Reinventing ITIL® in the Age of DevOps

Innovative Techniques to Make Processes Agile and Relevant

Abhinav Krishna Kaiser

Apress®

Reinventing ITIL® in the Age of DevOps

Innovative Techniques to Make Processes Agile and Relevant

Abhinav Krishna Kaiser

Apress®

Reinventing ITIL® in the Age of DevOps

Abhinav Krishna Kaiser
Bengaluru, India

ISBN-13 (pbk): 978-1-4842-3975-9 ISBN-13 (electronic): 978-1-4842-3976-6
https://doi.org/10.1007/978-1-4842-3976-6

Library of Congress Control Number: 2018965617

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: Matthew Moodie
Coordinating Editor: Shrikant Vishwakarma

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-3975-9. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

To my parents,

Sandhya and Krishna Sharma,

without whom none of my successes would be possible.

Table of Contents

About the Author

Abhinav Krishna Kaiser works as a senior manager in a leading consulting firm. He consults with organizations in the areas of Agile, DevOps, and ITIL and leads programs involving the development and implementation of Agile and DevOps solutions.

He is one of the leading names synonymous with ITIL on the Web, and his previous publication, *Become ITIL Foundation Certified in 7 Days*, is one of the top guides recommended to IT professionals looking to get into the service management field and become ITIL Foundation certified.

Abhinav started consulting with clients 15 years ago on IT service management, creating value by developing robust service management solutions. He is one of the foremost authorities in the area of configuration management, and his solutions have stood the test of time, rigor, and technological advancements.

Abhinav has trained thousands of IT professionals on DevOps processes, Agile methodologies, and ITIL expert-level certifications. He blogs and writes guides and articles on DevOps, Agile, and ITIL at `http://abhinavpmp.com`. His first book came out in 2015: *Workshop in a Box: Communication Skills for IT Professionals*.

While the life of a consultant is to go where the client wants him to be, Abhinav is currently in Bangalore. He is happily married to Radhika, and they have two children Anagha, daughter and Aadwik, son.

About the Technical Reviewer

Kwok Tsang is a certified professional IT architect, ITIL expert, and accredited trainer with a proven track record in IT service management, Agile DevOps, and enterprise and solution architecture design. He has held senior positions in IT consulting firms and played key roles for customer projects in the APAC region. He has worked in multicultural DevOps teams that focus on ITIL v3 service management systems and processes, project management, business analysis, Lean IT, enterprise architecture (TOGAF), IT governance (COBIT 5), strategy and road map, systems integration, information security, enterprise solutions cloud computing, and software-defined networks. He is responsible for end-to-end solution architecture and design, including presales consulting, proposal development, delivery, knowledge transfer, and project completion. He has worked on medium to large-scale projects including cross-country systems and has designed IT and network solutions for large corporations globally.

Introduction

Reinventing ITIL® in the Age of DevOps, my third publication, came more out of necessity rather than an idea that sprang into my mind one fine morning. My clients often ask me about the relevance of ITIL today with the advent of DevOps. They question on Yammer and other social networks whether the need to implement and maintain ITIL is made redundant in DevOps projects. After a few discussions over Skype and drinks, they are convinced that the world still needs ITIL, even if the DevOps clouds are to take over. Thus started the journey of this book with my intent to reach out to a wider community, not only convincing them of ITIL's relevance but also showing them how ITIL can work in DevOps projects.

In this book, I have not shied away from getting into the details of ITIL and DevOps and mapping activities between the two. I have provided my views on all 26 ITIL processes and 5 ITIL functions and how they fit into the DevOps scheme of things. For a few major processes such as incident and change management, I have delved so deep that one can pick up my book and start implementing with minor situational adaptations.

I don't expect you to be ITIL aware or DevOps aware or Agile aware. As long as you are in the IT field and understand the concepts around software development and support, you will definitely be able to understand all the concepts that are discussed in book.

