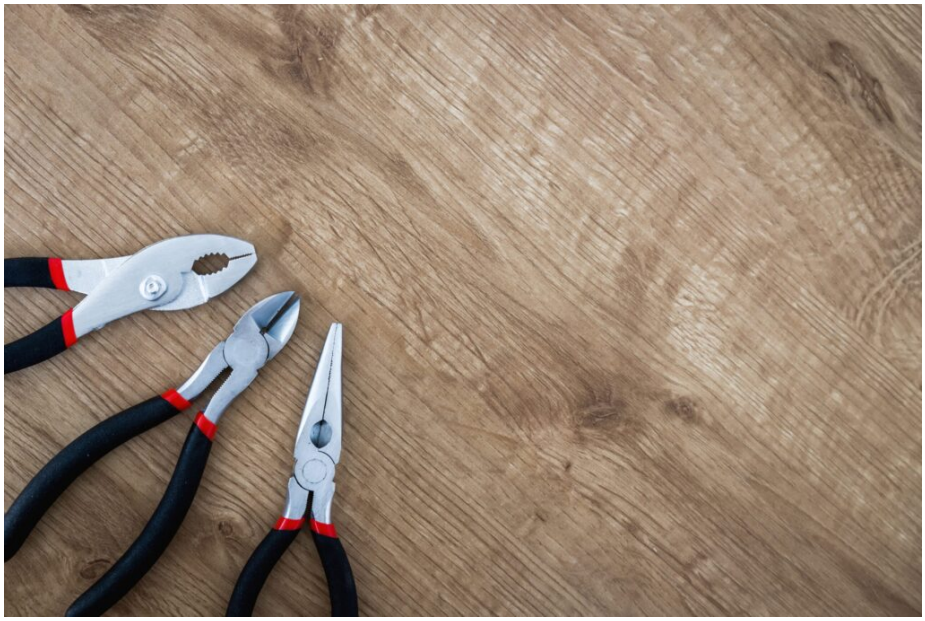
Issue 016: DevOps

Graham Lee

January 6th, 2020



Welcome to the sixteenth issue of *De Programmatica Ipsum*, dedicated to the subject of *DevOps*. In this edition:

- Graham explains why DevOps is the same thing as before[1], but this time is different.
- Adrian explores the current state of DevOps[2] at the end of this decade.
- In the Library section[3], Adrian reviews two major works by Bertrand Meyer[4]: *Object-Oriented Software Construction, 2nd Edition,* and *Agile! The Good, the Hype and the Ugly*.

Enjoy this issue! Please subscribe to our free newsletter[5] to stay updated about new releases, or contribute[6] if you would like to support our work.

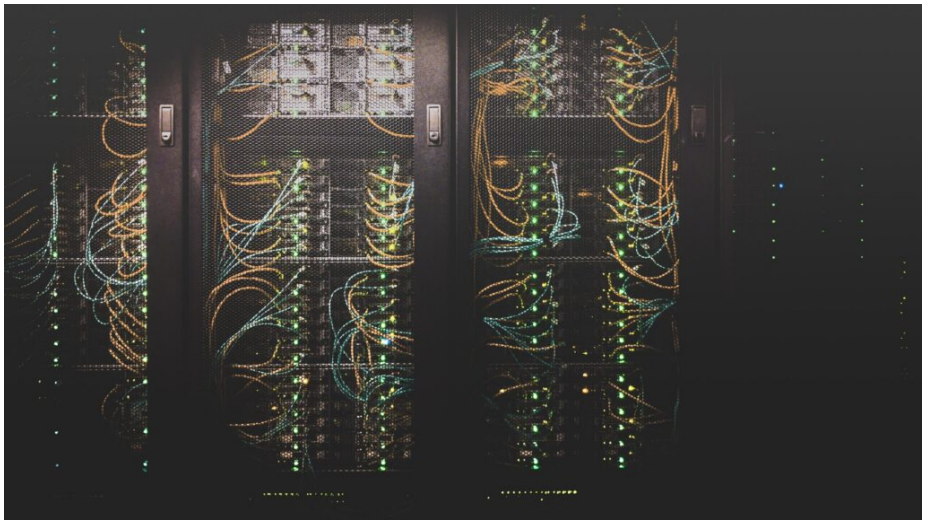
Cover photo by JESHOOTS.COM[7] on Unsplash[8].

