# DevOps Culture and Practice with OpenShift

Deliver continuous business with OpenShift through people, process, and technology

Tim Beattie, Mike Hepburn, Noel O'Connor, and Donal Spring
*Illustrations by* Ilaria Doria

Packt
www.packt.com

# DevOps Culture and Practice with OpenShift

## Deliver continuous business value with OpenShift through people, process, and technology

**Tim Beattie, Mike Hepburn, Noel O'Connor, and Donal Spring**
*Illustrations by* **Ilaria Doria**

Packt

DevOps, at its core, is all about collaboration - collaboration between different people and different teams which focus on different aspects of software delivery. Achieving success in DevOps adoption is completely dependent on how well people, process, and technology work together.

We're writing this book because we want to increase the understanding and capability to apply DevOps culture and practices within organizations. Our aim is to improve culture and engineering practices and to achieve continuous business value by adopting the OpenShift Container Platform and embracing DevOps principles.

Our book's purpose is to enable readers to learn, understand, and apply many different practices - some people-related, some process-related, some technology-related - to make DevOps adoption and in turn OpenShift a success within their organization.

This preview release provides an introduction to the book and how we came to write it. It introduces many DevOps concepts and tools that we use in subsequent chapters to connect DevOps culture and practices through a continuous loop of discovery, pivots, and delivery. All of this is underpinned by a foundation of culture, collaboration, and engineering.

We're releasing this preview to get some early feedback from you, our readers. We'd really appreciate your thoughts and feedback on this first section and what you hope to read about in subsequent sections. You can provide your valuable feedback via this simple form.

Thank you and enjoy the preview!

Tim, Mike, Noel, and Donal

# Table of Contents

# Section 1: Practices make perfect

# 1

# Introduction – Start with why

You've picked up this book and have started reading it – thank you very much!

Perhaps you read the back cover and it gave you just enough information to be inquisitive enough to open the book up and read some more. Maybe a friend or colleague told you about it and recommended it to you. Maybe you have stumbled upon it for another reason. Whatever the reason, we're very happy you've taken some time out of your day to start reading this and we hope you get some value from it and want to keep reading it.

Before going into any kind of detail regarding what this book is about and what it's going to cover, we want to start with why. This is a practice we use to create a common vision of purpose. Why have we written this book? What problems is it trying to solve and who is the intended audience?

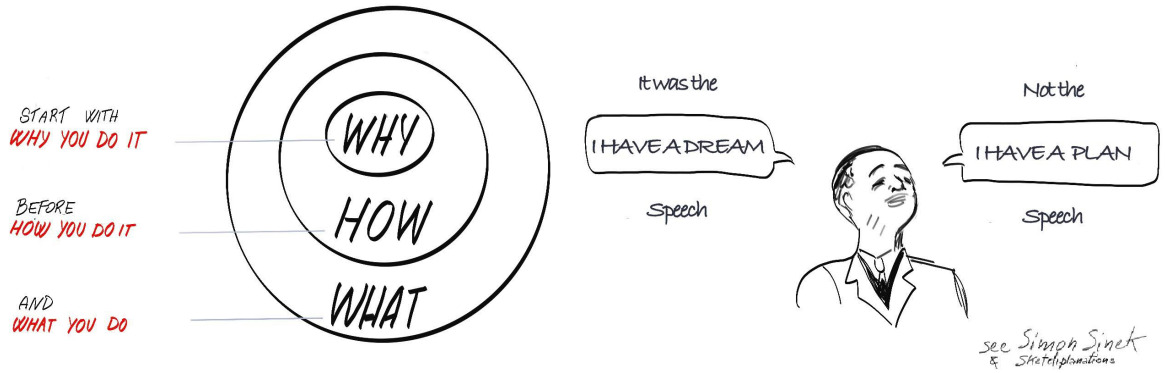


**Figure 1.1: Creating a common vision of purpose**

## Why: For what reason or purpose?

While this book may have been positioned as a book about technology, it is, at most, only one-third about technology. DevOps is really all about collaboration. We wrote this book because we want to increase your understanding of DevOps, collaboration, and cultural and engineering practices on a container platform such as OpenShift. We want to make moving to DevOps easier and provide a clearer path for you to applying DevOps and using OpenShift. We want to excite you when reading this and give you some inspiration as to how you can apply DevOps practices and principles. We want to equip you to go out and try these new techniques and practices.

As you progress through this book, we want you to continually measure the usefulness (impact/value) of using these new techniques. In fact, every time you try something out, we want you to think about and measure what impact it had.

That impact might be at an individual level: *What impact did trying that thing out have on me or another person? For example, has it reduced my cycle time to complete a set of delivery activities*? Alternatively, it might be an impact on a team or a department you work in: *Has team satisfaction been increased? What did we, as a group of people, achieve from that*? The impact might even be felt at organizational or societal level: *Has it reduced the number of operational incidents impacting customers*? We believe you will quickly start to see positive effects in all of these aspects. As a result, maybe you'll leave us nice reviews and tell all your friends about this book. If not, perhaps you can pivot and use this book as a doorstop or a monitor stand, which, of course, will give you a different type of value!

If you don't know where to start with how to go about measuring value, read on – we promise we'll cover that.

What we've just done is started using one of the practices and techniques used in writing this book. We have used the **Start with why** practice, which is something we always strive to do with every team or organization we work with.

So, what is a practice? A practice is an activity that helps teams achieve specific goals. It's not just an idea; it's something that you do repeatedly in order to hone or polish a skill. Practices have the following attributes:

- **Empowering**: The practices in this book will help teams discover and deliver iteratively.
- **Concise**: They can be read in a few minutes.
- **Agnostic**: Practices don't require the team to follow a specific framework.
- **Proven**: Practices have been tested in the real world.
- **Repeatable**: Practices can be used more than once.

Hopefully, throughout this book, you'll see many examples of us **practicing what we preach** through the many experiences, stories, and tips we will share from our real-world delivery experience, which includes stories such as these:

- The story about when we worked with an insurance company to rebuild one of their applications using DevOps and OpenShift but had a **stop the world** moment (a practice we'll talk about in the next section) when we realized we were redeveloping an app that users did not want and were not using!
- The story of when we worked with a European automotive company and kick-started modern application development and agile practices with one of their teams, only for the product owner to question how they were going to prove to management that this was a better way of working when management **only work with spreadsheets and numbers**.
- The story of the telecom company that suffered huge outages and non-functional problems over a festive period and were keen to learn new cultural and engineering practices to drive an auto-scaling and self-healing approach to their infrastructure and applications.

## So, why should I listen to these clowns?

Before you read any more from the four clowns writing this book, perhaps it's worth taking a step back and sharing a bit of background as to where all our anecdotes, theories, stories, and tips come from.

We all work for Red Hat. In particular, we are all a part of Red Hat's services organization, which means that we all regularly interact with, and deliver professional services to, Red Hat customers. This ranges from helping with installation and supporting the early adoption of Red Hat technology to driving large transformation programs underpinned by Red Hat technology and Red Hat's culture.

Red Hat's culture is relatively unique as it is entirely based on open source and open organizations (of which Red Hat is one of the largest examples). This means that the Red Hat organization is run under a set of characteristics that are closely aligned with open source culture and philosophy. They include collaboration, community, inclusivity, adaptability, and transparency. We highly recommend learning more about Red Hat's open organization philosophy by reading Jim Whitehurst's *The Open Organization*[1].

A lot of the experience that has informed this book and the stories and tips we will share emanate from engagements led by Red Hat Open Innovation Labs (or **Labs** for short). Labs provides an immersive and open approach to creating new ways of working that can help our customers and their teams develop digital solutions and accelerate business value using open technology and open culture. The main offering provided by Labs is called the residency, which is a four to twelve-week timeboxed engagement where client's engineers are matched one-on-one with Red Hat's technology and culture specialists.

Between the four authors, we've been involved in over 50 Open Innovation Labs' residencies around the world, in addition to many other professional services engagements. Due to the relatively short nature of Labs residencies, we get to learn very quickly different techniques, different approaches, and different practices. We get to see what works well and what doesn't work so well. We get to build up a huge collection of stories and tips. This book is all about sharing those stories and tips.

1 https://www.redhat.com/en/explore/the-open-organization-book

## Where did this book come from?

The title of this book is an evolution of a training enablement program that the authors have developed named *DevOps Culture and Practice Enablement*. This is an immersive training course run by Red Hat, providing enablement to Red Hat customers, partners, and employees.

We initially created the course because the services area of Red Hat we are working in was growing, and we needed a way to consistently increase the enthusiasm and shared understanding behind the practices and culture we were using globally, with our customers, and within our own organization. We wanted to do this by exploring all of the principal practices we had found to be successful in taking many products to market with our customers. This included practices to help understand the why and drive the discovery of products, as well as practices that would help us safely, securely, and confidently deliver in an iterative and incremental manner. And then there was the third outcome, which was having fun. We really couldn't see the point in all of this if you couldn't have some fun, banter, and enjoyment as you went along – it's one of the key ingredients of that mysterious word *culture*.

One of the key success factors behind this was injecting lots of experience and real-life stories into our delivery and using a lot of our practices on ourselves to deliver the course. Every time we run the course, we use the definition of done[2] practice to explain to participants that every practice we are going to teach on the course will be presented in a consistent way, following this process:

1. Introducing the practice with the theory and an overview of what it is, why you should use it, and how to use it
2. A hands-on practical exercise so everyone participating can leave the course having **had a go** at using the practice and having gained some learning and experience from it
3. A real-world example of the practice being used in action on a real customer delivery project or product development initiative

2 https://openpracticelibrary.com/practice/definition-of-done/

The core practices taught in this course vary from discovery practices, such as impact mapping and event storming, to delivery practices, such as sprint planning and retrospectives. They include a set of practices we've found to be very powerful in establishing high-performing, long-lived product teams, such as social contracts, team sentiment practices, and mob and pair programming. They include the engineering practices that many coming to the course would have most strongly associated with the term *DevOps*, such as continuous integration, continuous delivery, test-driven development, and infrastructure as code.