Specifically, Chapter 1 is DevOps 101 for those who are new to DevOps. Even if you are well versed in DevOps, I still encourage you to read this chapter as the understanding of DevOps varies slightly from one person to another, and there are no formal DevOps methodologies defined. While I connect the dots in the later chapters with the DevOps concepts, you will be in a position to understand the mapping from the word *go*. Chapter 2 is ITIL 101 where I introduce various ITIL concepts, the service lifecycle, processes, and other ITIL disciplines. If you are ITIL Foundation certified, you can safely skip this chapter. However, I have been witness to various misconceptions around ITIL concepts. So, it would be prudent to skim over the details of this chapter in the interest of the rest of the book.

Chapters 3 and 4 are dedicated to integration and alignment of DevOps processes with ITIL phases and processes. Yes, you read that right. Every single ITIL process (26 in total) has been analyzed from the perspective of DevOps. The various ITIL roles and DevOps roles that exist, the team structures, the synergies, and the irrelevancies are discussed in Chapter 5. Chapters 6 through 10 are dedicated to the major ITIL processes that play a significant role in DevOps projects: configuration management, incident management, problem management, change management, and release management.

The ideas presented in *Reinventing ITIL® in the Age of DevOps* are one way of combining the ITIL framework with DevOps practices. There are a number of other possible ways. Therefore, I want to hear from you, discuss more, and possibly collaborate. Please share your thoughts, feedback, and suggestions at `http://abhinavpmp.com`.

CHAPTER 1

Introduction to DevOps

New ways of working or new methodologies begin to unearth because of a problem---yes, it all starts with a problem. DevOps too had its own reasons. Businesses craved for fast turnarounds of their solutions. And often businesses found out in the midst of development that they didn't have all the information they needed to make the right decisions. They wanted to recommend a few more changes to the requirements and still expected the delivery to happen on time. DevOps was born to solve this problem.

Well, DevOps just didn't show up as the DevOps we have today. It has evolved over time. It was clear to those who started solving the problem around agility that DevOps has a lot more potential to not just solve the problem of agility but also increase productivity by leaps and bounds. Further, the quality of the software developed had the potential to be at a historic best. Thus, to this day, DevOps keeps changing and changing for the better.

DevOps is not just a methodology for developers. Operations too has its share of the benefits pie. With increased automation, operations went from being a mundane job to an innovative one. Operations folks just got themselves a new lease of life through various tools that can make their working lives a whole lot fun, and they could start looking forward to integrating and configuring tools to do advanced stuff, rather than the repetitive workload that's generally associated with operations. Here too, the productivity shot up, and human errors became a rarity.

The software development was carried out on the back of the software delivery lifecycle (SDLC) and managed through waterfall project management. On the operations front, ITIL ruled the roost. Through DevOps, development and operations essentially came together to form a union. In the mix, the waterfall methodology gave way to Agile methodologies, and still people who design DevOps processes did not have a good understanding of how ITIL would come into DevOps. So, a lot of noise started to circulate that the dawn of DevOps is the end for ITIL. This is plainly noise without any substance; you will find out through the rest of the book the value that ITIL brings to the table and why DevOps cannot exist in its entirety without a framework such as ITIL.

A. Krishna Kaiser, *Reinventing ITIL® in the Age of DevOps*, https://doi.org/10.1007/978-1-4842-3976-6_1

The book is structured around the ITIL service management framework and will explore what changes need to be made to ITIL if it needs to ease into DevOps projects. I have covered every single ITIL process and ITIL function with respect to DevOps in Chapter 4, and Chapters 5 through 10 provide in-depth analysis of major processes in ITIL around DevOps designs and implementations. You can use the contents of this book readily to start implementing ITIL in the most effective manner for it to create value in DevOps projects.

In this chapter, I briefly explain DevOps, including its principles, elements, and processes. Chapter 2 provides a snapshot of ITIL, including its lifecycle, phases, processes, and functions. I analyze DevOps and ITIL in Chapter 3, identifying the commonalities and conflicts, to support in the journey toward adapting ITIL for DevOps implementations.

