickly, algorithms do not have the same adaptive behaviors. Dealing with this constant change is an important part of alert and incident management.

Auditing your Tool Ecosystem

Before diving into selecting specific tools, you must examine the state of your ecosystem. In the appendix, you will find a checklist that will help you start assessing the current state of your tools. The first litmus test for many tools is whether the function exists in your environment. In this section we will establish the basic types of tools, the essential foundations that help measure the overall health of your ecosystem and will ensure support for the rest of your choices.

When auditing your environment to identify your tool ecosystem also include information about who has access and overall usage of the tool. Also include information about multiple tools within the same category or overlapping tools in your environment. This will inform areas of improvement that may require additional training or replacement of tools.

Alignment with process and individuals desire to use a specific tool is critical for effective tool usage. Too much process leads to a high cost to individuals as they are trying to maintain the intricate context around the processes. This effort can detract from project work. Too little process leads to lack in team cohesion with the proliferation of tools and methods of using the tools. This can also impact individuals. Time may be spent repairing understanding, and merging work or examining duplicated work. Finding this baseline is key to all aspects of identifying and selecting tools. It is even more important as we try to scale up and down organizations.

Communication

Examination of the tool ecosystem starts with communication. Work environment community is built from effective communication. Communication is a key part of being able to work together, and the tools and processes that your company uses for interpersonal communication can have noticeable effects on culture. In addition, every tool that you use has implicit communication inherent to its use.

Include the state of company policies in your examination of your tools and identify how those policies may impact the efficacy of the tools. These policies may be areas of improvement more crucial than improved tooling.

As mentioned in the Collaboration chapter, there are many factors to take into account with communication. These factors preclude finding a single communication tool that will meet all of the needs inherent to a healthy organization. As with any of the fundamental tools, examining the tools in place now is critical prior to making changes in the availability of tools.

It's likely that your communication needs will change as your company grows. While communicating with everyone via chat might make sense at a small startup when everyone can easily participate in one conversation, over time you might find teams leaning more towards email or team wikis.

Measuring Participation

It is critical to understand and measure the participation within the larger scope of the organization as your company grows. We will give this topic more focus within the Scaling your organization chapter, but the importance can not be stressed enough.

The process of finding the right tools at the right time is an iterative process. Ensuring that all voices are heard in the critical conversations ensures for a more healthy company. Silence should not be assumed to mean consent to the majority. As evidenced through research on smart teams, teams where everyone has a voice work are more effective and produce more.

When working with remote employees, invest in high-quality video conference solutions. Ensure that your team members have quality headsets as the onboard laptop microphones and speakers will lead to sub-par experiences that discourage individuals from using the medium.

It's a matter of choosing the tools, platforms, or methods based on the content, immediacy, context, and other factors of the communication itself. Once these needs are

identified based on the types of communication you find yourself or your team participating in, you can work on choosing the particular tool of that type based on other needs (such as paying for a chat program or using a free one).

The most important thing to keep in mind is that communication tools are like traveling companions in the convoy we introduced in Chapter 1. They can either be good companions that help us keep the compact we have made, or they can be poor companions that distract us or get in our way. This, of course, will depend on how each of our tools is used - trying to use a low-immediacy medium such as email for things that require immediate replies will cause problems, as will trying to describe something only with words when drawing a picture on a whiteboard would be quicker and more effective.

Finally, when choosing tools it is necessary to balance cohesiveness with flexibility. If there are too many communication methods used, people will likely find themselves having to do lots of searching for the information they need (was that in an email or a Google Doc or a Confluence wiki page?) or struggling with how to most effectively reach people (should I send an instant message or a text message or go to their desk for something like this). Too few methods, on the other hand, and you'll cause frustration - we've all heard the stories of companies who help meetings for everything when too often a quick email would have done the trick.

Our compact, with its focus on shared mutual understanding, will be aided if there are traditions or customs that people can draw upon to help decide on the most effective medium, but only if there is still enough flexibility to choose the right tool for the right job.

Local Development Environment

Often overlooked a consistent local development environment is critical. This is not to say that individuals should be locked into a single standard editor to get their job done. It means ensuring that individuals have the tools needed to get the job done.

It can be hard to qualify what a reasonable standard for tools can be based on the differing nature of jobs. For some who do a lot of image creation and manipulation, hardware and software requirements are vastly different than those who are focused on backend development focused on database schema design.

A reasonable standard for the average developer in 2015 to use some of the tools that will be described in this chapter is 16GB memory, and a reasonable sized solid state drive (SSD). These recommendations allow for sandbox environments that replicate production on the laptop, and overall decreased latency for every step of development along the way from compile times to opening up applications. Other aspects of minimal requirements may vary in your environment depending on individual preferen-

ces from multiple displays for increased collaboration, retina displays allowing for more comfortable long term viewing sessions, to specific keyboards.

Qualifying the current standard of your local development environment includes determining whether there is a consistent framework that teams share within the team and across teams. Reflect back on the topics covered in the Hiring chapter on onboarding, and the assessment of how long it takes an engineer to get onboarded within your environment. Further, how much consistency is there or do team members comment on having to establish their own way of doing things. This is a potential danger zone where knowledge is gained and isolated within individuals with regular incurred additional cost where individuals spend creativity time on fussing with special environment setup to get work done.

Identify a shared area for documenting the local development environment. This could be within a version control repository, or an internal wiki. Proficiency with tools will happen over time and use, so the goal isn't to have documentation that elaborates on every single detail, but enough to get individuals going and successful in the environment.

Invest in a quality editor(s). Depending on the size of your company, and teams, ensure there is budget to invest in quality editors. While individuals will be very attached to the editor that they have used over years, having the ability to choose an editor that will work more efficiently in your environment with the languages in use will be useful. Critical elements of a good editor include:

- syntax highlighting
- auto-completion of common commands
- line number visualization
- project organization

Some editors are more useful than others depending on the language used. Additional plugins may be available that will speed up and minimize mistakes.

For example, individuals using Chef may find that an editor like [Sublime Text](*http://www.sublimetext.com/*) is very helpful with the [SublimeChef](*https://github.com/cabeca/SublimeChef*) and [SublimeGuard](*https://github.com/cyphactor/sublime_guard*) plugins. SublimeChef includes several code snippets that autocomplete common chef resources which leads to faster development. Including SublimeGuard allows for continuous testing through showing output of [guard](*http://guardgem.org/*) within the editor ensuring that anything written complies to style and correctness early in the process rather than testing done during the automated test and build environments.

