Lessons from
29 DevOps Experts
On The Best Way to Make
The Transition to Continuous Delivery

Sponsored by: zend

TABLE OF CONTENTS

FOREWORD

Exploring Continuous Delivery

Innovation has changed. Gone are the days when a solitary genius holed up in a garage conceived a big idea, and then painstakingly perfected and brought it to market years later. Today, innovation is fluid, fast moving, and collaborative. Innovation is the engine for growth and value creation in the modern world, and software is the fuel.

The ability to create new, high-quality software applications and bring them to market more quickly is the "X factor" that defines industry leaders, and these leaders all have one thing in common: their IT organizations are leaving traditional approaches behind in favor of new, agile, collaborative approaches to the design, development, and delivery of applications.

At Zend, we are committed to helping companies deliver innovation more quickly. We've seen the dramatic results of this trend in working with Fiat, Hearst Corporation, BNP Paribas, Newell Rubbermaid, Prada, and other customers that are achieving faster and more frequent releases of more reliable software and, as a result, improving their business growth and profitability. Like other companies around the world, their success stems from the adoption of Continuous Delivery methodologies and best practices.

This e-book has been created for companies at virtually any stage of the journey toward Continuous Delivery. In the following pages, you'll find essays from software industry leaders whose experiences, insights, and solutions can make it a lot easier to get started, progress smoothly, and finish strong.

Wishing you the best success,
Andi Gutmans
CEO, Zend

zend

Zend helps businesses deliver innovation more quickly, on a larger scale, and across more channels than ever before. More than 40,000 companies rely on our solutions, including Zend Server, the integrated application platform for mobile and web apps. Zend Server provides superior tools for creating high-quality code, best-in-class infrastructure for moving applications from source control through deployment, and the best back-end platform for performance at Web scale. Zend helped establish PHP, which today powers more than 240 million applications and websites around the world. Visit us at www.zend.com.

Continuous Delivery is a Journey

Get buy-in

Integrate & Automate

Adopt best practices

ROI validation

Build the business case

Get started

We'll meet you wherever you are,
with the resources you need to succeed.

Learn More

> The ability to create new, high-quality software applications and bring them to market more quickly is the "X factor" that defines industry leaders.
>
> **Andi Gutmans**, CEO & Co-founder, Zend

zend

INTRODUCTION

Continuous Delivery isn't just a technical shift, it's a cultural one. Even though it takes hard work to make the transition, the benefits can't be ignored. Faster time to market, better quality product, competitive advantage, higher customer satisfaction and reduced cost of development are just a few of the benefits driving CD to become the new norm.

With the support of Zend, we reached out to 29 top DevOps professionals and asked them the following question:

Your friend has been tasked with transitioning her company's software development efforts to Continuous Delivery. She's extremely capable, but she's nervous about leading the transition. Please share a story from your own experience that will provide her with a critical piece of advice that will help her to be more successful.

The response was fantastic. Not only did we receive insightful essays, but the expert advice came from the very people who have been leading this revolution – people like Gene Kim, Andi Gutmans, Rebecca Parsons, Scott Hanselman and Andrew Yochum. The essays in this book roughly break down into six categories that range from understanding the business case for CD through actually making the journey. We hope the collective wisdom and hard-learned lessons contained in these pages will inspire you and help you take your own development efforts to a higher level.

All the best,
David Rogelberg
Editor

SECTION 1:
The Case for Continuous Delivery

BUSINESS SUCCESS—AND PERSONAL FULFILLMENT—THROUGH DEVOPS PRACTICES

GENE KIM
Author, Researcher at
IT Revolution Press

Gene Kim is a multiple-award–winning CTO, researcher, and author. He was the founder and CTO of Tripwire for 13 years. He has written three books, including "*The Visible Ops Handbook*" and "*The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*," and is part of the team writing the upcoming "*DevOps Cookbook*".

Twitter | Website | Blog

Among the benefits organizations gain when they adopt DevOps are faster time to market and reduced IT waste, both of which increase organizational effectiveness and market competitiveness.

Faster Time to Market

Back in 2007, at the IT Process Institute, we benchmarked 1,500 IT organizations to study what the high-performing IT organizations were doing differently and how they achieved their "good to great" transformation. Back then, we thought that 1,000 production changes per week was fast.

I learned from my friends at the Software Engineering Institute at Carnegie Mellon University that in every industry, high performers accelerate away from the herd. In other words, the best continue to get even better.

Thanks to practices such as continuous integration, continuous delivery and DevOps, high performers are often now doing thousands of production deployments daily! Amazon has gone on record that they are routinely performing over 23,000 deploys per day, while preserving world-class reliability, stability and security.

The ability to sustain high deployment rates (i.e., fast cycle times) translates into business value in two ways: how quickly the organization can go from an idea to delivering value to the customer and how many experiments the organization can perform simultaneously. If Organization A can perform only one deployment every nine months and its competitor can perform 10 deployments in a day, Organization A has a significant, structural competitive disadvantage.

KEY LESSONS

1. **WORK TO ACHIEVE FASTER TIMES TO MARKET.**
2. **REDUCE IT WASTE.**
3. **INCREASE ORGANIZATIONAL EFFECTIVENESS.**

> ***"Thanks to practices such as continuous integration, continuous delivery and DevOps, high performers are often now doing thousands of production deployments daily!"***

GENE KIM
Author, Researcher at
IT Revolution Press

Gene Kim is a multiple-award–winning CTO, researcher, and author. He was the founder and CTO of Tripwire for 13 years. He has written three books, including "*The Visible Ops Handbook*" and "*The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*," and is part of the team writing the upcoming "*DevOps Cookbook*".

Twitter | Website | Blog

High deployment rates also enable rapid and constant experimentation. Scott Cook, the founder of Intuit, has been an outspoken advocate for a "rampant innovation culture" at all levels of the organization. One of my favorite examples is:

> "Every employee [should be able to] do rapid, high-velocity experiments . . . . Dan Maurer runs our consumer division, including running the TurboTax website. When he took over, we did about seven experiments a year. By installing a rampant innovation culture, they now do 165 experiments in the three months of tax season. Business result? Conversion rate of the website is up 50 percent. Employee result? The folks just love it, because now their ideas can make it to market."

To me, the most shocking part of Scott Cook's story is that they were doing all these experiments during peak tax filing season! Most organizations have change freezes during their peak seasons, but if you can increase conversion rates and therefore sales during peak seasons when your competitor cannot, then that's a genuine competitive advantage. The prerequisites to doing so include being able to make many small changes quickly, without disrupting service to customers.

KEY LESSONS

1. **WORK TO ACHIEVE FASTER TIMES TO MARKET.**
2. **REDUCE IT WASTE.**
3. **INCREASE ORGANIZATIONAL EFFECTIVENESS.**

Reduced IT Waste

Mike Orzen and I have long talked about the enormous waste in the IT value stream caused by long lead times, poor hand-offs, unplanned work, and rework. We estimated how much value we could recapture by applying DevOps-like principles, calculating that if we could halve the amount of IT waste and redeploy those dollars in a way that could return 5 times what was invested, we would generate $3 trillion of value per year.

That's a staggering amount and an opportunity that we're letting slip through our fingers. That's 4.7 percent of annual global gross domestic product, or more than the entire economic output of Germany.

GENE KIM
Author, Researcher at
IT Revolution Press

Gene Kim is a multiple-award–winning CTO, researcher, and author. He was the founder and CTO of Tripwire for 13 years. He has written three books, including "*The Visible Ops Handbook*" and "*The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*," and is part of the team writing the upcoming "*DevOps Cookbook*".

Twitter | Website | Blog

This is important, especially when I think about the world my three children will inherit. The potential economic impact to productivity, standards of living, and prosperity almost makes this a moral imperative. However, there's an even greater cost. Working in most IT organizations is often thankless and frustrating. People feel as if they're trapped in an ever-repeating horror movie, helpless to change the outcome. Management abdicates their responsibility to ensure that IT is managed well, resulting in endless intertribal warfare between development, IT operations, and information security. Things only get worse when the auditors show up.

The inevitable result is chronic underachievement. As humans, we're wired to contribute and to feel like we're actively making a difference. Yet, all too often, when IT professionals ask their organization for support, they're met with "you don't understand," or worse, a barely masked, "You don't matter."

At the IT Revolution Press, our mission is to improve the livelihoods of 1 million IT workers by 2017. We hope that *The Phoenix Project* can help business and IT gain a shared understanding of the problem and that the *DevOps Cookbook* can help people fix the problem.

KEY LESSONS

1. **WORK TO ACHIEVE FASTER TIMES TO MARKET.**
2. **REDUCE IT WASTE.**
3. **INCREASE ORGANIZATIONAL EFFECTIVENESS.**

MEASURING EXPECTED ROI FOR CONTINUOUS DELIVERY

IDO BEN MOSHE

Vice President, Global Support & Professional Services at Zend

Ido Ben Moshe manages Zend's technical support, professional services, presales engineering, and solution consulting areas. In this role he leads a team of consultants that work with Zend's clients to implement Continuous Delivery. Before joining Zend, he managed the application problem resolution product line at BMC Software. Previously, he was the North America CTO at Identify Software, responsible for developing professional services, customer support, IT operations, and strategic partnerships. Ido holds a B.S. degree from Tel-Aviv University.

Website | Blog

Development leaders often approach me wanting to take their agile process to the next level and reap the rewards of implementing Continuous Delivery practices, but they need to make the business case to their boss, the chief information officer, or the chief financial officer. The business benefits of continuous delivery are very real, and the numbers prove it, time and again, for companies large and small across industries. Yet I've worked with many IT professionals as they grappled with how to measure the expected return on investment (ROI) of a process and methodology improvement. With this in mind, I created a workbook that anyone can use to calculate expected ROI for his or her organization and present their business case with confidence. The workbook is based on experiences of similar organizations and industry analysis.

KEY LESSONS

1. **USE THE STRATEGIES AND SUCCESSES OF OTHER ORGANIZATIONS THAT HAVE ALREADY IMPLEMENTED CONTINUOUS DELIVERY.**
2. **BEGIN THE MOVE TO CONTINUOUS DELIVERY BY IDENTIFYING AREAS OF GREATEST IMPACT FOR YOUR COMPANY.**

Draw on the Successes of Others

There's incredible value in having a proven methodology based on user surveys, best practices, and benchmarks from other organizations that have successful, mature DevOps practices. IT executives can use it to determine the potential revenue gains, productivity improvements, and cost reductions realized from Continuous Delivery. They can evaluate their unique application, IT staff, and environment parameters using the ROI model, and then develop a business case that is aligned with their organization's characteristics and goals.

In the workbook, gains used to calculate Continuous Delivery ROI fall into four key areas:

- Accelerated time to market with new functionality;
- Enhanced IT team productivity and reduced headcount waste;
- Reduced application failures because of increased application quality; and
- Increased flexibility in the IT environment.

> *"As you prepare to build your business case, identify the area or areas of greatest potential impact for your company, and go from there."*

MEASURING EXPECTED ROI FOR CONTINUOUS DELIVERY

IDO BEN MOSHE

Vice President, Global Support & Professional Services at Zend

Ido Ben Moshe manages Zend's technical support, professional services, presales engineering, and solution consulting areas. In this role he leads a team of consultants that work with Zend's clients to implement Continuous Delivery. Before joining Zend, he managed the application problem resolution product line at BMC Software. Previously, he was the North America CTO at Identify Software, responsible for developing professional services, customer support, IT operations, and strategic partnerships. Ido holds a B.S. degree from Tel-Aviv University.

