Continuous Delivery (CD) and DevOps are fast becoming the next big thing(s) in relation to the delivery and support of software.

This updated edition provides you with a clear and concise insight into what CD and DevOps are all about, how to go about preparing for and implementing them, and what quantifiable business value they bring.

You will be guided through the various stages of CD and DevOps adoption, the impact they will have on you and your business, how you can overcome common problems, and what to do once CD and DevOps have become embedded in your ways of working.

Included within are some real-world examples, tricks, tips, and observations that should help ease the adoption and allow you to fully utilize CD and DevOps to deliver quality software.

Who this book is written for
If you are an IT professional, software developer, or system administrator who wants to understand how to ship quality software regularly, effectively and efficiently, this book is for you. Previous knowledge of DevOps practices, Continuous Delivery, or using DevOps tools is not necessary.

What you will learn from this book
- Explore Continuous Delivery and DevOps in depth
- Understand the root causes of the problems and pain points within your product delivery process by choosing the right techniques and tools
- Understand the human elements of CD and DevOps and how intrinsic they are to your success
- Avoid the traps, pitfalls and hurdles you'll experience as you implement CD and DevOps
- Monitor and communicate the relative success of DevOps and Continuous Delivery adoption

Deliver quality software regularly and painlessly by adopting CD and DevOps.
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 "Evolution of a Software House"
- A synopsis of the book’s content
- More information on Continuous Delivery and DevOps – A Quickstart Guide
  Second Edition

About the Author

Paul Swartout has spent over 20 years working in the IT industry. Starting out as a developer with a small software house, he has filled a number of roles over the years, including software engineer, system administrator, project manager, program manager, operations manager, scrum master, Agile coach, and software development manager. He has worked across a number of different industries and sectors—from supply chain through manufacturing, education, and retail to entertainment—and within organizations of various sizes, from start-ups to multinational corporates.

He is passionate about software and how it is delivered. Since he first encountered Agile over a decade ago, he has been committed to the adoption and implementation of Agile techniques and approaches to improve the efficiency, output, and lives of everyone involved in software development.

Over the past few years, he has been heavily involved in the CD and DevOps movement, from heading the team within Nokia that implemented said ways of working to blogging, presenting, authoring, and evangelizing to whoever is in earshot. He strongly believes that CD and DevOps add massive value to the way software is delivered, and he wants to ensure as many people realize this as possible.

Paul lives in a small seaside town in the southwest of the UK with his wife, daughters, and two small yapping things.

He is a software development manager and Agile coach working for Microsoft, based in the MixRadio team in Bristol in the UK.

He has also worked on Continuous Delivery and DevOps: A Quickstart Guide.

You can contact Paul and find out what he's up to via www.swartout.co.uk
Acknowledgments

Firstly, I would like to say a big thank you to my darling wife, Jane, who has yet again had to put up with a husband, who, for the past few months, has done little more than spend every spare moment staring at a computer screen typing things, frowning, then typing more things—things that eventually turned into this book.

Next is my good friend John Clapham, whose level-headed approach and consistent vision helped make the implementation of CD and DevOps within MixRadio the success it was. Without that success, there would be little to write about.

A big thank you to John Fisher for allowing me to include his transition curve within the book again.

Thank you to everyone who purchased and read the first edition—without you, the opportunity for this second edition would never have come to pass.

Lastly, I want to thank the global CD and DevOps community for their never-ending commitment, passion, enthusiasm, and evangelism to bring this amazing way of working to the masses. Keep up the good work.
Continuous Delivery and DevOps – A Quickstart Guide

Second Edition

Continuous Delivery (CD) and DevOps is fast becoming the next big thing in relation to the delivery and support of software. Strictly speaking, that should read the next big things, as CD and DevOps are actually two complementary yet separate approaches:

- Continuous Delivery, as the name suggests, is a way of working whereby quality products, normally software assets, can be built, tested and shipped in quick succession—thus delivering value much sooner than traditional approaches
- DevOps is a way of working whereby developers and IT system operators work closely, collaboratively, and in harmony towards a common goal with little or no organizational barriers or boundaries between them

This book will provide you with some insight into how these approaches can help you optimize, streamline, and improve the way you work and, ultimately, how you ship quality software. Included in this book are some tricks and tips based on real-world experiences and observations; they can help you reduce the time and effort needed to implement and adopt CD and DevOps, which, in turn, can help you reduce the time and effort required to consistently ship quality software.