Version Control

Every organization should implement, use, train, and measure adoption of tool usage of version control. It gives teams the ability to deal with conflicts that come from having multiple developers working on the same file or project at the same time, and provides a safe way to make changes and roll them back if necessary. Using version control early in your product's development will facilitate adoption of good habits.

When choosing the appropriate tool in your environment for source control management, look for one that encourages the collaboration in your organization that you want to see.

Qualities that encourage collaboration include:

- opening and forking repositories
- contributing back to repositories
- curating contributions to your own repositories
- defining processes for contributing
- sharing commit rights

Some tools may lack the collaborative features but have inherent system knowledge within your environment due to long term use. In these cases identify the impact to not migrating for example hiring capability. With sufficient process, collaboration can still be implemented but it will not be as easy.

There are different types of repository models that version control systems can have, local, centralized or distributed.

In a local model, the repository is located on the node that is storing the files. The easiest to set up, it is also the most vulnerable to system failure.

In a client-server model, there is one main or central version of the repository, located on the server, and developers do their work locally, commit changes to the server in order to make those changes visible to other developers. Because clients do not store the entire repository locally, a centralized model can be useful for repositories that contain many large binary files, where storing local versions of every version of every file would quickly become costly for every client. On the other hand, the server becomes a single point of failure.

Distributed version control systems do not rely on a central server, and every client that copies or "clones" the repository has a full copy of it, including all metadata. Since most software projects are made of text files, and storage has become cheap, the overhead of storing large files is not as costly as it was decades ago. One advantage of having a full local copy is that all actions aside from pushing and pulling can be done offline, allowing developers to work without an internet connection - a great thing if

you have employees who want to work offline to focus or travel a lot. Developers can also spend time collaborating on changes together without pushing these changes to everyone, because everyone has their own local copy.

Metrics for Success

Lines of code is not an accurate measure of value. There are different types of developers, some that refactor hundreds of confusing lines into tens of lines of simple to read abstractions that can be built upon by others in the team. Others that focus their attention in finding the bugs hidden within code. Use quantitative measurements as informative trends to encourage the behaviors you want to see. For example, unless you have the skill to qualitatively examine code don't assume that more is better.

Artifact Management

An artifact is the output of any step in the software development process. Depending on the language, artifacts can be a number of things from jars, wars, libraries, assets, to applications.

An artifact repository should be

- secure
- trusted
- stable
- accessible
- versioned

Having an artifact repository allows you to treat your dependencies statically. You can store a versioned common library as an artifact separate from your software version control allowing all teams to use the exact same shared library. You build binaries once and only once (even though you could build the same binary again). This helps alleviate complexity by ensuring that the same binary is used throughout the test cycles and promotion between builds.

Artifact repositories allow you to store artifacts the way you use them. Some repository systems only store a single version of a package at a time. This can lead to problems describing the history of packages, increase the duplication factor of package storage to maintain a separate artifact repository per environment in your workflow.

When using an open source operating system like CentOS or Ubuntu, the default package manager will use external package repositories. This can lead to instability if depending on this external package repository. Network links being down, issues

with systems at the remote package store can lead to build issues if your systems rely on these external resources being available.

Beyond the OS more and more software dependencies and applications from vendors rely on downloading software from external providers like rubygems.org for ruby gems. In you environment you need to identify what the risks are and how you want to mitigate these risks. People generally trust external providers because the software works, other folks are using the software, and it's the easiest way to get software working.

Audit Code

Before putting external code into production, make sure that you have a process in place to audit and verify third party packages are not vulnerable.

Relying on third party packages? Ask for information about what steps they have in place to generate secure code, and store their binaries. Companies that care about the quality of their code will provide you this information.

While you can use version control to store binaries, this is not always an optimum use of resources. Developers don't modify binaries directly, so cloning a repository with a large number of binaries can affect bandwidth, as well as build times.

Table 5-1. Recognize Common Artifact Types

Purpose	Type
Linux	rpm,deb
Java	JAR, WAR
Ruby	gem
Windows	DLLs
General	tarball,zip files

While small, the need to pass certain security compliance requirements may not exist in your environment. As you grow, and pivot along product lines this may become a requirement. Having a dedicated local artifact repository allows you for a much smoother transistion to these requirements.

Ideally, your local development environment has the same access to your internal artifact repository as the other build and deploy mechanisms in your environment. This helps minimize the "it works on my laptop" syndrome because the same packages and dependencies used in production are now used in the local development environment. If the access is limited or blocked, this friction can lead to new ways of doing things that circumvent security and other policies.

Define Policies Early

Establish governance processes early to promote collaboration within the context of your environment and constraints. For example, identify who can push what artifacts, how are artifacts vetted, licensed, and secured. This will alleviate growing pains of out-of-date artifacts.

If you don't have access to the internet within your environment, you need to host your own universe. This includes software repositories, (ruby) gem servers, dependency management, and more. A lot of available shared services must be replicated.

Infrastructure Management

Infrastructure management may seem outside of the scope of the foundations of a strong tool ecosystem within your environment as it is just part of identifying the compute resources you need, and the function they have within your environment. Yet, these elements are the foundations of your company's health from email as the communication portal between internal and external individuals to the website that informs and educates about your product and possible services.

When IT is seen as a cost center this leads to underfunded and under-resourced operations teams. Single person ops teams supporting small companies may be the only individual involved in the selection of a tool that can ease the administration of all the systems that connect the company. This can lead to a very short sighted implementation as the person tasked with this enormous responsibility is the one most overwhelmed with ensuring that the systems and services operate at their best capacity.

Minimum Operations Team Size

As mentioned in the Hiring chapter, ensure that each team meets a minimum functional size. For Operations teams, the quorum is 3. Issues happen, life disrupts work. For small companies, this may mean that until you reach the size that allows for additional headcount that individuals share multiple roles. If you only have one person responsible for infrastructure automation, you will incur technical debt that will only grow as your team grows. One sure sign of a problem is the presence of a single subject matter expert that everyone goes to every time there is a problem. These single subject matter experts are single points of failure in your infrastructure.