Website | Blog

It's not unusual to see returns like these in industry surveys of organizations that have invested in DevOps and Continuous Delivery:

- **21 percent** increase in new software and services delivered;
- **22 percent** improved quality of deployed applications;
- **50 percent** fewer failures; and
- **19 percent** increase in revenues.

As you prepare to build your business case, identify the area or areas of greatest potential impact for your company, and go from there. Use the workbook to do the heavy lifting in terms of the data you need to gather and the formulas to use. You'll save time and effort and be well prepared to justify Continuous Delivery to your business leaders and decision makers. The workbook is available at https://pages.zend.com/continuous-delivery-economics.html.

KEY LESSONS

1. **USE THE STRATEGIES AND SUCCESSES OF OTHER ORGANIZATIONS THAT HAVE ALREADY IMPLEMENTED CONTINUOUS DELIVERY.**
2. **BEGIN THE MOVE TO CONTINUOUS DELIVERY BY IDENTIFYING AREAS OF GREATEST IMPACT FOR YOUR COMPANY.**

CHRIS HILTON
Lead Consultant at ThoughtWorks

Chris Hilton has been working with enterprise software as a build engineer and agile development advocate for more than 15 years. His four years at ThoughtWorks have seen him promote continuous delivery at a variety of enterprises in the United States and Canada. He has spoken on continuous delivery and his "Beyond Continuous Delivery" ideas in Seattle, Chicago, San Francisco, Calgary, Montreal, and Rome.

Twitter

I remember fondly my first experiences in programming. My mother worked for Apple in the 1980s and as an employee received a free Apple II+ computer. My father introduced me to BASIC and the power of GOTO. I spent hours typing on the keyboard, thinking up, writing, and modifying small programs—mostly for my brother's amusement. He was my first user. It was a thrill to have an idea, write the code to make it happen, and show him the result.

Of course, I grew up and learned better methods of programming than BASIC and GOTO. Software development was growing up, too, and adapted to all the complexities of the modern technological world. Ideas were formed, requirements were analyzed, designs were made, and code was written. Tests were performed, and integrations were tested; then, releases were created, deployments scheduled, and eventually a user got to see the result. Hopefully, the result resembled the original idea, and he or she liked it.

KEY LESSONS

1. **CONTINUOUS DELIVERY SHORTENS THE DISTANCE BETWEEN CREATOR AND USER.**
2. **STREAMLINING YOUR PROCESS HELPS ELIMINATE THE GUESSWORK AROUND YOUR DELIVERABLES.**

Eventually, the gap between idea and user became so big that something had to be done. Agile practices came along and started to turn back that tide. Working in iterations refines the design and development process more quickly. Unit testing gives greater confidence in coding and refactoring. Continuous integration provides rapid feedback on the state of the code. All are practices that make it that much easier to put an idea in front of a person without sacrificing any of the tenets of building modern, quality software.

The most valuable thing you do as a software developer is make that connection between creation and use. Continuous delivery takes that next step, shortening the distance between the creator's mind and the end-user's feedback. Don't guess what your customers need and like: streamline your process so that they can tell you themselves. Who knows what new ideas will arise to move us beyond even continuous delivery? Maybe someday we can all feel like kids tinkering away for our sibling's enjoyment again.

> ***"The most valuable thing you do as a software developer is make that connection between creation and use."***

START SMALL, LAUNCH EARLY, FAIL FAST, AND ITERATE: CONTINUOUS DELIVERY IN BIG SCIENCE

MIKE MILLER

Co-founder and Chief Scientist at Cloudant, an IBM Company

Mike Miller is co-founder and chief scientist at Cloudant, an IBM Company. He cut his teeth solving petabyte-per-second problems at the Large Hadron Collider. Mike holds a B.S. in physics and a B.A. in philosophy from Michigan State University and a Ph.D. in physics from Yale University. He is also an assistant professor at the University of Washington.

Twitter | Website

It's strange to think of continuous delivery as something new or intimidating: it's all I've ever known! I grew up programming for the ultra–real-time environment of experimental particle physics. When you're firing up a billion-dollar machine for the first time, even the best-designed systems have their commissioning quirks. Everything—down to the custom hardware—is continuously evolving. The first lines of production software I ever wrote went live a few minutes after commit. Thank goodness they passed the tests!

KEY LESSONS

1. **SHARED RESPONSIBILITY IS THE CORE OF A PROFESSIONAL ENTERPRISE.**
2. **START SMALL, LAUNCH EARLY, FAIL FAST, AND ITERATE.**

In big science, time is precious and expensive. Hours are measured in millions of dollars. That was a lot for a fresh graduate like myself to grasp. That pressure stemmed from opportunity, and ingraining continuous delivery mindset (the idea that "master" is essentially deployable at a moment's notice) was the best way to balance responsibility with agility. We didn't set out with a mandate to build a DevOps culture: we evolved based on our own system dynamics. Simply put, things always had to be in a production-ready state: if you wrote it, you darn well had to be there to get it running! That shared responsibility was the core principle by which our loosely federated, untrained teams were able to support such a professional enterprise.

At Cloudant, we've been able to grow that base culture into something core to our entire organization. Nobody is removed from the user experience. And we're not alone. Every market is now a land grab, and new product (new revenue!) is king. That requires rapid innovation, delivery, and operations in a way that simply isn't possible with the traditional enterprise application life cycle. In the time it takes to plan your next release, someone may have already captured the market. Adopting continuous delivery approaches for your team and aggressively weaving together cloud services is the best way to be effective in this brave new world. Start small, launch early, fail fast, and iterate. That's the mark of the new winner.

> ***"Simply put, things always had to be in a production-ready state: if you wrote it, you darn well had to be there to get it running!"***

Bring your code and user feedback closer together

Intuit founder **Scott Cook** is an advocate for a "rampant innovation culture" and allowing employees to do rapid, high-velocity experiments. Several years ago Intuit's Consumer Division took this to heart, and transformed the TurboTax website through **Continuous Delivery**.

The result?

They ran 165 experiments during the 3-month tax season. The website saw a **50% increase** in the conversion rate. The employees **loved it** because they saw their **ideas come to market**.

Gene Kim, Author and Researcher, IT Revolution Press discusses success through DevOps practices.

zend

SECTION 2:
Getting Started

IN THE DRIVER'S SEAT WITH CONTINUOUS DELIVERY

MAURICE KHERLAKIAN

Lead Architect for Continuous Delivery at Zend

Maurice Kherlakian is Zend's lead architect for Continuous Delivery and a seasoned PHP professional who has more than a decade of experience building and maintaining large-scale applications. He has also worked as a systems administrator for Windows and Linux platforms. As a key member of the Zend Professional Services team, Maurice helps customers optimize their use of PHP and Zend solutions. He specializes in PHP and Zend Framework architectural and performance audits and implementation of best practices.

Twitter | Website | Blog

Working with developers, I've seen firsthand the transformation that Continuous Delivery brings to their projects. Being able to iterate process and product changes with true agility puts them in the driver's seat, with more time for innovating and less time mired in backlogs and troubleshooting.

Even so, IT organizations that have traditionally been bound by processes, procedures, and workflows tend to see Continuous Delivery as a wholesale process change that will destabilize the IT environment and wreak havoc on their goals. They assume that a big-bang approach is required, but a phased approach to Continuous Delivery is not only preferable, it's infinitely more manageable.

KEY LESSONS

1. **WHEN STARTING THE CHANGE TO CONTINUOUS DELIVERY, START SMALL AND EXPAND SLOWLY.**
2. **GET ALL THE RELEVANT STAKEHOLDERS ON BOARD EARLY.**
3. **AUTOMATE WHEREVER POSSIBLE.**

Six Steps to Continuous Delivery

1. **Start Small.** This holds true whether your organization is embarking on its first Continuous Delivery effort or updating a legacy practice. Choose a project or work group in which change is manageable, goals can be set, and results can be measured.
2. **Get Buy-In.** Make sure the development and operations teams are on board. The Continuous Delivery paradigm is a big departure from a waterfall methodology: new tools are needed.
3. **Implement Continuous Integration.** Implement a continuous integration solution along with automating application deployment to testing and staging environments and implementing a basic set of automated unit tests that occur with every build.

> *"A phased approach to continuous delivery is not only preferable, it's infinitely more manageable."*

IN THE DRIVER'S SEAT WITH CONTINUOUS DELIVERY

MAURICE KHERLAKIAN

Lead Architect for Continuous Delivery at Zend

Maurice Kherlakian is Zend's lead architect for Continuous Delivery and a seasoned PHP professional who has more than a decade of experience building and maintaining large-scale applications. He has also worked as a systems administrator for Windows and Linux platforms. As a key member of the Zend Professional Services team, Maurice helps customers optimize their use of PHP and Zend solutions. He specializes in PHP and Zend Framework architectural and performance audits and implementation of best practices.

Twitter | Website | Blog

4. **Reinforce Best Practices, and Automate the Deploy-to-Production Pipeline.** A unit test must accompany every piece of code to ensure code functionality. Ideally, these unit tests will be automatically executed with every build, and builds will happen frequently. The entire team, including business management, should meet regularly to review which processes are working well and what needs improvement.
5. **Adopt Automation Across the Delivery Process.** Employ zero-touch continuous builds, automated system deployments, and comprehensive automated testing. Metrics, performance monitoring, and performance analytics are available to the entire team.
6. **Build on Your Successes.** Even if you start small, you can begin to reap the benefits of Continuous Delivery before a fully automated process is implemented and realize increasing benefits as your process matures. Compare your progress against release and quality metrics to see the incremental improvements along the way.

KEY LESSONS

1. **WHEN STARTING THE CHANGE TO CONTINUOUS DELIVERY, START SMALL AND EXPAND SLOWLY.**
2. **GET ALL THE RELEVANT STAKEHOLDERS ON BOARD EARLY.**
3. **AUTOMATE WHEREVER POSSIBLE.**

Continuous Delivery is an engine for innovation. In my experience, when developers get behind the wheel and learn the rules of the road, they can do amazing things.

LAYING ON A PIPELINE: IMPLEMENTING DEVOPS

OLIVIER JACQUES

HP IT–Distinguished Technologist at Hewlett-Packard

Olivier Jacques has more than 17 years of experience in software development, testing, and quality. His focus is on designing end-to-end solutions for application development that span multiple tools and processes, helping to keep HP's engineering in the "flow." He is a strong advocate of modern methods of developing and releasing applications through agile and DevOps strategies, applying them to massive-scale enterprises.

Twitter | Website

One of my passions is music. I love all kind of music, really, but I especially love electronic music. I have a band, too: we write music and lyrics and even tour Europe. Regardless of whether you like electronic music, I'm sure you have heard about one of the most successful electro-pop bands ever: Depeche Mode. In 1983, they released an album called "Construction Time Again," which features a track called "Pipeline." There's a lot to like about that track, including the massive use of sampled sounds—quite revolutionary back then. It's 2014, and you want to know everything about pipelines for your move to DevOps.

"We're laying on a pipeline."

In continuous delivery and DevOps, an important concept is the *deployment pipeline*. The deployment pipeline is your backbone—your assembly and delivery line. Code changes, infrastructure changes, and monitoring changes all flow through the deployment pipelines. You can allow some changes, deny others, or reroute them. You, supported by intelligent tests and automation, are always in control.