What This Book Covers

Chapter 1, Evolution of a Software House, introduces you to ACME systems and the evolution of their business from a fledgling start-up through the growing pains following acquisition by a global corporation, to the best of both worlds.

Chapter 2, No Pain, No Gain, introduces techniques that can be used to determine the current pain points within your software delivery process and they stem from.

Chapter 3, Plan of Attack, gives you some pointers on how the success of implementing CD and DevOps can be defined and how this success can be measured.

Chapter 4, Culture and Behaviors, highlights the importance of the "human" factors that must be taken into account if you want CD and DevOps to succeed.

Chapter 5, Approaches, Tools, and Techniques, will give you some options around the various tools and techniques (some technical, some not so) that can help with the implementation and adoption of CD and DevOps.

Chapter 6, Hurdles Along the Way, will give you some useful tips and tricks to overcome or avoid the bumps in the road during the adoption of CD and DevOps.
Chapter 7, Vital Measurements, focuses on the various metrics and measures that can be used to monitor and communicate the relative success of CD and DevOps adoption.

Chapter 8, Are We There Yet?, focuses on the sorts of things you should be looking out for once the adoption of CD and DevOps has become embedded in your day-to-day ways of working.

Chapter 9, The Future is Bright, will provide some insight into how you can take CD and DevOps techniques and experience beyond the traditional software delivery process.

Appendix A, Some Useful Information, provides you with some more detailed information on the tools referenced within the book and some useful contacts within the global CD and DevOps community.

Appendix B, Where Am I on the Evolutionary Scale?, provides you with one simplistic way to determine how advanced your CD and DevOps adoption is.

Appendix C, Retrospective Games, provides example agile games that can be used in conjunction with the techniques covered in Chapter 2, No Pain, No Gain.

Appendix D, Vital Measurements Expanded, provides some additional background on and advancement of the areas covered in Chapter 7, Vital Measurements.
Evolution of a Software House

As described in the Preface, both Continuous Delivery (CD) and DevOps are complementary ways of working. The former assists with shipping quality software quickly, the latter helps harmonize the teams that deliver and support said software. Both approaches can help you to optimize, streamline, and improve the way you work. Ultimately, both will help you ship quality software.

Before we get onto the meat of CD and DevOps, let me introduce you to ACME systems—a typical software business—and walk you through their trials, tribulations, and evolution. The topics we will cover in this chapter are as follows:

• How ACME systems started from humble beginnings
• The growing pains it went through to become successful
• The positives and negatives that came from success and dramatic growth
• The advantages that came with adopting CD and DevOps ways of working
• How it adapted to utilize what it had learned to drive their business into new markets and opportunities

Without further ado, let’s meet ACME systems.

A brief history of ACME systems

This fictional software business started out—as many successful tech companies do—in the garage of one of the founders. The founders were visionaries with big ambitions, good ideas, and a little bit of cash.
After a few years of hard work, determination, much blood, sweat, and tears, the dreams of the founders were realized. The business is recognized as a leader in its field and is then acquired by a multinational corporate. This acquisition brings with it the funding and resources needed to allow the business to grow and expand to become a global player. However, with corporate owners comes corporate responsibilities, rules, bureaucracy, and processes.

The ACME systems team start to find it increasingly difficult and painful to deliver quality software. They adopt and adhere to the parent company’s processes to improve quality and reduce risk, but this makes the seemingly simple task of delivering software, laborious and extremely complex.

They come to an evolutionary crossroad and have to make a decision either to live with the corporate baggage that they have inherited and potentially face extinction, or try and get back to the good old days and good old ways that had reaped rewards previously.