Fundamentally you should be able to provision elements of your infrastructure through code, treat this code just like the rest of your software, and recover your business through data backups, code repository, and compute resources. This is known as *infrastructure automation*. "Treat infrastructure code like the rest of your software" means the code developed using a common local development environment, versioned in version control, versioned artifacts in an artifact repository, tested, and verified before being put into production.

Infrastructure automation allows the definition and control of how a given system is set up to be described in code, from the system settings to the programs that are installed and running to user management and network configuration. Benefits of infrastructure automation include repeatable, consistent, documented, auditable and resilient processes that withstand some amount of failure. This frees up time, improves efficiency of staff, allows for more flexibility, and risk measurement. It also increases the degree of confidence that individuals have in the machine setup and deployment being identical reducing the amount of time spent debugging problems based on system differences.

Repetitious Work

Manual repetition of task driven work like infrastructure configuration can be a contributory factor to burnout. See the hiring chapter for more information on managing burnout.

Contrast infrastructure automation with an environment where a person must repeat a series of manual steps by hand on every single one of a group of many servers. Humans performing repetitive tasks lead to mistakes. Systems might be configured inconsistently due to a change in process not configured on older systems or a step in the checklist of manual steps could be missed. The solution isn't to institute more process and checklists, but to ensure that sufficient time is allocated to translate these

manual checklists into computer executable scripts. Computers are much better at repetitive tasks than humans are.

Even if this is a one time system, implementing in code is one step of disaster recovery. When one person deploys a system there is knowledge generated that creates a single point of knowledge (SPOK).

Establishing a common method of describing infrastructure in code across your teams so having a infrastructure as code system reduces the number of "special snowflake" servers that exist - servers that were set up by hand one at a time, often becoming so unique that others are wary of making any changes to them for fear of irreparably breaking whatever long-forgotten incantations were typed to get them there. This also allows for a higher server-to-people ratio because the time to configure and maintain each individual server is significantly less.

A Site Configuration Engine

With the proliferation of systems, and the inherent complexity of managing the configuration and state of systems, Mark Burgess shared **cfengine** with the community through a paper titled "A Site Configuration Engine". He proposed mechanizing a tool to systemize administration and configuration of a system. Cfengine was the early prototype encapsulating the ideas behind **infrastructure as code**.

Finally, infrastructure automation makes it easier for more people to understand and be able to perform the setup of machines necessary for them to do their jobs. No longer is bringing up a test server some arcane ritual only understood by a few system administrators. By abstracting away many of the details, infrastructure as code systems allow developers and other non-operations engineers to gain a high-level understanding of the systems they need.

With there being so many tangible benefits to using infrastructure as code, it makes sense that it would be one of the first tools that companies pursuing a devops initiative would look into and need to decide upon. As mentioned earlier, tools can only be understood in use. Depending on the environment, the specific culture and beliefs of the environment can impact the efficacy of the tool. Which infrastructure automation works best for you will depend on your specific needs.

Even as a small startup with small number of systems, it's absolutely critical to not build technical debt through the creation of automation through specially handcrafted snowflake systems. Investing in individuals with operational skills that understand the difference between snowflake shell scripts and infrastructure automation will make the difference in whether you are spending cycles on specializing outside of the area you are competing in. Even if you are contributing software and tools back to the community to expand the fundamental features of software that exists in this

space is better than architecting and maintaining systems that provide infrastructure automation at the core of the system.

As companies grow, complexity increases through additional compute resources, staff, software and services. The more people interacting with those compute resources leads to more interesting boundaries and edge cases. As an organization crosses the threshold from startup into a well-established company (or absorbed into another company with their own automation standards), the effectiveness of managing infrastructure becomes critical, and so do the qualities that an effective solution needs.

These qualities include:

- **Management of configuration drift**: Configuration drift is the phenomenon where servers will change, or drift, away from their desired configuration over time. This can be due to manual changes, software updates or errors, or entropy. A good solution will have a way of preventing this, often by having an individual node regularly check the desired configuration against its actual configuration and self-correcting any inconsistencies.
- **Elimination of snowflake servers**: A snowflake server is one that has gotten to its current, desired configuration by way of many manual changes, often requiring a combination of command line sorcery, configuration files, hand-applied patches, and even GUI configurations and installations. Snowflakes are difficult to manage and would be quite difficult to set up again if they were to have some kind of hardware failure. Infrastructure automation solutions can avoid creation of snowflake servers by making sure all changes are clearly and deterministically defined. These servers can also be eliminated by applying configuration management to snowflakes in small batches, adding management to one piece of the system at a time until the same configuration management recipe can be used to recreate the server from scratch in its desired configuration.
- **Versioned artifacted infrastructure code**: A good infrastructure automation solution will tie in nicely to a version control system and artifact repository. This ensures the code that defines the server configuration can be versioned with all the benefits that come from that, such as being able to easily roll back changes to a known good version, or have post-commit hooks that run test against the infrastructure-defining code. It's also a familiar process such that all team members can contribute towards improving the infrastructure code as they feel comfortable with the process.
- **Minimizing complexity**: By specifying specific version of configuration per platform type or version, infrastructure automation solutions should allow individuals despite their official title to manage a heterogeneous environment with a minimum of overhead.

Moving Beyond the Basics

These fundamental tools are so essential because it would be prohibitively difficult to work effectively without them. Any type of operations without infrastructure automation would mean significant amounts of extra work for operations staff preventing, checking for, and mitigating configuration drift and snowflake servers, and development work without version control is pretty much asking for trouble when work gets lost and a bad set of changes goes live and has to be rolled back. And communication is present in every part of any job that involves working with other people at all.

Beyond the basics there are several other types of tools that often come up in the process of creating, deploying, and running software. The absence of these other types of tools might not be as readily apparent as trying to work without version control, for example, but they bear mentioning for how they can, if used effectively, also help negotiate the compact.

Sandbox Automation

A sandbox is a testing environment that allows an individual to test code changes, and experiment with different infrastructure elements without impacting production. Sandbox automation is the process of encapsulating the definition of a sandbox so that an individual can quickly replicate or share the specifications that make up a sandbox.

