How to Engineer People, Process and Technology for DevOps?

This paper is derived from selections from my 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)

May 8, 2020

© Engineering DevOps Consulting 2019, 2020

All rights reserved

**How to Engineer People, Process and Technology for DevOps?**

DevOps at its best is engineering greatness. In my experience, well- engineered DevOps always starts by defining a great goal. Greatness is won by careful inspirational leaders, measured tactics, superior technology, a culture of collaboration, courage, and unwavering persistence to applying skills to the task until the goal is attained.

**Does DevOps Engineering Require People to be Engineers?**

Before we get too far into this chapter, we need to address the elephant in the room. DevOps belongs in the category of software engineering more than computer science, and that distinction matters to DevOps. Whether software belongs in the science or engineering category has been the subject of expert debate for many years that continues to this day. Back in 1977, my rural high school guidance counselor was unable to explain the difference between computer science and software engineering sufficiently for me to make a choice of which university program I should apply to. At first, I thought it was simply the lack of knowledge on the part of my high school counselor, so I started my own research efforts to get an answer. After talking to other university counselors and real-world computer scientists and engineers, I determined that there was no consensus.

This is more than just an academic debate. A recent ruling by the Canadian Province of Quebec is a clear example of the way this debate is relevant beyond the halls of academia.

“Quebec engineers win court battle against Microsoft—The software giant is penalized over its use of the word “engineer” in its professional certification program. Just when you thought it was over, an old debate is reignited.

According to the Institute of Electrical and Electronic Engineers (IEEE), software engineering is “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software”.

According to the Association for Computer Machinery (ACM), “Computer science (CS) is the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society.

So is computer science is about “studying,” while engineering is about “applying” software? This seems to be an oversimplification based on my own observations of computer science graduates and software engineering graduates working side by side, performing the same jobs in real industry software projects. Nevertheless, both academia and industry continue to distinguish professional software engineering degree programs from computer science degree programs. These degree programs have overlap, but there are key differences in their curricula.

The “Software Engineering 2014 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering was undertaken by a Joint Task Force on Computing Curricula—IEEE Computer Society and Association for Computing Machinery February 2015. This joint task force consisted of a distinguished panel of engineering and computer scientist experts, which concluded that “particular attention has been paid to incorporating engineering practices into software development so as to distinguish software engineering curricula from those appropriate to computer science degree programs.

 Whereas scientists observe and study existing behaviors and then develop models to describe them, engineers do the following:

1) Engineers use such models as a starting point for designing and developing technologies that enable new forms of behavior.

2) Engineers proceed by making a series of decisions, carefully evaluating options, and choosing an approach at each decision point that is appropriate for the current task in the current context. Appropriateness can be judged by trade-off analysis, which balances costs against benefits.

3) Engineers measure things, and when appropriate, work quantitatively. They calibrate and validate their measurements, and they use approximations based on experience and empirical data.

4) Engineers emphasize the use of a disciplined process when creating and implementing designs and can operate effectively as part of a team in doing so.

 5) Engineers can have multiple roles: research, development, design, production, testing, construction, operations, and management in addition to others such as sales, consulting, and teaching.

6) Engineers use tools to apply processes systematically. Therefore, the choice and use of appropriate tools is a key aspect of engineering.

7) Engineers, via their professional societies, advance by the development and validation of principles, standards, and recommended engineering practices.

8) Engineers reuse designs and design artifacts.

To summarize the key points of the above joint task force, the key distinction between software engineering and computer science primarily has to do with methodologies of software design and development. Software engineering methodologies, while having some overlap with computer science, align more perfectly with DevOps methodologies.

For designing and developing, software engineers are more apt to use models as a starting point; evaluate each decision point; measure quantitatively; calibrate and validate their measurements; use a disciplined process; have multiple roles; use tools to apply processes systematically; advance by the development and validation of principles, standards, and recommended engineering practices; and reuse designs and design artifacts.