While trying to decide which way to go, they discover they have another choice—implement CD and DevOps—which could give them the best of both worlds. As luck would have it that is exactly what they did.

Over the next few pages, we’ll go through this evolution in a little more detail. As we do, you may recognize some familiar traits and challenges.

The name ACME is purely fictional and based upon the ACME Corporation, first used in Road Runner Cartoons in the 1950s—just in case you were wondering.

We’ll start with the initial incarnation of the ACME systems business, which for want of a better name, will be called ACME systems version 1.0.

**ACME systems version 1.0**

Some of you have most probably worked for (or currently work for) a small software business. There are many such businesses scattered around the globe and they all have one thing in common—they need to move fast to survive and they need to entice and retain customers at all costs. They do this by delivering what the customer wants just before the customer needs it. Deliver too soon and you may have wasted money on building solutions that the customer decides they no longer need, as their priorities or simply their minds have changed. Deliver too late and someone else may well have taken your customer—and more importantly, your revenue—away from you. The important keyword here is *deliver*. 
As mentioned earlier, ACME systems started out in humble beginnings; the founders had a big vision and could see a gap in the market for a web-based solution. They had an entrepreneurial spirit and managed to attract backers who injected the lifeblood of all small businesses—cash.

They then went about sourcing some local, keen, and talented engineers and set about building the web-based solution that bridged the gap in the market, which they had seen before anyone else could.

At first, the going was slow and the work was hard; a lot of pre-sales prototypes needed to be built in a hurry—most of which never saw the light of day—some went straight into production. After many long days, nights, weeks, and weekends, things started to come together. Their customer base started to grow and the orders started rolling in; as did the revenue. Soon the number of employees was in double figures and the founders had become directors.

So, I hear you ask, "What has this got to do with CD or DevOps?" Well, everything really. The culture, default behaviors, and engineering practices of a small software house are what would be classed as pretty good in terms of CD and DevOps. For example:

- There are next to no barriers between developers and operations teams—in fact, they are generally one and the same
- Developers normally have full access to the production environment and can closely monitor their software
- All areas of the business are focused on the same thing, that being to get software into the production environment as quickly as possible and thus delight customers
- Speed of delivery is of the essence
- When things break, everyone swarms around to help fix the problem—even out of hours
- The software evolves quickly and features are added in incremental chunks
- The ways of working are normally very agile
There is a reason for stating that the culture, default behaviors, and engineering practices of a small software house would be classed as *pretty good* rather than *ideal*. This is because there are many flaws in the way a small software house typically has to operate to stay alive; for example:

- Corners will be cut to hit deadlines, which compromises software design and elegance
- Application security best practice is given short shrift or even ignored
- Engineering best practices are compromised to hit deadlines
- The concept of technical debt is pretty much ignored
- Testing is not in the forefront of the developer’s mind and even if it were, there may not be enough time to work in a test-driven development way
- Source and version control systems are not used religiously
- With unrestricted access to the production environment, tweaks and changes can be made to the infrastructure with little or no audit trail
- Software releasing will be mainly manual and most of the time an afterthought of the overall system design
- At times, a rough and ready prototype may well become production code without the opportunity for refactoring
- Documentation is scant or nonexistent—that which does exist is most probably out of date
- The work-life balance for an engineer working within a small software house is not sustainable and *burn out* does happen

To emphasize this, let's have a look at a selection of typical conversations between three individuals within the ACME systems team: Stan, the manager; Devina, the developer; and Oscar, the operations guy.
Chapter 1

Don't worry, you'll get time to rewrite it. The prototype we are now using in production keeps crashing. You're not going home until it's fixed. I'll get the pizza!

The manual hack I did last week has caused the disks to fill up and the production server has crashed. You and Devina aren't going home until it's fixed. I'll get the pizza!

We would like to invest time in implementing a source control system. Is it free? On second thoughts we don't have the time. Let's look at it later.