Test Kitchen is an implementation of sandbox automation that can run on a individual's laptop and integrates with a number of different cloud providers and virtualization technologies including Amazon EC2, CloudStack, Digital Ocean, Rackspace, OpenStack, Vagrant, and Docker. It has a static configuration that can be easily checked into version control along with a software project.

Work Visualization and Planning

One of the key elements of Lean manufacturing, as discussed more in-depth in the "What is Devops" chapter, is the idea of just-in-time production as a way of limiting the manufacture of parts to only what was needed for the current batch of whatever final product was currently being produced. This eliminated waste by not having surplusses of materials being created and sitting around needlessly, which helped to streamline the overall manufacturing process. This concept of limiting the amount of work in progress to a manageable amount has also carried over into Lean software development, helping to ensure that development teams don't take on more work than they can reasonably expect to finish.

Most organizations use some kind of ticketing or bug-tracking system to keep track of the work that they are both currently doing and planning for the future. The best systems will include some kind of visualization tool as a way of representing work.

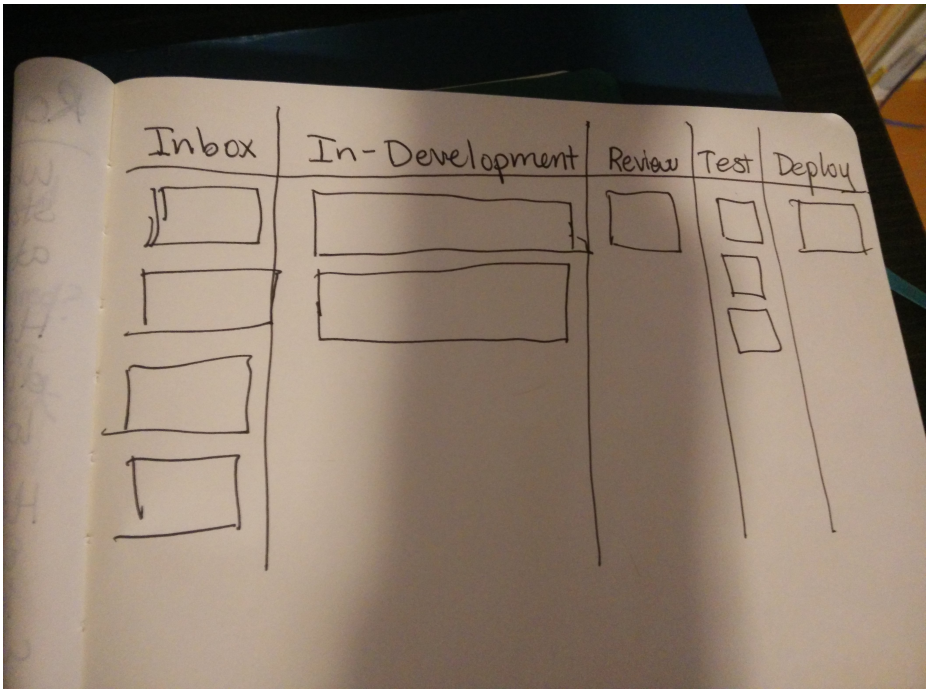

Figure 5-1. A Visualization of Work in Progress

Visualizing work in this way allows both individual contributors and managers to keep an eye on how much work a team or group is currently doing, so they can adjust their work as needed as time goes by. Having this information easily accessible is key to making this work. Jira, for example, allows users to create dashboards that have various visualizations of work broken down by status, team or individual, due date, and so on. Trello, while not a full-featured bug tracker, has the advantages of being very affordable and mobile-friendly.

Kanban

Kanban, which also originated with Japanese manufacturing processes, is a way of visualizing and managing work while avoiding overloading the people doing the work. A Kanban board has "cards" for each unit of work, and a column representing each state the work can be in. For software development, this might include states such as planning, in development, under code review, sent to QA for testing, and deployed to

production. Many teams will use a Kanban board, either physical or digital, as a way of managing and visualizing their work. If you find your teams are constantly under too much stress or it seems like features never make it all the way to completion, visualizing work in this way can be an excellent starting place.

Some work visualization tools will provide ways of automatically limiting the sizes of different work states, only allowing a specified number of items to be designated as "in progress". Teams may also decide to limit the size of their inbox or backlog, using this as a way of making them prioritize which work is most important. This automation can be helpful for teams just starting out with work visualization and planning, instead of people having to remember their limits themselves. These limits can of course be changed over time - adding a couple new team members will likely mean an increase in the amount of work in progress once the new employees get up to speed.

Being able to move work in between teams or projects can also be very important. This kind of flexibility allows for more collaboration between teams, especially if the tools allow for easy in-place communication of why something is being moved. Letting different teams see what each other are working on can also lead to collaboration, or at the very least avoid unnecessary duplication of work.

Deployment

Deployment was often a point of contention back when software development teams who wrote the code were siloed off from the system administrators whose job it was to deploy and maintain the code in production. Originally, deployment might have involved a system administrator taking the code, on some kind of physical media, to a server or mainframe and installing it by hand - not only a slow and cumbersome process but one that was likely to be error-prone as well, as manual processes tend to be. Modern deployment tools come in a variety of forms today but the common theme is that they try to automate the process as much as possible to cut down on time spent and the possibility of errors.

The software deployment process will obviously vary based on what kind of software is being deployed. Code that runs on very small systems embedded into other devices such as printers or televisions will be deployed differently than a mobile application (which has to go through the mobile operating system's application process) will be deployed differently than a website. The key pieces of a robust deployment system include:

- Clearly defined steps. To get started with setting up a deployment tool, you'll want to specify all the steps required for a deployment. It's a common saying that you can't automate something until you can define it, and that is certainly true for deployments. Once all the steps of the process are clear, it becomes much

more straightforward to start automating pieces of the process. Documenting the steps required also makes it easier to bring new employees up to speed on the process as well as troubleshoot if something goes wrong.

- Plenty of error checking and error handling. With as important as deploying code is, the last thing you want is for the process to fail but nobody find out about it until it's too late and there are customer-facing impacts. A good deployment toolset will check for possible errors every step of the way (defining the problems that can arise is an important part of defining the steps of the process) and make those errors visible. Part of this is having a way to back out of a deploy if an error occurs in any part of the process to avoid having a bad change being unstoppably deployed to production.
- Minimal overhead. A long, cumbersome deploy process means that deploys will happen less frequently, and the more intervention required on the part of an operator means the more likelihood there is of an issue occurring. While it might not be possible or desirable to automate the entire deploy process, the more steps that can be done without human intervention, the smoother the process will be.

