2020

# Continuous Deployment Transitions at Scale

Laurie Williams
*North Carolina State University*

Kent Beck
*Facebook*

Jeffrey Creasey
*LexisNexis*

Andrew Glover
*Netflix*

James Holman
*SAS Institute Inc.*

*See next page for additional authors*



## Recommended Citation

Laurie Williams, Kent Beck, Jeffrey Creasey, Andrew Glover, James Holman, Jez Humble, David McLaughlin, John Thomas Micco, Brendan Murphy, Jason A. Cox, Vishnu Pendyala, Steven Place, Zachary T. Pritchard, Chuck Rossi, Tony Savor, Michael Stumm, and Chris Parnin. "Continuous Deployment Transitions at Scale" *Tools and Techniques for Software Development in Large Organizations: Emerging Research and Opportunities* (2020): 168-181. https://doi.org/10.4018/978-1-7998-1863-2.ch006

# Chapter 6
# Continuous Deployment Transitions at Scale

**Laurie Williams**
*North Carolina State University, USA*

**Kent Beck**
*Facebook, USA*

**Jeffrey Creasey**
*LexisNexis, USA*

**Andrew Glover**
*Netflix, USA*

**James Holman**
*SAS Institute Inc., USA*

**Jez Humble**
*DevOps Research and Assessment LLC, USA*

**David McLaughlin**
*Twitter, USA*

**John Thomas Micco**
*VMWare, USA*

**Brendan Murphy**
*Microsoft, UK*

**Jason A. Cox**
*The Walt Disney Company, USA*

**Vishnu Pendyala**
*Cisco Systems Inc., USA*

**Steven Place**
*IBM, USA*

**Zachary T. Pritchard**
*Slack, USA*

**Chuck Rossi**
*Facebook, USA*

**Tony Savor**
*Facebook, USA*

**Michael Stumm**
*University of Toronto, Canada*

**Chris Parnin**
*North Carolina State University, USA*

## ABSTRACT

*Predictable, rapid, and data-driven feature rollout; lightning-fast; and automated fix deployment are some of the benefits most large software organizations worldwide are striving for. In the process, they are transitioning toward the use of continuous deployment practices. Continuous deployment enables companies to make hundreds*

DOI: 10.4018/978-1-7998-1863-2.ch006

*or thousands of software changes to live computing infrastructure every day while maintaining service to millions of customers. Such ultra-fast changes create a new reality in software development. Over the past four years, the Continuous Deployment Summit, hosted at Facebook, Netflix, Google, and Twitter has been held. Representatives from companies like Cisco, Facebook, Google, IBM, Microsoft, Netflix, and Twitter have shared the triumphs and struggles of their transition to continuous deployment practices—each year the companies press on, getting ever faster. In this chapter, the authors share the common strategies and practices used by continuous deployment pioneers and adopted by newcomers as they transition and use continuous deployment practices at scale.*

## INTRODUCTION

Continuous deployment is a software engineering process where incremental software changes are automatically tested and deployed to production environments without manual steps in the deployment pipeline (Rahman et al. 2015). Continuous deployment enables companies, such as Facebook (Savor et al. 2016), to make hundreds or thousands of software changes to live computing infrastructure every day, while maintaining service to millions of customers. Such ultra-fast changes create a new reality in software development.

Over the past four years, we have held the Continuous Deployment Summit, hosted at Facebook (Parnin et al. 2017) (2015), Netflix (2016), Google (2017), and Twitter (2018). For three years from 2015 to 2017, representatives from eleven companies, Cisco, Disney, Facebook, Google, IBM, LexisNexis, Microsoft, Netflix, SAS, Slack, and Twitter, have shared the triumphs and struggles of their transition to continuous deployment practices—each year the companies press on, getting ever faster. In this paper, we share the common strategies and practices used by continuous deployment pioneers and adopted by newcomers as they transition and use continuous deployment practices at scale. Every company is still making this journey toward continuous deployment.