One of the unique aspects of this course was its appeal to a broad audience. It was not exclusively for technologists or designers. In fact, we embraced the idea of having cross-functional groups of people spanning from engineers to project managers, from infrastructure experts to user experience designers. We felt this course offered the opportunity to break down silos. We intentionally do not run different tracks for different types of people. The aim is for participants to have a shared understanding of all of the practices that can be applied to truly appreciate and enable a DevOps culture.

Having run this course more than a hundred times globally, we've learned volumes from it and have continuously improved it as we've gone along.

Faced with the opportunity to write a new book about DevOps with OpenShift and to apply new learnings and more up-to-date technologies from Stefano Picozzi, Mike Hepburn, and Noel O'Connor's existing book, *DevOps with OpenShift – Cloud Deployments Made Easy*, we considered what the important ingredients are to make DevOps with OpenShift a success for any organization choosing to adopt the technology.

The success factors are all based on people, processes, and technology through the application of the many practices we've used with our customers globally and, in particular, the kinds of practices we were introducing and enabling using DevOps culture and practice enablement.

This book's purpose is to enable you to understand and be ready to apply the many different practices – some people-related, some process-related, some technology-related – that will make DevOps culture and practice with OpenShift a success within your organization.

## So, who exactly is this book for?

This book is intended for a broad audience – anyone who is in any way interested in DevOps practices and/or OpenShift or other Kubernetes platforms. One of the first activities for us to undertake was to get together and list the different personas and types of reader we intended to write for. These included the following:

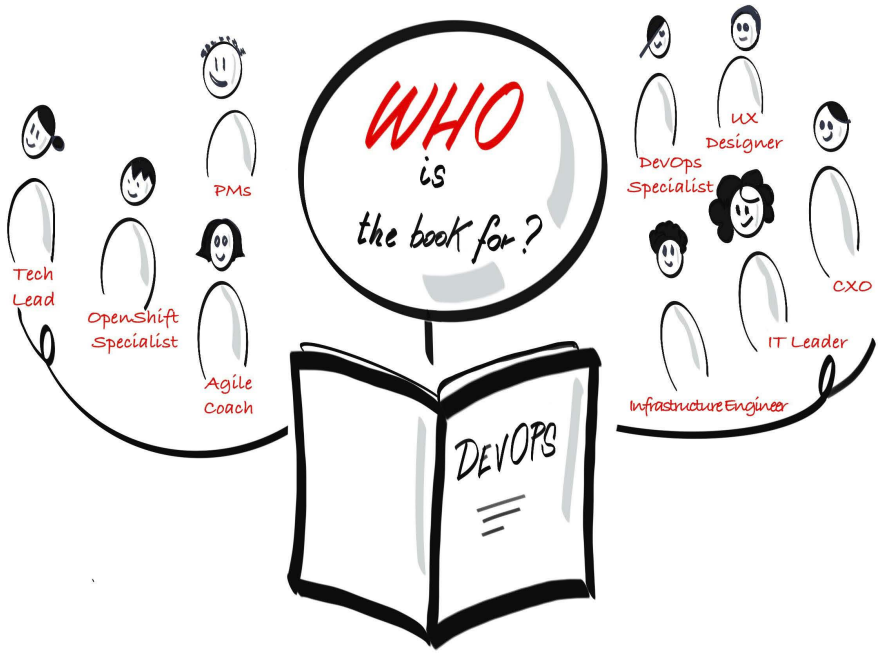


Figure 1.2: The intended audience

- Caoimhe, a technical lead who looks after a team of people who develop software. She wants to learn more about DevOps so she can help adopt great DevOps practices.
- Fionn, a project manager who is responsible for a set of legacy software applications and wants to modernize his team's approach to make use of this **DevOps** thing he's heard lots of people talking about.
- Padraig, an Agile coach who is very experienced in applying Agile delivery frameworks such as **Scrum** and wants to further his skills and experience with DevOps. He feels that this will really add value to the teams he is coaching.

- Tadhg, a user experience designer who wants to better understand what other people in the company's development team do with his designs and how he can collaborate with them to deliver products.
- Séamus, who is an IT leader executing his company's technology strategy to adopt containers and cloud-native technology across the company's entire IT estate. He has chosen **OpenShift Container Platform** (**OCP**) as the strategic product to support this. He wants to ensure that OCP generates a fast return on investment and that there is a large uptake across all IT teams in his organization.
- Aroha, the CIO of the organization. She wants to ensure that the company's people are aligned with company strategy and getting the very best out of the technology and the organizational decisions being made to drive the strategy. She's motivated for the business to become more agile and adapt quickly if and when market conditions change. She wants to read about what similarly sized organizations in different industries (including in her own industry) have successfully done and what they saw as being the critical success factors.
- Siobhán, an infrastructure engineer who has been using Kubernetes for many years and is now part of a team introducing OCP to her organization. She wants to ensure that the platform is configured to support her team's goals and wants to know how she can best work with development teams so that they get the maximum value out of the technology.
- Eimar, a project manager who has spent two decades delivering IT projects through up-front planning, tracking deliverables against plans, and managing risks, issues, and dependencies with strong project reporting and stakeholder management skills. She gets frustrated by the amount of time it takes to ship software and not being able to address user needs and fixes quickly. She sees the benefit of moving to a more product-centric approach rather than a project-centric one. She would like to re-skill herself to be a product manager. In doing this, she wants to be able to test and adapt quickly, ship deliverables quicker, adapt to changing market conditions, and also improve performance, uptime, recovery times, and more.
- Finn, a system tester who takes great pride in quality assuring software before it is shipped to customers. His business analysis background helps him develop comprehensive testing approaches and scripts and, over the years, he's also led performance testing, security testing, and operability testing. He's keen to learn how he can introduce more automation to his work and branch out to other forms of testing.

### From "I" to "T" to "M"

With this book, we want people to move away from being **I-shaped**, where they are a specialist in one skill or one field. We want them to become more **T-shaped**, where they still have a depth of skill and experience in a particular field (such as infrastructure or UX design), but they also have an appreciation and breadth of knowledge across all the other skills that people bring to make up a cross-functional team. This could be a frontend engineer, for example, who also works side by side with the API engineer.

A great cross-functional team is one where the full team holds all the skills and experience they need. They are empowered to take a new requirement from a user or business stakeholder through to production. A team could be made up of lots of **I-shaped** people, but this type of team quickly becomes dependent on specific individuals who can be a blocker when they are not available. For example, if a database change is needed to expose a new API but only one team member has the knowledge to be able to do this, the team can quickly become stuck. If the team is full of more **T-shaped** members, there is a greater opportunity for collaboration, sharing, and partnerships across the team and less reliance on individuals:

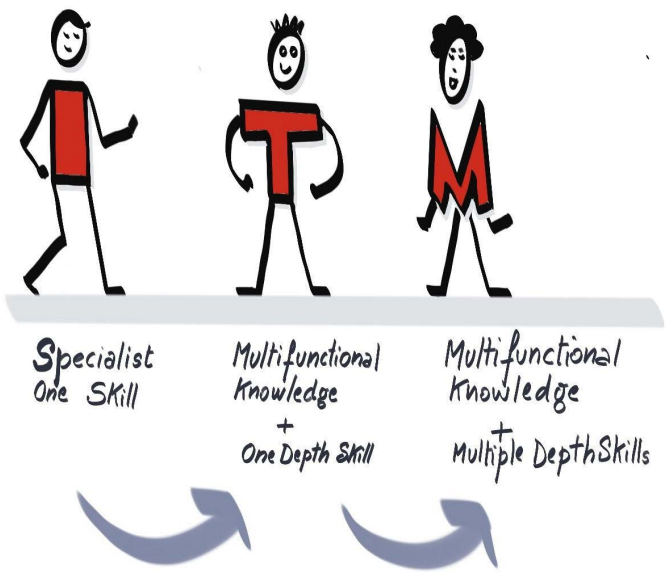


Figure 1.3: Skills transformation

We want this book to help **I-shaped** people become more **T-shaped** and perhaps even become **M-shaped**. **M-shaped** people are inspired to deepen their learning, take it into other fields, and hold multiple skills, thereby building stronger cross-functional teams.

## Conclusion

This chapter presented a brief overview of why we wrote this book and who it is intended for.

We introduced ourselves and how we will be using our applied knowledge, experience and learnings to write this book full of stories and examples.

We examined the different personas we are targeting in this book and how we intend to help move these focused I-shaped people into more T-shaped or M-shaped to build stronger cross functional teams.

In the next chapter we will introduce DevOps and some tools we will use during the book to organize and explain DevOps practices.

# 2

# Introducing DevOps and some tools

## What does it mean to be DevOps in a container world?

People have different perceptions about DevOps, what it means and how it works.

In this chapter, we are going to explain our view on DevOps and the bottlenecks and challenges that DevOps focuses on addressing. We will introduce the idea of a value chain in software product delivery and how we can use different techniques from lean, agile and DevOps communities to optimize and speed up the value chain.

We will also introduce some tools, such as the Mobius Loop and the Open Practice Library that we will use to navigate our way through the many practices utilized in the rest of the book.

*DevOps* is a bit of a buzzword at the moment! It seems that for every decade in technology, there is a new buzzword associated with it.

Throughout the 2010s, *Agile* was that buzzword. **This is going to be an Agile project**, or **We're going to use Agile to deliver this**, or **We're going to use the Agile methodology** are common phrases that many of us have heard. It was (and still is) often used incorrectly about delivering software faster. In fact, Agile is focused more around delivering business value earlier and more frequently and driving a culture of continuous learning. Agile has now officially grown up – it had its 18th birthday in February 2019. Even after all this time, we still love to use the *values* and *principles* of the Agile Manifesto[1] created back in 2001.

Containers is another buzzword these days. We see it being used by individuals without them necessarily understanding the full meaning of what a *container* is and why people, teams, and organizations would benefit by utilizing them.

So, with this book being about DevOps and OpenShift (a container management platform), we're going to de-buzzify these terms and talk about very practical, real-world experience and examples of the real value behind DevOps and OpenShift containers.

Let's take a look back in time and see where we believe these phenomena came from.

We all have worked in IT for a number of decades (some more decades than others!). While chatting over a beer and looking back at our experiences of delivering IT projects, we recognized some common characteristics in all our IT projects that have been constant. We also identified a set of gaps in the value chain of delivering IT projects that, for us, seemed to slow things down.