KEY LESSONS

1. **THE DEPLOYMENT PIPELINE HARNESSES ALL THE TOOLS IN A MEANINGFUL AND VISIBLE WAY.**
2. **AUTOMATE MOST BUT NOT NECESSARILY ALL OF THE PROCESS.**
3. **ENABLE COLLABORATION ACROSS FUNCTIONAL BOUNDARIES.**

- **Tools.** "Get out the crane/Construction time again." Recent technology evolutions, and specifically cloud computing, have made deployment pipelines even more relevant in an IT world. We are now able to treat infrastructure (compute, storage, network) as code and benefit from the same processes that have been applied to source code for years, such as team collaboration and change management (including traceability, peer reviews, defect management, and automated testing). The deployment pipeline harnesses all the tools in a meaningful and visible way.

> ***"To implement DevOps, you need to enable collaboration across functional boundaries. Here, people are key."***

LAYING ON A PIPELINE: IMPLEMENTING DEVOPS

OLIVIER JACQUES

HP IT–Distinguished Technologist at Hewlett-Packard

Olivier Jacques has more than 17 years of experience in software development, testing, and quality. His focus is on designing end-to-end solutions for application development that span multiple tools and processes, helping to keep HP's engineering in the "flow." He is a strong advocate of modern methods of developing and releasing applications through agile and DevOps strategies, applying them to massive-scale enterprises.

Twitter | Website

- **Process.** "On this golden day/work's been sent our way." A deployment pipeline leads to a process that is not only faster but safer. You're looking to automate most but not necessarily all of the process. Deployment procedures, controls, tests, triggers, reactions to monitoring alerts—the deployment pipeline structures your process, helps you communicate it, and (most importantly) improve it.
- **People.** "From the heart of the land to the mouth of the man." To implement DevOps, you need to enable collaboration across functional boundaries. Here, people are key. Deployment pipelines enable collaboration, continuous improvement, and lower the walls between the boundaries. Everything becomes visible: the changes going through the pipeline, the triggers, the feedbacks, the gates. Deployment pipelines bring people together.

You want to implement DevOps? Then, you want to know everything about deployment pipelines.

KEY LESSONS

1. **THE DEPLOYMENT PIPELINE HARNESSES ALL THE TOOLS IN A MEANINGFUL AND VISIBLE WAY.**
2. **AUTOMATE MOST BUT NOT NECESSARILY ALL OF THE PROCESS.**
3. **ENABLE COLLABORATION ACROSS FUNCTIONAL BOUNDARIES.**

MANAGING THE CHANGE

FREDERIC RIVAIN

Head of Software Development at Betclic Everest Group

Frederic Rivain is a dedicated, highly motivated chief technical officer and head of software development at Betclic. He has extensive expertise in content and media—mobile, television, or Web—from e-commerce to gaming.

Twitter | Blog

I have traveled the path to DevOps and continuous delivery with various levels of success. I have come to realize that this is not only about tools, scripts, and automation (which is in reality the easy part) but above all about a certain state of mind and change management.

As strange as it may seem, the benefits of continuous delivery are not obvious for many people. Most prefer to rely on what they're used to—even tedious, repetitive tasks—because this is the way it has always been. So, you will first need to gain the trust of your team and get them to agree to the vision.

This can be tricky, depending on the actual organization and the size of the company. You may want to start with a few pioneers on an isolated project, where you will be in a position to run a pilot to demonstrate how the new system can work.

KEY LESSONS

1. **FIRST, GAIN THE TRUST OF YOUR TEAM, AND GET THEM TO AGREE TO THE VISION.**
2. **START WITH A FEW PIONEERS ON AN ISOLATED PROJECT THAT YOU CAN USE AS A PILOT.**
3. **RELY ON YOUR BUSINESS SPONSORS.**

Showing a strong success and visible benefits is key to getting others to agree to try your way of doing things. I also encourage you to rely on your business team as sponsors, because they will be your fiercest allies and the first to advocate all the advantages of continuous delivery (after they start reaping the rewards).

Then, you can start to expand across the whole organization. Use your pilot team to explain and train other willing people, and spread the practice until you have reached a critical point where the rest of the company more naturally joins the trend.

Of course, you will have ups and downs: this is a journey with turns and bumps, but once again, the secret is that this is all about a state of mind and managing the change in the company.

> ***"Showing a strong success and visible benefits is key to getting others to agree to try your way of doing things."***

BABY STEPS TO IMPROVEMENT

ALEX SCHWARTZ

DevOps Evangelist at HERE, a Nokia business

Alex Schwartz is a speaker, coach, and lean manager who focuses on DevOps, continuous delivery, and agile testing. His journey in the industry started 20 years ago as a programmer; over the years, he assumed different roles in different organizations. Most recently, he has been working in the trenches for eBay and later for Nokia to introduce DevOps and continuous delivery to those organizations.

Twitter | Website

Imagine yourself stuck in firefighting mode, with no time left for necessary improvements. In such cases, the "baby steps" technique can be your good friend.

Here's an example. A few years ago, I joined Nokia to handle release management for a dozen Representational State Transfer (REST)-ful services. I learned quickly that releasing was non-trivial, integration testing was complex and time consuming, and deployment automation was rudimentary. As a first step, my team decided to reduce handovers by merging the quality assurance, release management, and operations teams for those services. The resulting delivery team owned the value stream to production.

During the first month, we spent more than 80 percent of our time resolving problems. Within 12 months, we spent 10 percent of our time on unplanned work. Meanwhile, we increased the number of releases by 20.

KEY LESSONS

1. **USE "BABY STEPS" TO IMPLEMENT SMALL BUT CRUCIAL CHANGES.**
2. **GIVE YOUR IMPROVEMENTS HIGH PRIORITY.**

We achieved this progress mainly by aggressively using the baby steps technique. Because we didn't have time to come up with a huge roadmap of improvements, we never created one. A big roadmap would actually have caused more harm than good, inducing more stress. After we extinguished one of our usual fires, we took a break, had a coffee, and afterwards as a little celebration we spent 10 minutes coming up with three small improvement ideas. After selecting the one with best potential—that is, the best ratio of overhead versus potential time savings—we implemented it over the next few days. We assigned a high priority to this task; because it was tiny (two hours maximum), we actually managed to resolve it before the next fire broke out.

To ensure that the baby step was tiny, we relaxed some of the usual rules for development. For example, we allowed a bit more code duplication and removed that duplication later. In favor of the KISS principle (i.e., "keep it simple, stupid"), we also tried hard to avoid generic, configurable code and used hard-coding as much as possible. Step by step, we improved our performance, our automation, and our product. Along the way, we established our belief in improvements, so we built trust in our ability to turn the situation around.

> ***"To ensure that the baby step was tiny, we relaxed some of the usual rules for development. For example, we allowed a bit more code duplication and removed that duplication later."***

ASKING QUESTIONS TO MOVE TOWARD CONTINUOUS DELIVERY

MATHIAS MEYER

CEO of Travis CI

Mathias Meyer is the CEO and "Happiness Officer" at Travis CI. He has an interest in infrastructure, distributed systems, bacon, and coffee—the latter in unnatural amounts.

Twitter | Website | Blog

Continuous delivery is a culture shift in most teams. You go from big, weekly, biweekly, even monthly releases to shipping in small increments. Rather than focus on big tasks, teams break them down into smaller pieces, shipping them whenever they're ready directly to production.

This flow poses a challenge for teams coming from the "old way" of shipping. Just like testing and continuous integration, such a transition won't happen overnight, but gentle nudges in the right direction will help foster it.

Start by focusing on how you can find out what's keeping your team from shipping sooner and faster. How can you make the build fast enough to get quick feedback? How can you improve your deployment process to get changes to production quickly? How can you ship new changes without breaking existing features for your customers?

KEY LESSONS

1. **ASK YOUR TEAM QUESTIONS, ESPECIALLY WHY THEY'RE CONCERNED WITH WORKING ON SMALL CHANGES.**
2. **DO WHAT YOU CAN TO BUILD CONFIDENCE IN YOUR TEAMS.**
3. **SEEK FEEDBACK FROM YOUR TEAMS.**

Continuous delivery starts with asking "how?," with asking your team questions. People tend to be worried about working on small changes, shipping them quickly. Start by asking them why and how you can improve the situation to give them more confidence. You'll find that beyond encouraging people, feature flags and thorough monitoring help a lot to get code to production quickly. Together with a thorough and fast test suite, they give you confidence.

Confidence is the key to implementing continuous delivery successfully. There will always be doubts, but working together, you can find solutions to remove them. With enough confidence, your team can focus on what's important—customer happiness, fixing bugs, adding new features gradually but confidently.

When you sit down with your team and start to ask questions, bring up ideas, and ask for feedback, you can start moving toward continuous delivery.

> ***"You'll find that beyond encouraging people, feature flags and thorough monitoring help a lot to get code to production quickly."***

SECTION 3:
Integrate and Automate

PAUL M. DUVALL
CEO at Stelligent

Paul M. Duvall is the CEO of Stelligent, which offers cloud delivery solutions in Amazon Web Services. Winner of the 2008 Jolt Award, he is the principal author of *Continuous Integration: Improving Software Quality and Reducing Risk* (Addison-Wesley, 2007), *DevOps in the Cloud* (Addison-Wesley, 2012) and two IBM developerWorks series on automation, DevOps, and cloud computing. He is passionate about software delivery and the cloud and actively blogs at www.stelligent.com.

Twitter | Website | Blog

It's about releasing software to users. When starting my career, one of my first jobs was as a programmer, but I didn't just write application code. I was involved in writing systems code and configuration across the entire software systems stack, including security, systems administration, diagnostics and monitoring, user role management, and later an application framework that more than 70 software developers used. I was writing code in many languages—C++, PowerBuilder, and several others I've since forgotten. In this role, I learned how the entire system was constructed across team and system silos.

The other lesson I learned is what motivates us. Releasing and receiving software motivates both the creators and the users, not silly Hawaiian shirt days or contrived "team-building" exercises. It's about releasing software.

KEY LESSONS

1. **RELEASING AND RECEIVING SOFTWARE IS WHAT MOTIVATES CREATORS AND USERS, RESPECTIVELY.**
2. **INTEGRATION IS THE KEY TO CONTINUOUS DELIVERY.**

After two long years on this project, I was thrilled when the software I helped create was used at a hospital logistics center. I immediately knew that this was what motivated my work. I think if you asked most people who develop and deliver software systems, and then talk with users, you'd get near universal agreement. Yet, we waited two years to release the software system to users. The primary reason for this delay was system complexity: a 70-person development team, software that had to be manually installed at the logistics centers, and so on. So, the work got batched and delayed and delayed. I found it maddening that it would take us days to get just the application portion (not the environment or databases) of the software system built and deployed within a simple shared integration environment.

It was also while I was on this project that I saw a single-page industry magazine advertisement with a picture of a keyboard button and the word Integrate on it. To me, it symbolized the means to an end. The more we could integrate the complete software system with recent changes, the better the chance that we could regularly release software to users while collectively motivating the software users and the software creators. This is what motivates me every day and why we create continuous delivery systems for our customers today. With continuous delivery, you're integrating the entire software system (application code, configuration, infrastructure, and data) with every code commit. In doing so, you're reducing the time it takes to release software to users while providing rapid and actionable feedback to team members, getting both the users and the creators closer to the collective goal.

> ***"I found it maddening that it would take us days to get just the application portion (not the environment or databases) of the software system built and deployed within a simple shared integration environment."***

WHAT YOU ARE CAPABLE OF WITH AUTOMATION

SCOTT HANSELMAN

Principal Program Manager at Scott Hanselman

Scott Hanselman is a web developer who has been blogging at hanselman.com for more than a decade. He works in open source on Microsoft ASP.NET and the Microsoft Azure Cloud out of his home office in Portland, Oregon. He has written many books and spoken in person to almost half a million developers worldwide.