## PERSISTENT AND INCREMENTAL PRACTICE ADOPTION

As Einstein advises, "Persistence is the most powerful force on earth, it can move mountains." The uniting factor among all the Summit companies was the persistent movement toward becoming more efficient, improving customer satisfaction and

business results and increasing release frequency through the incremental adoption of continuous deployment practices. Each year, the Summit companies demonstrated measurable increases in the adoption of the practices.

Some of the Summit companies, such as Google, Facebook, and Netflix, were "born" using continuous deployment practices. Older companies, such as Microsoft, IBM, Cisco, and SAS, have large legacy products in their portfolio that were "born and raised" with a waterfall-type software development process. Disney supports a wide range of software products—from websites to safety-critical software that runs theme-park rides. These older companies could have decided continuous deployment was not appropriate for some of their products. Instead, these giants took demonstrable steps each year to "turn their ship around."

Each company found its unique way to bring about continuous change. Disney attributes its success with the use of continuous deployment practices to their company's values established by Walt Disney himself: Curiosity, Confidence, Courage, and Constancy. The developers are curious to see if the practices could help them with their business results; they are confident in their abilities, systems, and checks so they dare to make changes. Constancy helps them continue to incrementally adopt more practices. Microsoft has a range of product types from Yammer and Bing, which use continuous deployment practices similar to those of Google, Facebook, and Netflix; to its monolithic software, such as Microsoft Exchange and Windows operating system. Inspired by continuous deployment practices, Microsoft Exchange now deploys to beta customers using a ring deployment model, where a release is deployed to a new ring level every week, finally reaching beta customers in the sixth week—if no problems are detected. Finally, Facebook has applied this principle to changing their release process for all developers in the company.

## MOBILE FIRST

Summit companies recognize that worldwide growth in the use of mobile applications exceeds that of web and other cloud-based applications. This growth trend motivated Facebook CEO, Zuckerberg, to announce a "Mobile First!" strategy in 2012, which directed new development to occur first for mobile applications before developing for the other platforms. Mobile First! strategy is followed by other Summit companies, such as Google. However, the frequency of updates of mobile software has traditionally lagged that of web applications for many reasons. Mobile versions can only be released through the Apple and Google app stores that control the frequency of releases and impose constraints on development. Users may not auto-install updates and can decide when and if to upgrade; conceivably every release of a mobile app that ever existed could be installed across their user base.

The need to support and test hundreds of Android hardware variants increases the computational cost complexity and speed of the verification process, thereby further slowing down deployment. Finally, quality requirements are higher for mobile apps as there are more limited options for taking remedial action through deploying a new version when a defect is detected, compared with web- and cloud-based apps.

Chuck Rossi, the director of release engineering at Facebook, delivered the keynote at the 2017 Summit. At Facebook, mobile applications are used by over a billion people each day (Rossi et al. 2016). Rossi shared that over a period of four years, Facebook has decreased the deployment speed from 6 weeks to 4 weeks to 2 weeks to 1 week. Mobile applications are deployed more frequently to its internal users during a one-week stabilization phase that occurs the week after development is complete to conduct "dogfood" testing. Summit companies also use tools, such as the Gatekeeper tool, and feature flags in the code to dynamically control from the cloud the features that users see in an app. Even though the customer installations of new versions of the app will occur periodically, the companies can still control the incremental rollout of new individual features across their user base and can disable problematic changes in the advent of unexpected behavior without requiring customers to update their apps.

## DEVELOPER PRODUCTIVITY METRICS (LOOK WITHIN)

Companies are increasingly looking inward at their productivity, to evolve practices or improve tool infrastructure for developers. These opportunities offer a much richer source of information beyond simple metrics, such as lines of code produced, and are more deeply tied to customer behavior.

At Google, searching for internal libraries is a common task and deeply integrated into developer tooling and culture. Given that many possible library choices may exist, one determining factor may be signals (Trockman 2018), information cues that indicate attributes, such as quality, that may bias a developer towards one particular library. Google has recently integrated metrics that serve as signals into project dashboards. For example, the metrics include pre-submit speed (i.e. time to run tests before committing to a repository), release frequency (hypothesizing that projects with higher frequency are healthier), green builds/week (builds with fewer failures), and number of post-release patches (how error-prone is the code). A project with good project health metrics (called PH-levels) can be perceived as more reliable and thus might be more likely to be adopted. Developers are encouraged to strive for healthy PH-levels. However, some metrics are considered controversial for certain teams, who want to opt-in/opt-out of certain metrics. Despite these challenges, PH-levels can help maintain a shared sense of productivity.