What Exactly Is DevOps?

There are multiple perceptions about DevOps in the core. In fact, if you search the Web, you will be surprised to find multiple definitions for DevOps and that no two original definitions converge on common aspects and elements.

I have trained thousands in the area of DevOps, and the best answer I have is that it brings together the development and operations teams, and that's about it. Does bringing two teams together create such a strong buzz across the globe? In fact, if it actually was just the culmination of two teams, then probably DevOps would have been talked of in the human resources ecosphere, and it would have remained a semicomplex HR management process.

During the beginning of the DevOps era, to amuse my curiosity, I spoke to a number of people to understand what DevOps is. Most bent toward automation, some spoke of *that* thing they do in startups, and there were very few who spoke of it as a cultural change. Interesting! Who talks of culture these days, when the edge of our seats burn a hole if we don't act on our commitments? A particular example made me sit up and start joining the DevOps dots, and it all made sense eventually.

DevOps with an Example

Let's say that you are a project manager for an Internet banking product. The past weekend you deployed a change to update a critical component of the system after weeks of development and testing. The change was deployed successfully; however, during the post-implementation review it threw out an error that forced you to roll back the change.

The rollback was successful, and all the artifacts pertaining to the release were brought to the table to examine and identify the root cause the following Monday. Now what happens? The root cause is identified, a developer is pressed into action to fix the bug, and the code goes through the scrutiny of various tests, including the tests that were not originally included that could have caught the bug in the functional testing stage rather than in production. All tests run OK, a new change is planned, it gets approved by the change advisory board, and the change gets implemented, gets tested, and is given all green lights.

These are the typical process activities that are undertaken when a deployment fails and has to be replanned. However, the moment things go south, what is the first thing that comes to your mind as the project manager? Is it what objective action you should take next, or do you start thinking of the developer who worked in this area, the person responsible for the bug in the first place? Or do you think about the tester who identified the scenarios, wrote the scripts, and performed exploratory testing? Yes, it is true that you start to think about the people responsible for the mess. Why? It is because of our culture. We live in a culture that blames people and tries to pass the buck.

I mentioned earlier about some respondents telling me that DevOps is about culture. So, what culture am I talking about with the context of this example? The example depicts a blameful culture where the project manager is trying to pin the blame on the people within his team directly responsible for the failure. He could be factually right in pinning the blame on people directly responsible, but I am focusing on the practice involving blaming individuals.

How is this practice different from a DevOps culture? In DevOps, the responsibility of completing a task is not considered as an individual responsibility but rather a shared responsibility. Although an individual works on a task, if the person fails or succeeds, the entire team gets the carrot or the stick. Individuals are not made responsible when

we look at the overall DevOps scheme of things, and we don't blame individuals. We follow a blameless culture. This culture of blamelessness culminates from the fact that we all make mistakes because we are humans after all and far from being perfect. We make mistakes. So, what's the point in blaming people? In fact, we expect that people do make mistakes, not based on negligence but from the experimentation mind-set. This acceptance (of developers making mistakes) has led us to develop a system where the mistakes that are made get identified and rectified in the developmental stages and much before they reach production.

How is this system (to catch mistakes) built? To make it happen, we brought the development and operations teams together (to avoid disconnect), we developed processes that are far more effective and efficient than what is out there (discussed in the rest of the book), and finally we took umbrage under automation to efficiently provide us with feedback on how we are doing (as speed is one of the main objectives we intend to achieve).

DevOps is a common phrase, and with its spread reaching far and wide, there are multiple definitions coming from various quarters. No two definitions will be alike, but they will have a common theme: culture. So, for me, DevOps is a cultural transformation that brings together people from across disciplines to work under a single umbrella to collaborate and work as one unit with an open mind and to remove inefficiencies.

Note Blameless culture does not mean that the individuals who make repeated mistakes go scot-free. Individuals will be appraised justly and appropriately but in a constructive manner.