Twitter | Website | Blog

If you do it twice, automate it. I'm consistently shocked when I find large companies opening two file windows and deploying mission-critical sites from Joe's laptop. The reasons usually come down to, "We didn't have the time" or "We lacked organizational will."

The most powerful tool we have as developers is automation: I can't stress this point enough. Humans are lousy at doing repetitive work, but computers excel at it. Look hard for the opportunity to automate processes down to a single button push. Even better, can you remove the button entirely? When you automate something, see if you can automate even further.

Years ago, I worked on a large banking system. We were proud to have automated the build and test, and then eventually got to the point where we "deployed" the binaries into a folder. However, we realized quickly that we don't ship binaries: we ship complete solutions.

KEY LESSONS

1. **EVERYTHING GETS BETTER WHEN YOU'RE NO LONGER AFRAID OF YOUR DEPLOYMENT SYSTEM.**
2. **GOOD CONTINUOUS DEPLOYMENT SYSTEMS BREED CONFIDENCE, BOTH IN THE PRODUCT AND IN THE ORGANIZATION.**

We changed our definition of *deployment* and expanded the continuous deployment system to generate a fresh virtual machine (VM) for the sales department after each build. Sales loved demoing new features but never had the time to build new demo banks. Of course not: that was our job!

We automated the build system to pop completely configured, fresh VMs out the other end. The added benefit, of course, was that we now had a build product that looked exactly like the product we sold.

Our rapidly maturing continuous deployment expanded to include better unit tests, better integration tests, and better custom acceptance tests. In fact, everything gets better when you're no longer afraid of your deployment system. Imagine the confidence we felt in our code! We made a code change and a bank appeared an hour later downstream. If a bank came out that we didn't trust, we updated our tests and deployments and tried again.

Good continuous deployment systems breed confidence, both in the product and in the organization. All that time we didn't think we had? We gained time when we got this process down. All that organizational will we thought we lacked? We gained it and more when we realized what we were capable of.

> ***"The most powerful tool we have as developers is automation."***

AUTOMATE QA FOR SUCCESSFUL CONTINUOUS DELIVERY

MICHAEL DEHAAN

CTO at Ansible

Michael DeHaan is the creator of popular DevOps automation tools like Cobbler and Ansible and co-creator of Func. He serves as CTO of Ansible, Inc., and lives in Raleigh, North Carolina.

Twitter | Website | Blog

One of the events that led to me creating Ansible was working at a hosted web application company that did infrequent releases and the challenges that doing so caused. Each release involved five or six people stuck in a conference room, manually pulling servers out of monitoring systems and load balancers, manually kicking off updates, and putting them back into load balancing and monitoring.

This process took hours, and invariably something went wrong. Update steps were skipped. Although there was manual quality assurance (QA), regression testing was difficult.

The secret to continuous deployment is automating the process. More importantly, it's about building up a culture of automated testing at both the developer and QA level:

- Deploy development environments with automation that is the same as that used in production to ensure that developers run code in an environment that simulates production.
- When developers check in source code, the continuous deployment system should run unit tests. Coverage and requiring unit tests for each piece of functionality are critically important.
- If the code passes those unit tests, deployments go to a staging environment that is a close mirror of production. Automated integration tests are then run against the staging environment.
- If those tests succeed, run the same automation against production. Embed tests in the rolling update process to ensure that servers that fail are not added back to a load-balanced pool.

Although it takes some time to get there, automated QA coupled with an automation system designed for embedded tests and rolling updates can then let you deploy a dozen times an hour with confidence. In addition, join your local DevOps group! You'll find many like-minded individuals seeking many of the same solutions.

KEY LESSONS

1. **AUTOMATE THE DEPLOYMENT PROCESS.**
2. **BUILD A CULTURE OF AUTOMATED TESTING.**
3. **JOIN A LOCAL DEVOPS GROUP.**

> ***"The secret to continuous deployment is automating the process. More importantly, it's about building up a culture of automated testing at both the developer and QA level."***

PACKING APPS FOR EASY DEPLOYMENT

KRIS BUYTAERT

CTO of Inuits

Kris Buytaert is a long-time Linux and open source consultant. In fact, he's one of the instigators of the DevOps movement. Currently with Inuits, Kris spends most of his time working to bridge the gap between developers and operations, with a strong focus on high availability, scalability, virtualization, and large infrastructure management projects. His goal is to build infrastructures that can survive the "10th-floor test," better known today as the *cloud,* while actively promoting DevOps concepts.

Twitter | Website | Blog

When talking about continuous delivery, people invariably discus their delivery pipeline and the different components that need to be in that pipeline. Often, the focus on getting the application deployed or upgraded from that pipeline is so strong that teams forget how to deploy their environment from scratch.

After running a number of tests on the code , compiling it where needed, people want to move forward quickly and deploy their release artifact on an actual platform. This deployment is typically via a file upload or a checkout from a source-control tool from the dedicated computer on which the application resides. Sometimes, dedicated tools are integrated to simulate what a developer would do manually on a computer to get the application running. Copy three files left, one right, and make sure you restart the service. Although this is obviously already a large improvement over people manually pasting commands from a 42 page run book, it doesn't solve all problems.

KEY LESSONS

1. **PACKAGE YOUR APPS AS .DEB OR .RPM ARCHIVES FOR EASY DEPLOYMENT.**
2. **WHEN PROVISIONING THE COMPUTER, SPECIFY WHICH VERSION OF YOUR APP YOU WANT TO DEPLOY BY DEFAULT.**

For example, it doesn't take into account that you want to think of your servers as cattle and be able to deploy new instances of your application fast. Do you really want to deploy your five new nodes on Amazon Web Services with a full Apache stack ready for production, then reconfigure your load balancers only to figure out that someone needs to go click your continuous integration tool to deploy the application to the new hosts? That one manual action someone forgets?

There's an easy approach to this process, and it comes with even more benefits. It's called *packaging*. When you package your artifacts as operating system (e.g., .deb or .rpm) packages, you can include that package in the list of packages to be deployed at installation time (via Kickstart or debootstrap). Similarly, when your configuration management tool (e.g., Puppet or Chef) provisions the computer, you can specify which version of the application you want to have deployed by default.

So, when you're designing how you want to deploy your application, think about deploying new instances or deploying to existing setups (or rather, upgrading your application). Doing so will make life so much easier when you want to deploy a new batch of servers.

> ***"When you package your artifacts as operating system (e.g., .deb or .rpm) packages, you can include that package in the list of packages to be deployed at installation time."***

“WHERE THE MONEY’S AT”: POSITIVE SOFTWARE EVOLUTION

AXEL FONTAINE

CEO at Boxfuse

Axel Fontaine is an entrepreneur, public speaker, and continuous delivery expert based in Munich, German. He specializes in continuous delivery and hates complexity with a passion. A regular speaker at technical conferences, Axel is the founder and project lead of Flyway—Database Migrations Made Easy. He currently works on Boxfuse, radically simplifying the deployment of applications by turning them into ultracompact, perfectly isolated, secure virtual machines within seconds and deploying them on any hypervisor with a single command.

Twitter | Website | Blog

Over the past 15 years of working with clients to improve their software systems, I found that one major challenge of building such systems is not an inability to write the code—that’s the easy part—but being able to understand the system and use that knowledge to develop the system in an efficient and cohesive way. That’s “where the money’s at.” It’s also where the going gets tough.

The complexity of a system is directly proportional to the number of moving parts. In software, the vast majority of these moving parts originate in configuration—a quick setting here, a convenient way to change something there. It all seems harmless, yet the complexity devil is already creeping in and quickly starts to create an explosion of potentially software-breaking combinations.

Taking a closer look at this, I distinguish two types of configuration in the systems. The first type is the configuration that genuinely expresses the differences between the environments. After all, you wouldn’t want to connect to your development database in production, would you? The second type consists of the elements that may change some day. They have traditionally been made configurable to compensate for slow release cycles and to provide a faster backdoor to change the system in case of emergency. Although the former configuration type is a necessary evil, the latter is not. When introducing continuous delivery; rapid, reliable releases; and short cycles to my clients, I relegate this second type of configuration to the dark corners of history. In a world where a new production deployment is just one code commit away, new possibilities open. The backdoors of the past are eradicated and replaced by a simple and reliable alternative.

In the world of continuous delivery, hard-coded is the new configuration.

> ***“The complexity of a system is directly proportional to the number of moving parts. In software, the vast majority of these moving parts originate in configuration.”***

KEY LESSONS

1. **UNDERSTANDING SOFTWARE SYSTEMS IS PARAMOUNT TO THEIR POSITIVE EVOLUTION.**
2. **MOST OF THE “MOVING PARTS” IN A SOFTWARE SYSTEM ARE CONFIGURABLE ELEMENTS.**
3. **WHEN INTRODUCING CONTINUOUS DELIVERY, RELY ON HARD-CODING.**

THE LAZY MAN'S GUIDE TO SOFTWARE SUCCESS

ANDI GUTMANS

CEO and Co-Founder of Zend

Andi Gutmans provides vision and direction for Zend, the leading provider of solutions for the rapid and Continuous Delivery of mobile and web applications. Zend's goal is to help companies deliver innovation faster. Since 1997, Andi and Zend have been key contributors to the evolution of PHP, which today powers more than 240 million websites and applications. Under Andi's leadership, Zend has partnered with IBM, Oracle, Microsoft, and other companies to advance enterprise PHP adoption. Andi holds a B.S. degree in computer science from the Technion, Israel Institute of Technology.

Twitter | Website | Blog

Years ago, I worked on F-15 fighter jet avionics simulations. The environment was complex, with more than 100 application binaries and hardware drivers written in C/C++ and other languages. When I joined the team, creating new development and production environments was a manual and tedious process. There was no easy way to set up a debug environment, and certain code changes required that many binaries be recompiled. This environment not only increased the potential for errors and inconsistencies in the code–test–debug process but also had a negative impact on software delivery velocity.

KEY LESSONS

1. USE AUTOMATION TO CREATE BUILD ENVIRONMENTS QUICKLY AND EFFICIENTLY.
2. LET AUTOMATION DO THE HEAVY LIFTING IN YOUR DEVELOPMENT ENVIRONMENT.

I quickly became frustrated by the lack of productivity and all the manual work involved. Fortunately, in my previous role as a systems administrator at a hosting provider, I had learned a lot about scripting and the environment. I like to work fast and dislike doing the same thing twice, so I was always building tools that would save time and steps and make my life easier. More time for breaks!

With this in mind, I rolled up my sleeves and decided to revamp the entire build and debug environment for the avionics simulation. I leveraged my automation skills to deliver a one-click build environment. One command was all I needed to rebuild the complete environment and install it into production. Every developer on the team had full control of his or her own environment, and no manual copying was involved. Getting an entire debug environment up and running was never more than one click away. A single code change could find its way into the test environment with one command. Looking back, my early work experience and a bias toward automation were what enabled me to have a big impact on productivity and quality for myself and my team.

In hindsight, I realize that I had adopted DevOps practices and implemented a Continuous Delivery process long before these terms existed—in an environment that was fundamentally the antithesis of Continuous Delivery. Today, with the methodology and all the great tools and best practices that exist to support DevOps and Continuous Delivery, there's no excuse not to embrace it fully. Go ahead: be a little lazy, let automation do the heavy lifting so your team can get the quality and productivity results they want, faster, without breaking a sweat.