As we will see in this chapter's case study, there are reasons to use an existing deployment tool or to create your own.

Monitoring

Monitoring is how you keep tabs on what is going on with your product and infrastructure. Infrastructure monitoring, taking the form of checking on the states and statuses of server hardware, operating system processes, and network functionality can be taken care of by tools like Nagios, Cacti, and Ganglia. Application-level monitoring is often more complex as it depends on the specifics of your application.

Organizations taking advantage of cloud offerings such as Platforms as a Service, Infrastructure as a Service, or anything else that runs outside of their own infrastructure will often have to rely on those providers for detailed monitoring. From their end, they will be able to see whether or not a provider's service is functioning or not, and possibly get some error codes, but more detailed monitoring will likely be unavailable. You will have to decide whether or not you have the bandwidth to develop and maintain your own in-house monitoring, and whether or not the additional detail and customization you can get from that is worth the overhead.

Tools such as statsd and graphite allow developers to choose what metrics they want to monitor in those applications and graph those metrics with relatively little overhead. Monitoring should not be considered the sole purview of an operations department, something that only system administrators worry about once software has been deployed in production. It is something that should be considered from the beginning of product development, so choosing a tool that developers can easily use is vital.

Alerting

While alerting tools used to be limited to the ability to send emails that was built in to monitoring systems such as Nagios, or later to send text messages with the advent of email-to-SMS gateways, tools are being developed these days to make more featured-rich or easy to use alerting systems. Whereas using something like Nagios's built-in email feature usually requires running your own mail server as well, services such as PagerDuty or VictorOps will handle sending emails, SMSs, and even push notifications to their mobile applications or making phone calls for you.

One of the other benefits to a more robust alerting solution is that of schedule and rotation management. In previous decades, when an organization might only have one system administrator who handled all on-call responsibilities, this was much less of a concern. These days, with larger and more complex organizations, as well as with web apps that are expected to be available 24/7, organizations are likely to have one or more teams that share in on-call rotations. An application such as PagerDuty that allows for easy scheduling, whether you want your shifts to last multiple hours or multiple days, and takes care of sending the correct alerts to the correct people automatically, can cut down on a lot of manual work.

Again, using an external service means that you are relying on someone else for part of the reliability of your infrastructure, so again you will have to consider the tradeoffs and decide what makes the most sense for your organization, but we've found most alerting providers to be quite reliable and well worth getting rid of having to manually set and change alerting schedules by hand. And as we've discussed in the Hiring chapter, burnout is a serious concern, so in all likelihood you'll want to have at least a few people on your on-call rotation to avoid that, making an alerting solution even more attractive.

Logging

Logging is of great importance to organizations - it can be incredibly helpful when troubleshooting application issues, but can also be necessary for compliance regulations that require keeping records or audits for certain periods of time. When considering a logging solution, there are several factors to consider:

- **Retention**: Referring to how long you want to keep logs around, this will likely vary for different parts of your infrastructure. Server or application error logs can be helpful in debugging, but might lose their usefulness not too long after incidents occur, so keeping them for many months or years might not be a great use of storage space. Logs of financial transactions you are likely interested in (or required to) keeping them around for longer. The tradeoff with the convenience of having logs still stored to search through is that the storage space required has costs that will have to be considered, whether it goes to a cloud provider or is disk space and power in your own data center.

- **Search**: Generally one of the points of storing logs is to be able to search through them for useful information at some later point, so ease of searching is often key. You will have to decide which fields are likely to be searched enough that you will want to index them, with the consideration that indices also require storage space. The ease of the search interface for people, especially less technical people, to use is also a key factor. More advanced search features are one of the reasons that search solutions like Splunk are so popular.
- **Cost**: We've mentioned costs in relation to storage and power usage, but if you are using an external logging provider, there will be a monetary cost as well. Enterprise solutions like Splunk, being very robust and full-featured, tend to be quite pricey as well. Cloud-based solutions such as Loggly or LogEntries are becoming more popular, and offer usually much lower-cost pricing solutions based on log volume and the length of your retention period. There may be a cost for a support contract as well.
- **Maintenance**: If you are running your own logging solution in-house, you will be paying for it more in engineers' time and energy as opposed to simply writing a check every month. If there are bugs in the software you use, you'll likely have to wait for an upstream fix or find one yourself, though some software companies do have support contracts available. You'll own your own availability and customization, but you'll also be responsible for maintenance and capacity planning.

Many companies will find that the best logging solution for them will change over time as their organization size and requirements change. A larger company might find that they have more engineering time available to maintain an internal solution, while a smaller one might find that the pricing of a logging-as-a-service makes the most sense for them.

Optimization: Selection and Elimination of Tools

Many of the factors to use when analyzing your tool usage are common across the types of tools that you have identified within your environment.

Some selection factors are common across tool types when analyzing tool usage within your environment. These factors include:

- Product Development
- Community Health
- In-House Customization

Product Development

An actively developed product will be quicker to get new features, support newer operating systems and platform versions, and deal with any security vulnerabilities.

Community Health

An active community can be even more beneficial. As discussed earlier in this chapter, one of the benefits of open source software is that you don't have to reinvent the wheel by coming up with your own solutions to problems that other people have already dealt with. An open source solution with a strong community contributing to it can make its implementation even more effective.

In-House Customization

A tool that can be easily customized and contributed to will make for a robust solution that is well suited to both the technological and human aspects of an environment. This is especially important in organizations with a large number of people working with the tool. A tool that deals well with that kind of scale is one that will be able to grow along with your organization, as well as making engineering work easier.

Version Control

To demonstrate some of the differences that you might consider when choosing a version control system, we will look at two examples: Git, which is a distributed system, and Subversion, which is centralized. We chose these for our examples because they are two of the most popular version control systems being used today, and because they are free and open source tools that can be used on Linux, OS X, and Windows. Unless you have very specific requirements (the ability to cache objects that were built by other clients instead of rebuilding them locally is included in very few version control systems, for example), we would likely recommend picking one of the more well-known open-source systems, both because of the benefits of using open-source software that we described in Chapter 1, but also because it will make it more likely that developers will be familiar with your environment.