Why DevOps?

What gave rise to a new culture called DevOps, you might ask. The answer is evolution. If you take a timeline view of software, from the 1960s up to the advent of the Internet in 1990s, developing software was equivalent to a building project or launching a space shuttle. It required meticulous planning and activities that were planned to be executed sequentially. The waterfall project management methodology was thus born with five sequential steps, as indicated in Figure 1-1.

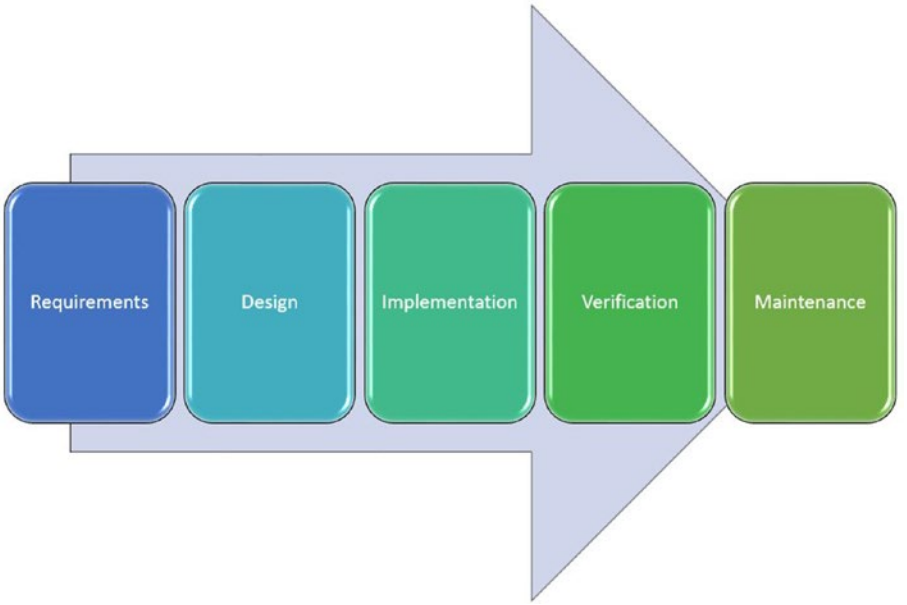

Figure 1-1. *Waterfall project management methodology*

When the Internet boomed, the software was far more accessible than the earlier era, and this generated demand not seen earlier. When the software industry started to expand, the waterfall model's limitations were exposed. The need to complete a detailed planning exercise and the sequential practice of flow did seem like an impediment to the advancement of the software industry.

Then in 2001 at a ski resort in Utah, the Agile Manifesto was born. A number of prevalent Agile methodologies came together to form a common goal that would remove the cast-in-stone waterfall sequential activities.

Agile was more fluid because they did not conceive of all of the requirements at the beginning and try to solve world hunger overnight. It was an approach that was based on iterations, where all the activities of project management just cycled over and over again. In between, if a requirement was to change, that's okay because there were provisions to make changes that were not bureaucratic nor tedious in nature. In fact, the Agile methodology puts the emphasis on the response to changes in requirements rather than a map to be followed.

The flexibility and dynamism that came about through Agile spread its wings across the software industry. A number of software projects migrated to the Agile way of working, and to this day, there are projects that are undergoing serious coaching during this transformational phase.

The Agile methodology is pretty simple where you keep things small enough to manage and large enough to be rendered meaningful. The time frames that defined iterations in Agile did not carry too much wriggle room. From an efficiency perspective, Agile was far better than the waterfall model. However, the demands from the market were out of sync with what Agile could provide. While the market shouted out for faster deliveries, the need to increase quality (reducing defect rate) was perennially being pursued. The Agile project management methodology needed something, like an elixir to run things faster. It needed automation. Enter DevOps!

Automation by itself is like giving a drone to a kid without really teaching him the process to make it fly. Generally speaking, technology by itself has no meaning if there are no underlying functional architecture, process, and embedded principles. DevOps, therefore, is not just automation but a whole lot more. You will find out the nitty-gritty details in the coming sections.