> ***"In hindsight, I realize that I had adopted DevOps practices and implemented a Continuous Delivery process long before these terms existed—in an environment that was fundamentally the antithesis of Continuous Delivery."***

ARCHITECTURE AND CONTINUOUS DELIVERY

REBECCA PARSONS

Chief Technology Officer at ThoughtWorks

Dr. Rebecca Parsons is ThoughtWorks' CTO. She has more than 30 years' experience in leading the creation of large-scale distributed, services based applications, and the integration of disparate systems. Her interests include parallel and distributed computation, programming languages, domain specific languages, evolutionary architecture, genetic algorithms, and computational science. Rebecca received a BS in Computer Science and Economics from Bradley University, and both an MS and Ph.D. in Computer Science from Rice University.

Twitter | Blog

Reminiscing about past deployments, both good and bad, is a common occurrence when current and former operations people get together. Sadly, the focus is most often on the things that went right or wrong with getting the system live, not on the value (or lack thereof) of the functionality in the release. The bad times always seem to have involved that one little step that didn't get done properly, which actually isn't all that surprising, because these deployments tend to happen in the middle of the night.

Although the focus used to be on getting the instructions clear and right, automated deployment scripts won't miss a step, or mistype a configuration, or any of the other ways these manual deployment go horribly wrong. So, how can architecture help? In my experience, picking the right tools and knowing exactly what's running where are two of the big ways.

KEY LESSONS

1. **CHOOSE THE RIGHT TOOLS TO TAKE ADVANTAGE OF AUTOMATION.**
2. **KNOW WHAT'S RUNNING WHERE IN YOUR INFRASTRUCTURE.**
3. **USE STANDARD ENVIRONMENTS THAT HAVE KNOWN SOFTWARE VERSIONS AND CONFIGURATIONS.**

So many tools vendors seem to think that graphical interfaces are easy to use. They may be prettier, but they're much more difficult to automate. We've worked hard at times, even resorting to test automation tools, to automate deployment steps with such tools. Those making tool selections need to give high priority to the ease of deployment automation when making choices. Pretty interfaces might look nice and be easy to use in the middle of the afternoon, but things look very different at 2:00 a.m. Safe deployments are far easier to guarantee when you take humans out of the critical path.

Standard environments that have known versions of systems software and known configuration parameters also make deployments far easier. Guessing is not easy to do at 3:00 a.m., either. Being able to establish an environment to its known good state is critical. Scripted environment setup and tear-down mean you're always sure that the environment you're getting is the one you're expecting. Debugging issues on an unknown configuration brings far too many variables into the debugging equation. Ensuring environmental continuity allows troubleshooting to focus on the code that changed rather than worrying about whether it's just the environment that's problematic.

Deployment celebrations should be about the value of the new features, not joyous relief that nothing went horribly wrong. When these two things haven't been true, my life has been much harder.

> ***"Deployment celebrations should be about the value of the new features, not joyous relief that nothing went horribly wrong."***

Continuous Delivery

Six Steps to Faster Releases without Breaking Anything

More Innovation • Better Quality • Earlier Feedback • Faster Releases

Start off on the right foot.

Read the White Paper ▸

zend

SECTION 4:
Getting the Team on Board

DON'T FORCE THE CHANGE: GETTING BUY-IN FROM YOUR TEAM FOR CONTINUOUS DELIVERY

KATE MATSUDAIRA

Founder of Popforms

Kate Matsudaira is an experienced technology leader. She worked in companies like Microsoft and Amazon and three successful startups (Decide, acquired by eBay; *Moz*; and Delve Networks, acquired by Limelight) before starting her own company, Popforms. Her early career was as a software engineer, where she worked on distributed systems, cloud computing, and mobile. But she's more than just a technology leader: she managed product teams and research scientists as well as built her own profitable business. Kate is an author and keynote speaker and has been honored with Seattle's top 40 under 40 award.

Twitter | Website | Blog

Continuous delivery has many advantages. You can get things into production more quickly, it forces you to make smaller changes (which, in turn, minimizes risk), and it federates control, empowering a smart and capable team. That said, change is always difficult.

Humans like to be able to predict things. In fact, for people in operations, it's considered a strength to be able to notice potential problems before they happen. As a result, getting an operations team on board with a new idea can often mean a lot of friction. The best thing you can do to ensure a successful transition is get everyone on board early. Doing so probably means more work for you, but the effort will be well worth it.

KEY LESSONS

1. GET EVERYONE ON BOARD EARLY.
2. DON'T FORCE THE CHANGE TO CONTINUOUS DELIVERY.
3. MAKE YOUR TEAM MEMBERS PART OF THE PROCESS.

In one of my previous roles as a startup chief technology officer, I was tasked with moving the whole technology stack from one big code spaghetti-ball we had moved to the server into services that could be independently deployed and decoupled. Many of the old team members were disgruntled about the changes and favored the old system.

I knew forcing the change would only alienate these people I barely knew (not a great way to start off as a new manager), so instead I spent time with each of them. I made an effort to listen to their point of view, talk through solutions, and make them part of the process. When the changes were finally rolled out, no one was surprised, and many of the dissenters actually got on board with the solution because they had been part of creating it.

One of the most powerful things I did was ask for their help. By enlisting them to participate in the solution, there was no question about whether they were "for" or "against" it; instead, they were already involved, and we made progress as a team.

> ***"Getting an operations team on board with a new idea can often mean a lot of friction. The best thing you can do to ensure a successful transition is get everyone on board early."***

REMEMBERING THE BENEFITS OF CONTINUOUS DELIVERY

PATRICK KUA

Principal Consultant at ThoughtWorks

Patrick Kua is a speaker, writer, and consultant and stills gets to write code for customers where continuous delivery is the norm. He is the author of *The Retrospective Handbook: A Guide for Agile Teams* and bridges the worlds between the technical and nontechnical realms. Patrick is passionate about working closely with teams, helping them grow and learn with sustainable and long-term change. He is fascinated by elements of learning and continuous improvement, always helping others to develop enthusiasm for these same elements.

Twitter | Website

Continuous delivery is the normal way in which we work with clients, but it's easy to forget the benefits that continuous delivery can bring to them. Last year, I worked with a client to deliver a software system that was used to demonstrate the capabilities of a standard, an otherwise dry and theoretical topic that no one really understood.

The client used the system heavily during meetings with large, well-known international companies to demonstrate the potential benefits of global acceptance of the standard. During one meeting with an important company, the company representative asked whether the standard would enable a particular key feature. It sounded like without this feature, support for the standard would fall by the wayside.

Our team knew that this feature was trivial to implement but had been deprioritized in favor of work on other system features. One of our developers implemented the feature; with our continuous delivery pipeline, it was quickly made available in the production system. Instead of simply talking about what the standard supported, we could demonstrate how the standard enabled the feature.

The company representative who asked this question was impressed, not only by what the standard enabled but also by how quick software could be turned around. Their experience with software releases often meant three-month release cycles, endless paperwork, and frustration that the software they asked for was far from what they expected.

By demonstrating the benefits that continuous delivery can bring, the client opened a separate conversation about how software can be delivered in a different manner.

KEY LESSON

1 **BY DEMONSTRATING THE BENEFITS THAT CONTINUOUS DELIVERY CAN BRING, THE CLIENT OPENED A SEPARATE CONVERSATION ABOUT HOW SOFTWARE CAN BE DELIVERED IN A DIFFERENT MANNER.**

> ***"One of our developers implemented the feature; with our continuous delivery pipeline, it was quickly made available in the production system."***

PROTOTYPING FOR CONTINUOUS DELIVERY

MATTHEW SKELTON

Director and Co-founder of Skelton Thatcher Consulting Ltd

Matthew Skelton is an independent software systems consultant who helps organizations to evaluate, adopt, and sustain good software delivery and operations practices, such as continuous delivery, DevOps, and ITIL, with a focus on software operability. He founded and runs the London Continuous Delivery meet-up group and instigated PIPELINE, the continuous delivery conference. Matthew is director and co-founder of Skelton Thatcher, a software operations consultancy.

Twitter | Website | Blog

Two things that can really help to make continuous delivery happen are empathy and a careful prototype. I recently helped to introduce continuous delivery at a large online travel retailer in the United Kingdom. We began by focusing on a set of online tools and applications lying outside the main booking flow, partly because these applications had previously been built and deployed manually but also because we felt that we would make more rapid progress with these components than with the main booking engine.

Many changes to the build, test, and deployment of the applications were needed, including new ways of using remote machines to trigger and control deployments. These changes required new firewall and network configurations that had not been used before, and we knew that unless we had a convincing case, such changes were likely to be rejected as too risky.

KEY LESSONS

1. **DEVELOP END-TO-END PROTOTYPES TO CAPTURE IMPORTANT WORKFLOWS.**
2. **EMPATHIZE WITH THOSE RESPONSIBLE FOR BUILD, DEPLOYMENT, AND TEST.**

Using ThoughtWorks GO to model the deployment pipeline, we produced an end-to-end prototype that captured some important workflow, auditing, and approval steps that up to that point had been done manually. We realized that many people involved in the build–deployment–test process would find the level of automation scary at first. We tried to empathize with their situation and, using role-based security in the deployment pipeline, uncovered enough information to give them a sense of visibility and control.

We then approached key people individually (such as security, application support, and infrastructure), walking them through the prototype-deployment pipeline and showing them the kinds of controls and checks we had considered and prototyped. After a few iterations, we had a deployment pipeline that all involved were happy to try, and we deployed to production soon afterward. The scheme used for building and deploying the online tools in this way became a model for other components to follow.

By empathizing in advance with the concerns of other people and addressing these concerns in a visible, browser-accessible way in a prototype deployment pipeline, you can remove many of the blockers to moving to continuous delivery.

> ***"We tried to empathize with their situation and, using role-based security in the deployment pipeline, uncovered enough information to give them a sense of visibility and control."***

SECTION 5: Changing Culture and Building Confidence

GETTING YOUR ORGANIZATION EXCITED ABOUT CONTINUOUS DEVELOPMENT

TOMMY TYNJÄ

Continuous Delivery Consultant at Diabol AB

Tommy Tynjä is a continuous delivery consultant and open source software contributor who is passionate about continuous delivery, test-driven development, automation, and tools that boost developer productivity and get software shipped faster. Working primarily with Java-based solutions, he has several years of developer and architect experience, ranging from small start ups to enterprises and everything in between.

Twitter | Website | Blog

Continuous delivery is about making sure an application is always in a releasable state, where manual, repetitive, and error-prone tasks related to a release, such as setting up servers and middleware, have been fully automated. Many discussions on continuous delivery focus on the automation and tooling aspects, but it's important to remember that continuous delivery requires more than just tools and products to succeed. To successfully implement continuous delivery, you need to change the culture of how an entire organization views software development efforts.

KEY LESSONS

1. **CONTINUOUS DELIVERY REQUIRES MORE THAN JUST TOOLS AND PRODUCTS TO SUCCEED: YOU MUST CHANGE THE ORGANIZATION'S CULTURE.**
2. **EVERY MEMBER THE INITIATIVE AFFECTS MUST UNDERSTAND THE BENEFITS OF THE IMPLEMENTATION AND HOW IT WILL AFFECT THE WAY THEY DO THEIR DAILY WORK.**

Change is easy to introduce in an organization. The difficult part is making it stick. This is especially true when starting an effort to introduce continuous delivery in an organization, whether it's imposed by management or an initiative from developers tired of their unproductive, old-fashioned way of working. Every member the initiative affects must understand the benefits of the implementation and how it will affect the way they do their daily work. This is typically not done through one or a couple of formal meetings, where the organization's management presents the continuous delivery efforts. This is something that has to be done continuously; it has to be rooted in every employee working in the software development departments.