But wait! Most modern software applications use software from open source. If DevOps is software engineering, then open source software development must follow software engineering practices. It turns out that it does. According to an interesting article, “Open Source Software Engineering: An Introduction to Open Source Tools, “software engineers make tools and applications that enable users in many domains to perform their work more effectively and efficiently, yet frequently. Thanks to open source, we not only get the source code for development but also the tools to deliver high-quality products.”

!! Key Concept !! DevOps Is More Software Engineering than Computer Science

**Engineering methods are key to DevOps success. Arguments that software is more of an art form than something that fits engineering disciplines apply to the creative side of developing software products, while DevOps has to do with disciplined methodologies for designing, development, and production of software. A well-engineered DevOps in no way inhibits creativity. Indeed, a well-engineered DevOps facilitates creative design because it reduces production bottlenecks from the software creator.**

It is my argument here that DevOps should be classified under software engineering and performed using engineering disciplines. I want to emphasize that this does not mean that only software engineers with an engineering degree should be doing the work of DevOps. I hope that is obvious, but I fear I better clarify that in case some readers think I am a snobbish engineering curmudgeon.

So why does it matter that DevOps is software engineering? The approach to understanding and implementing DevOps from a software engineering perspective emphasizes DevOps blueprint models and specifications, disciplined measurable processes, calibrated tools, systematic progress tracking, principles and recommended engineering practices, validation, strict governance, and artifact reuse.

DevOps has often been referred to as a journey. Not all journeys are alike. Too often I have seen organizations with DevOps journeys that are following a meandering, ad hoc path with few measurable progress milestones, resulting in backtracking or getting lost and not getting to the destination at all. The engineering approach described in this book provides a more certain way to survey the best path, build a clear roadmap based on measurable milestones, and enable a speedy route for the organization to accomplish its goals.

**DevOps People, Process, and Technology Engineering Maturity Levels**

In the context of engineering, measurements are a critical component. Knowing where you are and where you should go next in an organized, stepwise fashion is a key tenet of engineering. One of the most confusing things about describing how DevOps is engineered is that there are different levels of DevOps maturity, and those levels of maturity are not defined in a standard way. How can you describe a DevOps implementation in any concrete, measurable engineering terms without a measurement guideline? I am not attempting to define standards for DevOps in this book; however, organizations can and should define their own versions of maturity definitions and then use those to calibrate their own DevOps implementations and progress towards higher levels of maturity.

The Software Engineering Institute defined a Capability Maturity Model (CMM) for software with five levels of maturity as depicted in ***Figure 1—Capability Maturity Model***. This model, first published in IEEE Software in March 1988, is a de facto industry standard model for defining the maturity of processes and can be applied to DevOps nicely. The CMM contains five levels of software process maturity: Initial, Repeatable, Defined, Managed, and Optimizing.

 **Initial** indicates the lack of a stable environment for developing and maintaining software. Few stable software processes are in place, and performance can only be predicted by individual, rather than organizational, capability.



Figure 1: Capability Maturity Model

**Repeatable** indicates basic level software management controls are in place at the project level. Example measurable elements include the following:

• Software configuration management

• Software quality assurance

• Software subcontract management

• Software project tracking and oversight

• Software project planning

• Requirements management

**Defined** indicates a standard process for developing and maintaining software is followed across the organization. Example measurable elements include the following:

• Peer reviews • Intergroup coordination

• Software product engineering

• Integrated software management

• Training program

• Organization process definition

• Organization process focus

**Managed** indicates quality and processes are measured and analyzed quantitatively across the organization. Example measurable elements include the following:

• Quality management

• Process measurement and analysis

**Optimizing** indicates the organization is practicing continuous process improvement. Example measurable elements include the following:

• Process change management

• Technology innovation

• Defect prevention

**The CMM levels can be used to create a maturity model for DevOps**. In doing so, it is important to include measurable elements for the three major dimensions of DevOps—People, Process, and Technology—at each maturity level. ***Figure 2—Engineering DevOps Maturity Levels*** is an example of a DevOps maturity model and measurable elements that I use.