Finally, participants cautioned about direct interpretations of developer productivity metrics. Some participants at the Summit argued that simply increasing release frequency (say, 8 weeks to 4 weeks) does not necessarily improve developer productivity. Instead, the increased frequency forces upgrades in tooling and automation, which in turn reduces errors and inefficiencies in the process. In another example, a common low-hanging fruit that an organization may target for optimization is increasing the speed of tooling. However, Microsoft provided several cases where tools were made faster, but observed no tangible benefit in productivity gains: Instead, developers simply changed when they ran the tool (from night-time to day-time). Ultimately, the participants recommended instead of simply striving to hit or game metrics, organizations should target desirable changes in developer behavior.

## TOOLS EMBODY CULTURE

Creating a shared sense of culture and maintaining architectural integrity in a large organization can be difficult; especially when the number of developers can be counted in the thousands and with teams operating in small independent units. Traditionally, many software organizations have relied on centralized architecture teams to help manage standards (Parsons 2005). However, an alternative paradigm has emerged, where architectural principles can be enforced through strong investments in tooling.

At the Summit, companies shared various ways in which tooling played a central role in creating a shared engineering culture. Perhaps the most illustrative example is the introduction of chaos engineering at Netflix. At Netflix, developers mostly work in small teams that support a single feature or microservice. Given Netflix's anti-process culture and lack of centralized architectural teams but high interdependence of microservices, there needed to be some way to communicate and enforce architectural principles across the whole organization. Chaos engineering (Basiri et al. 2016), is the practice of introducing small changes or unexpected events into production environments to analyze how these changes or events could impact the behavior of the system. For example, by introducing a chaos monkey, a tool that randomly turned off AWS instances during working hours, the tool could help enforce architectural principles of maintaining stateless and resilient microservices.

Enforcing cultural changes through tools can result in adoption barriers. For example, Microsoft wanted to introduce stronger coding practices that could reduce potential security problems. In one instance, trying to turn-on compiler errors for uninitialized variables (a potential security concern) as a general policy resulted in a large pushback from many development teams. While understanding the security implications, many developers often viewed these compiler findings as false positives

and did not want them turned on as errors for their projects. To combat a similar problem at Google, the static analysis tool, Tricorder (Sadowski et al. 2015), allows developers to give feedback on any finding (e.g., "Does not work in IE8"). Further teams can opt-out of specific types of findings or even opt-in specialized findings. If a finding is found not to be useful 10% of the time, the tool findings may eventually be disabled across the company.

Tooling allows developers to share common workflows across the company and even between companies. Some companies, such as Google and Facebook, invest in their web-based IDE. By having all developers share the same interfaces for developing code, the companies can ensure that all developers share the same workflow for processes such as code review, code search, and reviewing findings from static analysis tools. At the Summit, the participants noted the increasing importance of partnership and investment of tooling across multiple companies and open source communities. Open-source tools, such as Spinnaker (which supports specifying and customizing deployment workflows), have been developed in partnership between Netflix and companies such as Microsoft, Google, and Pivotal. Some parameters and decisions can be highly variable between teams and products: How long is a canary experiment; at what step do you sign-off on a deploy; how does your particular service handle state? Scale differences between companies and communities introduce a complication. For example, at Twitter, upstream open source patches often end up breaking Twitter's production environment because the open source community operates at a much lower scale. Despite these challenges, companies cite numerous benefits, such as attracting talent and improving tool value. As one participant stated: "It makes sense to work together when you're the only two companies in the world that face the same issue."

## TESTING AND RELEASE IN PRACTICE

Operating continuous deployment pipelines at scale requires numerous shifts in technology and practices. Traditional problems are amplified, while new problems and pain points emerge. At Google, the demand for continuous integration (CI) services double each year, with over 4.5 million tests being run daily—if not properly optimized, this demand would require more servers to run than Google's primary product itself: search.