It is also important that all employees feel that they have the proper forum in which to discuss how continuous delivery will affect their daily ways of working. Even after continuous delivery has been introduced, team members will need reminders of why, for example, manual changes to environments are bad or how shortcutting the deployment pipeline negatively affects the overall quality of the system and undermines the continuous delivery principles as a whole.

The biggest boost an organization can get when implementing continuous delivery is to get people excited and passionate about it.

> ***"To successfully implement continuous delivery, you need to change the culture of how an entire organization views software development efforts."***

SMALL STEPS TOWARD FULL AUTOMATION

JAMIE INGILBY

Software Development Manager

Jamie Ingilby is a software development manager based in the United Kingdom. He has worked in many large and small technology organizations, supporting Web services. He is passionate about software development and is a huge advocate of continuous delivery as a means of producing quality software at pace with confidence. He enjoys good coffee and tea, particularly Earl Grey.

Twitter | Website | Blog

I worked in a large organization that operated in a heavily regulated environment where uptime was critical. Conventional wisdom had embedded a mindset that fewer changes meant lower risk, while the business wanted to become more reactive to the marketplace.

A small number of pilot delivery teams were tasked with moving toward a continuous delivery approach, but the teams initially found it difficult to gain traction. Automated functional testing and deployment were easy to get started with, but the teams found it difficult to automate for such parts of the pipeline as security, configuration, and infrastructure, which other internal teams historically owned.

Continuous delivery requires cultural adoption of the principles across the organization. This is especially important in areas of the business not traditionally used to adopting the continuous delivery mindset. Specialist infrastructure teams, for example, were still used to putting major changes expected to boost performance in production first. Our initial approach was to try to obtain buy-in from across the organization by reinforcing the benefits and principles of continuous delivery, but it's easy for people to brush this message off without real cultural change. We realized that empowerment and ownership with those who encounter the pain of manual processes would help. We built role-combined delivery teams that were encouraged to take full ownership of their own delivery pipelines and that worked to automate as much as possible. Small steps toward continuous delivery in fully automated testing and deployment, helped to increase the frequency of and our confidence in our releases.

KEY LESSONS

1. **CONTINUOUS DELIVERY REQUIRES CULTURAL ADOPTION OF THE PRINCIPLES ACROSS THE ORGANIZATION.**
2. **EMPOWERMENT AND OWNERSHIP WITH THOSE WHO ENCOUNTER THE PAIN OF MANUAL PROCESSES HELPS.**
3. **AS YOU REFINE YOUR PIPELINE, SHINE A SPOTLIGHT ON AREAS THAT ARE OUTSIDE THE SCOPE OF THE EXISTING PIPELINE.**

As we refined our immature pipeline, it helped to shine a spotlight on areas that were outside the scope of the existing pipeline; teams began to challenge the existing processes. The quick wins and early steps had helped to initiate a cultural shift, which in turn helped all areas of the organization understand what we were trying to achieve. When we had a common understanding of what we wanted the result to be, we were able to consider new approaches for automating the parts that remained. Continuous delivery in some environments can be a long road; don't be afraid to start with small, incremental steps that demonstrate value, and use that to build momentum.

> ***"Small steps toward continuous delivery in fully automated testing and deployment, even with manual steps built into the process, helped to increase the frequency of and our confidence in our releases."***

AVOIDING THE EASY WAY: SELLING CONTINUOUS DELIVERY IN YOUR BUSINESS

MARK NELSON

Architect at Oracle

Mark Nelson is an architect on the Platform Architecture team in Oracle Development. Mark's focus is continuous delivery, configuration management, and provisioning, making it simple to manage the configuration of complex environments and applications built with Oracle Database, Fusion Middleware, and Fusion Applications on-premises and in the cloud. Before joining the Platform Architecture team, Mark had been a senior member of the A-Team since 2010 and has worked in the IT industry since 1994.

Twitter | Blog

When you introduce continuous delivery into your company, you're giving people a new capability that will save them time and improve reliability, visibility, repeatability, and quality. But that does not mean that they will use it—or even want to.

No matter how significant the benefits, no matter how well you demonstrate and quantify those benefits, people tend to be resistant to change. You should be prepared to "sell" it to all of the stakeholders and to take an active, hands-on role—to sit with people, one after another, and help each one develop the understanding and comfort he or she needs to make the investment of time to move to something new.

KEY LESSONS

1. **BE PREPARED TO "SELL" CONTINUOUS DELIVERY TO STAKEHOLDERS.**
2. **DON'T COMPROMISE ON QUALITY OR RELAX THE RULES.**

You're also going to shine a light on and possibly exacerbate many existing problems in your current build–test–release processes—even in your source and binary management processes. It's easy for people to blame these "problems" on the new approach, certainly much easier than admitting that the issues were always there. Introducing continuous delivery provides an opportunity to go back and correct some of those problems from the past. Many of today's issues are steps that were taken for perfectly valid tactical reasons, but somehow no one ever got around to replacing them with something more strategic.

Be steadfast on the goal: do not compromise on quality. If you relax the rules, even a little, and let people find a way around things that are difficult, they will take the easy way out every time. Remember, one of the key tenets of continuous delivery is to bring the things that are difficult forward in the project to reduce risk.

It's going to be a bumpy ride, so keep a firm hand on the wheel.

> ***"Introducing continuous delivery provides an opportunity to go back and correct some of those problems from the past."***

THE FREEDOM TO BE CREATIVE

STEIN INGE MORISBAK

Manager at BEKK Consulting AS

Stein Inge Morisbak is practice lead for continuous delivery and DevOps at BEKK. He is a true agile evangelist, with 15 years of experience contributing to and helping others to become better at producing excellent software together with demanding customers. He is also an experienced speaker at conferences and the founder of DevOps Norway Meet-Up.

Twitter | Website | Blog

Ever since I started in software development, I have worked with skilled and highly educated colleagues. They have been product owners, project managers, architects, testers, systems administrators, and programmers. Having all these roles on a project, you would think that every aspect of a software project would be covered and that we would succeed every time. Yet our projects failed, as unfortunately most software projects do.

At some point, I was asked to be tech lead on a smaller project. The project sponsor wanted to follow the project closely, so we formed a close-knit team around her. I chose seven other people for the project and asked for servers to host the software we were going to build. The project was amazingly successful. We delivered a valuable product and with fewer errors than anything I had been involved with before. Adding new features was easy, and the project sponsor was extremely satisfied when she saw her ideas running in production soon after they were born. We were practicing continuous delivery.

KEY LESSONS

1. **CAREFULLY CONSIDER ANY ROLES INVOLVED IN A PROJECT THAT DO NOT DIRECTLY CONTRIBUTE TO THE GOAL.**
2. **DELIVER SOFTWARE INCREMENTALLY TO FAIL AS EARLY AS POSSIBLE SO THAT YOU CAN CORRECT ERRORS BEFORE DEPLOYMENT.**

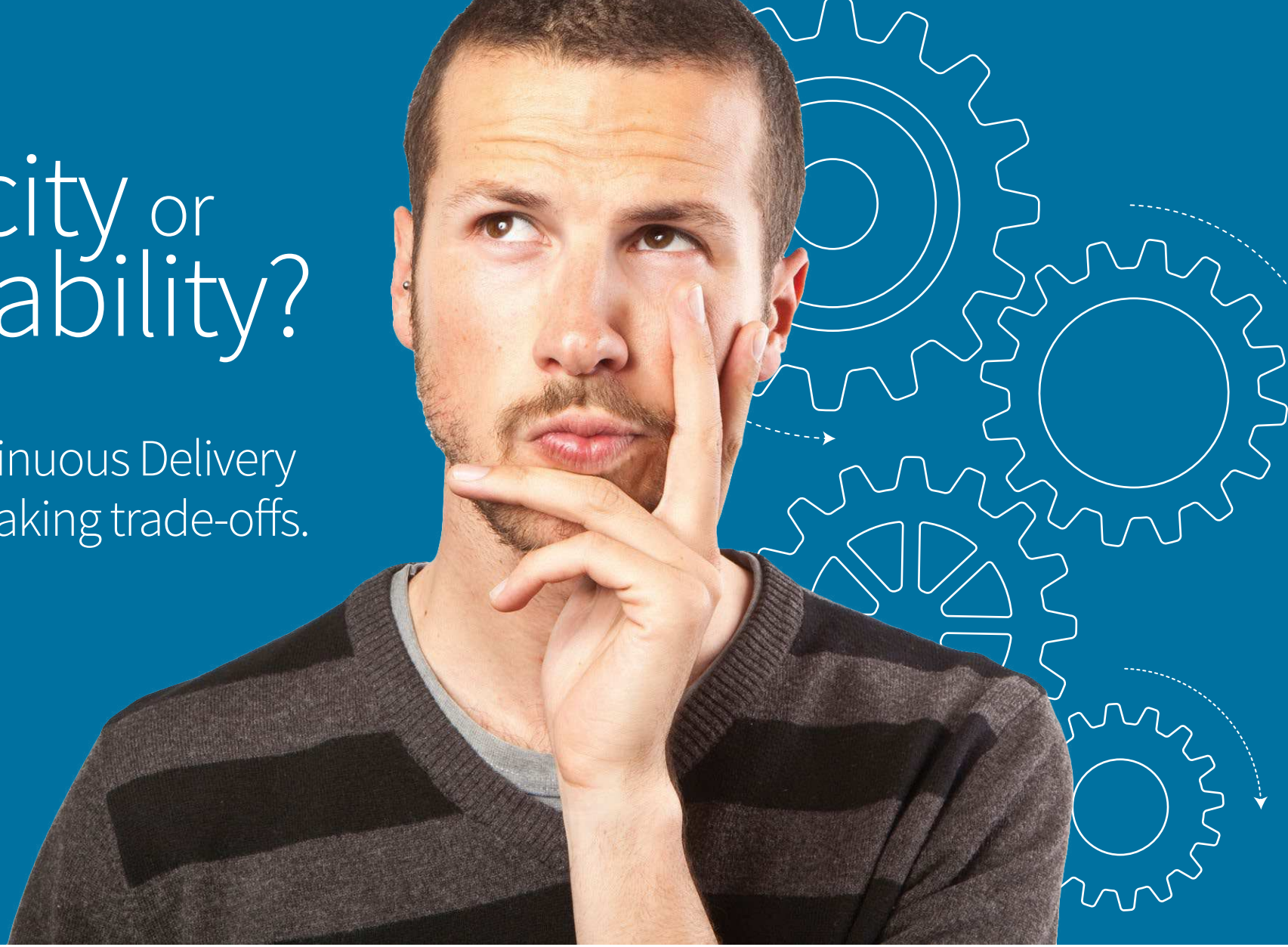
The reason we were able to deliver high-quality software with great value continuously was not that we were superhuman developers. I realized that the main reason for our success was that we only had three roles: the sponsor, the users, and the "techies." I found out that any roles involved in a project that do not directly contribute toward the goal of putting valuable software in the hands of users as quickly as possible should be carefully considered. We were not being continually measured by project managers or pushed beyond our limits. Architects were not telling us what kind of products or patterns to use. Testers did not obstruct our changes from getting into production. Systems administrators were not trying to prevent changes that could destabilize their servers. We had to take responsibility for the testing ourselves, and we worried about the infrastructure because we were the only ones who could be blamed. We delivered software incrementally to fail as early as possible so that we could correct errors before they got out of hand and get a feel for what the users really wanted.