**Chaos** **is the initial level of DevOps maturity** and very similar to the Initial level of CMM. There is a lack of a stable environment for developing and maintaining software. Teams are separated into distinct silos with little cross-team communication between them. Few stable software processes are in place, and performance can only be predicted by individual, rather than organizational, capability. Most of the processes including software builds and tests and environment setups are manual and error prone.

**Continuous Integration** **is the second level of DevOps maturity,** which correlates to the Repeatable level of CMM. At this level, the focus is to get the front end of the pipeline on solid engineering basis. It would be foolhardy to proceed to automated delivery and deployment until builds and tests and production of release artifacts are repeatable. At this level, there is some cross-team information sharing to support automated builds, release artifacts, and tests sufficient to manage integration.

**Continuous Flow is the third level of DevOps maturity**, which correlates to the Defined level of CMM. This level is equivalent to The First Way of DevOps discussed in The Phoenix Project. At this level, a standard end-toend process exists in the form of a highly orchestrated and automated pipeline for developing and delivering software. DevOps knowledge and skill levels across the cross-function team are substantial and supported with DevOps training programs. Infrastructure is orchestrated as code. Release acceptance metrics and analysis are automated.

**Continuous Feedback is the fourth level of DevOps maturity,** which correlates to the Managed level of CMM. This level is equivalent to The Second Way of DevOps discussed in The Phoenix Project. People, Process, and Technology dimensions are measured and analyzed quantitatively, and the analytics drive actions such as automated deployments and rollbacks.

**Continuous Improvement is the fifth and highest level of DevOps maturity**, which corresponds to the Optimizing level of CMM. This level is equivalent to The Third Way of DevOps discussed in The Phoenix Project. At this level, there is a culture of continuous experimentation and improvement. Risks and costs are continuously optimized. Technology solutions support zero downtime deployments, immutable infrastructures, and resiliency.

If these DevOps maturity levels and measurable elements do not fit your organization, that is okay. You are free to create your own if you like. The key point here is that you will need to have a definition of maturity and measurable elements to support an engineering approach to DevOps. And it is critical that you obtain a high level of consensus in your organization before proceeding to use it. Many organizations make the mistake of over-focusing on technologies for DevOps without ensuring associated people and process requirements are kept in step. In this paper, I refer to Three Dimensions of Engineering DevOps because all three apply to each of the Nine Pillars of Engineering DevOps.



Figure 2: Engineering DevOps Maturity Levels

**Three Dimensions of Engineering DevOps— People, Process, and Technology**

As shown in the ***Figure 3—Three Dimensions of Engineering DevOps*** materials that are required for DevOps come in the categories of People, Process, and Technology. Jennifer Davis and Katherine Daniels in their book Effective DevOps—Building a Culture of Collaboration, Affinity, and Tooling at Scale state that “successful DevOps culture requires the intersection of people, process, and tools.” If you try implement DevOps without considering all three you will be missing key dimensions. Building anything with missing dimensions doesn’t work!

**People—Leadership, Organization, Teams, and Culture**

The People dimension of DevOps includes all the human elements that make DevOps successful. This includes leadership, organization, teams, and culture. Effective DevOps requires “an organization that embraces culture change to affect how individuals think about work, value all the different roles that individuals have, accelerate business value, and measure the effects of the change. Conway’s Law (named after computer scientist Melvin Conway) indicates that software tends to reflect the organization structure that created it. Organizations that have trouble communication between team members will generally create software that has communication problems. The following paragraphs characterize the People dimension of DevOps for each of the five maturity levels:

• At the Chaos level of DevOps maturity, teams and organizations exist within separate departments (silos) with little communication between them. There is a little clarity regarding tasks that require shared accountability. This results in a culture of finger-pointing and blame. There is an overdependence on individuals and experts to perform critical tasks. Training and cross-training are not high priorities. Leadership is primarily focused on local department goals and spends little time bridging across departments.