1 www.agilemanifesto.org

### The value chain...

Every project we've ever worked on has had some kind of end customer or user. Sometimes they have been external users, such as an online shopper wanting to use their mobile app to buy the latest set of Justin Bieber bedsheets! Other times, they have been teams internal to an organization, such as an operations team or a particular department within a company. One common denominator we all agree on is that the objective of our work was always having smiley, happy customers:

**Figure 2.1: Happy customers - The ultimate goal of organizations**

Between us, we have helped many organizations, from the public sector and finance to retail and charities. We've seen it all! As we reminisced, we discussed the end result of some of our projects; we thought about our why – there was almost always some kind of monetary value aspect associated with the reason for us being there. There were other motivations, too, such as increased customer satisfaction, reduced risk, and improved security and performance, but the bottom line is that an essential part of any of our commercial customers' business is to make money and reduce costs.

So, in the end, value was often linked to money in some shape or form. Three of us authors are Irish and the fourth is from New Zealand, so we felt it was appropriate to reflect this as a pot of gold!

**Figure 2.2: Profits - A common goal of every commercial organization**

The 1990 book *The Machine That Changed the World*, written by James Womack, Daniel Jones, and Daniel Roos, first introduced the term **value stream**. The idea was further popularized by the book *Lean Thinking*, written by the same authors. According to them, the value stream *is the sequence of activities an organization undertakes to deliver on a customer request. More broadly, a value stream is the sequence of activities required to design, produce, and deliver a good or service to a customer, and it includes the dual flows of information and material. Most value streams are highly cross-functional: the transformation of a customer request to a good or service flows through many functional departments or work teams within the organization:*

**Figure 2.3: Customers dreaming of the pot of gold**

Let's visualize this as our customers dreaming of that pot of gold. They're constantly thinking about how they can get the most out of their products or ideas to generate the most gold. So, how do they go about doing this?

## Let's look at the gaps

### The big list of things to do

The first gap in the software development process that we consistently saw was the process of collecting information from end customers and forming a list of customer requirements:

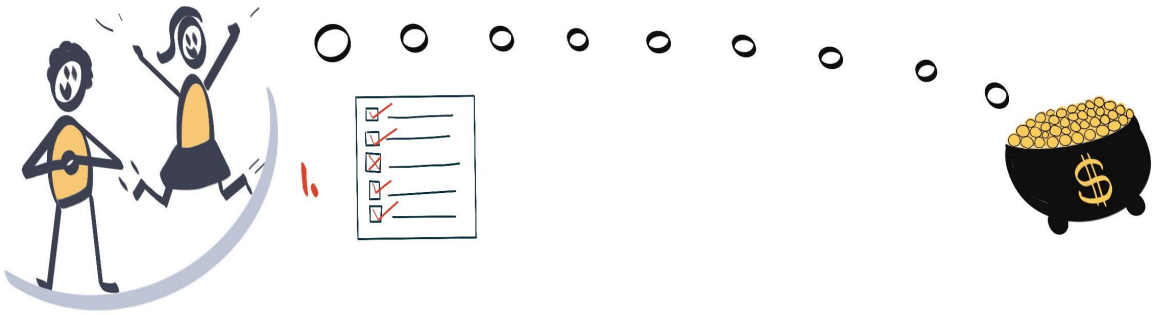

Figure 2.4: Understanding and collecting customer requirements

Our early projects often involved long phases of business analysts documenting every possible requirement they could conceivably think of into epic volumes of business requirements – documents. The goal was to pre-empt every conceivable customer journey or scenario and to cover all the bases by building specifications that included every possible eventuality. Sounds rigid, right? What if we made an incorrect assumption?

### Demonstrating value and building the right thing

The second gap revolved around demonstrating value to customers. Usually the **project** being undertaken was set up to include all of the features and ideas needed so that they could be released together. Once the project was in production, it would only have a small operations budget to support minor enhancements and problem resolution. Sounds like it might take a long time to get the application into the end users' hands, right?

There are two reasons we call these *gaps*. First, the process was lengthy – months, sometimes years, would elapse between starting a project and signing off on the requirements. Second, trying to collect every possible requirement before delivering anything would mean no real benefit to the end customer for years and often, the wrong functionality was built and delivered to an unhappy customer:

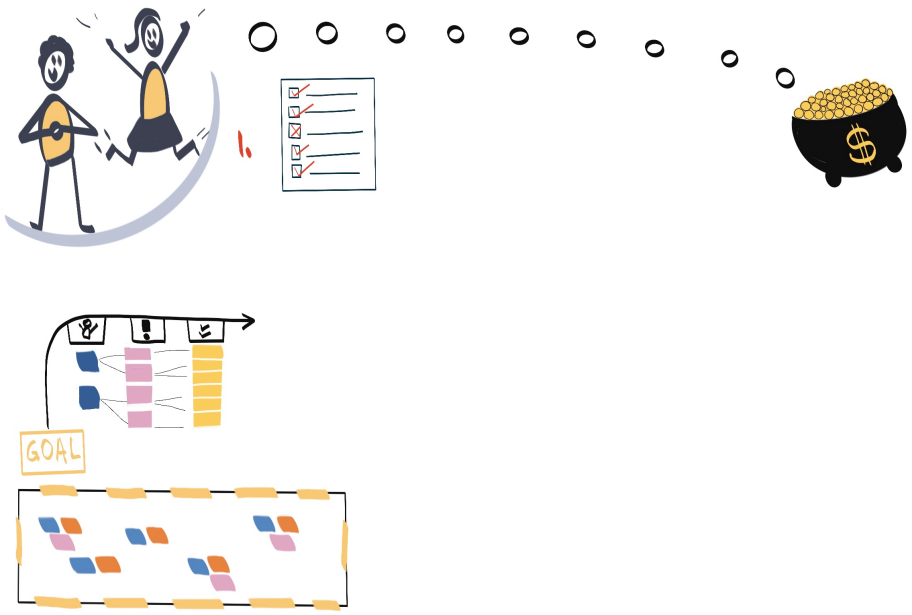


Figure 2.5: Using human-centered practices to understand customer needs

This gap of *not building the right thing* has been plugged in recent years by the emergence of human-centered design and design thinking. These are a set of practices that put the end user at the center of capturing the needs and requirements of a product.

We gather the information by talking directly to users and forming greater *empathy* with them:

**empathy** noun

Save Word

em·pa·thy | \ ˈem-pə-thē \

**Definition of *empathy***

**1** : the action of understanding, being aware of, being sensitive to, and vicariously experiencing the feelings, thoughts, and experience of another of either the past or present without having the feelings, thoughts, and experience fully communicated in an objectively explicit manner

*also* : the capacity for this

Figure 2.6: Merriam-Webster definition of 'empathy'

https://www.merriam-webster.com/dictionary/empathy

In this book, we'll explore how techniques such as impact mapping, event storming, and human-centered design can aid the software development process. We'll also explore other practices to help us define solutions and features and crucially ensure that the solution is connected to business value. We'll show how the act of coupling research activities such as user interface prototypes and technical spikes with experimentation inform product backlogs that are well prioritized according to delivered business value. We will show you how using just enough information can lead to a better-understood product.

**How do we do the things on our list?**

Let's consider the second gap in delivering value to users. This gap focuses on moving from a shopping list of TODO items into working software.

The traditional approach is to sign off and confirm a finite set of requirements that have undergone the lengthy process of business analysis and capture. The scope of the project is locked down and a stringent change control process and governance is put in place for dealing with any deviation from the documented requirements.

A team of software designers and architects then gets to work, producing a **high-level design** (**HLD**) that will deliver a solution or set of solutions according to the business requirements specified. These requirements also go through a formal review process by key project stakeholders and, once signed off, become the reference source for the solution scope.

Often, different design documents are written in the next phase – detail design documents, program specifications, data designs, logical architecture blueprints, physical architecture solutions, and many more. Each of these is written to support a defined, dated, and signed-off version of the HLD, which itself, is signed off against a defined set of business requirement specifications:

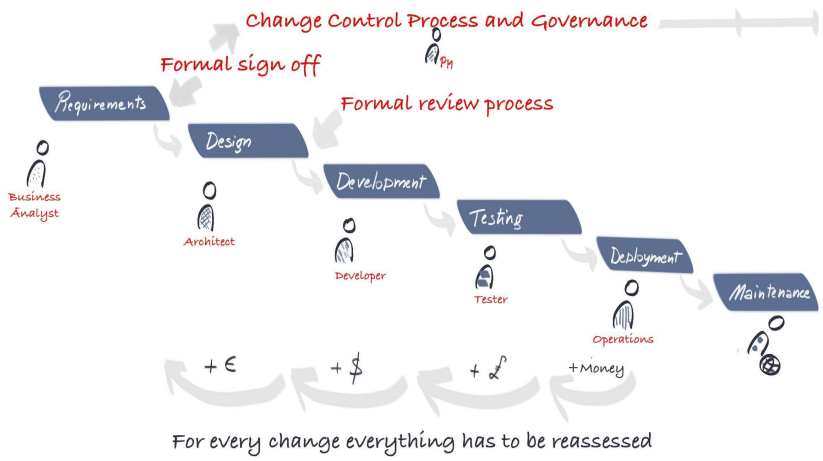


**Figure 2.7: Traditional application development lifecycle**

Any changes to the earlier documents have direct time and cost implications for reassessing and updating each of the following design documents. Software development teams may have been involved in the production or review of some of these documents. However, they are often encouraged not to start any coding or development activities until these designs have been locked down. Some organizations reduce project costs by not onboarding development teams until this stage. Development is often siloed by function and unaware of the big picture with limited automated testing.

At a predefined point in the project plan, all developers are expected to have delivered their coded components to a testing environment. Perhaps each developer manually builds and deploys their own code to the testing environment. Some larger programs seek economies of scale by setting up build infrastructure teams who do this on behalf of all developers. Once all components had been delivered, a separate team of testers starts executing the hundreds of test scripts they had been writing in the preceding weeks and months to test the solution according to business requirements and HLD documentation. This is the first time some components are integrated and tested together. Of course, problems and bugs drive reworking by development teams and designers to fix such issues.