It might be tempting to hire engineers based in part on which version control system(s) they are most familiar with. However, familiarity with a tool doesn't guarantee that someone can use it effectively - more than digging into nitty-gritty technical details of your tool of choice, we would recommend having candidates speak to the workflows and use patterns that they liked and disliked as well as their reasons why. Also keep in mind that engineers don't always have a say in what tools they worked with. Your team should be able to teach engineers the new skills and best practices they need to be successful.

Let's take a look at some common version control workflows:

Example 5-2. Creating a Repository with Git

```
cd /path/to/project
git init
```

Creating a repository with git This will create a repository in the current directory, which can then be cloned by other developers by using a command of the form `git clone git@server:/path/to/project.git`. This command doesn't have to be run on any particular server, because Git is distributed. The repository can be created anywhere and then cloned as needed.

Example 5-3. Creating a Repository with Subversion

```
cd /path/to
svnadmin create project
```

Because Subversion is a client-server model, this command must be run on the central SVN server, not on any machine that is going to be a client.

Making and Committing Changes

Example 5-4. Making and Committing Changes with Git

```
git clone git@server:/path/to/project.git
cd project
vim file1
touch file2
git add file1 file2
git commit -m "Message describing your changes"
git push
```

This shows how to make, commit, and push changes back upstream with Git. Cloning the project is how a developer creates a local copy of the repository - again, because Git is distributed, this creates a local copy of the entire repository, including all the project metadata. Left out of this example is configuring where the code gets pushed to. Git's distributed nature allows users to define any number of remote respositories. They could push their changes to another developer, to a local git server, or to something like Github - part of the flexibility of Git is that it allows for the use of a central server, but unlike client-server version control systems, it does not require it. Pushing the changes pushes the differences between your local copy and the repository being pushed to, so commits containing big binary files will get quite large.

Example 5-5. Making and Committing Changes with Subversion

```
cd /path/to/project
svn checkout svn://server/project .
```

```
vim file1
touch file2
svn add file1 file2
svn commit -m "Message describing your changes"
```

With Subversion, there are slight differences in how we need to think about our workflow. First, subversion is going to ensure that all of my files are up to date. It compares my working copy with the latest revision of the project in the repository. The working copy is the directory of versioned files along with metadata about the files. If any of my files are out of date, they're automatically converged to the latest version. After editing the files, we schedule the changes to be uploaded and added to the next commit. Finally we commit our changes back to the repository as a single atomic transaction with a log message. Either all of the changes are accepted or none of them are.

Up to this point, you may notice that both of these examples look fairly similar - and for basic workflows, they do in fact behave very similarly. It is when we start to get into more complex and collaborative workflows that the differences between these two types of tools become more apparent. Recall from Chapter 1 the idea of the compact and working towards shared mutual understanding, both of the individual and shared goals that people of teams have, and of the terms of the compact itself. A more explicitly collaborative workflow will make these understandings more explicit.

Branching a Repository and Curating Contributions

Let's consider an example with George and the General. The General has a project that she is working on, and George has some ideas that he wants to contribute. He and the General haven't worked together on this project before, and his changes are pretty substantial - he'll want to make sure that he can develop and test them thoroughly before committing them to the main project, to avoid breaking things for the General and anyone else who might be using this project.

First, George is going to create a feature branch, then make all of his changes.

Example 5-6. Branching a Repository with Git

```
cd /path/to/project
git checkout -b georges-feature master
vim file1
git add file1
git commit -m "file1 in george's feature"
git push -u origin georges-feature
vim file2
git add file2
git commit -m "file2 in george's feature"
git push
```

First, with the `git checkout -b` command, George creates a feature branch called `georges-feature` based on the master branch of the General's project. This branch gets created on his local machine, and because this is Git it contains a full copy of the entire repository. He makes some changes, commits them, then pushes them upstream for safekeeping. Until he has pushed, the files and their changes live only on his local workstation - it is only with the push command that they exist on the server at all for other people to be able to see them. The `-u` flag adds his branch as an upstream source so that he won't have to specify which branch to push to on subsequent commits, as is seen with his commit and push of his changes to file2 - this is very convenient on projects with many many commits.

Now that he's done with his work and ready for it to be merged into the master branch, which the General maintains, let's take a look at her workflow.

Example 5-7. Merging Contributions with Git

```
git checkout master
git pull
git pull origin georges-feature
git push
```

Once she's looked at George's changes and decided she's ok to merge them, the General checks out her master branch and pulls, just to make sure she's got the latest version of her code checked out. She then pulls the branch `georges-feature` from the origin of the repository, where we saw he pushed it after he was done with his changes. This command merges his branch into the master branch that she has checked out. Finally, she pushes the updated master branch back to the origin, so any other contributors to the project can get these changes.

With Github, a web-based Git hosting service, this can be done via pull requests in the web UI. This allows developers to request that their branches (or forks, if they are using Github's fork feature) be merged with a couple clicks of a button, and offers things like color-coded diffs for easy comparison.

Now, let's look at how this workflow would look if this project were managed with Subversion.

Example 5-8. Branching a Repository with Subversion

```
svn copy svn://server/project/trunk svn://server/project/georges-feature
cd /path/to/project
svn checkout
svn update
svn switch svn://server/project/georges-feature .
vim file1
svn add file1
```

```
svn commit -m "file1 in george's feature"
vim file2
svn add file2
svn commit -m "file2 in george's feature"
```

With Subversion being a client-server model, the `svn copy` command copies the repository to a different branch on the remote server. At this point, the branch exists only on the server, as does trunk. The `svn switch` command is what switches the environment so that George's changes will be applied to that branch. When he makes his changes and adds them, those changesets exist locally, but when he does the commits, each commit applies those changes to the server, as happens with the client-server model. You'll notice that there is no comment of having local commits that only get pushed upstream later the way there is with Git.

Now, on the server, the General can merge George's branch into the trunk.