• At the Continuous Integration level of DevOps maturity, leadership is proactively involved in directing communications between department silos, especially for activities that involve cross- functional participation, including peer reviews, build artifact testing, and software integration. Workload backlogs are systematically managed. Cross-team training is a priority but generally not formalized.

• At the Continuous Flow level of DevOps maturity, The First Way of DevOps, leadership visibly and proactively advocates for and sponsors DevOps practices. Collaboration between cross- functional teams is engrained in the culture. DevOps training programs are formalized to ensure team members have DevOps knowledge and skills.

• At the Continuous Feedback level of DevOps maturity, The Second Way of DevOps, the culture is largely self-directed, using shared metrics with a focus on removing bottlenecks. Service Level Indicators (SLIs), Service Level Objectives (SLOs), and Service Level Agreements (SLAs) are used to measure the performance of DevOps systems. The data results drive actions. DevOps training involves the use of advanced concepts such as mentors and guilds.

• At the Continuous Improvement level of DevOps maturity, The Third Way of DevOps, there is a culture of continuous experimentation and improvement. There is a prevailing confidence in the organization and its ability to deliver products quickly and without risk. Training programs that emphasize mastery of the DevOps craft, including reaching out of the organization for expertise, bringing in industry experts, and proactive participation in industry events, is strongly encouraged.



Figure 3: Three Dimensions of Engineering DevOps

**Process—Value Streams and Workﬂows**

The Process dimension of DevOps includes all the workflows that support the pipeline of activities required for the end-to-end value stream of an organization. A value stream is the series of activities creating a flow of value realized by a product or service that the customer gets. Workflow is a sequence of processes through which a piece of work passes from initiation to completion. The following paragraphs characterize the Process dimension of DevOps for each of the five maturity levels:

• At the Chaos level of DevOps maturity, requirements, planning, and tracking processes are poorly defined and operated manually. The results are unpredictable, and systems—where they exist—are designed to be reactive.

• At the Continuous Integration level of DevOps maturity, processes that implement workflows are defined within the silos within which they operate to support software development, testing, and integration. There are no standards for end-to-end processes. Systems can repeat what is known, but reaction to unknown circumstances require human intervention.

• At the Continuous Flow level of DevOps maturity, The First Way of DevOps, value streams are defined and processes that make up the end-to-end continuous delivery pipeline are automated. Standards exist for cross-functional workflows across the organization for applications, release processes, and infrastructure.

• At the Continuous Feedback level of DevOps maturity, The Second Way of DevOps, proactive monitoring systems are in place for all the key elements of the value stream. Metrics are systematically collected and analyzed against business goals. Performance indicators are visible and used proactively to manage processes for applications, pipelines, and infrastructure elements.

• At the Continuous Improvement level of DevOps maturity, The Third Way of DevOps, self-service automation is available to developers. Risk and cost optimization are measured and pro- actively managed. There is a high degree of experimentation with new processes to improve performance.

**Technology—Products and Tools**

The Technology dimension of DevOps includes technical design and capabilities for products and services; infrastructures; and the tools that are used to plan, create, build, test, package, deploy, and support them. The following paragraphs characterize the Technology dimension of DevOps for each of the five maturity levels:

• At the Chaos level of DevOps, maturity tools tend to be “hand tools” rather than production-grade automated tools. Infra- structures and tools that are used to support planning, designs, builds, tests, and deployments are typically are operated manually, with few (if any) disciplined playbooks. The lack of automation or concrete playbooks results in variances and inconsistent results.

• At the Continuous Integration level of DevOps maturity, application source code is maintained with a version management system. Software builds, the production of images and release artifacts, and tests are largely automated. This results in repeatable releases, but the release and deployment end of the pipeline requires manual effort, which is error-prone and painful.