Just as there are different levels of design documentation, testing often undergoes different levels of testing, with one starting when the previous phase is completed. A test manager would sign off on a set of test results, signaling that the next level of testing could start. Testing would range from a set of component integration testing to wider system integration testing, security and penetration testing, performance testing, failover and operability testing, and finally, user acceptance testing!

The final stage before the big bang go-live of a solution would often be user acceptance testing, involving a set of focus users and the test system. In many cases, it could often be months or years before this first user saw the implemented system. Once user acceptance of the solution was signed off, the green light was given to deploy to the production environment. Finally, with the software in the hands of real end users, business revenue could hopefully be generated from all this work.

You're probably thinking that this process sounds long and drawn out – well in truth, it was! Many programs hit delays at different points along the way and what started out as a multi-month project plan ended up being years long. For the curious, there is even a list of some epic failures on Wikipedia: https://en.wikipedia.org/wiki/List_of_failed_and_overbudget_custom_software_projects.

Often, business conditions would change during the development period. New feature requests would be generated. During testing, gaps in the requirements would emerge that no one considered during the analysis and requirements capture. The market didn't stand still during development and competitor companies may have started to innovate quicker. The competition would even provide more feature requests, in a process akin to a feature comparison war.

Of course, there was always some kind of change control procedure to handle new scope like this. In a complex program of work, the lead time to get features added to the work plan could range from months to years. In order to get something into production, program executives would simply say no to any more change and just focus on getting to the end of the project plan.

This meant that solutions finally delivered to production were somewhat underwhelming to users several years after the first requirements were discussed. Time and industry had moved on. The biggest frustration of these programs was that they were frequently delivered late, were over budget, and often delivered a solution that lacked user satisfaction or quality.

Stepping back a little, we had this huge gap of converting lists of features into a software deliverable. The process known as **Waterfall** due to the nature of separate phases of work flowing down to the next phase was associated with very lengthy times:

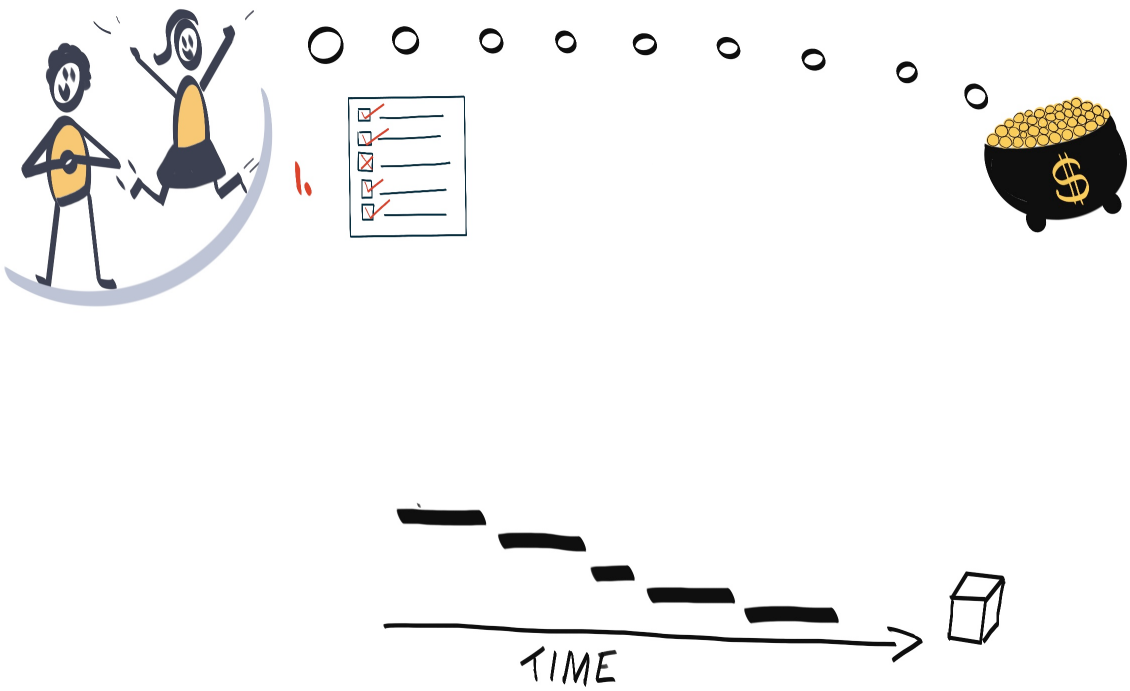


**Figure 2.8: Traditional deliverables with its drawbacks failed to achieve customer satisfaction**

Let's think about how we plug that second gap with more modern software development processes. How do modern developers manage to translate user needs into working software solutions much more quickly compared to previous ways of working?

The formation of the Agile movement in 2001, led by the 17 IT individuals who wrote the Agile Manifesto, has triggered alternative approaches and mindsets toward delivering software. Many of the individuals involved in writing the Agile Manifesto had been tackling many of the problems described by Waterfall development. Jeff Sutherland and Ken Shwaber had created the Scrum framework for software development, which included delivering small incremental releases of value much more frequently – they used the term **sprint**, which was a fixed time box ranging from 1-4 weeks (usually being 2 weeks), during which a set of events and roles would work together such that big solutions could be delivered iteratively and incrementally. Kent Beck and Ron Jefferies led much of the **eXtreme Programming** (**XP**) movement, focusing on delivering faster releases of value and working on key practices that helped drive more efficiency into review, testing, and release processes, using better collaboration and increased automation:

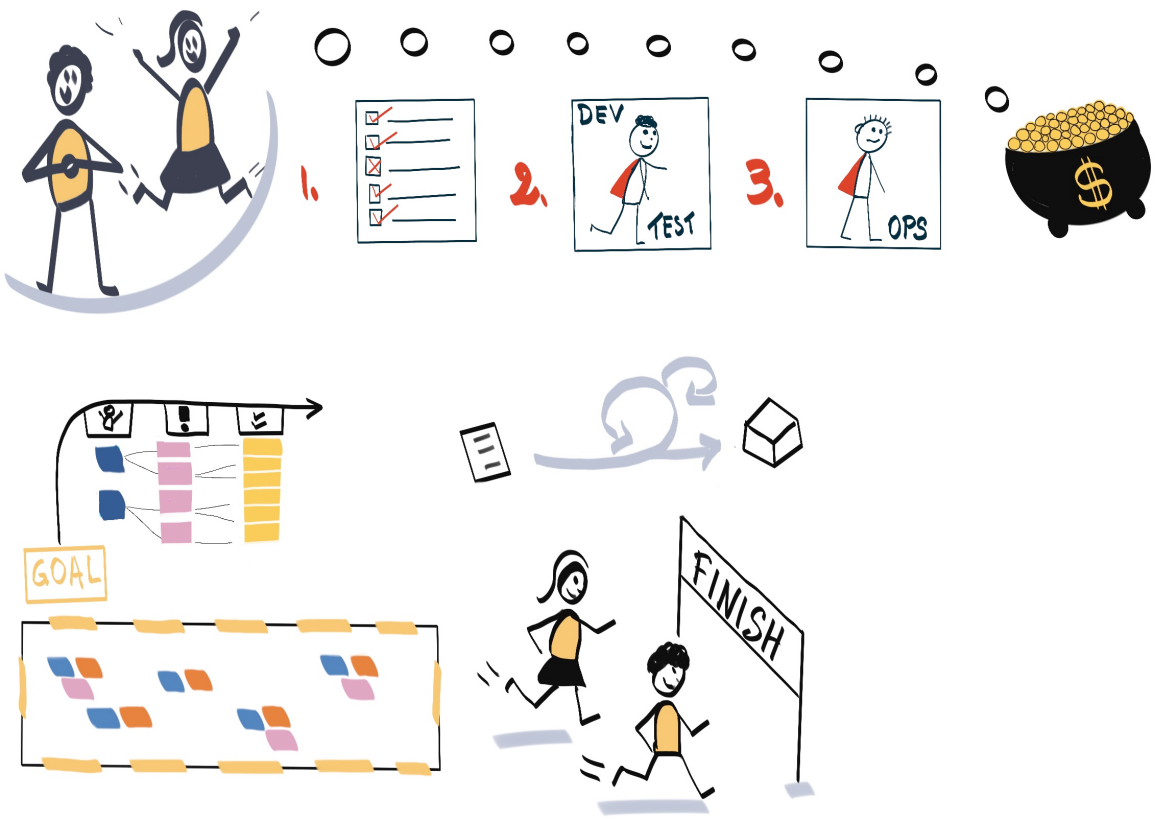


Figure 2.9: Implementation of DevOps practices leading to faster delivery and better products

In this book, we'll show you different software delivery practices and how our experience using a mixture of different practices from Scrum, Kanban, XP, Lean, and some scaling frameworks helps deliver value quicker. All the underlying practices are simply tools to help close the gap between an idea or requirement being captured and it being delivered. This has been an area we have sought to continuously improve to a level where the gaps are minimized and we're working in a mode of continuous delivery.

**Development to operations**

There is one more gap to plug in our efforts to optimize the software delivery process. The third gap is the one between development teams and operations teams.

In our Waterfall process, we had reached the point where the signed-off solution exited user acceptance testing and went through a big bang go-live. So, what happened next?

Often, a whole new team responsible for maintenance and support would then pick up the solution. The people who work in this new team were not involved in any of the design, development, or testing, so additional time would be built into the project plan for knowledge transfer. The delivery team would write lengthy documentation in the hope that this would be a useful resource for future operations teams.