[1] https://deprogrammaticaipsum.com/play-it-again-sam/
[2] https://deprogrammaticaipsum.com/antonomasia/
[3] https://deprogrammaticaipsum.com/category/library/
[4] https://deprogrammaticaipsum.com/bertrand-meyer/
[5] https://deprogrammaticaipsum.com/newsletter/
[6] https://deprogrammaticaipsum.com/contribute/
[7] https://unsplash.com/@jeshoots?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[8] https://unsplash.com/s/photos/maintenance?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Antonomasia

Adrian Kosmaczewski

January 6th, 2020



Antonomasia is a word derived from the Greek ἀντονομάζειν, which means, "to name differently." It is a rhetoric figure of speech, or metonymy, in which a concept is named after another closely related. Classical examples are, for example "King" for Elvis Presley, or "Fab Four" for The Beatles. In technology, "Xerox" for photocopying, "Big Blue" for IBM, "Google" for searching, but (surprisingly enough) not "iPhone" for smartphone. In the software industry, a similar phenomenon happens: "C++" for object orientation; "Java" for web apps; "Jenkins" for CI/CD; "Python" for machine learning; "Scrum" for Agile; and Kubernetes[1] for DevOps.

DevOps, as a word or as a concept, did not exist 10 years ago: it was born as part of a conference name in 2009[2]. Back then, Android and iOS were still niche players in the mobile industry, Facebook had not yet been acknowledged as an effective threat to democracy, and the Marvel Cinematic Universe consisted of... two movies. A very different world, yet all too similar.

Avid readers of this magazine might remember that in the first issue, published in October 2018, the author of these lines wrote[3] that "Hype spreads like cancer." The thought is still valid, and DevOps should be studied to find whether it is just a temporary hype, or a fad. But if recent experience is to be believed, DevOps might be the strongest new trend in the software development industry, at least since Agile became a manifesto[4] almost 20 years ago. Where other approaches have failed, DevOps has stayed and spread.

This is a fact. If you are doing DevOps between 2019 and 2020, you are most probably using Kubernetes.

[1] https://kubernetes.io/
[2] https://legacy.devopsdays.org/events/2009-ghent/
[3] https://deprogrammaticaipsum.com/mainstream-is-the-new-hype/
[4] http://agilemanifesto.org/

The "container orchestration wars" are over, my general, and Kubernetes won. In any of its incarnations: K3s[5], OpenShift[6], Rancher[7], NetApp[8], you name it. They all won. Applications became small Docker[9] containers, lightweight virtual machines that can start and stop in a heartbeat. The programming language does not matter anymore. The IDE does not matter anymore. The framework does not matter anymore. The build tool does not matter anymore.

And when I say Kubernetes won, I mean its whole ecosystem: Terraform[10]. Prometheus[11]. Helm[12]. Do not worry if you have never heard these names before. They all come in the same bus as Kubernetes, only in the back seats. They might not be immediately visible. Like Knative[13] back there, texting with Envoy[14] and Fluentd[15] who stayed at home[16] watching TV.

The container matters, and as long as it is exposing an HTTP port, you are on the way to DevOps. It matters so much that IBM bought Red Hat for 34 billion USD.

Kubernetes (a word derived from the same Greek word that begat "Government" and "Cybernetics") is a beautiful black box, working as a declarative engine; you want a load balancer and three instances of your application running? Kubernetes will watch the state of your system for you, and launch enough "pods" so that you have exactly what you described. Yes, "pods." An abstraction layer on top of those Docker containers we talked about previously.