• At the Continuous Flow level of DevOps maturity, The First Way of DevOps, a CI/CD toolchain orchestrates and automates builds and tests and packaging deliverables. Infrastructure is orchestrated as code. Products, services, and tools are instrumented to report metrics as logs. Automated metrics and analysis are used extensively for release acceptance and deployment.

• At the Continuous Feedback level of DevOps maturity, The Second Way of DevOps, applications, pipelines, and infrastructure components are fully instrumented. Activities are driven by metrics and analytics dashboards. Deployments and roll-backs are orchestrated and automated using metrics and analytics to guide deployment and roll-back decisions.

• At the Continuous Improvement level of DevOps maturity, The Third Way of DevOps, deployments are using zero downtime methods such as Green/Blue, A/B, and Canary methods. Infrastructures are immutable using containers (such as Docker) and cluster deployment tools (such as Kubernetes). Resiliency of pipelines and infrastructures is actively tested for by forcing failures and enforced by using policies and automation.

**Twenty-Seven DevOps Engineering Critical Success Factors**

The components to implement the Nine Pillars of DevOps for each of the Three Dimensions of DevOps need to be appropriate for the level of maturity that is being implemented, and at any time they must be kept in a balance. This seems obvious when you think about it. Did you ever see a gold toilet in an outhouse, or an outhouse in a five-star hotel?

**Well- engineered systems are designed carefully to strike an optimal balance between different factors to accomplish the goals of the project.**

It amazes me how many organizations busy themselves over- engineering some parts of DevOps at the expense of others—and then they wonder why the end-to-end system is not balanced. I think this occurs when the implementers have a siloed perspective of DevOps rather than a big-picture engineering blueprint in mind and a clear understanding of the pillars, dimensions, and maturity levels of DevOps needed to clearly define a well-engineered solution.

As an aide-memoire (I took French in school—are you not impressed? Oui ou non?), think of engineering DevOps like a three-dimensional game, as shown in the ***Figure 4—DevOps Engineering 3D Game***. The game levels are the Three Dimensions (People, Process, and Technology). You get twenty- seven pieces to place on squares of the game. The game is “won” when all the pieces for all the pillars, for each of the dimensions, line up within the target maturity level. It is also okay if you have some pieces at higher levels of maturity, but that’s not desired because that’s over-engineering. Another way to think about the correct relationship between the Nine Pillars and Three Dimensions of well-engineered DevOps for any one maturity level is the cube puzzle illustrated in the ***Figure 5—DevOps Engineering Cube Puzzle***. The puzzle is “solved” when each of the Nine

Pillars are in line with the Three Dimensions. If you have specifications for DevOps pillars that are missing any DevOps dimension, then you do not have a solution! The correct solution resolves to twenty-seven combinations of Nine Pillars times Three Dimensions. I refer to these as twenty-seven DevOps engineering critical success factors, because the omission of even a single factor jeopardizes the solution.

I know what some (most?) of you are thinking. “Twenty-seven things! I’m not good at solving cube puzzles. I have trouble keeping track of any more than three things. Coffee, milk, and sugar is enough to get right each morning. How can I be expected to balance twenty-seven things?!” My answer is simple, folks. You want the benefits? You deal with it. Do you really think leaders and practitioners of complex engineering projects like designing and building castles, airplanes, boats, web services, and almost every other technology-centric professional doesn’t have to engineer at least twenty-seven things to get their products and services right?



**Figure 4: DevOps Engineering 3D Game**



**Figure 5: DevOps Engineering Cube Puzzle**

**Summary**

In summary, the answer to the question: “How to Engineer People, Process and Technology for DevOps?

has been explained in this white paper.

**Successful implementations of DevOps are engineered by balancing 27 critical success factors. The critical success factors are a cross product of the Nine Pillars of DevOps with Three Dimensions - People, Process and Technology. Five DevOps maturity levels : Chaos, Continuous Integration, Continuous Flow, Continuous Feedback and Continuous Improvement, which are defined in terms of People, Process and Technology, provide a useful guide for engineering DevOps.**

**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?**