Let's Look at the Scope

The word *DevOps* gives away the scope through its conjunction of two parts of a software lifecycle. While Agile existed mainly to put an end to the rigidity brought forth by the waterfall model, it was said that the methodology can be used for operations as well. But, without an overarching process or a framework, using Agile for operations with the same rigor was not going to work. DevOps bridged this gap by bringing in the operational phases along with the developmental activities under a single umbrella and applying common processes and principles to be employed across the board.

DevOps comes into play when you get started with the software development process, which is the requirement gathering phase. It ends when the software retires from service. DevOps spans the entire wavelength of a software lifecycle, and if you read between the lines, you cannot just implement and execute DevOps until deployment and be done with it. It will continue to function until the software is used by its designated users. In other words, DevOps is here to stay, and stay for as long as services are delivered. So, in practice, the operational phase runs perpetually, and DevOps will deliver the required optimization and automation. The processes to run operations will be borrowed from the ITIL service management framework, and the present format of the ITIL framework will be highly customized to fit the DevOps bill. It is this specific ITIL fit into a DevOps project that is at the heart of this book.

Note The word DevOps came into existence thanks to Twitter. The first Devopsdays conference was held in Ghent, Belgium, in 2009. While people tweeted about it, the #devopsdays tag ate way 11 characters out of a possible 140. In a way of shortening it, one of the tweeters used #devops, and others followed suit. This led to the advent of what we know today as DevOps.

Benefits of Transforming into DevOps

A number of software companies have been delivering applications for a number of years now. Why do we need DevOps to tell us how we must develop now?

Our services are being delivered to a number of customers including top banks and mines around the globe. I am running just fine with my service management framework. Why DevOps?

People have lived for thousands of years now. They did just fine, reproducing and surviving. What has changed in the past 100 years? We have changed the modes of transport for better efficiency, we communicate faster today, and overall our quality of life has gone up several notches. *Something is working* is not a barrier for improvements to be brought about and to transform. DevOps introduces several enhancements in the areas of working culture, process, technology, and organization structure. The transformation has been rooted in practices that were developed by some like-minded organizations that were willing to experiment, and the results have vastly gone in the favor of DevOps over other ancient methodologies that still exist today.

Amazon, Netflix, Etsy, and Facebook are some of the organizations that have taken their software deliveries to a whole new level, and they don't compete anymore with the laggards. They have set new benchmarks that are impossible to meet with any of the other methodologies.

At the 2011 Velocity conference, Amazon's director of platform analysis, Jon Jenkins, provided a brief insight into Amazon's ways of working. He supported it with the following statistics:

During weekdays, Amazon is able to deploy every 11.6 seconds on average. Most organizations struggle to deploy weekly consistently, but Amazon does more than 1,000 deployments every hour (1,079 deployments to be precise). Further, 10,000 hosts receive

deployments simultaneously on average, and the highest Amazon has been able to achieve is 30,000 hosts simultaneously receiving deployments. Wow! These numbers are really *out of this world*. And these are the statistics from May 2011. Imagine what they are able to do today!

It's just not the speed of deployments. There are several added advantages that Amazon went on to claim during the conference.

- Outages owing to software deployments have reduced by a whopping 75 percent since 2006. Most outages are caused by new changes (read software deployments), and the reduction in outages points to the success achieved in deploying software changes.
- The downtime owing to software deployments too has reduced drastically, by about 90 percent.
- On an average, there has been an outage for every 1,000 software deployments, which is about a 0.001 percent failure rate. This looks great for a moderate software delivery organization, but for Amazon, the number seems high because of the 1000+ deployments every hour.
- Through automation, Amazon has introduced automatic failovers whenever hosts go down.
- Architecture complexity has reduced significantly.

Insight from State of DevOps Report

Puppet Labs publishes an annual whitepaper called the "State of DevOps Report." The report provides insight into the world of DevOps, the statistics, the changes brought about, and all the new innovations that have come about in the past year.