We would like to invest time in developing a fully automated test suite. Is it free? On second thoughts we don't have the time. Let's look at it later.

The prototype we are now using in production keeps crashing. You're not going home until it's fixed. I'll get the pizza!

This prototype is rough and ready and needs to be rewritten before we hand it over to our customers.

Don't worry, you'll get time to rewrite it.

I can manually hack the production server to improve performance and stability to overcome the issues we're having. I fully trust your judgment on this, just get it done quickly.

That will only slow things down and we don't have the time.

We want to work in a test-driven development mode.

We would like to invest some time in implementing automated server provisioning.

Is it free? On second thoughts we don't have the time. Let's look at it later.

We would like to invest some time in developing a fully automated test suite.

We want to work in a test-driven development mode.

We'll now have a look at the software delivery process for ACME systems version 1.0, which, to be honest, shouldn't take too long.
Software delivery process flow version 1.0

The following diagram gives an overview of the simple process used by ACME systems to deliver software. It's simple, elegant (in a rough-and-ready kind of way), and easy to communicate and understand.

![Diagram of software delivery process flow version 1.0](image)

An overview of the ACME systems version 1.0 software delivery process

Let's move forward a few years and see how ACME systems is doing and gain some insight into the benefits and pitfalls of being the leader in the field.

**ACME systems version 2.0**

The business has grown in size and turnover. The customer base is now global and the ACME systems software platform is being used by millions of customers on a daily basis. ACME systems is well established, well renowned, and recognized as being at the forefront in its area of expertise.

So much so that the board of ACME systems is approached by a multinational corporation and discussions are entered into regarding an acquisition. These discussions don't take long and the acquisition is completed within weeks. The board members are extremely happy, and the business as a whole sees this as a positive recognition that they have at last reached the big time.

At first, everything is good; everything is great in fact. The ACME systems team now has the backing they need to invest in the business and be able to scale out and obtain a truly global reach. They can also focus on the important things such as building quality software; scaling out the software platform; and investing in new technologies, tools, and R&D. The drier side of business—administration, program, project management, sales, marketing, and so on—can be passed to the new parent company that has all of this in place already.
Chapter 1

The ACME systems team moves forward in excited expectation. The level of investment is such that the software engineering team doubles in size in a matter of months. The R&D team—as they’re now called—introduces new tools and processes to enable speedy delivery of quality software. Scrum is adopted across the R&D team and the opportunity to fully exploit engineering best practices is realized. The original ACME systems platform starts to creak and is showing its age, so further investment is provided to re-architect and rewrite the software platform using the latest technologies. In short, the R&D team feels that it’s all starting to come together and they have the opportunity to do it right.

In parallel to this, the ACME systems operations team is absorbed into the parent’s global operations organization. On the face of it, this seems a very good idea; there are data centers filled with cutting-edge kit, global network capabilities, and scalable infrastructure. Everything that is needed to host and run the ACME systems platform is there. Like the R&D team, the operations team has more than they could have dreamed of. In addition to the tin and string, the operations team also has resources available to help maintain quality, control change to the platform, and ensure the platform is stable and available 24/7.

Sitting above all of this, the parent company also has well-established governance, program, and project management functions to control and coordinate the overall end-to-end product delivery schedule and process.

On the face of it, everything seems rosy and the teams are working more effectively than ever before. At first, this is true, but very soon, things start to take a downward turn. Under the surface, things are not that rosy at all.

We’ll shift forward another year or so and see how things are:

- It is becoming increasingly difficult to ship software—what took days, now takes weeks or even months
- Releases are getting more and more complex as the new platform grows and more integrated features are added
- Despite the advances in re-architecting and rewriting the platform, there still remains large sections of legacy code deep within the bowels of the system, which refuse to die
- Developers are now far removed from the production environment and as such are ignorant as to how the software they are writing performs, once it eventually goes live
- There is a greater need to provide proof that software changes are of the highest quality and performance before they can go anywhere near the production servers
Evolution of a Software House

