What is Engineering DevOps?

This paper is derived from selections from this book.



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from [**https://engineeringdevops.com/documents**](https://engineeringdevops.com/documents)

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

© EngineeringDevops 2019, 2020

All rights reserved

**What Is Engineering DevOps?**

What are the key ingredients to assure success with DevOps?

Engineering requires the entire team to have a common set of instructions. Can you imagine the chaos that would ensue if the stonemasons, ironmongers, and carpenters building Camelot castle each had their own construction drawings? As they designed and built their components of the structure, they would have become frustrated when their work didn’t fit together. There are many ways to label and layout DevOps components and the connections between them. Having a common “big picture” of DevOps components and relationships avoids construction chaos.

Figure 1— ***DevOps Engineering Blueprint*** is the approach I use in my practice and in my book “Engineering DevOps” to illustrate the major components associated with DevOps, and the relationships between them, in a manner consistent with recommended engineering practices.



Figure 1—DevOps Engineering Blueprint

Different organizations have different definitions of components and may not have all the components shown in this blueprint, and yet they can still be doing DevOps. What is critical is that the entire team agrees to follow the same DevOps blueprint and strives to work in accordance with shared practices that have been proven to lead to success.

All blueprint components are defined in the Appendices and explained in detail later in the book. In this paper, for the sake of explaining the blueprint, I will simply refer to the components without going into detail about them.

The background labeled “Governance, Continuous Security, and Site Reliability Engineering” indicates these items apply to all other items in the diagram. Items positioned higher in the blueprint have higher levels of control and observations than items lower in the blueprint. For example, Value-Stream Management can control and observe Application Release Automation, Pipeline, and Elastic Infrastructure components. In a real DevOps implementation, the Value-Stream Management component may or may not connect to an Application Release Automation (ARA) component. It may have connections to any of the lower layer components directly. For example, it may connect directly to Elastic Infrastructure components. Similarly, in a real DevOps implementation the ARA component may or may not connect directly to Elastic Infrastructure components. Items in the component labeled “Pipeline” are shown in a series of “stages” with arrows from left to right. The arrows indicate increasing relative time from stage to stage. The Plan stage is completed before the Continuous Integration (CI) stage, and then the Continuous Delivery/ Deployment (CD) stage follows, and finally the Operations stage. The CI and CD stages are further subdivided into stages themselves. The CI stage takes input from the Plan stage and includes stages for Backlog and Design, Code and Test, Commit and Merge, and Build and Test, resulting in a repository of Artifacts depicted by the barrel symbol for database. The CD stage takes as input Artifacts and includes SAT and UAT, Approve Release, Deploy to Prod, and Post-Prod, resulting in an output to Operations. The Elastic Infrastructure component has several components noted within it. Data Center and the Cloud components labeled “IaaS,” “PaaS,” “SaaS,” and “FaaS” are not in a time order.

A PowerPoint version of the blueprint can be found on www.EngineeringDevOps.com if you would like to download it or modify it to the suit terminology and layout of DevOps components for your organization.

**A DevOps Engineering Blueprint that clearly defines abstract labels and layers of DevOps components and their relationships is an important reference diagram to ensure all team members have a common understanding that guides the implementation.**

**DevOps Engineering Tenets and CALMS**

DevOps engineering tenets are the core underlying principles that guide well-engineered DevOps implementations. Figure 2—CALMS Model is a conceptual model that explains DevOps tenets. CALMS is an acronym that stands for Culture, Automation, Lean, Metrics, and Sharing. The CALMS model is useful to explain qualitative tenets of DevOps. It does not, however, provide enough precision to engineer DevOps solutions, and so it is limited in scope. It is important to understand the limitations of the CALMS model to find a more precise model to guide DevOps work.



Figure 2—CALMS Model (credited to Jez Humble, co-author of The DevOps Handbook)

**Culture** — Leadership and Organization: The culture element emphasizes leadership and team organization. DevOps favors cultures that are highly collaborative, leaders that advocate and sponsor DevOps practices, and learning and work arranged with small cross-functional teams with very short communication paths and shared accountability.