In the 2017 report, Puppet Labs surveyed about 3,200 professionals including executives, developers, testers, and other IT professionals. It is abundantly clear from the report that the number of people working in DevOps has gone up steeply. In 2014, 16 percent of the respondents worked in DevOps, and in 2017, it was 27 percent. DevOps is indeed quickly reaching the far-cornered IT companies, and I wouldn't be surprised if the jumps are exponential in the coming years.

- *Insights from development*: IT companies that have implemented DevOps are able to deploy 46 times faster than non-DevOps organizations. Their change lead time is a dramatic 440 times faster as well.
- *Insight from operations*: Organizations practicing DevOps have been able to recover from failures 96 times faster than their non-DevOps peers and have reduced the change failures by 80 percent.
- *Employee attraction*: It was reported that potential employees find it more appealing to work in organizations that follow DevOps methodologies than otherwise.

DevOps Principles

DevOps principles or the guidance toward the true north is in a state of constant evolution. In fact, there are multiple versions of the principles. The most widely believed set of principles is represented with the acronym CALMS. Figure 1-2 represents a mug from a marketing campaign for DevOps featuring CALMS.

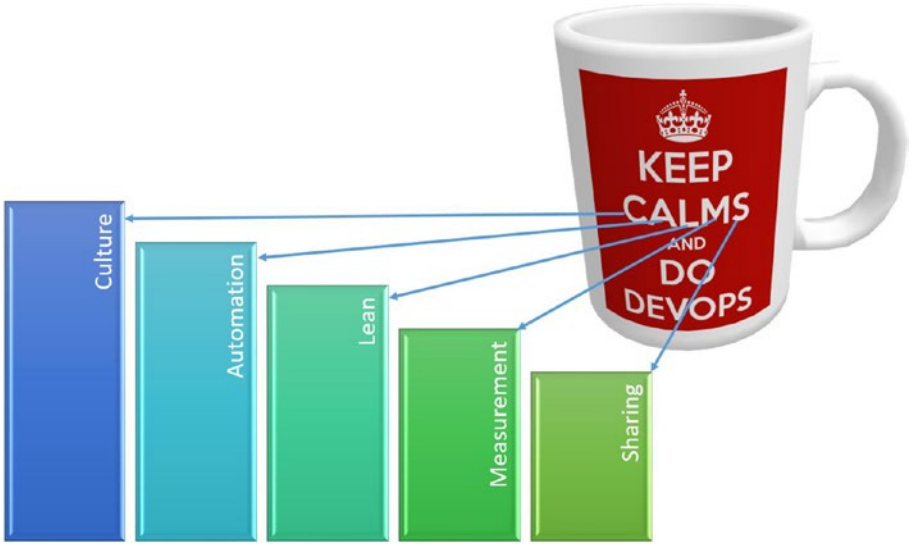

Figure 1-2. *DevOps principles (credit: devopsnet.com)*

CALMS stands for the following:

- Culture
- Automation
- Lean
- Measurement
- Sharing

Culture

There is a popular urban legend that the late Peter Drucker, known as the founder of modern management, famously said, "Culture eats strategy for breakfast." If you want to make a massive mind-boggling Earth-shaking change, start by changing the culture that can make it happen and adapt to the proposed new way of working. Culture is something that cannot be changed by a swift switching process. It is embedded into human behavior and requires an overhaul of people's behavior.

These are some of the behavioral traits that we want to change with DevOps:

- Take responsibility for the entire product and not just the work that you perform
- Step out of your comfort zone and innovate
- Experiment as much as you want; there's a safety net to catch you if you fall
- Communicate, collaborate, and develop affinity with the involved teams
- For developers especially, you build it, you run it

Automation

Automation is a key component in the DevOps methodology. It is a massive enabler for faster delivery and also crucial for providing rapid feedback. Under the culture principle, I talked about a safety net with respect to experimentation. This safety net is made possible through automation.