Speaking about a description, YAML[17] went from being the humble configuration format used in Ruby on Rails applications, to be the configuration format of deployments, services, ports, storage, and anything composed thereof. Everything is defined in YAML, and that YAML is fed to a command line tool, usually written in Go[18], so that this everything defined in YAML is "deployed" to the "cloud." And because Kubernetes is yet another layer of abstraction, it does not matter if you are in AWS, Azure, Google Cloud, or even in Alibaba's own cloud. Your containers just run™®©. Kubernetes is watching.

Teams then can concentrate on the "descriptive" part of DevOps, and they push their code to a git[19] repository, because who is using Subversion[20] these days, anyway. Your commit wakes your CI/CD system up, who in turn clones the repo, builds the binaries, executes the unit tests, creates and pushes the Docker containers, and deploys a new version of your application to your staging environment a few minutes later. That CI/CD thingy might even send a message on Slack[21] to someone in the testing team, because, who reads e-mails these days, anyway.

---

[5] https://k3s.io/
[6] https://www.openshift.com/
[7] https://rancher.com/
[8] https://cloud.netapp.com/kubernetes-service
[9] https://www.docker.com/
[10] https://www.terraform.io/
[11] https://prometheus.io/
[12] https://helm.sh/
[13] https://knative.dev/
[14] https://www.envoyproxy.io/
[15] https://www.fluentd.org/
[16] https://www.cncf.io/
[17] https://yaml.org/
[18] https://golang.org/
[19] https://git-scm.com/
[20] https://subversion.apache.org/
[21] https://slack.com/

This is how, in 2017, DevOps became GitOps[22] and everything was hype again.

Want to push a new version to production? If you are using something like Git Flow[23], just push to `master` and your CI/CD system will automatically execute the `kubectl` command and deploy your code to production. Or if you are using GitLab[24] (who is not these days?) your `.gitlab-ci.yml` file can trigger a deploy simply after pushing a humble tag.

Deploy to production 20 or 30 times a day? No problem. Still using SFTP to deploy your PHP apps? Poor soul. I hope you are using Semantic Versioning[25] for those tags, though.

Of course, not all developers can be as "DevOpsy" or "GitOpsy" or "DevSecOpsy" as they wish. Mobile developers, a group dear to the heart of this author, still has to juggle with manual deployments (usually via a pitifully broken web interface or IDE.) But if you have to wait for a human to approve your app, why would you want to automate your deployments, anyway?

The closest thing that mobile developers have to Kubernetes (sorry, DevOps) is Fastlane[26], but then again, one wonders endlessly about the level of automation that mobile apps could reach, if their respective vendors actually cared.

Cover photo by Taylor Vick[27] on Unsplash[28].

[22] https://www.gitops.tech/

[23] https://nvie.com/posts/a-successful-git-branching-model/

[24] https://gitlab.com/

[25] https://semver.org/

[26] https://fastlane.tools/

[27] https://unsplash.com/@tvick?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[28] https://unsplash.com/s/photos/servers?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Play It Again, Sam

Graham Lee

January 6th, 2020

While the phrase "Play it again, Sam" never occurs in the film Casablanca, the sentiment does. Sam is asked to play "As time goes by", a song about how some things remain the same despite the passage of time. The song was originally part of a 1931 Broadway musical, and perhaps ironically due to a musician's union strike when Casablanca was released, the studio were unable to release a new version of the song as a record. Instead, they re-issued the original version, recorded over a decade earlier.

DevOps is the "play it again, Sam" of software methodologies. Its central message is that development and operations staff should work together closely on the shared goal of producing working software, and that other activities should be subordinate to that goal.

This echoes very closely the earlier principles from the Agile manifesto, that working software should be the primary measure of progress, and that the best systems arise from self-organizing teams. These principles came to Agile from other, earlier sources: Extreme Programming[1], Crystal[2], DSDM[3], Rapid Application Development[4].