 **Automation and Orchestration** — Automation is applied to workflow tasks that can be made faster to remove bottlenecks in the continuous delivery pipeline flow. With DevOps, “Orchestration” refers to a special type of automation that is applied to creating and releasing ephemeral resources. This tenet is key to engineering because all tools in a DevOps toolchain need to support automation capabilities and the ability to integrate with each other in an automated end-to-end workflow.

**Lean** — Shift-Left, Fail-Early, Fail-Often, and Test-Fast: Lean refers to lean engineering, which strives to “perfect process “ ‘flow’ ” in a value stream by removing waste (Muda) and minimizing time-to-market (Pull). DevOps tenets “Shift-Left” and “Fail-Early” refer to the lean tenet that doing any work that is necessary to creating value in a value stream is best done as early as possible in the continuous delivery pipeline to avoid that work causing bottlenecks later in the pipeline. Similarly, the tenets of “Fail-Often” and “Fail-Fast” emphasize that test activities are a primary source of waste, or “Muda,” for continuous delivery pipelines, so it is best to run tests often with different configurations to detect and repair failures quickly before they become a bottleneck later in the pipeline.

**Metrics** — Focus on the Most Relevant: Metrics is a critical tenet of engineering DevOps. In DevOps there are many things to measure, including all the elements that make up applications, pipelines, and infrastructures. The challenge is to be precise about what data to measure, how to analyze the data, and what to do with the analysis. Too much information creates noise. Therefore, another key tenet is to engineer metrics, analysis, and communication to ensure that the most relevant information is collected. The most relevant analysis is computed and then communicated to the most relevant people or processes that need to take action as a result of the analysis.

Sharing, Collaboration, and Yokoten — Sharing refers to the tenet that lean engineering recommended engineering practices (Kaizen) are proactively shared with other people (Yokoten) horizontally across an organization so they can immediately learn from each other without barriers imposed by slower hierarchical organization policies.

**Origins of DevOps from an Engineering Point of View**

While the word “DevOps” has only been in use since 2009, concepts and practices associated with DevOps have been in existence for many years before. Contrary to popular myth, the book that most popularized DevOps, The Phoenix Project, published in 2013, is not the origin of DevOps. While it may not be an engineering DevOps “how-to” reference manual, it is an entertaining page-turner. The book illustrates how culture, human interactions, and organization silos often undermine the success of IT projects. It also nicely lays out three major phases of DevOps maturity with something called “The Three Ways,” which draw from lean manufacturing principles. The First Way has to do with realizing “Continuous Flow.” This occurs when pipeline stages are connected and operate smoothly without major interruptions. The Second Way, “Continuous Feedback,” occurs when the Continuous Flow pipeline is instrumented with metrics to help identify bottlenecks in the flow. The Third Way, which I refer to as “Continuous Improvement,” occurs when the pipeline is stable enough to risk some experimentation.

I do agree with many DevOps experts that DevOps completely depends on having the right kind of culture. No technology without the right culture will achieve success with DevOps.

**I often say that “*culture is a door to Engineering DevOps*,” because if leaders and people do not accept it, then no tool or engineering process will be successful. But without an underlying engineering basis, culture is not enough to realize success with DevOps.**

Nevertheless, I and everyone in the DevOps industry owe a huge debt of gratitude to The Phoenix Project and the tireless promotions by its primary author, Gene Kim. Gene is perhaps deserving more credit than anyone for popularizing DevOps through writings, DevOps conference events like the DevOps Enterprise Summit, and motivating other IT leaders to embrace DevOps as a core strategic component at the center of modern digital technology transformations.

A nice article written by Steve Mezak and published on DevOps.com on January 25, 2018, called “The Origins of DevOps: What’s in a Name?” does a masterful job of explaining how the word “DevOps” came into being during 2009 at the O’Reilly Velocity Conference, during which two Flickr employees (John Allspaw, senior vice president of technical operations, and Paul Hammond, director of engineering) gave a now- famous presentation: “10+ Deploys per Day: Dev and Ops Cooperation at Flickr.”

There is no need to repeat the whole history here, but I will point out points relevant to this book. The references to lean manufacturing principles and practices is of high importance to DevOps and its links to engineering. DevOps publications often overlook, or at least gloss over, the extent to which success with DevOps requires, at its core, a strong engineering basis.