The objective is to automate whatever possible in the software delivery lifecycle. The kinds of activities that can be efficiently automated are those that are repetitive and those that don't ask for human intelligence. For example, building infrastructure was a major task that involved hardware architects and administrators, and most importantly building servers took a significant amount of time. This was time that was added to the overall software delivery. Thanks to the advancement of technology, we have cloud infrastructure today, and servers can be spun up through code. Additionally, we don't need hardware administrators to do it. Developers can do it themselves. Wait, there's more! Once the environment provisioning script is written, it can be used to automate spinning up servers as many times as necessary. Automation has really changed the way we see infrastructure.

Activities involving executing tasks such as running a build or running a test script can be automated. But, the activities that involve human cognizance are hard to automate today. The art of writing the code or test scripts require the use of human intelligence, and the machines of today are not in a position to do it. Tomorrow, artificial intelligence can be a threat to the activities that are dependent on humans today.

Lean

DevOps has borrowed heavily from the Lean methodology and Toyota Production Systems (TPS). The thinking behind the Lean methodology is to keep things simple and not to overcomplicate them. It is natural that the advent of automation can decrease the complexity of architecture and simplify complicated workflows. The Lean principle aids in keeping us on the ground so we can continue working with things that are easy to comprehend and simple to work with.

There are two parts to the Lean principle. The primary one is not to bloat the logic or the way we do things; keep it straightforward and minimal. An example is the use of microservices, which support the cause by not overcomplicating the architecture. We are no longer looking to build monolithic architectures that are cumbersome when it comes to enhancements, maintenance, and upgrades. A microservice architecture solves all the problems that we faced yesterday with monolithic architectures; it is easy to upgrade, troubleshoot (maintain), and enhance.

The second part of the principle is to reduce the wastage arising from the methodology. Defects are one of the key wastes. Defects are a nuisance. They delay the overall delivery, and the amount of effort that goes into fixing them is just sheer waste of time and money. The next type of waste focuses on the convoluted processes. If something can be done by passing the ball from A to B, why does it have to bounce off C? There are many such wastes that can be addressed to make the software delivery more efficient and effective.

Measurement

If you ought to automate everything, then we possibly need a system to provide feedback whenever something goes wrong. Feedback is possible if you know what the optimum results are and what aren't. The only way we can find out whether the outcome is optimum or not is by measuring it. So, it is key that you measure everything if you are going to automate everything!

Measurement principle provides direction on the measures to implement and the tabs to keep to feel the pulse of the overall software delivery. It is not a simple task to measure everything. Many times, we don't even know what we should measure. Even if we do it, the *how* part can be an obstacle. A good DevOps process architect can help solve this problem. For example, if you are running static analysis on your code, the extent of passable code must be predetermined. It is not a random number, but a scientific reasoning must be behind it. A number of companies allow a unit test to pass even if it parses 90 percent of the code. We know that ideally it must be 100 percent, so why should anybody compromise for 90 percent? That's the kind of logic that must go behind measuring everything and enabling fast feedback to be realistic about the kind of feedback that you want to receive.

In operations, monitoring applications, infrastructure, performance, and other parameters come under this principle. Measurements in monitoring will imply when an event will be categorized as a warning or an exception. With automation in place, it is extremely important that all the critical activities, and the infrastructure that supports them, be monitored and optimized for measurement.

There are other measurements as well that are attached to contracts and SLAs and are used for reporting on a regular basis. These measurements are key as well in the overall scheme of things.

Sharing

The final principle is sharing, which hinges on the need for collaboration and knowledge sharing between people. If we aim to significantly hasten the process of software delivery, it is only possible if people don't work in silos anymore. The knowledge, experience, thoughts, and ideas must be put out into the open for others to join in the process of making them better, enhanced, and profound.

One of the key takeaways of this principle is to put everyone who works on a product or a service onto a single team and promote knowledge sharing. This will lead to collaboration rather than competition and skepticism.