At this point, the package of software would metaphorically be thrown over the wall from the army of developers to the army of operation engineers. The operations teams often had to learn about the software the hard way by investigating production incidents, addressing bugs that were not found previously, and handling new scenarios not considered during the requirement planning stage:

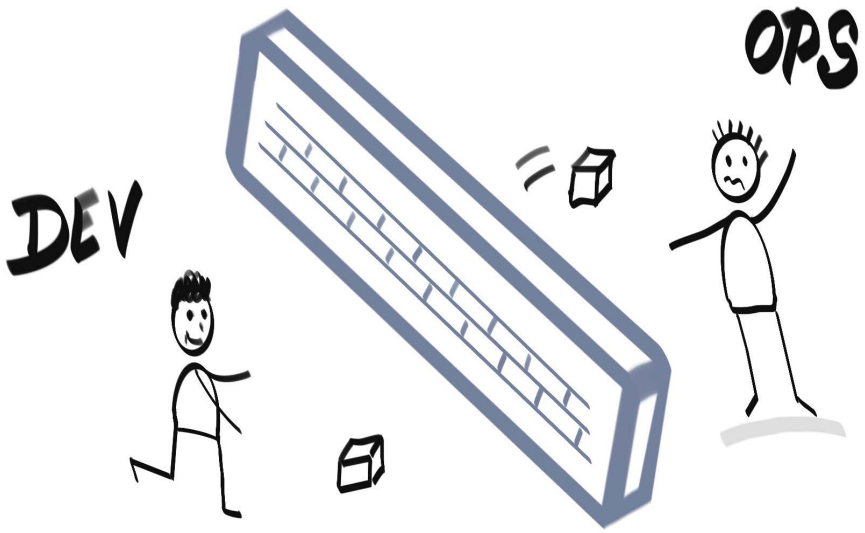


**Figure 2.10: Aspiring to bring down the wall between development and operations teams**

To plug this gap, we must bring development and operations teams together. Tear down that wall and remove the silos! Bringing down the wall forms new teams that are focused on development and operations activities. These teams are collectively responsible for the whole solution and can design the solution according to each others' needs.

The term **DevOps** was coined by this idea that we no longer have siloed development and operations teams. In recent years, we've seen various other terms emerge from this idea, such as these:

- **DevSecOps**: Cross-functional teams that develop solutions, operationally support them, and ensure security compliance
- **BizDevOps**: Cross-functional teams that encompass shared understanding across the team of the business value of features, the development of those features, and the operational support of the components containing those features
- **DesOps**: Cross-functional teams that consider the user experience and user interaction design of proposed ideas, the development of the resulting features, and the operational support of the components containing those features
- **BizDesDevSecOps** (**!**): What we really mean by a cross-functional product team: a group of people who, between them, can take an idea and conduct all of the research and experimentation activity, develop solutions, and operationally support them

BizDesDevSecOps is a bit of a mouthful, so we're going to use the term **product team** to describe it throughout this book. It addresses the ultimate goal of plugging all gaps in the software development process and bringing down all the walls:


Figure 2.11: Plugging the gaps in the software delivery process

Note that we will not use the term **DevOps team** – the idea of having a team or even an individual purely focused on DevOps runs counter to what the DevOps philosophy is all about – collaboration, cross-functionality, and the removal of silos. How many times have you seen ads on LinkedIn or other sites looking for DevOps engineers? The invention of the DevOps engineer or the DevOps team could be seen as creating just another silo.

## People, process, and technology

DevOps is really all about collaboration. It's about taking pride in, and ownership of, the solution you're building by bringing down walls and silos and by removing bottlenecks and obstacles. This speeds up the value stream connecting the customer's perceived need to the product delivery.

Technology alone will never solve all your business problems. No matter how good the platform or software product you are evaluating or being sold, unless your organization has learned to adopt the correct balance of people aspects, process changes, and technology adoption, the objectives will not be met.

This book is about finding the right combination of *people*, *process*, and *technology* changes needed to maximize business outcomes on a continuous basis. This requires changes in mindset and changes in behavior. This book will look at the behavioral change that we have seen be most effective with the hundreds of organizations we have collectively worked with. We've observed that such mindset and behavioral change is needed across all roles and that we need to break down the silos we see inside organizations, which, as we saw previously, is what drives the gaps and inefficiencies in software development:

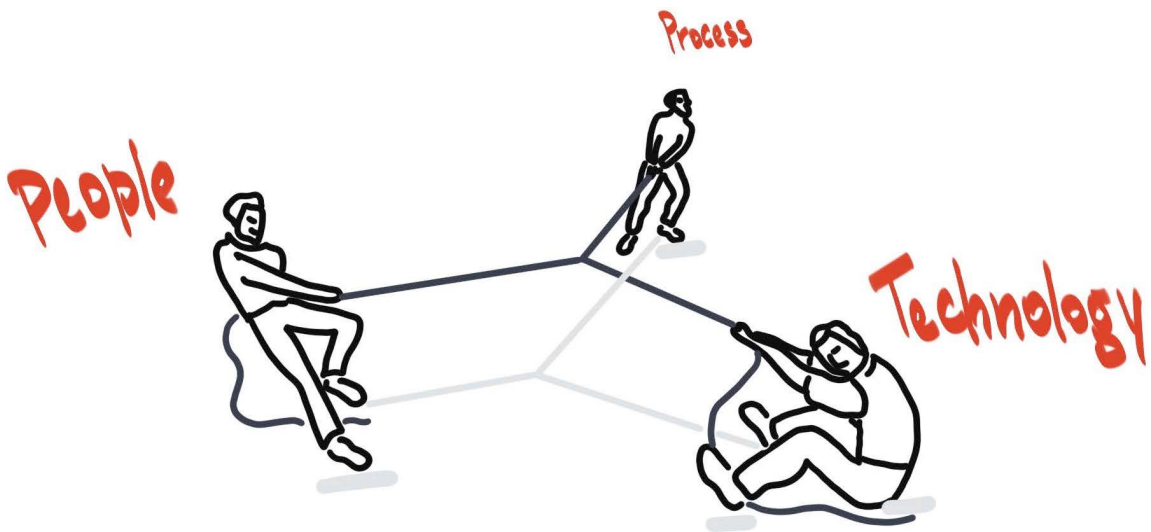


**Figure 2.12: A healthy balance between people, process, and technology**

Everyone in an organization should care about people, process engineering, and technology in order to drive the desired outcomes. We want to break down the silos between these three pillars and bring them closer together. A reader who may be more interested and focused in one of these three things will get as much (if not more) value from learning about the other two things.

This means a hardcore software engineer or architect can pick up some great insights and guidance on why people, culture, and collaboration are equally important for their role.

Someone who has previously been an expert in project management methodologies and is now learning about more agile delivery practices such as Scrum can also use this book to learn about modern technology approaches such as GitOps, CI/CD, and serverless. They can learn why these are important to understand and appreciate so that they can articulate the business value such approaches bring to organizations.

A leader who is concerned about employee retention can learn how the mastery of these modern tech practices of iterative and incremental delivery strategies can maximize the opportunities for organizational success through the delivery of highly valuable products being used by happy customers.

## The Mobius loop and the Open Practice Library

In this book, we're going to explore lots of different practices. We're going to explain what they are and why we use them. We're going to give you some guidance on how to use them. We're going to share some real-world examples of how we've used them and, where possible, we'll even show them in action. Using our Pet Battle case study (more on that later), we're going to bring them to life in a fun way and we'll share the best tips that we've picked up in the field.

A problem we hit a few years ago when working with our customers' new teams was explaining how and when you might want to use different practices and in what order. What practice should we start with? What practice links nicely to the output produced from a previous practice, and so on?

To help with this, we have made use of an open source navigator tool called Mobius. This was created by Gabrielle Benefield and Ryan Shiver. There is a huge amount of great material, including a number of open sourced canvases and artifacts, available at www.mobiusloop.com. Red Hat Open Innovation Labs makes use of this open source material in all of its residencies and in its DevOps culture and practice enablement courses. We will use it in this book to structure the content and the sections.


**Figure 2.13: The Mobius loop**

Mobius is a framework that connects discovery and delivery and can be used to connect strategy to products to operations. The common denominator is measurable outcomes. Mobius is used to understand, align, and share measurable target outcomes so they can be tested and validated.

There are a number of principles that underpin the Mobius navigator:

- **Outcomes over outputs**: We focus on delivering tangible impacts or outcomes to people as opposed to delivering lots of features that may not drive outcomes.
- **Multi-options strategy (options pivot):** We look to build a list of options, a list of research initiatives, experiments, and implementation features that can be used to test hypotheses about whether those research initiatives, experiments, and implementation features will indeed drive the anticipated outcomes.
- **Rapid delivery**: We aim to use short iterations of delivery with regular feedback and measurement as we strive toward the idea of continuous delivery.
- **Continuous learning and improvement** happens throughout the cycle so that our next set of options yield an even better impact on outcomes.

There are **seven core elements** to the Mobius approach across a continuous and never-ending flow. They can be visualized on a single canvas that is open source and made available under a creative commons license at www.mobiusloop.com:

**Why** describes the purpose. Why are we doing this? What is the problem we are trying to solve? What is the idea we are trying to pursue?

**Who** focuses on the end users. Who are we trying to solve the problem for?

**Outcomes** are where we want to get to with these people and how we will measure the customer and business impacts delivered.

**Options** are the potential solutions that could deliver these outcomes. They help define the hypotheses we can go on to test and help us find the simplest way to achieve the desired outcome with the least amount of effort or output.

**Deliver** is the cycle where we run experiments to deliver a solution or set of solutions to users so we can measure the impact.

**Measure** is where we assess what happened as a result of delivering the solution or set of solutions. We check whether the impact of the solution delivered the desired outcomes and assess how much of an impact we achieved.

**Learn** is the feedback loop that takes us back to the options pivot. We learn from what we delivered and assess what to do next. Have we delivered enough to make an assessment? Do we go right back around the delivery loop again? Have we reached our target outcomes or invalidated assumptions from our learnings? Do we return to the discovery loop?

Personas such as Tadhg, our user experience designer, would typically spend a lot of time in the discovery loop. Personas such as Caoimhe, our technical lead, would traditionally be focused on the delivery loop. Personas such as Fionn, our project manager, would typically spend a lot of time here establishing outcomes and gathering options. But, as we seek to move to cross-functional teams of T- or M-shaped people, we really benefit from everyone being involved at every stage of the Mobius loop. And Mobius creates a common language based on targeted measurable outcomes.

You can apply the same principles of outcome-driven thinking for strategy, product, and services delivery to enabling business and technical operations – we'll return to this idea later in the book.

Mobius is powerful because it's framework-agnostic. It integrates with many existing frameworks and methods you may already be familiar with – Scrum, Kanban, design thinking, Lean UX, Business Model Generation, Lean startup, and many other great frameworks that have surfaced during the last couple of decades. You don't have to reinvent the wheel or replace everything you already like and that works for you.