DevOps also borrows from the Lean Software Development[5] movement, using ideas like empowering the whole team, optimizing the whole, and deciding as late as possible. Lean Software Development in turn got these ideas from lean manufacturing, a movement developed from the Toyota Production System[6] worked out in Japan during its post-war indus-

[1] http://www.extremeprogramming.org/
[2] https://en.wikipedia.org/wiki/Alistair_Cockburn
[3] https://en.wikipedia.org/wiki/Dynamic_systems_development_method
[4] https://en.wikipedia.org/wiki/Rapid_application_development
[5] https://en.wikipedia.org/wiki/Lean_software_development
[6] https://en.wikipedia.org/wiki/Toyota_Production_System

trial reconstruction.

We can draw a shorter line from the manufacturing industry to DevOps. The business novel, The Phoenix Project[7], which tells a fictional example of an IT department transformed through discovery and implementation of the DevOps principles, is a straightforward retelling of an earlier book. Eliyahu Goldratt's business novel The Goal[8] tells the story of a manager at a failing manufacturing plant who turns it (and, bizarrely, his marriage) around by applying Goldratt's Theory of Constraints[9] to his operation. Perhaps the biggest difference between the two novels is that Goldratt's protagonist gets promoted *after* demonstrating the success of his techniques, while Kim's is promoted while it's still clear to him and his manager that he has no clue what he's doing. This ass-backwards career progression is necessary to make the book realistic in a software development context.

And DevOps is not the end of this chain of retellings. DevSecOps tells us exactly the same thing as DevOps, but with security[10] in the mix. It reminds us that security is not achieved by adding a penetration test or compliance-ticking activity at the end of our development process, but is part of the design, implementation, delivery and operation of a successful software system.

We will see this again. Perhaps the blog post that is seen as the seed of the next post-DevSecTestDocBlockchainUXAIOps revolution has already been published, and hasn't been recognised yet. The thing is, this repetition is necessary. It's necessary for multiple reasons.

- Contexts change. The world in which DevOps was introduced was not the world in which Agile was introduced. Through the prism of distance, people understanding software development methodology now wouldn't necessarily see what the big deal is about Agile and what was so bad about what we were doing before, after all don't Windows, Macintosh, Unix, the web, the Internet, and everything all predate it?
- People change. Plenty of programmers now weren't alive when the Agile manifesto was published. Almost all of us weren't around to watch Japan being rebuilt after 1948. Some of us who were once junior engineers are now senior, or have moved into QA or documentation, or out of in-house teams into shrinkwrap product development, and need to understand how to apply our principles to our new environments.
- Stories grow stronger through repetition. If one person tells you about a new way of doing things, they're a crackpot. Ten people are a cult. A thousand are a movement. A million are a revolution. A story told today is a flash in the pan. If it's still being told next year, it's a school of thought. If it's still being told in two thousand years, it's a culture.

So here's to DevOps, to DevSecOps, and to whatever comes next. And to Lean, Agile, Toyotism, and whatever came before. Long may we keep retelling our stories, and listening to others'.

Cover photo by Markus Gjengaar[11] on Unsplash[12].