There are a number of collaboration tools on the market today that help support the cause. People don't even have to co-located to share and collaborate. Tools such as Microsoft Teams and Slack help in getting the information across not only to a single person but to all those who matter (such as the entire team). With information being transparent, there will be no reason for others to worry or be skeptical about the dependencies or the outcome of the process.

Elements of DevOps

DevOps is not a framework; it's a set of good practices. It got started out of a perfect storm that pooled several practices together (which will be discussed later in this chapter), and today we consider them under the DevOps umbrella. You might have seen the elephant in Figure 1-3. The IT industry around software development is so vast that a number of practices are followed across the board. This is depicted as the elephant in the figure. DevOps, which is a cultural change, can be applied to any part of the software industry and to any activity that is being carried out today. So, you can identify any part of the elephant (say testing) and design DevOps-related practices and implement them and you are doing DevOps!

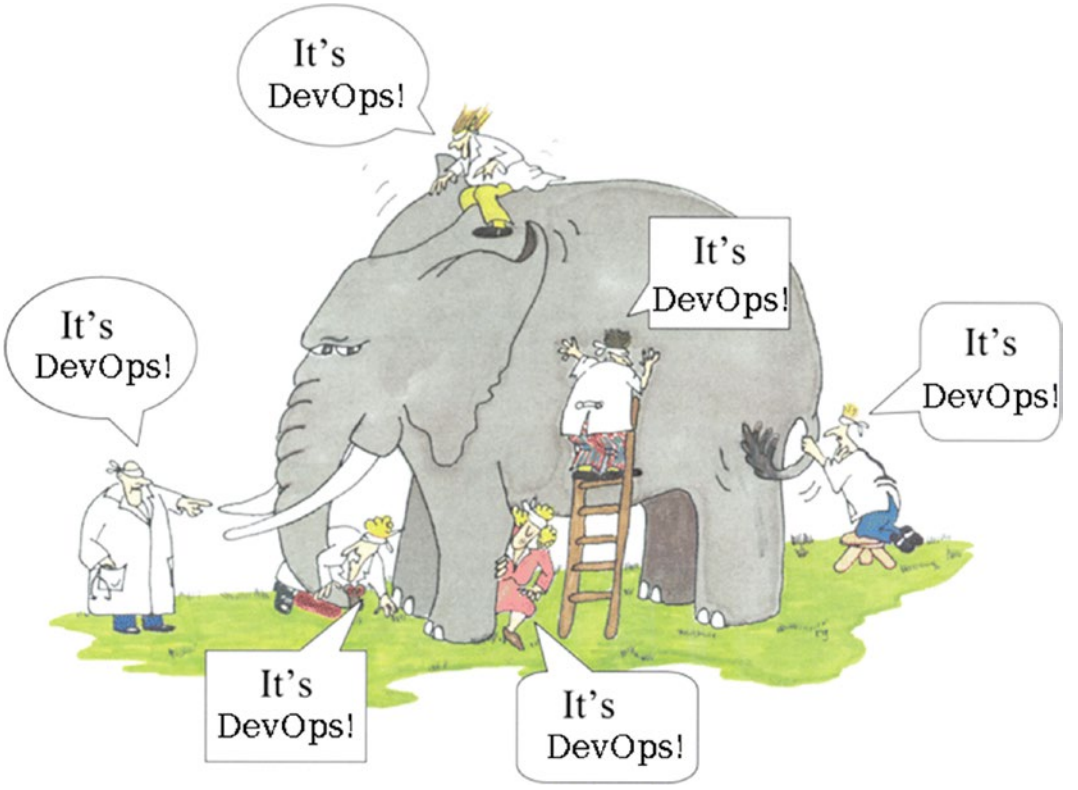

Figure 1-3. *The DevOps elephant (credit: devopsdays.org)*

No matter where you want to implement DevOps, there are three common elements that support and enable the culture change. The three sections are indicated by a Venn diagram in Figure 1-4.