- Quality is starting to suffer as last minute changes and frantic bug fixes are being applied to fit into release cycles
- The technical debt amassed during the fast and loose days is starting to cause major issues
- Project scope is being cut at the last minute as features don't fit into the release cycles, which is leaving lots of redundant code lying around
- More and more development resources are being applied to assisting releases, which is impacting on the development of new features
- Deployments are causing system downtime—planned and unplanned
- Deadlines are being missed, stakeholders are being let down, and trust is being eroded
- The business's once glowing reputation is being tarnished

The main problem here, however, is that this attrition has been happening very slowly over a number of months and not everyone has noticed—they're all too busy trying to deliver.

Let's now revisit the process flow for delivering software and see what's changed—it's not a pretty picture.

**Software delivery process flow version 2.0**

As you can see from the following diagram, things have become very complicated for the ACME systems team. What was simple and elegant has become complex, convoluted, and highly inefficient. The number of steps and barriers have increased, making it extremely difficult to get software delivered. In fact, it's increasingly difficult to get anything done. The following diagram gives you an overview of the ACME systems version 2.0 software delivery process:
An overview of the ACME systems version 2.0 software delivery process
Not only has the process become very inefficient—and to all intents and purposes broken—but the dialogue and the quality of the communication have also broken down. Let's again review a typical discussion between Devina, Oscar, and Stan regarding a live issue.

---

Some time later

```
We've no idea which one running the secure web login service.
You have to be more specific, we've got hundreds of servers. Maybe it's listed on a project plan?

We've checked through an old project plan and it's listed at DC03MM01DB16
That sounds more like it. We've found it. What information do you need?

Okay—for now I'll set the deployment parameters to use 8 GB, can you update the system configuration?
That's an infrastructure change. You'll have to raise an infrastructure change ticket.

What about spinning up a couple of new servers so we can spread the burden?
That's a DC infrastructure change. You'll have to raise a DC infrastructure change ticket.

---

Stan, can you tell Oscar I have obtained permission to view the server configuration values for one of the production servers?
I'll email him later, I'm tied up in meetings for the next hour or so and getting beaten up about these performance issues.

We've no idea which one running the secure web login service.
You have to be more specific, we've got hundreds of servers. Maybe it's listed on a project plan?

We've checked through an old project plan and it's listed at DC03MM01DB16
That sounds more like it. We've found it. What information do you need?

Okay—for now I'll set the deployment parameters to use 8 GB, can you update the system configuration?
That's an infrastructure change. You'll have to raise an infrastructure change ticket.

What about spinning up a couple of new servers so we can spread the burden?
That's a DC infrastructure change. You'll have to raise a DC infrastructure change ticket.

---

That's not good, no wonder there are performance issues. Can we increase it up to 16 GB? That's the minimum space the application needs.
You'll have to raise a change ticket!

As the heap size has changed, we need to see the results from integration, performance, and functional tests as this deployment could have an adverse impact on the production platform.

---

Devina, I'm being hassled to get this performance issue sorted today—what's the hold up?
I'myppppppHNNNN HHUH HHHHH!

Some time later

Stan, can you tell Oscar I have obtained permission to view the server configuration values for one of the production servers?
I'll email him later, I'm tied up in meetings for the next hour or so and getting beaten up about these performance issues.

We've no idea which one running the secure web login service.
You have to be more specific, we've got hundreds of servers. Maybe it's listed on a project plan?

We've checked through an old project plan and it's listed at DC03MM01DB16
That sounds more like it. We've found it. What information do you need?

Okay—for now I'll set the deployment parameters to use 8 GB, can you update the system configuration?
That's an infrastructure change. You'll have to raise an infrastructure change ticket.

What about spinning up a couple of new servers so we can spread the burden?
That's a DC infrastructure change. You'll have to raise a DC infrastructure change ticket.

---

That's not good, no wonder there are performance issues. Can we increase it up to 16 GB? That's the minimum space the application needs.
You'll have to raise a change ticket!

As the heap size has changed, we need to see the results from integration, performance, and functional tests as this deployment could have an adverse impact on the production platform.

---

We now have all the tickets raised and signed off. We're now ready to deploy this fix.

Arrrrrggggghhhhh! I quit

---

Some time later

```

Okay, so this might be a little over the top, but it just serves to highlight the massive disjoint between the R&D and Operations team(s)—who you'll remember were pretty much one and the same in the early days of ACME systems. It should also be noted that this communication is now normally done via e-mail.