Technically skilled people are used to solving challenges. The more freedom and responsibility they have, the more creative they will be—as long as their goal is known: to satisfy the sponsors through early and continuous delivery of software valuable to users.

> ***"Any roles involved in a project that do not directly contribute toward the goal of putting valuable software in the hands of users as quickly as possible should be carefully considered."***

Velocity or Stability?

Adopt Continuous Delivery and stop making trade-offs.



Learn what every DevOps team needs to know to increase velocity and produce more stable code.

Read the White Paper ▶

zend™

SECTION 6:
Continuing the Journey

MAKING ONE-DAY TURNAROUNDS POSSIBLE

JEFF SUSSNA

Founder and Principal
at Ingineering.IT

Jeff Sussna is founder and principal of Ingineering.IT, a boutique consulting firm that facilitates adaptive IT through teaching, coaching, and strategic design. Jeff has more than 20 years of IT experience and has led high-performance teams across the DevOps–quality assurance spectrum. He is a highly sought-after speaker and was recognized as a Top 50 Must-Read IT Blogger for 2012 and 2013 by *BizTech Magazine*. His interests focus on the intersection of development, operations, design, and business.

Twitter | Website | Blog

When my clients ask me how to adopt continuous delivery, I tell them, "Don't do continuous delivery; do *more* continuous delivery." Achieving true continuous delivery requires tremendous discipline and maturity. It also requires a shift in mindset that has to happen over time. Getting there is a path, not a jump.

At its heart, continuous delivery is about small batch sizes. You should always ask yourself how you can make your batch sizes smaller. For example, if you're building a software component with a front, middle, and back tier and it takes two days to build the whole thing, what happens if you build and release the back tier by itself? Because it's invisible without the other tiers, there's no harm in putting it into production alone. If you do, you've suddenly shrunk your batch size from two days to a few hours.

KEY LESSONS

1. **ALWAYS ASK YOURSELF HOW YOU CAN MAKE YOUR BATCHES SMALLER.**
2. **CONTINUOUSLY QUESTION YOUR ASSUMPTIONS ABOUT WHAT'S POSSIBLE.**

The key to following the continuous delivery path is to continually question your own assumptions about what's possible. I recently helped an agile development team test a new, stand-alone service. They were used to releasing every two weeks. One day, after I helped a developer validate a simple bug fix for the service, he said, "This is ready for release next week." I responded by asking, "Why not release it today? It doesn't affect anything else. We're not going to test it any more than we just did."

At the following morning's stand up, he announced that we'd fixed, tested, and released a bug fix, all in one day. Everyone's eyes got big! No one realized that one-day turnaround was even possible. Suddenly, they started thinking and talking about how to do it more often.

> ***"The key to following the continuous delivery path is to continually question your own assumptions about what's possible."***

SURVIVAL OF THE FITTEST: BEING RESPONSIVE TO CHANGE

EIKE THIENEMANN-DEHDE

Agile Product Lead at CoreMedia

Eike Thienemann-Dehde has worked on continuous delivery since 2012, when he was a product owner for tools and automation, support, and documentation efforts. He has worked in various roles, from software development to program management, over the past 16 years and is passionate about solving next-generation customer problems in an international and challenging environment.

Twitter | Website

I consider continuous delivery an ongoing journey. When you have mastered the principles of continuous integration and release automation, including deployment and quality assurance, and adopt your own practices, you quickly recognize that there will be always something to improve further.

After my company's successful agile transition and the migration to a distributed source code management system some years ago, we ramped up a dedicated team driven by the need to move quickly and incorporate support for automation tools like Chef and related processes directly into the product we were deploying in various operating environments, with their individual tool chains. When tackling change management challenges, it helps to create a community of practice made up of team delegates.

Being able to bootstrap continuous integration slave machines within minutes was the key to initially embracing the concept of Infrastructure as Code. In conjunction with the adoption of the Jenkins Job DSL plug in, it helped coordinate among all product development teams.

Baking build and test or production machines from well-defined Packer templates for any environment (e.g., Vagrant and vSphere or Amazon Web Services) increased reproducibility significantly and helped establish a fast in-house feedback loop so that we could spot issues such as commits that broke the deployment in some operating environments early.

KEY LESSONS

1. START WITH WEEKLY RELEASES WHILE PREPARING FOR MORE ON-DEMAND ROLLOUTS.
2. CONSIDER YOUR INITIAL CONTINUOUS DELIVERY TRANSITION SUCCESSFUL IF SUDDENLY YOUR PROCESS IS WAITING FOR YOUR PEOPLE.
3. LEARNING AND MOVING FAST IS A COMPANY DIFFERENTIATOR.

> ***"When you have mastered the principles of continuous integration and release automation, and adopt your own practices, you quickly recognize that there will be always something to improve further."***

SURVIVAL OF THE FITTEST: BEING RESPONSIVE TO CHANGE

EIKE THIENEMANN-DEHDE

Agile Product Lead at CoreMedia

Eike Thienemann-Dehde has worked on continuous delivery since 2012, when he was a product owner for tools and automation, support, and documentation efforts. He has worked in various roles, from software development to program management, over the past 16 years and is passionate about solving next-generation customer problems in an international and challenging environment.

Twitter | Website

So, to conquer the maturity level of automated tests, it helps to start with a dedicated code-freeze or release branch approach to avoid an initial moving target effect. Depending on the complexity of your regression tests in all supported operating environments—and especially the benefits versus cost of automated user interface tests versus manual tests—it might be feasible to start with weekly releases while preparing for more on-demand rollouts. You can consider your initial continuous delivery transition successful if suddenly your process is waiting for your people!

I suggest leveraging open source software infrastructure tools and innovating on a higher level by embracing a culture of "proudly found elsewhere" (preventing the "not invented here" syndrome). Collaborate in the open to give and get instant feedback. The impact on productivity and quality will create more freedom and options for experimentation.

Learning and moving fast can also be your company's differentiator, especially in a highly competitive environment. Always remember, "It is not the strongest of the species that survives, nor the most intelligent, but those most responsive to change" (Charles Darwin).

KEY LESSONS

1. **START WITH WEEKLY RELEASES WHILE PREPARING FOR MORE ON-DEMAND ROLLOUTS.**
2. **CONSIDER YOUR INITIAL CONTINUOUS DELIVERY TRANSITION SUCCESSFUL IF SUDDENLY YOUR PROCESS IS WAITING FOR YOUR PEOPLE.**
3. **LEARNING AND MOVING FAST IS A COMPANY DIFFERENTIATOR.**

COLLABORATION. AUTOMATION. DISCIPLINE.

ANDREW YOCHUM

Enterprise Technology Architect

Andrew Yochum strategically leads technology in digital agencies, finance, and start ups. With a background in highly scalable Web development and a passion for data, Andrew sees his work as an art and a science. His current focus is on emerging technologies in cloud computing and big data. He guides the adoption of agile methodologies as a Scrum Master and Coach.

Twitter | Website | Blog

I'm a big advocate of agile methodologies, having experienced vast improvements on development teams over my career. Many teams adopt agile practices, like Scrum, DevOps, and continuous delivery, to bring agility to the business. Each of these practices plays a role in a team's realization of the Agile Manifesto's principles, which should be interwoven to achieve even greater results. The key to achieving agility is being able to deliver confidently in a repeatable fashion—every time. But, how?

Collaboration. Automation. Discipline.

One team I lead for a Software as a Service company had multiple, interdependent applications supporting consumers, business owners, customer service, and internal administration. When I joined, the applications were deployed manually to physical servers. Deployments were time consuming, requiring downtime and carefully executed processes from systems administrators who knew little about the applications themselves. Troubleshooting was tedious when things went wrong. Testing the deployment was equally challenging. Often, the result was failed deployments and downtime.

I knew it was time to bring agile methodologies to the team, fostering collaboration with Scrum and DevOps. The goal: add automation with automated testing, builds, and deployment, and through these elements, bring a disciplined methodology to achieve agility.

KEY LESSONS

1. **FOSTER COLLABORATION AMONG DEVELOPERS, QA TEAM MEMBERS, AND SYSTEMS ADMINISTRATORS.**
2. **ADD AUTOMATION WITH AUTOMATED TESTING, BUILDS, AND DEPLOYMENT.**
3. **INTEGRATE DATABASE SCHEMAS INTO THE AUTOMATION PROCESS.**

> ***"The benefits were clear right away. Committing code kicked off a build, deployment, and automated tests. The results of the build showed up in the developers' inbox and the integrated development environment."***

COLLABORATION. AUTOMATION. DISCIPLINE.

ANDREW YOCHUM

Enterprise Technology Architect

Andrew Yochum strategically leads technology in digital agencies, finance, and start ups. With a background in highly scalable Web development and a passion for data, Andrew sees his work as an art and a science. His current focus is on emerging technologies in cloud computing and big data. He guides the adoption of agile methodologies as a Scrum Master and Coach.

Twitter | Website | Blog

We took things to the next level by adding a continuous build and test process internally on a virtualized infrastructure that our systems administrators had set up. The benefits were clear right away. Committing code kicked off a build, deployment, and automated tests. The results of the build showed up in the developers' inbox and the integrated development environment. Over time, the team added unit and functional tests to ensure that everything worked as expected. Our quality assurance (QA) team evolved from manual testing to automated testing. They became integrated with the development team, writing tests and code as development occurred. We set up a dashboard for me and the executive team to review the state of the development at a glance.

The next big step was adding Infrastructure as a Service cloud servers, using the same deployment process we used internally. Because our systems administrators had become intimately familiar through the initial automation, the leap from physical to cloud servers was reasonably easy.

KEY LESSONS

1. **FOSTER COLLABORATION AMONG DEVELOPERS, QA TEAM MEMBERS, AND SYSTEMS ADMINISTRATORS.**
2. **ADD AUTOMATION WITH AUTOMATED TESTING, BUILDS, AND DEPLOYMENT.**
3. **INTEGRATE DATABASE SCHEMAS INTO THE AUTOMATION PROCESS.**

But, there was a snag. Over time, as new features were developed that required changes to the database schemas, tests failed. Developers had been manually propagating changes to the database schemas ahead of their commits and builds: a big no-no! The schemas hadn't been integrated as part of the automation process. We took a deep dive into it during our Scrum sprint retrospective, adding user stories to our backlog to address the issue. The team collaborated to apply tools and automate the same methods we used in other parts of the development process.

We accomplished huge improvements overall. The collaboration among developers, QA, and systems administrators fostered greater shared knowledge and improved morale. With full end-to-end automation in place, deployment times went from hours to minutes, with few failures. The team's adoption of agile methods brought understanding and appreciation for discipline, which allowed them to better their craft. As the team achieved agility, customers and management saw new features reliably delivered more quickly.

DON'T LOOK BACK: PERSEVERING THROUGH THE TRANSITION TO CONTINUOUS DELIVERY

MATTHIAS MARSCHALL

CTO of gutefrage.net GmbH

Matthias Marschall is a software engineer "made in Germany." His four children make sure that he feels comfortable in lively environments and stays in control of chaotic situations. A lean and agile engineering lead, he's passionate about continuous delivery, infrastructure automation, and all things DevOps. Matthias is CTO at gutefrage.net group, helping to run Germany's biggest Q&A site among other high-traffic sites. He's the author of the *Chef Infrastructure Automation Cookbook*.