At the Summit, companies discussed numerous pain points related to testing and shared various strategies that could help address them. One of the most common pain points expressed was flaky tests, that is, tests that intermittently fail due to random factors, such as resource availability (Luo et al. 2014). At Google, an internal analysis of failing tests found that 84% of the time a failure is due to a flaky test.

Several strategies were discussed to combat flaky tests. Companies have started calculating the flakiness of tests or try to tag flaky tests based on historical data. At Google, tests are kept below 1–5% flakiness or are quarantined. At Facebook, the current practice is to simply delete flaky tests without mitigation. Several companies reported reliability issues of running tests in Jenkins due to resource exhaustion or inconsistent state of the workspace. To improve the reliability of running tests in Jenkins, IBM and Netflix are moving towards running tests in containers. Finally, participants at the Summit discussed the goal of moving toward predicting failing builds and the presence of flaky tests. For example, if the dependency chain between a changed source file and a failing unit test is more than ten hops away, it is likely to be a flaky test.

Companies also discussed various issues and strategies for deploying releases into production. At LexisNexis, releases occur every three weeks during off-hours. Each release requires manual coordination and blessing of released features—a customized Gantt chart is used to coordinate the order of flips for new versions of shared services. Once everything is in place, manual testers verify the release; meanwhile, developers of each service/module are on standby to patch any problems. At Disney, release management was more frequent, with three release windows per week. However, developers did not have full autonomy for making release decisions; a highly centralized process is used and overseen by executives for no/go decisions on each release. Meanwhile, Netflix remained at the head of the pack with 4000 deployments a day.

## HOLDING ONTO SCHEMAS

For some companies, the biggest barrier to full continuous deployment adoption is a lack of an effective strategy for deploying schema changes to relational databases (or their usage at all). For example, in many database engines, a simple operation such as renaming a column in a table would require locking all rows and thus prevent any new data from being stored, while the rename operation took place. Major schema changes could effectively shutdown an application for many hours (de Jong et al. 2017).

This challenge was especially apparent in companies that supported legacy applications. For example, IBM used to take a month to migrate a system to a new version at a customer's site. The primary challenge was coordinating code and database changes with on-premises instances. Eventually, IBM shortened the process to one hour. For LexisNexis, a 200-year-old company with software components that are over 15 years old, deploying database changes remains one of the most challenging aspects of continuous deployment. For every deployment to

production, the deployment process is often on hold for several hours as they wait for the DBA to clear the release. In addition to schema changes, other issues can make deployment with databases problematic.

For SAS, dumping and restoring databases to accommodate schema changes can take hours. At Microsoft, database rollbacks are avoided at all costs, especially if the failure rate is low. Companies that have built continuous deployment-ready architectures often discard relational databases entirely or develop new storage technologies that can handle schema changes. For example, Netflix uses a key-value based store, Cassandra, and microservice architecture. Any changes to a database are handled by managing access to versioned calls at the service layer. Graph databases, such as Facebook's social graph, avoid these locking issues entirely by being able to add new nodes and edges, then removing old edges and nodes when done without any downtime. Still, even the most advanced architectures cannot escape issues related to schema changes. At Facebook, changes to the schema for storing messages and photos required a year-long migration to a more efficient schema.

## INTENTIONAL FEATURE EXPERIMENTATION

Companies have, for decades, used telemetry to capture usage of their software to identify quality issues or to help improve deployed features. Since the inception of the Lean Startup (Ries 2011) practice, Internet-based and other companies have been using data obtained via feature experimentation instrumentation to make data-driven decisions on whether a new feature or algorithm should "pivot or persevere" in the released product. Specifically, companies are removing features from their code if these features do not have a positive impact on their customers. Five of the Summit companies have evolved their continuous deployment processes to include feature experimentation.