**A few brave men and women**

As was previously stated, not everyone noticed the attrition within the organization—luckily a few brave souls did. A small number of the ACME systems team are able to see the issues within the overall process as clear as day and they become determined to expose them and, more importantly, sort them out—it is just a question of how to do this while everyone is going at full pelt to get software delivered at all costs.

At first, they seek out a like-minded manager who has influence within the business and helps them to form a small virtual team. They then start identifying and breaking down the immediate issues and go about implementing the following tooling to ease some of the pain:

- Build and test automation
- **Continuous Integration (CI)**
- Automated deployment and monitoring solutions

This goes some way to address the issues but there are still some fundamental problems that tooling cannot address—the culture of the organization itself and the many disjointed silos within it. It becomes obvious that all the tools and tea in China will not bring pain relief; something more drastic is needed.

The team refocuses and works to highlight this now obvious fact to as many people as they can up and down the organization, while the influential manager works to obtain backing from the senior leadership to address it—which luckily is forthcoming.

We're now going on to the third stage of the evolution where things start to come back together and the ACME systems team regains their ability to deliver quality software when it is needed.
ACME systems version 3.0

The CD team—as they are now called—gets official backing from up high and becomes dedicated to addressing the problematic culture and behaviors, and developing ways to overcome and/or remove the barriers. They are no longer simply a technical team; they are a catalyst for change.

The remit is clear—do whatever is needed to streamline the process of software delivery and make it seamless and repeatable. In essence, implement what we now commonly refer to as CD and DevOps.

The first thing they do is to simply talk with as many people across the business as possible. If someone is involved in the process of getting software from conception to consumer and support it when it’s live, they are someone you need to speak with. This not only gathers useful information but also gives the team the opportunity to evangelize and form a wider network of like-minded individuals.

The team has a vision, a purpose, and they passionately believe in what needs to be done, and have the energy and drive to do it.

Over the next few months, they embark on (among other things):

- Running various in-depth sessions to understand and map out the end-to-end product delivery process
- Refining and simplifying tooling based upon continuous feedback from those using it
- Addressing the complexity of managing dependencies and order of deployment
- Engaging experts in the field of CD to independently assess the progress being made (or not as the case may be)
- Arranging offsite CD training and encourage both R&D and Ops team members to attend the training together (it’s amazing how much DevOps collaboration stems from a chat in the hotel bar)
- Reducing the many handover and decision-making points throughout the software release process
- Removing the barriers to allow developers to safely deploy their own software to the production platform
- Working with other business functions to gain trust and help them to refine and streamline their processes
- Working with R&D and operations teams to experiment with different agile methodologies such as Kanban
Chapter 1

• Openly and transparently sharing information and data around deliveries and progress being made across all areas of the business
• Replacing the need for complex performance testing with the ability for developers to closely monitor their own software running in the production environment
• Evangelizing across all areas of the business to share and sell the overall vision and value of CD and DevOps

These initiatives are not easy to implement and it takes time to produce results but, after some months, the process of building and delivering software has transformed to the extent that a code change can be built, fully tested, and deployed to the production platform in minutes, many times per day—all at the press of a button and initiated and monitored by the developer who made the change.

Let's look again at the software delivery process flow to see what results have been realized.

**Software delivery process flow version 3.0**

As you can see from the diagram, the process looks much healthier. It's not as simple as version 1.0 but is efficient, reliable, and repeatable. Some much needed checks and balances have been retained from version 2.0 and optimized to enhance rather than impede the overall process.
This highly efficient process has freed up valuable DevOps resources so that they can focus on what they are best at—developing and delivering new software features and ensuring that the production platform is healthy and customers are again delighted.