Example 5-9. Merging Contributions with Subversion

```
cd /path/to/project/trunk
svn update
svn merge --reintegrate ^/project/georges-feature
```

As these last example workflows show, the distributed Git version control system allows for many more collaborative efforts than the client-server based Subversion. It is certainly possible for developers to collaborate using something like Subversion, but as you can see there is a lot more friction. One of the goals of an effective configuration management tool is to reduce the frictions that get in the way of engineers doing their best work, and that means looking for and using tools that make working together easier instead of getting in the way.

Two key parts of the idea of devops as a compact are defining boundaries and repairing when there are conflicts, both of which can be emphasized with the right version control solution. Because branching in a distributed version control system like Git is a first-class consideration, it is easier to both create forks or branches for experimentation or feature development and merge different branches together in a DVCS. Engineers can more easily define who can contribute to their projects and which changes they want to accept. It might not be worth the effort of migrating existing projects to a different version control system, but for new projects, a distributed option has much more strength and flexibility in terms of the collaboration it allows.

Infrastructure Automation

Most of the established infrastructure automation solutions will be similar in terms of overall functionality even though their implementations differ. As with all of the tool categories, each tool may reduce or encourage different aspects of collaboration.

As an example, we will illustrate the more advanced workflows that are possible with Chef and some of its plugins. This workflow could be improved further by keeping the cookbooks in version control. We have chosen to focus on the elements of infrastructure automation.

The General and George are both working on their organization's apache cookbook, which defines the Apache web server configurations for their website. Using the Chef environment abstration, they have defined *environments* that reflect the workflow in use within their organization. This allows the team to define the configuration in production, while also working towards the next release of software that could include new Apache modules. This ensures that new production systems can be deployed that are identical to current existing systems without introducing new systems with the development version of configuration.

The production environment will have specification of versions of cookbooks to be used until they decide to change it. This means anyone can experiment with the cookbook in different versions in the development environment. With George and General both updating the Apache cookbook at the same time, they might run into version conflicts. How can they most effectively resolve, or even better, avoid these conflicts? They use a community developed plugin, *knife-spork* that allows for them to avoid cookbook conflicts.

Example 5-10. Avoiding Cookbook Conflicts with knife-spork

```
> knife spork check apache
Checking versions for cookbook apache...

Local Version:
  1.0.0

Remote Versions: (* indicates frozen)
 *1.0.0
 *0.9.9

> knife spork bump apache
Successfully bumped apache to v1.0.1!

> knife spork upload apache
Freezing apache at 1.0.1...
Successfully uploaded apache@1.0.1!
```

When George makes his changes, he uses `knife spork check` to check what version of the cookbook is on the server. This shows him that the server has version 1.0.0, which is the same version he's been working on which indicates a potential conflict.

The Chef server is an artifact repository that provides the feature of *freezing cookbook artifacts*. Freezing ensures that an artifact that is uploaded with a specific version can

not be overwritten with a second artifact that is different. This ensures that if you deploy version 1.0.0 into production, 1.0.0 will always have the same elements.

Prevent re-use of version identifiers

Elimination of re-use of version identifiers will eliminate some unexpected side effects. With Chef this means freeze your cookbooks. While it may seem like a quick fix is sufficient to re-use version identifiers, this can introduce unexpected problems due to other software misidentifying the need for applying a change.

The organization has taken advantage of this feature to freeze the cookbook artifacts, where the server will not overwrite the same version of a cookbook with different contents - this helps prevent unnoticed changes from causing problems. George uses spork to update the cookbook locally to a new version, 1.0.1, then uploads it to the Chef server. His new version is automatically frozen by knife-spork to prevent changes being unknowingly overwritten.

What happens when the General goes to push her changes to the Chef server? Without this plugin, she might inadvertently push her changes over George's, especially if he hadn't manually frozen his new cookbook version. With knife-spork, however, her workflow goes something like this:

Example 5-11. Avoiding Cookbook Conflicts with knife-spork

```
> knife spork check apache
Checking versions for cookbook apache...

Local Version:
  1.0.0

Remote Versions: (* indicates frozen)
 *1.0.1
 *1.0.0
 *0.9.9

> knife spork bump apache
Successfully bumped apache to v1.0.2!

> knife spork upload apache
Freezing apache at 1.0.2...
Successfully uploaded apache@1.0.2!
```

Here, `knife-spork check` discovers George's 1.0.1 version of the cookbook on the Chef server, even though he hadn't told the General that he was working on the cookbook at that time. It then bumps the version to 1.0.2, avoiding any conflicts with George's version. Knife-spork plugins can also be used to automatically announce

these changes in a chatroom so other developers can easily see them without much in the way of extra effort on either end.

This workflow can easily be repeated and defined by continuous integration software which lends itself to the build and release of cookbook artifacts that follow standard software practices. This means that if a member of an external team wants to contribute to the cookbook, and only knows about the version control processes in place they can still provide meaningful contributions. The continuous integration software would be the process that did the knife-spork workflow after tests passed.

People wanting to take full advantage of the benefits that their current infrastructure automation can offer would do well to read a more in-depth book such as Alessandro Franceschi's *Extending Puppet* or Jon Cowie's *Customizing Chef*, both available in the O'Reilly online store.

If you are in the process of choosing a specific solution, make sure that you have identified the elements in your environment that will reflect your workflows to fulfill your organization's configuration and collaboration needs now and as your teams grow.

Artifact Management

Artifact management can be as simple as a web server with access controls that allows file management internal to your environment to a more complex managed service with a variety of extended features.

Much like early version control for source code, artifact management can be handled in a variety of ways based on your budgetary concerns. Generally an artifact repository can serve three functions:

- central point for management of binaries and dependencies
- configurable proxy between organization and public repositories
- integrated depot for build promotions of internally developed software

When choosing between a simple repository and more complex feature-full repository, understand the cost to support additional services as well as inherent security concerns.

Work Visualization

To demonstrate work visualization, we will use kanban to illustrate the process of selecting and implementing a work visualization.

The essentials of implementing kanban within an organization include a kanban board, cards, some mechanism to flag, and another mechanism to individuals who are currently working on a specific task.

You can implement kanban virtually or within the physical world. In initial planning it is useful to try out different steps physically to see the impact and value of the board layout.

The kanban board is generally a big white magnetic dry erase board. You can use any other surface on which you are prepared to create markings through the judicious usage of tape. You need sufficient room to attach additional meta-information about the board and its lanes.