To enable experimentation, feature toggles may be implemented in the software to create multiple experiences for different customers. Feature toggles are essentially conditional blocks – if/else statements that can be used to enable or disable a feature selectively (Schermann, Cito, & Leitner, 2018). For example, when Facebook released Live video, they realized an individual live video could receive up to 2500 comments per second. Facebook built experiments to evaluate multiple algorithms for filtering and ranking comments to choose the algorithm that performs best at elevating comments with high engagement. Data scientists work with the development team to design experiments, develop hypotheses, collect metrics, and analyze collected data. The paper documenting the 2015 Continuous Deployment Summit contained the adage, "Every feature is an experiment." (Parnin et al. 2017) However in later summits, the reality of the experiment complexity and the sheer amount of data

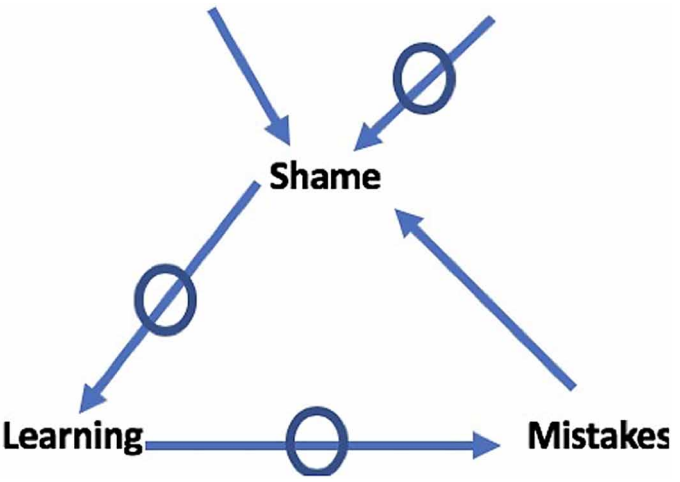
needed to be collected has made Summit companies more intentional in their choice of when to run an

experiment. For example, Google is cautious of experiments that may have implications to ad revenue, so typically small, incremental experiments are run. Naively, companies with large user bases may initially believe they would obtain feedback on a new feature rapidly, such as in hours or a small number of days. However, users behave differently throughout the day, on different days of the week, and at different times of the month. Representative experiments on stable features take longer than expected. Microsoft has analyzed 21,220 experiments applied in Bing (Kevic et al. 2017). Their results indicate that an experiment runs an average of 42 days before a "pivot or persevere" decision is made. As discussed above, feature rollout to mobile customers are delayed relative to online customers, making mobile experiments slower and more technically challenging. Summit companies did not use feature experimentation for bug fixes, infrastructure changes, or architecture changes.

## SHAMELESS RETROSPECTIVES

Retrospectives are meetings in which a team inspects and adapts their methods and teamwork after completing a unit of work. Retrospectives enable learning, act as a catalyst for change, and generate action (Derby and Larsen 2006)—as long as the environment for retrospective discussion is safe. Allow shame and blame to enter the retrospective, and these benefits are obliterated. Shame crushes our tolerance

*Figure 1. Cycle of shame*



for vulnerability, thereby killing engagement, innovation, creativity, productivity, information flow, and trust (Brown 2012).

In the 2016 Summit, Kent Beck who was at Facebook at the time, delivered a keynote about the role of shame in software development as depicted by the Cycle of Shame (Figure 1). The Cycle of Shame uses the notation of influence diagrams (Weinberg 1992). With influence diagrams, a regularly directed arrow indicates that more of the source activity tends to create more of the destination activity (i.e. an amplifier), such as more mistakes generate more shame. A directed arrow with a circle over it indicates that more of a source activity tends to create less of the destination activity (i.e. an inhibitor), such as more shame drives less learning. Starting from the regular arrow into Shame in Figure 1, more shame drives less learning which drives more mistakes which drives more shame. Conversely, starting from the arrow with the circle, less shame drives more learning which drives fewer mistakes which drives less shame.