You can capture key information on a discovery canvas, an options canvas, and a delivery canvas – all of these are open source artifacts available under **Creative Commons** at www.mobiusloop.com:

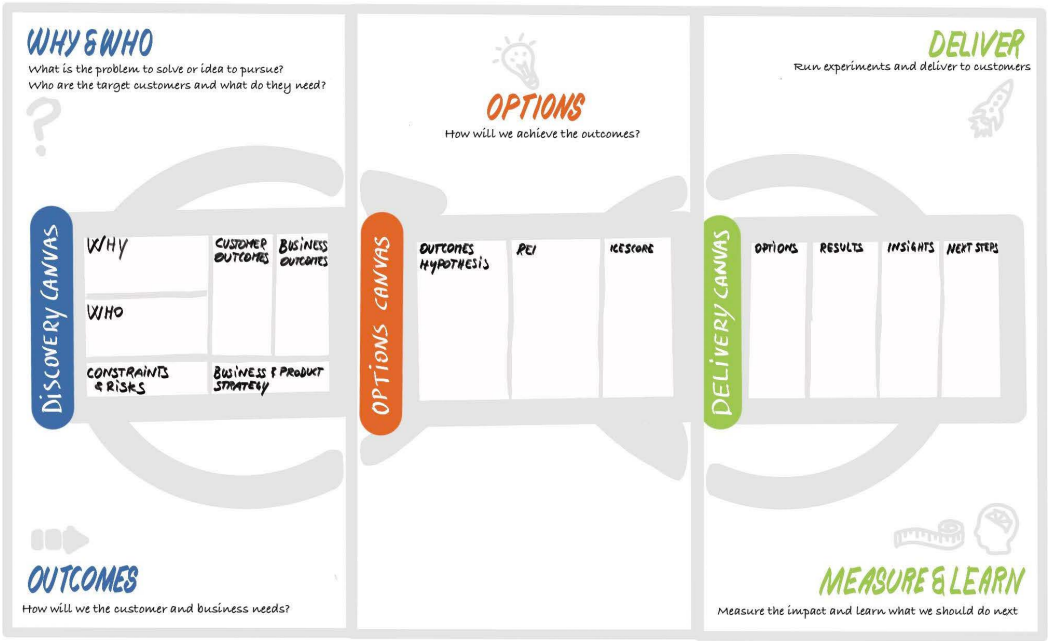


**Figure 2.14: Using the Discovery, Options, and Delivery canvases of the Mobius loop**

When Red Hat Open Innovation Labs started using Mobius, we placed all of our practices around the Mobius loop. Some practices clearly aligned with the discovery loop and, in particular, the **Why & Who** end of the discovery loop. Practices such as impact mapping, start-at-the-end, and empathy mapping are great at uncovering the answers posed in this section of the loop. We'll get into the detail of these practices in subsequent chapters of this book.

Practices such as event storming and user story mapping were very helpful in establishing and visualizing outcomes on the other side of the discovery loop. Again, we'll look at these practices in detail and share some great examples of their effect.

Practices such as design sprints, how-might-we, and product backlog refinement would help determine and organize the series of options available attempting to drive toward outcomes.

Practices such as sprint planning would help plan and execute the incremental delivery of products toward outcomes. We'll explore these iterative delivery practices and how different Agile frameworks can be used with Mobius.

Practices such as showcases and retrospectives would help with capturing measure-and-learn data from incremental delivery.

We still had a large number of practices that we did not feel naturally fitted into one of the loops or the options pivot. When we laid out all of the remaining practices that we had all used with numerous customers very effectively, we found they fitted into one of two areas. One set of practices were all focused on creating culture and collaboration. The other practices were all technical engineering practices that supported the concept of continuous delivery.

When explaining these practices to others, we talked about these being very important practices to put in place, but not necessarily practices that you would schedule. For example, you will learn that practices such as impact mapping on the discovery loop are important scheduled workshops that you execute and occasionally revisit in the future. Practices such as sprint planning, showcases, and retrospectives on the delivery loop are also tightly scheduled when working in an iterative delivery framework. But, the practices associated with culture and collaboration or those associated with technical engineering were more like practices that you use all the time, continuously.

Practices such as social contracts and definition of done are not one-time-only practices where you bring the artifact out on a schedule. These are living and breathing artifacts that teams use all the time in their day-to-day work. Likewise, continuous integration, test automation, and infrastructure as code – these are not the types of practices you schedule one or two times a week. These are practices that you do all the time. They are practices in the foundation of where and how we're working. In order to effectively practice continuous delivery and continuous discovery as presented by the Mobius loop, we need to have a strong foundation of culture, collaboration, and technical engineering practices.

To visualize this, we added the foundation to the Mobius loop:

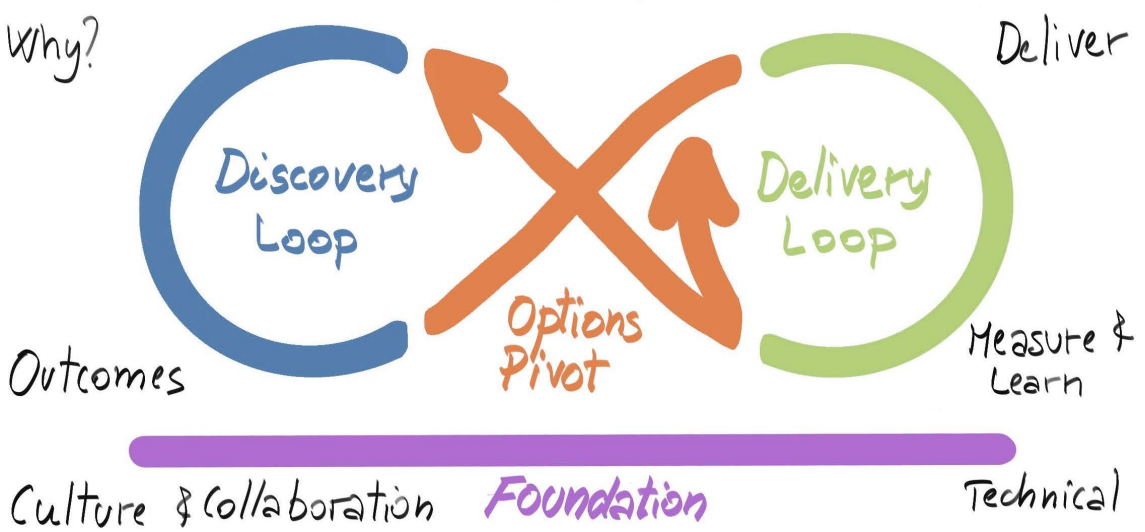


**Figure 2.15: Adding a foundation to the Mobius loop**

This graphic has become a simple visualization tool that helps us navigate the ever-growing list of practices and techniques we use to achieve continuous discovery and continuous delivery of digital products:

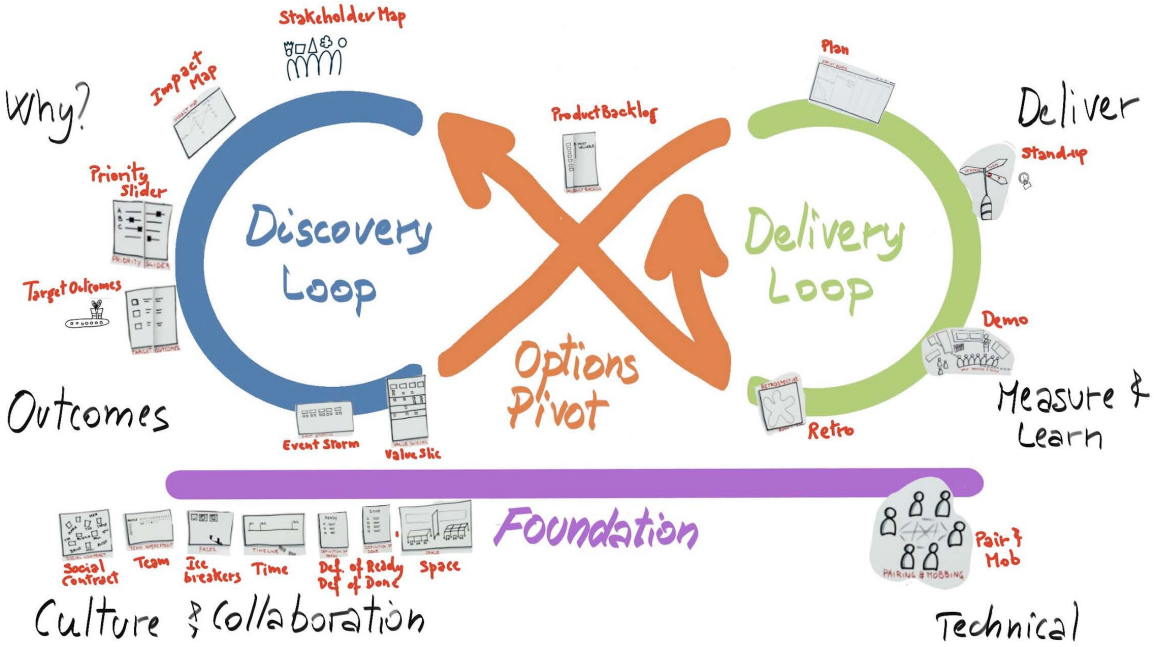


**Figure 2.16: Practicing continuous discovery and delivery through the Mobius loop**

Open Innovation Labs Residencies involves traveling around the Mobius loop a few times, usually starting from discovery before proceeding to delivery and then pivoting a few times to either more delivery or returning to discovery continuously. We find that, in order for this to be sustainable, you *must* build a foundation of culture and collaboration and you *must* build a strong foundation of technical engineering practices.

Open Innovation Labs kick-started an open source, community-driven project called the Open Practice Library. The Open Practice Library is a community-driven repository of practices and tools. These are shared by people currently using them day-to-day for people looking to be inspired with new ideas and experience.

All of the practices you read about in this book have been contributed to the Open Practice Library and, throughout the book, we will use the Mobius loop and the foundation of culture, collaboration, and technical practices as a reference point to determine where and how all our open practices fit together to deliver great DevOps culture and practice with OpenShift.