The Goal by Eliyahu M. Goldratt, a very entertaining and informative book published in 1984, is a great explanation of lean engineering principles and practices that are so vital to DevOps. The Goal predates DevOps and therefore does not correlate specific engineering practices to a DevOps blueprint, but its underlying principles and practices are relevant. My book “Engineering DevOps” points out how to correlate them to modern DevOps blueprints.

Going further back, the great “wizard” (statistical quality guru and consultant) W. Edwards Deming is credited with much of the “magic” behind the Japanese industrial recovery including the benchmark lean Toyota Production System. Deming wrote several books, including “Out of the Crisis”, first published in 1982, which is a great summary of scientific and engineering principles and practices that are foundational for lean manufacturing and quality assurance and have been applied to software. “Out of the Crisis” also pre-dates DevOps and does not correlate its engineering principles and practices to the DevOps blueprint as this book does.

My own engineering career, starting in 1974 as an engineering student at Queen’s University in Kingston, Ontario, exposed me to software design, building, testing, and deployment—all of which I experienced firsthand. My undergraduate thesis project taught me that testing is an essential part of design and construction. Our small team, consisting of two engineering classmates and I, challenged ourselves to design and implement one of the world’s first microprocessor-based private telephone switching systems and some compatible telephone handsets. This we had to do part-time in between other classes and studies. We knew from the start that we had little room for mistakes, as success with the project was essential to graduation! From the start we decided we needed a parallel track to design and implement code and hardware for both “product (switch with handsets)” and “test-ware.” This was to make sure we could test the system and verify the operation as we created it. When we finished on time, both the switch and test-ware performed well, and the test-ware served as a great demo tool also. Although the project pre-dated the word DevOps by more than 30 years, it used the same underlying lean engineering practices that DevOps uses. The proof was that we graduated on time!

Glenford Myers’ book “The Art of Software Testing”,published in 1979, espoused that developers who write software should not test their own code because they would be biased against finding their own failures. In my own experience, I found this idea is not true as indicated by examples in the next paragraph, but nevertheless it took hold in the software industry, at least for functional and system-level quality assurance. Slow, error-prone, back-end loaded “waterfall” processes with large “independent” quality assurance teams became accepted practice. This occurred despite the lessons learned by lean manufacturing that have shown testing should be conducted early to avoid bottlenecks at the back-end of pipelines. It is as if Deming’s earlier findings that quality is everybody’s responsibility and not to be relegated to some other person was forgotten or ignored “because software is different.” My own experience has shown that there is an INVERSE relationship between quality of engineering projects and the size of the independent quality assurance team.

I observed while developing and leading projects and quality assurance tools development at Bell-Northern Research (BNR) during the 1980s, and collaborating on other engineering projects since then, that the most efficient projects and highest quality products occur when the engineering team takes responsibility for the end-to-end process, including the quality and deployment stages. To do this requires an end-to-end quality and responsibility mindset. It also requires engineering leadership, training, incentives, and engineering design practices; smart testing methods; and good tools. These added engineering “costs” are too often ignored yet consistently have yielded return on investment (ROI) and advantages for business stakeholders when applied correctly.

Kent Beck’s book “Extreme Programming Explained”, first published twenty years after “The Art of Software Testing” in 1999, advocated “test-first development,” which finally put testing back in the hands of developers. Test-Driven Development is now a best practice. Soon after that, The Manifesto for Agile Software Development was published in 2001, and “Agile” became the software world’s favorite new buzzword. Four key tenets of the manifesto are “Individuals and inter- actions over processes and tools,” “Working software over comprehensive documentation,” “Customer collaboration over contract negotiation,” and “Responding to change over following a plan.” I have an opinion that this did little to help the situation from an engineering point of view because many practitioners have interpreted this (incorrectly) to mean, “Software developers should not concern themselves with engineering plans or tests.” The newer “Scaled Agile Framework (SAFE)”, initially introduced in 2011 is much improved and espouses development teams to “generate tests for everything—features, stories, and code—ideally before the item is created, or test-first. Test-first applies to both functional requirements (Feature and Stories) as well as non-functional requirements (NFRs) for performance, reliability, etc.”