The ACME systems team has gotten its mojo back and is moving forward with a new-found confidence and drive. They now have the best of both worlds and there's nothing stopping them.

**ACME systems version 4.0**

The ACME systems team have come through their challenges stronger and leaner but their story doesn't end there. As with any successful business, they don't rest on their laurels but decide to expand into new markets and opportunities—namely, to build and deliver mobile optimized clients to work with and complement their core web-based propositions.

With all they have learned throughout their evolution, they know they have an optimal way of working to allow them to deliver quality products that customers want, and they know how to deliver quickly and incrementally. However, the complexities of delivering code to a hosted web-based platform are not the same as the complexities of delivering code to an end consumer's mobile device—they are comparable but not the same. ACME systems also realizes that the process of delivering code to its production platform many times per day is under its control—code is being deployed to its infrastructure by its engineers using its tools—whereas it has little or no control over how its mobile clients are delivered, nor if and when the end consumer will install the latest and greatest version from the various app stores available. ACME systems also realizes that delivering a new version of its mobile clients many times per day is not viable nor welcome.

All is not lost—far from it. The ACME systems team has learned a vast amount throughout their evolutionary journey and decide to approach this new challenge as they did previously. They know they can build, test and deliver software with consistent quality. They know how to deliver change incrementally with little or no impact. They know how to support customers, and monitor and react quickly to change. They know their culture is mature and that the wider organization can work as one to overcome shared challenges. With this taken into account, here are a few of the things they decide to do:

- Agree on a realistic delivery cadence to allow for regular incremental changes without bombarding the end consumer
- Invest in new automated build, CI, and testing tools, which seamlessly integrate with and enhance the existing tooling
• Invest time and effort in nonfunctional features that will allow for greater visibility of what is running out in the wild, which again seamlessly integrates with the existing tooling and monitoring approach

• Ensure that the engineers delivering the mobile clients work closely with the backend engineers (DevOps) so that the client integrates seamlessly and doesn't cripple the existing production platform

As the ACME systems team start to look into applying their established and proven approach to the new venture, they also discover another side effect of their newly rekindled success; they need to scale their platform and they need to do it as soon as possible. Given the timescales and urgency, the ACME systems team decides to move away from relying on their own datacenter and move towards a globally distributed "cloud-based" solution. This brings with it new challenges; the infrastructure is completely different, the provisioning tools are new, and the tools used to build and deliver software are incompatible with the existing ACME systems tools. Again, they take this in their stride and forge ahead with confidence using the ways of working, techniques and approaches that are now part of their DNA.

Could the ACME systems version 1.0 business have taken on these new challenges and succeeded? It's possible, but the results would have been mixed, the risks would have been much greater, and the quality much lower. It's pretty obvious that the ACME systems version 2.0 business would have had major struggles and by the time the products had hit the market, they would have been outdated and fighting for the market share with quicker and leaner competition.

If you would like to understand where you and your business sits within the CD and DevOps evolutionary scale, please see Appendix B, Where Am I on the Evolutionary Scale?
The evolution in a nutshell
Throughout this chapter, we have been following the evolution of ACME systems; where it started, the growing pains that came from success, how it discovered that being acquired brings with it negatives as well as positives, how it overcame its near extinction by adopting CD and DevOps, and how it regained its mojo and confidence to move forward. All of this can be represented by the following simple diagram:

An overview of ACME systems evolution

Summary
The ACME systems evolution story is not atypical of the many software businesses out there today. As stated previously, you may recognize and relate to some of the traits and challenges, and you should be able to plot where you, your business, or your employer currently sit within the stages detailed.

We’ll now move from storytelling mode and start to look in more detail at some of the practical aspects of adopting CD and DevOps, starting with how one identifies the underlying problems that can—and do—stifle the delivery of quality software.
Where to buy this book

You can buy Continuous Delivery and DevOps – A Quickstart Guide

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.