Within the context of the Cycle of Shame, Beck remarked positively about how little shame there was in the engineering culture at Facebook. An engineer can freely share the details about a mistake that he or she has made, owning the mistake—and most importantly not blaming anyone else for the mistake. The engineer shares the consequences of the mistake, details the remedial action, and provides suggestions for how that type of mistake could be avoided in the future. In sharing this information, the engineer does not feel shame, benefiting his or her learning and that of the team members. The practice of shameless retrospectives resonated with Summit companies as an essential component of the continuous process improvement needed while adopting continuous deployment practices, which are often disruptive changes to the organization.

## LEVERAGING CULTURE AND PRACTICES TO ENHANCE SECURITY

Alongside continuous deployment practices, organizations are increasingly adopting software security practices. However, from a frequency of adoption perspectives, firms most often adopt software security practices for reasons, including responding to a security event, detecting vulnerabilities, and preventing vulnerabilities (Williams et al. 2018). Integrating software security practices in a continuous deployment environment is challenging because teams must integrate these practices at speed, perhaps in an environment that chooses speed over deliberate, methodical approaches to testing, security, and quality (McGraw 2017).

Many of the Summit companies have their software security group "silo'ed" into a separate organization, as is also common in most non-Summit companies.

Some of the smaller organizations, such as Twitter and Slack, have stronger partnerships between the developers and their software security group, moving towards a DevSecOps model, in which the security silo is broken down. At Slack, teams often use Trello for collaborative, team-based project management. Based upon the perceived risk of a new feature or product, their security team puts cards on the team's Trello board to signify the software security practice or reviews that are needed to take place before the release. At both Slack and Twitter, the security group partners with the development team starting with the requirements and design phases. The philosophy of the security groups is that rather than taking the role of fishing for security vulnerabilities when development is complete, the role of the security team is to "teach the development team to fish" whereby the development team specifies, designs, and implements secure products. All Summit teams desire better automated security tools that could detect both architecture/design- and code-level vulnerabilities with fewer false positives, a call for security researchers and tool vendors.

Continuous deployment practices can enhance the security of a product. The use of feature toggles is prevalent by Summit companies to support dark launches and feature experiments. Dark launches release new features into production surreptitiously, without any real users noticing them (Schermann, Cito, & Leitner, 2018). The system still duplicates the user requests to evaluate the new features in the clandestine releases. Summit companies, such as Twitter, use feature toggles to prevent features with security and/or privacy implications from being accessible to external users until the security team has conducted their checks. Using this procedure, developers can still continuously integrate code to these important features, but a separate security/privacy process can take place before the public launch. Teams instrument their code and constantly monitor the behavior of users to enable feature experimentation. This same instrumentation and monitoring can be used to detect anomalous behavior by attackers. Finally, organizations can use their normal process to rapidly deploy security fixes that will more likely be installed by customers. In the middle of 2016, security researchers found critical vulnerabilities in both Chrysler and Tesla automobiles. Tesla was able to deploy their fix over the air, while Chrysler sent USB sticks to its customers due to the lack of a better deployment process.

## CONCLUSION

The eleven companies that participated in the annual summit, reveal their commitment toward adopting software development practices that move them closer to continuous deployment. All the companies at the Summit have experience applying continuous deployment practices and are aware of the challenges in applying these practices. At

one end of the spectrum, the adoption may be challenging yet feasible for deploying hundreds or thousands of times of day, supporting feature experiments that can drive data-driven decisions. On the other end of the spectrum, legacy products may be deployed multiple times per year rather than once per year with a corporate strategy shifting toward more cloud-based solutions that can be deployed more frequently. Regardless of where they are on the spectrum, the Summit companies share a bond of a commitment to continuous process improvement and sharing technical solutions, approaches, and use of tools.

## ACKNOWLEDGMENT

One of the authors who now works at VMWare had worked at Google during the time of the Summits. Google has reviewed and approved the contents of this paper.

## REFERENCES

Basiri, A., Behnam, N., De Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J., & Rosenthal, C. (2016). Chaos engineering. *IEEE Software*, *33*(3), 35–41. doi:10.1109/MS.2016.60

Brown, B. (2012). *3 Ways To Kill Your Company's Idea-Stifling Shame Culture*. Fast Company.

de Jong, M., van Deursen, A., & Cleve, A. (2017, May). Zero-downtime SQL database schema evolution for continuous deployment. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)* (pp. 143-152). IEEE. 10.1109/ICSE-SEIP.2017.5

Derby, E., Larsen, D., & Schwaber, K. (2006). *Agile retrospectives: Making good teams great*. Pragmatic Bookshelf.