**I often say that “testing is the primary key to Engineering DevOps” because only testing provides enough tangible information to measure the quality and completeness of artifacts with engineering precision as they move through the pipeline.**

Effective lean practices require that testing be “shifted left” as much as possible to the earliest stages of the pipeline to avoid costly bottlenecks caused by latent error detection in later stages. Jez Humble and David Farley’s book “Continuous Delivery”, published in 2011, is still regarded as one of the best “how-to” reference books for implementing DevOps, even though it barely mentions the word DevOps and does not use the word “engineering” at all. The book refers to DevOps as a “movement” with the same goal of “encouraging greater collaboration between everyone involved in software delivery to release valuable software faster and more reliably.” While the book does meet the goal, it really goes much further by laying out underlying engineering processes for controlling, testing, packaging, releasing, and deploying software changes to production. However, the book does not cover software engineering requirements for planning, design, and post-release “operations,” which are also critical for engineering DevOps. Such topics are covered by a scattering of other books and articles.

My book “Engineering DevOps” emphasizes that a well-engineered, high-performance DevOps requires a complete engineering blueprint with engineering practices that cover the end-to-end set of activities—not just a CI or CD pipeline.

The “DevOps Handbook”, first published in 2016, is another valuable contribution to the literature that everyone interested in DevOps needs to have. The book includes a wealth of suggestions and case studies for areas beyond Continuous Delivery, including some aspects of software design and, to some extent, operations. However, it does not provide a measurable definition of DevOps, nor does it mention engineering or provide a comprehensive engineering blueprint for implementing DevOps that starts with planning and runs through to operations from any starting point.

I have personally worked with many clients who, despite having read all the great published suggestions and case studies, are still confused about what exactly DevOps is and how they should be implementing it in their organizations. Given this historical perspective, it is my thesis that

**DevOps can and should be framed as an engineering problem that has an engineering solution. A complete engineering solution requires a blueprint that covers all the people, process, and technology end to end from conception through to operations, not just a CI/CD pipeline. This is the thesis of my book Engineering DevOps”.**

**The Dilemma of Deﬁning Engineering DevOps**

Business leaders and practitioners need to know, in concrete engineering terms, when DevOps has been accomplished to justify investments that support business goals and how to set limits on projects. Pinning down a universal definition of DevOps that satisfies the diverse community of stakeholders and survives the test of time has proven elusive. DevOps spans a broad and evolving body of knowledge covering multiple human, lean process engineering, and technology disciplines. As explained in “The Phoenix Project”, DevOps is never really “done” because the highest level of maturity, “The Third Way” (Continuous Improvement), defies setting final boundary conditions. To reconcile this dilemma, some have defined DevOps using vague or transient terms such as a “cultural movement.” Others have skirted the problem of definition with long paragraphs describing what is involved to do DevOps, rather than state a definition in concise and concrete terms. Despite these challenges, acceptance of a definition by stakeholders is a key first step in engineering any successful DevOps transformation. Tracking DevOps project progress requires clear definitions that can be measured in concrete engineering terms.

**Deﬁnition of Engineering DevOps**

My definition of Engineering DevOps is broad enough to stand the test of time, yet specific enough to answer key questions addressed by a good definition, including “What is it?,” “How it is done?,” and “Why do it?” while being concise. When applied to a specific project, the what, how, and why provide tangible things that can be measured in precise engineering terms.

**“Engineering DevOps is the application of lean engineering practices (Continuous Flow, Feedback, and Improvement) to the Nine Pillars of Engineering DevOps (Leadership, Collaborative Culture, Design for DevOps, Continuous Integration, Continuous Testing, Elastic Infrastructure, Continuous Monitoring, Continuous Security, and Continuous Delivery) and Three Dimensions (People, Process, and Technology) for the benefit of agility, stability, efficiency, quality, security, availability, and satisfaction.”**

Whether you create or choose another definition or decide to use mine, it is important that your cross-functional teams agree on one definition on which to align their goals and implementations and set a basis to measure DevOps progress and business outcomes.

**Summary**

In summary, the answer to the question: “What are the key ingredients to assure success with DevOps?” has been explained in this white paper. Engineering DevOps provides guidance, templates, blueprints and recommended engineering practices to assure successful DevOps implementations.

**Learn More**



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from <https://engineeringdevops.com/documents>

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

© EngineeringDevops 2019, 2020

All rights reserved

**What Is Engineering DevOps?**