An important characteristic of Mobius and the Open Practice Library is that it is not prescriptive. It is not a methodology. It does not tell you exactly which practice to use when and where. Think of the Open Practice Library as a box of tools – a really well-organized toolbox with lots of compartments and shelves. The practices have been organized into compartments that help with discovery and, in particular, the *why* and *who*, followed by deriving outcomes. There is a drawer containing all the tools that help form, organize, and prioritize options and how to pivot later in the cycle. There is a portion of the toolbox with all of the tools that help with delivery – whether that be iterative and incremental delivery associated with Agile practices or single delivery associated with Waterfall. There are tools to help capture and understand the measurements and learning from delivery. Finally, there is a huge drawer of tools used to establish culture, collaboration, and technical engineering excellence. These are often the first tools we go to grab when starting a piece of work.

## Conclusion

In this chapter we introduced the value chain in software product delivery and explored how traditional ways of working brought inefficiencies, bottlenecks and gaps between users, business stakeholders, development teams and operational teams.

We explored some of the techniques that have been used to plug these gaps and how a balanced focus on people, process and technology is needed by all involved.

Finally, we introduced the open source navigator tool called Mobius that connects discovery and delivery in an infinite loop and can connect strategy to product to operations with a common denominator of measurable outcomes. The Open Practice Library uses mobius on a foundation of culture and technology to navigate between an evolving number of open practices - many of which will be explained in subsequent chapters.

In the next chapter, we're going to outline how we'll approach the rest of the book by introducing our case study and the structure for the remaining sections.

# 3

# The journey ahead

As we conclude the first section of this book, this chapter will explain the journey we intend to take you through the remaining sections.

This will include how we intend to not just tell you about practices and techniques but also show them in action and apply them. We'll introduce a fun case study and real-world stories to do this.

One of the challenges of writing a book intended to be read by a diverse group of people with different skill sets and backgrounds is how to write it in such a way that means it can be consumed, understood, and appreciated by all. From tech leads, infrastructure engineers, and OpenShift specialists, to Agile coaches, user experience designers, and project managers, to IT leaders and CXOs, we want you to grasp a shared understanding of what's behind all the practices being taught and the principles that underpin then.

The topics covered are going to range from how to capture behaviors in an empathy map using human-centered design practices to considering observability within applications using performance metrics. It will look at ways to help product owners prioritize value versus risk while also addressing instrumentation for applications, image tagging, and metadata!

Similar to the definition of done practice we use on our DevOps culture and practice enablement course, we're going to use a few different approaches in this book to help you with your journey:

1. Explaining the culture and practice
2. Showing the culture and practice
3. Applying the culture and practice

To explain the culture and practice, we will introduce what the practice is and why and where we've chosen to use it, and give some guidance on how to use it. In some ways, this is the easy part.

We have a saying among us that we prefer to **show, not tell**. It's easy to research and write a load of words. It's far more compelling to visually show a practice in action and the effect it is having. To show the culture and practice, we have a few techniques:

1. As much as possible, we'll aim to make use of visualization techniques such as sketch notes, diagrams, and other charts. You will have seen a few of these, beautifully drawn by Ilaria Doria, in this section already, and hopefully they have helped bring the words to life a little.
2. Where we can show you a practice in action through photographs or reproduced artifacts, we will do so. Where possible, we have made the diagrams and other visual artifacts open source, and they are available at https://github.com/devops-culture-practice-with-openshift.
3. We find stories and real-world examples the best way to explain a practice and the value it brings. So, from time to time, we will break away and tell a story that one or more of the authors have experienced connected with these practices. We'll visually tell these stories on their own by having a box around the story. Let's start with one now:

## A story about telling a practice

In December 2005, I was working in the billing workstream of a large telecommunications billing system replacement program in the UK. I'd been working on this program for 18 months already at this point. It was a 10-year program to replace all legacy billing systems with a more modern COTS software and introduce some new business capability enabling flexible and changeable product management.

I lead the billing interfaces workstream and was responsible for the delivery of interfaces between billing systems and third parties such as banks, BACS, and the treasury.

Our workstream was having our Christmas dinner in a pub near the office. We'd chosen this pub because most of us had been to the same pub 12 months previously for last year's Christmas dinner. It was funny to see so many of us in the same place around the same time of year 12 months on.

When I was there, I reflected on the last 12 months and what had been achieved by our expensive team of consultants during the period. It dawned on me that 12 months ago, we were entering the design phase of a major release of the program. We were running a series of workshops over several weeks to map out the different work products and deliverables required for the release.

12 months on, we were still in that design phase. A workstream of over 60 people had spent a year writing, rewriting, refining, and writing again design documents, more design documents, variations of design documents, technical clarification notes against design documents, and even change requests against design documents. At this point, no code had been written, no tests had been run, and no software had been released. All we had produced from 12 months was lots of design documents and lots of meetings.

I remember feeling somewhat underwhelmed by the impact of what we'd achieved in the last year. I said to myself then, *There has to be a better way of delivering software.*

Finally, we want to apply some of the culture and practices for real. To help us do that, we are going to use a simple, fun case study about a small start up organization going through some of the challenges and hurdles associated with creating a DevOps culture and establishing DevOps practices. This story will represent an anonymized account of some of the things we've seen in the field with our customers using these practices.

We'll regularly return to this story of applying DevOps culture and practices using OpenShift in shaded boxes. Let's get this rolling with the backstory – we hope you're ready for this!

## Pet Battle – The backstory

Pictures of domestic cats are some of the most widely viewed content on the internet[1]. Is this true? Who knows! Maybe it's true. What we do know is that they make a great backstory for the example application we use in this book to help explain a number of DevOps practices:

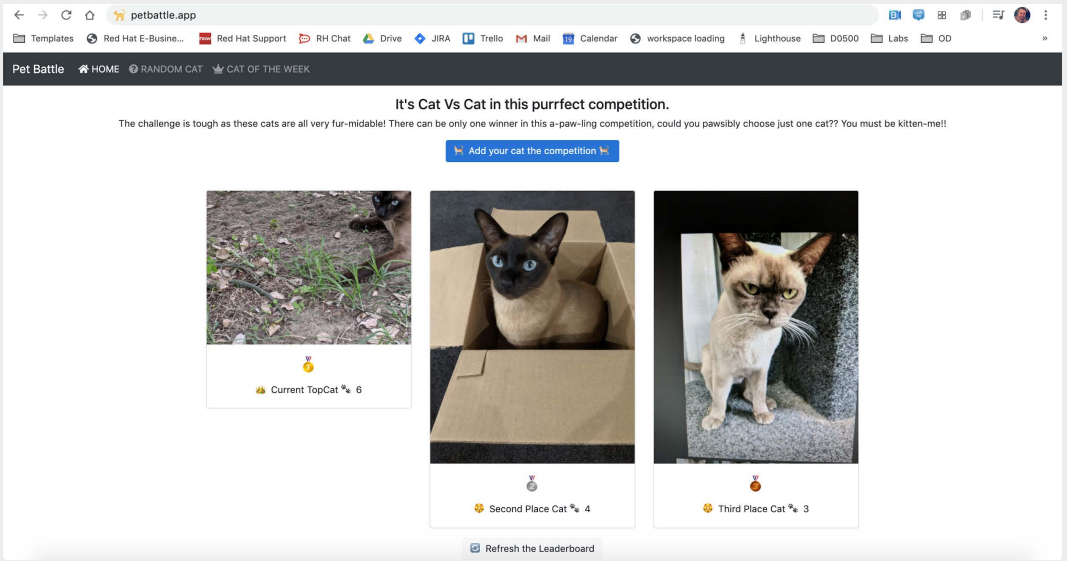


**Figure 3.1: Pet Battle – The backstory**

1 https://en.wikipedia.org/wiki/Cats_and_the_Internet

Pet Battle is a hobbyist app, started for fun, hacked around with so that the authors can **Cat versus Cat** battle each other in a simple online forum. A **My cat is better than your cat** type of thing. There are very few bells and whistles to the initial architecture – there is a simple web-based user interface and an API layer coupled with a NoSQL database.

PetBattle begins life deployed on a single virtual machine. It's online but not attracting a lot of visitors. It's mainly frequented by the authors' friends and family.

While on holiday in an exotic paradise,one of the authors happens to meet an online influencer. They date, they have a holiday romance, and PetBattle suddenly becomes Insta-famous! Nearly overnight, there is a drastically increased number of players, the PetBattle server crashes, and malicious pictures of **not** cats start appearing on the child-friendly application.

Back from holiday, the authors suddenly find themselves needing to earn a living from PetBattle and decide that developing a business and a production-ready version of the hobbyist app is now a viable thing to do.

## What about legacy systems?

People often associate Agile and DevOps with greenfield, brand-new development and see it as only applicable to start-ups and those with the luxury to start again. *What about legacy systems*? is a common question we get asked.

We'll show throughout this book that the Mobius loop and the foundation can apply to any kind of project and any kind of technology; greenfield or brownfield, small web app or large mainframe, on-premises infrastructure delivery or hybrid cloud technology.

We tend to start our journey on the Mobius loop at the *discovery* part (after building the base foundation of culture, collaboration, and technical practices). But you don't have to start there. In fact, you can start anywhere on the loop. The most important tip is to make sure you regularly travel around all parts of the loop. Do not get stuck in delivery loops and never return to discovery to revisit hypotheses and assumptions previously made. Do not get stuck in discovery where you're moving so slowly you're getting stuck in **analysis paralysis** and risk missing market windows and never delivering value. Most importantly, never forget to keep building on the foundation of culture, collaboration, and technical practices.

## Borrowing brilliance

Before we start to dive deeper into the detail, we should take a moment to point out that we did not write or dream up any of the practices in this book. The practices in this book and in the Open Practice Library are a growing list of contributions of some brilliant minds. We have borrowed that brilliance and will attribute it to the brilliant minds that came up with it. We hope we have attributed everyone correctly and any omissions are purely accidental.

What we have attempted to do with this book is show how these practices, when connected together, have delivered some highly impactful outcomes for organizations and show some of the underlying principles needed to enable those outcomes.