Twitter | Website | Blog

If you're working in an environment where you do infrequent, big releases, you might find yourself in a situation where the pressure to deliver faster becomes unbearable. When I was building an Internet platform for the construction industry, it was best practice to do a big release twice a year. Unfortunately, each such release was a big disaster followed by weeks of fixing critical bugs and customers threatening to sue us. We knew that the huge amount of code changes forming a big release was an issue, so we cut down our release cycles to bimonthly.

KEY LESSONS

1. **CONSIDER RELEASING INDIVIDUAL FEATURES LIKE CRITICAL BUG FIXES.**
2. **CONCENTRATE ON AUTOMATING YOUR TESTS AND RELEASE PROCESS TO INCREASE QUALITY.**

Although the releases shrank, the shortened release cycles put a lot of additional pressure on us. Not rethinking our process, we simply had less time to do all the upfront analysis and manual regression testing we considered mandatory at that time. Of course, we also had to make critical bug fix releases all the time, without a lot upfront analysis and definitely without manual regression tests of the whole product.

One day, we stood together and wondered how we could solve our troubles. A colleague raised the key question, "What's the difference between a feature and a critical bug fix? Why do we force ourselves to do a lot of upfront analysis and manual regression testing for features, while we're releasing critical bug fixes at any time?"

The question was spot on. Eventually, there was no real difference. Why not release individual features like critical bug fixes? We tried it.

In the beginning, it hurt—badly. We were even thinking about going back to bigger releases. But we persevered. Instead of going back, we concentrated on automating our tests and release process to increase quality. Slowly, we were seeing the benefits of small releases: they became a nonissue. And instead of suing us, our clients were happy to get new, working features much more quickly than before!

The key in such a transition to continuous delivery is to expect things to get worse before you'll be able to make them better. But if you persevere, you'll be able to make things so much better that you'll never want to look back.

> ***"The key in such a transition to continuous delivery is to expect things to get worse before you'll be able to make them better."***

TRAINING FOR THE WIN: TRANSITIONING TO CONTINUOUS DELIVERY

MAX LINCOLN

DevOps Instigator at ThoughtWorks

Max Lincoln works in the Continuous Delivery group at ThoughtWorks. He passionately promotes the use of continuous delivery and DevOps to achieve technical success, lean startup to drive business success, and user experience to deliver enjoyable products. He is a frequent contributor to open source cloud and quality assurance projects.

Twitter | Website | Blog

Continuous delivery is often compared with physical exercise. It's difficult at first but becomes easier as you develop skills and form habits. When I first heard this analogy, I thought of weight training. I should have thought of training for a marathon.

I've noticed techniques used by people who are training for a marathon that are similar to the techniques used for successful transitions to continuous delivery. This isn't surprising: both continuous delivery and running a marathon are ambitious goals that require a sustainable pace rather than heroic sprints. Both share three techniques for ensuring success: continuous improvement, forming positive habits, and dealing with setbacks.

The first week of a typical 16-week marathon training program looks more like a training regime to run a better 3.1 mile race than one for a 26.2 mile race. The lengths of the runs start short but increase each week. This is the surest way to reach an ambitious goal: as a series of achievable improvements starting from your current state. A sudden leap into enterprise-wide continuous delivery may be just as foolish as attempting 20 miles on your first run.

KEY LESSONS

1. **TRANSITION TO CONTINUOUS DELIVERY REQUIRES A SUSTAINABLE PACE.**
2. **DEVELOP A ROUTINE, THEN TRY NOT TO BREAK IT.**
3. **LEARN HOW TO DEAL WITH SETBACKS.**

The same training program involves four runs per week, always on the same days. Setting a routine minimizes the overhead of planning and maximizes the opportunities to learn which changes are working. Runners learn about diet and hydration, while we learn about topics like automation and feature toggles. The routine also ensures that there is never too big a gap. I would rather deploy at least weekly, then deploy daily but occasionally have a multi-week deployment freeze. Breaking the routine is risky.

The biggest lesson is how to deal with setbacks. Our resolve toward continuous delivery is always put to the test after a rough deployment. Someone usually suggests that our problem was deploying too often.

Runners face this situation, as well. It seems logical to think, "I should run less because I am too fatigued to finish." This is a dangerous idea. Running fewer total miles compromises your goal; running more miles per run risks further fatigue or injuries. Even worse, it does not address the root cause.

The key to success is a routine, fixing the root cause of all setbacks, whether it's better hydration for runners or better automation and communication when deploying software.

> ***"The biggest lesson is how to deal with setbacks. Our resolve toward continuous delivery is always put to the test after a rough deployment."***

CONTINUOUS DELIVERY VS. DELIVERING CONTINUOUSLY

MICHAEL HÜTTERMANN

CEO of huettermann.net

Java Champion Michael Hüttermann is a freelance delivery engineer and expert in DevOps, continuous delivery, and software configuration management/ application life-cycle management (ALM). He wrote some of the first books on DevOps (*DevOps for Developers*, 2012) and agile ALM (*Agile ALM*, 2011).

Twitter | Website

Continuous delivery is a new mindset for development and delivery of software to add value to the system and satisfy the customer. It is also possible to "deliver continuously" (for some people, delivering once a year is "continuously," too) just by throwing changes to production randomly, with bad quality. Thus, a systematic process of staging of software is crucial, along with introducing collaboration between development and operations to reduce cycle time and fulfill holistic business goals, often called *key performance indicators*. Without DevOps, there's no continuous delivery.

Whereas DevOps deals with collaboration between development and operations, continuous delivery focuses on the concept of bringing changes to production continuously and with minimized cycle time—and being able to do so frequently. Particularly crucial elements for implementing DevOps are:

KEY LESSONS

1. **FIND THE RIGHT BALANCE OF MANUAL WORK AND AUTOMATED PROCESSES.**
2. **ALIGN THE CHANGES IN THE DELIVERY PIPELINE WITH THE BUSINESS, NOT THE TECHNIQUE.**
3. **FOSTER THE MINDSET THAT A SINGLE CHANGE MAY RESULT IN A *POTENTIAL* RELEASE.**

- Aligning development and operations with the same goals, which are in turn aligned with overall business goals;
- Fostering a mind shift and introducing slack time and room for experimenting; a bit of firefighting is okay—you learn a lot through firefighting—but time for innovation is even better;
- Aligning processes and tools and finding the optimal tradeoff for configuration management;
- Finding the right balance of manual work and automated processes (be aware of the irony as well as the paradox of automation);
- Aligning the changes in the delivery pipeline with the business, not the technique;
- Realizing that the delivery pipeline is not a one-direction fire-and-forget tube but rather an integrated life cycle;
- Fostering the mindset that a single change may result in a *potential* release and *may* start the whole process; and
- Finding semantic containers for your release, including all artifact types, that are versioned and executable.

> ***"Without DevOps, there's no continuous delivery."***

CONTINUOUS DELIVERY VS. DELIVERING CONTINUOUSLY

MICHAEL HÜTTERMANN

CEO of huettermann.net

Java Champion Michael Hüttermann is a freelance delivery engineer and expert in DevOps, continuous delivery, and software configuration management/ application life-cycle management (ALM). He wrote some of the first books on DevOps (*DevOps for Developers*, 2012) and agile ALM (*Agile ALM*, 2011).

Twitter | Website

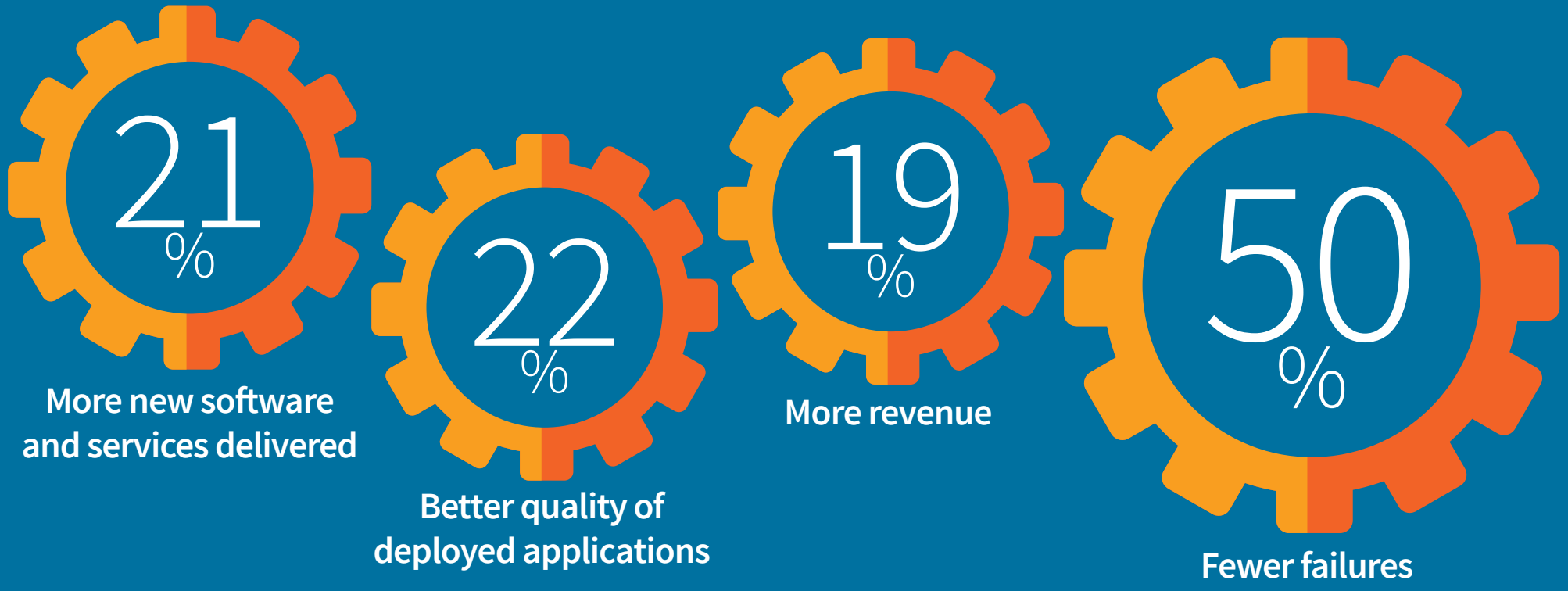
Continuous delivery spans the software life cycle and is based on *continuous integration* (that is, checking changes into version control multiple times a day, with the constraint that developers can check out current states of the software at any time without having any build errors locally afterwards), *continuous deployment* (that is, the frequent deployment of changes to target environments), and *continuous inspection* (that is, the inspect-and-adapt pattern to keep up the internal and external quality of the software).

Continuous delivery is not necessarily part of a DevOps initiative. In other words, if you implement continuous delivery, you'll most likely have to implement some sort of DevOps, too. I wish you much fun and success with your continuous delivery initiative—and don't forget to shape your DevOps approach.

KEY LESSONS

1. **FIND THE RIGHT BALANCE OF MANUAL WORK AND AUTOMATED PROCESSES.**
2. **ALIGN THE CHANGES IN THE DELIVERY PIPELINE WITH THE BUSINESS, NOT THE TECHNIQUE.**
3. **FOSTER THE MINDSET THAT A SINGLE CHANGE MAY RESULT IN A *POTENTIAL* RELEASE.**

Continuous Delivery
Get a handle on the Economics



Learn how to measure expected ROI for your company, get buy-in, and go full speed ahead.

Get the Workbook

> This workbook draws upon experiences of other organizations and industry analysis. Use it to calculate expected ROI of Continuous Delivery for your organization and present your business case with confidence.
>
> **Ido Ben Moshe,** Vice President, Global Support & Professional Services

zend