Kevic, K., Murphy, B., Williams, L., & Beckmann, J. (2017, May). Characterizing experimentation in continuous deployment: a case study on bing. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track* (pp. 123-132). IEEE Press. 10.1109/ICSE-SEIP.2017.19

Luo, Q., Hariri, F., Eloussi, L., & Marinov, D. (2014, November). An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 643-653). ACM.

McGraw, G. (2017). Six Tech Trends Impacting Software Security. *Computer*, *50*(5), 100–102. doi:10.1109/MC.2017.143

Parnin, C., Helms, E., Atlee, C., Boughton, H., Ghattas, M., Glover, A., ... Stumm, M. (2017). The top 10 adages in continuous deployment. *IEEE Software*, *34*(3), 86–95. doi:10.1109/MS.2017.86

Parsons, R. I. (2005). Enterprise architects join the team. *IEEE Software*, *22*(5), 16–17. doi:10.1109/MS.2005.119

Rahman, A. A. U., Helms, E., Williams, L., & Parnin, C. (2015, August). Synthesizing continuous deployment practices used in software development. In 2015 Agile Conference (pp. 1-10). IEEE. doi:10.1109/Agile.2015.12

Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., & Stumm, M. (2016, May). Continuous deployment at Facebook and OANDA. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)* (pp. 21-30). IEEE. 10.1145/2889160.2889223

Schermann, G., Cito, J., & Leitner, P. (2018). Continuous experimentation: Challenges, implementation techniques, and current research. *IEEE Software*, *35*(2), 26–31. doi:10.1109/MS.2018.111094748

Trockman, A., Zhou, S., Kästner, C., & Vasilescu, B. (2018, May). Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *Proceedings of the 40th International Conference on Software Engineering* (pp. 511-522). ACM.

Weinberg, G. (1992). *Systems Thinking, Quality Software Management* (1st ed.). New York: Dorset House.

Williams, L., McGraw, G., & Migues, S. (2018). Engineering Security Vulnerability Prevention, Detection, and Response. *IEEE Software*, *35*(5), 76–80. doi:10.1109/MS.2018.290110854

## ADDITIONAL READING

Arachchi, S. A. I. B. S., & Perera, I. (2018, May). Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. In 2018 Moratuwa Engineering Research Conference (MERCon) (pp. 156-161). IEEE. doi:10.1109/MERCon.2018.8421965

Laukkanen, E., Paasivaara, M., Itkonen, J., & Lassenius, C. (2018). Comparison of release engineering practices in a large mature company and a startup. *Empirical Software Engineering*, *23*(6), 3535–3577. doi:10.100710664-018-9616-7

Mahdavi-Hezaveh, R., Dremann, J., & Williams, L. (2019). Feature Toggle Driven Development: Practices usedby Practitioners. *arXiv preprint arXiv:1907.06157*.

Ravichandran, A., Taylor, K., & Waterhouse, P. (2016). *DevOps for Digital Leaders*. doi:10.1007/978-1-4842-1842-6

Schermann, G., Cito, J., Leitner, P., & Gall, H. C. (2016, May). Towards quality gates in continuous delivery and deployment. In *2016 IEEE 24th international conference on program comprehension (ICPC)* (pp. 1-4). IEEE. 10.1109/ICPC.2016.7503737

Schermann, G., Cito, J., Leitner, P., Zdun, U., & Gall, H. (2016). *An empirical study on principles and practices of continuous delivery and deployment (No. e1889v1)*. PeerJ Preprints.

Shahin, M., Zahedi, M., Babar, M. A., & Zhu, L. (2019). An empirical study of architecting for continuous delivery and deployment. *Empirical Software Engineering*, *24*(3), 1061–1108. doi:10.100710664-018-9651-4