## What to expect from the rest of this book

So, you've almost made it to the end of this first section. Thanks for sticking with us so far! We hope you're feeling suitably enthused and motivated to read some more and have enough trust in us to want to read what we've written.

If you need just a little bit more information about what to expect, here's a short overview.

### Section 2: Establishing the foundation

In this section, we'll look much deeper into the importance of establishing a foundation both culturally and technically. We'll look again at the purpose motive – the **start with why** that we kicked off this book with and how that should be the starting point for any product or any team. We'll look at some of our favorite and most powerful practices we've used to help create the foundation of culture of collaboration – social contracts, stop-the-world andon cords, real-time retrospectives, creating team identity, and getting into the habit of visualizing everything and having a cycle of inspection and adaptation. A relentless focus on creating an environment of psychological safety is a key success factor when establishing the foundation. We'll explain what this is and how we can help achieve it.

We'll explore how executive sponsorship can enable and impede a successful foundation and explore deeper what it means to be open in terms of technology and culture. We'll look into how Agile decision making works and some of the useful tools and practices that can help with this. And we'll look at the adoption approach and how to convince the doubters and skeptics!

From a technical foundation perspective, we're going to share some of our most successful approaches, including the visualization of technology through a big picture, the **green from go** philosophy, and how we treat **everything as code**. We'll introduce some of the baseline metrics we've used to measure the success and impact of DevOps culture and practices. We'll even set the scene for some of the technical practice trade-offs and approaches to consider when creating your foundation – GitFlow- versus Trunk-based development, setting up development workflows, considering different types of testing, and setting up an environment for pairing and mobbing.

To show and not tell, establishing the foundation is about turning the picture on the left into the picture on the right:

**Figure 3.2: Collaboration within the organization**

## Section 3: Discover it

Here, we'll dive into the discovery loop of Mobius and look at some of the best ways to use it. We'll share some of our favorite and most impactful practices from the Open Practice Library that have helped us in the discovery loop, including impact mapping, human-centered design, and event storming.

We'll look at how this relates to technology and the idea of emerging architecture and enabling true continuous delivery.

From a business perspective, we'll explore the difference between outcomes and outputs and how we're trying to move from the idea of more features being better to creating powerful outcomes with fewer features. We'll explore some practices for how we can continuously measure outcomes and how we can radiate information from the entire discovery loop on open source canvases.

To show and not tell, we'll look at moving discovery from looking like what you see on the left to what you see on the right:


Figure 3.3: Practicing discovery through impact mapping, human-centric design, and event storming

## Section 4: Prioritize it

Here, we'll dive into the options pivot of Mobius and see why living, breathing, and always-changing options are important. We'll explore practices such as user story mapping and value slicing that help us with this and share some of the **gotcha** stories we have of where this has been misunderstood and misused. We'll look at how we go about building that initial product backlog using discovery that leads to options pivot practices. We'll look at different types of items that end up in product backlogs, which range from research work to experimentation work and implementation work. We'll look at some economic prioritization models and how to assess the trade-offs between value and risk with the mindset of continuous experimentation and continuous learning. We have lots of stories to share – some with a specific focus area and some with a thin thread of learning across many areas.

To show and not tell, we'll see how prioritization can go from looking like what's on the left to what's on the right:

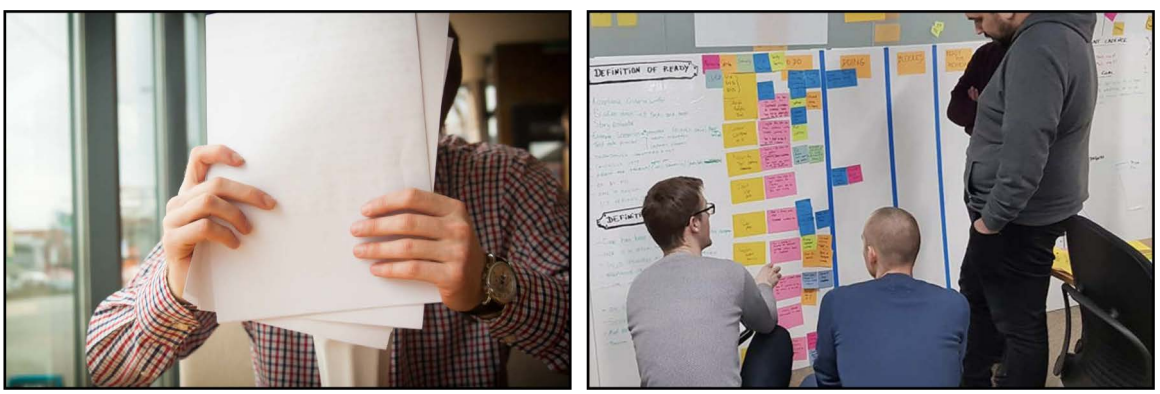


Figure 3.4: Using the Options pivot to prioritize backlog items

## Section 5: Deliver it

In this section, we'll look at Agile delivery and where and when it is applicable according to levels of complexity and simplicity. We'll also look at Waterfall and the relative merits and where it might be appropriate. We'll explore different agile frameworks out there and how all of them relate to the Open Practice Library and Mobius loop. We'll explore the importance of visualization and of capturing measurements and learning. Technology-wise, we'll look at how advanced deployment techniques now available help underpin some of the experimentation and learning approaches being driven.

To show and not tell, we'll see about getting delivery from looking like the picture on the left to something like the picture on the right:


**Figure 3.5: Practicing delivery through visualization and measurements**

## Section 6: Build it, run it, own it

This section really focuses on technology as an enabler and why it is important to have an application platform.

We'll return to the philosophy of everything-as-code and look at Git and Helm as enablers for this. We'll dive deeper into containers and the cloud-native (the cloud, the platform, and the container) ecosystem. We'll explore OpenShift and Cloud IDE, as well as pipelines that enable continuous integration, including Jenkins and Tekton. We'll explore emerging deployment and configuration approaches, such as GitOps through ArgoCD, with guidance on how and where to store configuration. We'll explore advanced deployment techniques, such as A/B testing, feature toggles, canary deployments, and blue/green deployments, and how these are used with business outcome experimentation. We'll look at non-functional aspects of DevOps practices, including OpenPolicyAgent, the scanning of images, DevSecOps, BaseImage, and chain builds. We'll look at some functional and non-functional testing. We'll explore operational aspects such as app chassis, image tagging, metadata and labeling instrumentation, Knative and serverless, and observability in terms of business versus app performance metrics. We'll reference Service Mesh and focus on operators for management and day 2 operation considerations.

To show and not tell, we'll explore taking building and running from being what you see on the left to what you see on the right:

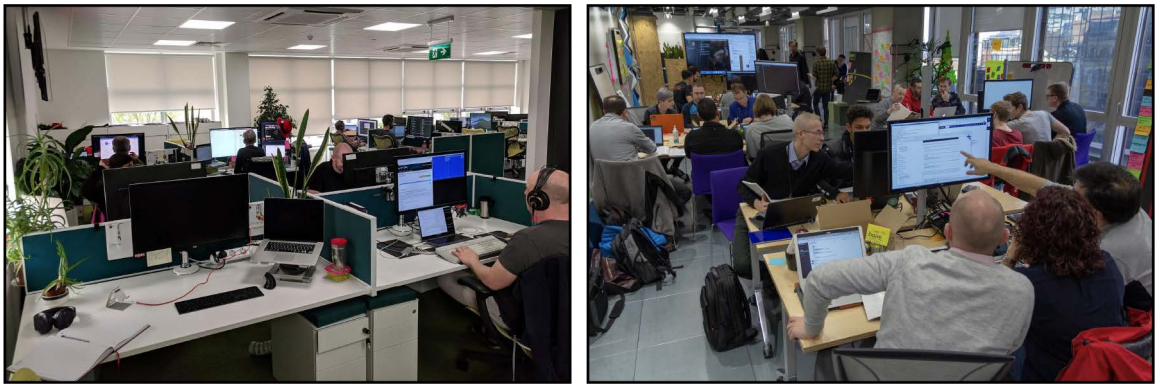

Figure 3.6: Creating the right environment for doing DevOps

## Section 7: Learn from it

As we come out of the delivery loop, we'll ask, *Did we learn enough? Should we pivot or go round the delivery loop again*? We'll see how we're entering a continuous learning cycle – not a one-time thing. Assumptions are proven or disproven during delivery loops. We explore the world of technical debt and how we can bring qualitative and quantitative metrics from the platform, the feature, and the app dev workflow to help radiate this. We'll seek how to take measurements and learning from delivery back into discovery artifacts, such as event storms, metrics-based process maps, and user research.

## Section 8: Sustain it

We'll learn how to blend everything covered in the discovery loop, the options pivot, the delivery loop, and the foundation to help sustain this way of working. This is what enables double-loop learning for continuous discovery and continuous delivery.

Long lived, cross-functional product teams learn to build it, run it, and own it. In this section, we'll look at some of the practices that help them sustain it.

What role does leadership have to play in all of this? We'll show how to visualize the connection between leadership strategy, product development, and platform operations, all being driven by intent and informed by information and metrics.

We'll explore approaches to scaling everything described in the book and how a culture of following principles is more important than pure religious use of practices.

### Some words about the world of "open"

The term **open** has been used several times in this book already and it will be used many times more. We work for an open organization, a company built on open source principles and characteristics. We're using the experiences of Open Innovation Labs to tell many of our stories, and all the practices we're using are captured and will continue to evolve in the Open Practice Library.

We strongly believe that open culture and open practices using open technology makes the best cocktail for successful transformation:


**Figure 3.7: Default to open**

## Conclusion

In this chapter, we introduced Pet Battle and the backstory of the hobbyist app that will form our fun case study we'll use throughout this book.

We also introduced how we'll regularly break out into real stories and examples from work we've done with our customers.

Finally, we set out the remaining sections of the book and what we'll explore in each of those sections.

## Thank you for reading this preview of the first section.

*To practice what we believe in, i.e. the importance of feedback loops and taking measurements early and often, we're releasing this preview to get some early feedback from you, our readers. We'd really appreciate your feedback on this first section and what you hope to read about in subsequent sections. You can provide feedback via this simple form.*