The cards are generally sticky notes of varying colors and sizes. The colors generally indicate different types of tasks. For example a green sticky note could represent all user story tasks and a yellow sticky note could represent all feature tasks.

The flags are generally sticky flags or circle stickers of varying colors. The colors generally indicate special information about the task such as the task having a specific start date, or end date.

The mechanism to identify individuals assigned to a card is generally obtained by creating an avatar of the user either through a cartoonish figure or a picture.

The key principles of kanban are:

- **Visualize** Make work visible, identify work queues and potential bottlenecks.
- **Limit WIP** Each individual increases their focus. Priorities are set, and load is managed.
- **Flow management** Flow is a metric of productivity. Smooth flow makes work more predictable.
- **Explicit Policies** Document each implicit assumption. What do cards and flags mean, how do cards move on the board.
- **Implement feedback loops** Continuous improvement is driven by sufficient feedback loops that inform behavior management.

The first step in the adoption of kanban is to get everyone in the team aware of the basic principles of kanban. There are a number of workshops at different conferences, or customized trainings available. Dominica DeGrandis is one of the key authorites on work visualization with kanban. Along with providing kanban for devops training, she has created the Kanban for Devops game that is available for play at a variety of DevOpsDays.

Once the team has an initial idea of the terminology and premise of kanban, the next step is to get the team together to discuss their current workflow. Prior to the meeting a few indivdual roles should be assigned: champion, gatekeeper, and recordkeeper.

The champion will be the individual responsible for ensuring that the team makes it to the finish line, whether that is with a successful implementation or selection of a different process of work visualization.

The gatekeeper should be assigned with the responsibilities of ensuring everyone has the opportunity to speak up in the meeting, that no one person dominates the whole conversations, and that the team discusses the current state of work and not the ideal state.

The record-keeper should also be assigned with the responsibilities of ensuring that the end result is accurately recorded.

During the meeting, the team discusses the workflow, i.e. the process of a specific request coming to the team to complete. Sample boards are drawn out with columns reflecting the different stages a specific task flows through.

Figure 5-2. Workflow Visualization

It's really important that the team have a clear and common understanding of what each state means. Each of the states needs to have a clear boundary with the exit and entrance conditions defined. In the above diagram, there are 4 states. The words used to describe the states could be interpreted differently depending on the individual. What does "Accepted" mean as a state in this workflow?

Giving further definitions helps clarify intent and get everyone on the team to a common understanding. The "Accepted" state is when the team understands the work that has been requested and the complete state known. The "WIP" state is when a team member has started work and clarified the goals of the specific task. The "Done" state is the completion of understood tasks. The "Verified" state is the check for completion, external from the work done.

The team needs to categorize task types, common task sizes, task states, and how tasks move among states. Key to this meeting is that everyone gets the opportunity to express their feelings about the workflow and to get down the process that is currently occuring.

Once the current process is documented clearly, everyone agrees to the process of incremental evolutionary change. All roles will be respected for now, but as process evolve new roles may be assumed based on improvements in the environment. Central to positive change is that everyone is a leader and has the capacity to pull tasks that they feel they have the skills and resources to complete.

Sometimes it is helpful for management to facilitate the meeting and then leave to ensure that individuals speak candidly about the work environment. After the meeting, the manager should take the notes from the meeting and follow up with individuals to ensure that any additional feedback is captured.

Every sticky note represents a task that a team or an individual has to perform. You simply write what has to be done on the sticky note (name the task). Sticky notes can be of different colors for different types of tasks (User story, feature, defect …). On every note, you can also write additional data, like the estimated scope of the task in hours, a unique task ID, task owners and other information.

After you have the board and sticky notes, you simply stick the notes in one of the columns, depending on the phase the task is in. Now you have a nice visual representation of what needs to be done, the works in progress and tasks that are being completed.

Little's Law

Fundamental relationship between work in progress, cycle time and throughput.

Cycle Time = Work in Progress (WIP) / Average Completion Rate (throughput)

Metrics

Some things that you might want to monitor have simple yes/no answers, such as is the site currently up. Most things that you'll want to measure will have some numerical value that you'll want to keep track of instead - metrics allow you to measure these

things. Metrics are most often gathered and stored in a time-series database so that the changes to the metric over time can be easily tracked and reasoned about.

Analyzing metrics in order to get useful information out of them usually requires a fair amount of contextual knowledge. If your traffic patterns vary based on time of day or time of year, what is normal at one point in time might be problematic at another, and new employees without this context or background information might not be able to make decisions in as informed a manner. This is why software to detect anomalies is not a solved problem and requires humans to make more intelligent contextual decisions around metrics.

Another thing to consider is that obtaining and storing metrics does not come without a cost. Calculating more complex metric values costs CPU cycles, as does sending them to an external server for collection. Storage space for metrics on whatever servers you use also has a price tag associated with it. While many systems will have ways to age out data, keeping less and less granular information for metrics the older they get, you will still have to figure out how much detail you will need for each metric in order to have it be useful, as well as how long you will want or need to keep historical data.

Improvements: Planning and Measuring Change

Remember that lasting change will take time.

Identify the problem that you are solving. Before tackling a specific change, look around and examine what needs to be done. Determine who is interested in the project, who has time, and the overall value of the project. Visualize the various options and identify possible projects. Prioritize the projects.

Break down the specific project into smaller pieces that can be accomplished and tracked.

Identify who you are solving the problem for. What are their needs and motivations? How often are they going to use the solution?

Describe the solution, focus on the end goal. Talk to the stakeholders and ensure buy-in. This generally takes time and effort.

Identify possible tooling. Probe strengths and weaknesses. Sometimes you have to invent and develop the tooling. In-house development may seem cheaper but include the time and resources to support long term.

Focus on the process first. Do what makes sense. Make sure work is visible.

Now that we've taken a deep dive into the considerations around a variety of tools that you'll likely have as part of your business, we'll take a look at real-world examples. These companies target different industries, differ in specific tool implementa-

tions and processes, yet embrace similar principles. Seeing these use cases are examples for how you might identify your own implicit values and adopt desired principles in your own environment as you examine current tools, optimize the process of selection and elimination, and measure and iteratively improve your tool ecosystem.