[7] https://itrevolution.com/book/the-phoenix-project/

[8] https://en.wikipedia.org/wiki/The_Goal_%28novel%29

[9] https://www.tocinstitute.org/theory-of-constraints.html

[10] https://deprogrammaticaipsum.com/issue/issue-003-security/

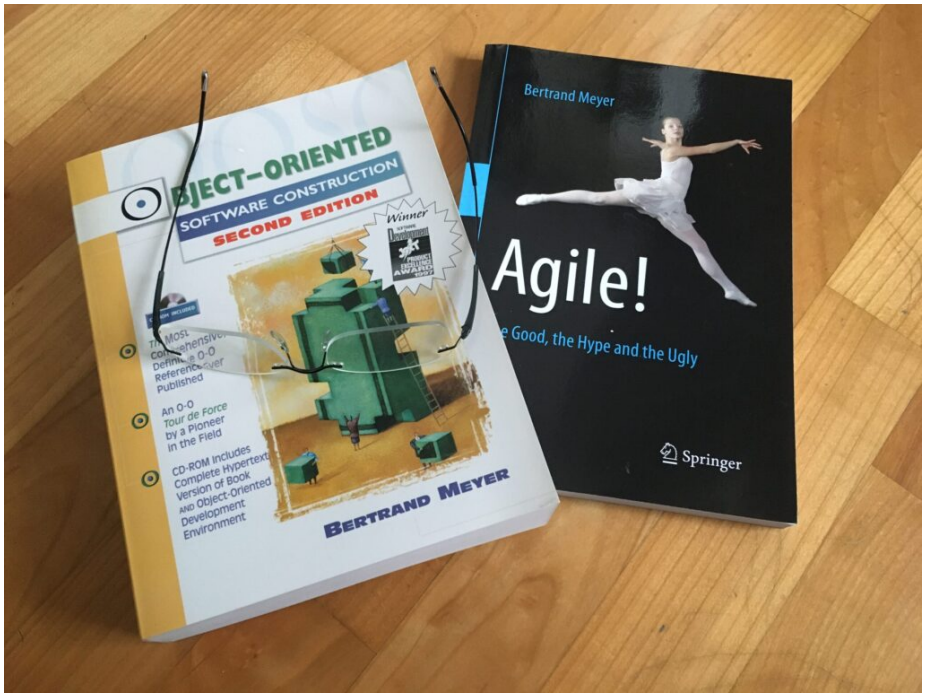
[11] https://unsplash.com/@markus_gjengaar?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[12] https://unsplash.com/s/photos/piano?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Bertrand Meyer

Adrian Kosmaczewski

January 6th, 2020



When the author of these words started his career as a software developer, "object orientation" was all the rage. "Serious" programming languages were object oriented. "Professional" programming environments allowed one to view "objects" and "classes" in all of their glory. Inheritance, not composition, was the way of the future. Design patterns names were the answer to actual interview questions.

As many other developers, the path of this one started, however, with a humble programming language, sadly not yet dead[1], called VBScript. In its version 5, released in 2000, this little mutant language featured a somewhat new concept called "classes," which at that time were more akin to C structs than anything else. These "classes" could not be inherited; well, to be fair they could not even have methods. They consisted of only data fields, and were more or less useful as a way to pass around large amounts of data back and forth from a `Sub` to a `Function`. In later versions (5.8 being the last one, apparently) this feature grew stronger and more complete[2].

At that time, however, more interesting things were happening, like Java and its (at the time) clone called C#. Because, honestly, reading Java code starting with `public static void main(String args[])` was not much more difficult than reading `public static void Main(string[] args)` in C#. Thankfully things got more interesting starting with C# 2.0, released in 2005. But let us not digress.

[1] https://isvbscriptdead.com/
[2] https://docs.microsoft.com/en-us/previous-versions//4ah5852c%28v%3dvs.85%29

The priority in 1999 was, then, to learn object orientation (and XML, as well.)

As fate wanted things to be, perusing the shelves of a bookstore in Buenos Aires, this author ended up buying a copy of Grady Booch's Object-Oriented Analysis and Design with Applications, 2nd Edition[3] (OOADA.) Not a bad choice, actually; after all Booch is one of the authors of UML, so he knows a thing or two about objects. The best part of OOADA is the introduction, drawing ideas from psychology and sociology to explain what classes and objects meant. The final parts, featuring C++ code, are more arcane and difficult to follow for developers new to the language and the concepts (although a more recent 3rd Edition is said to feature Java code instead of C++.)

With hindsight, a much better choice would have been to acquire Bertrand Meyer's[4] Object-Oriented Software Construction, 2nd Edition[5] (OOSC) instead. Not only because of breadth and depth (as suggested by a quick comparison of the page count, from 500 to 1200) and by language choice (Eiffel beating C++ in readability, particularly for a beginner in the field) but also, and most importantly, because of its *style*.

Bertrand Meyer is, hands down, the best writer in the computer field, because of a simple reason: his books have both great content *and* great prose. Not all authors of computer books (and certainly not the one you are reading now) can make the same claim.

OOSC is now considered a historic milestone in the history of object orientation, a major reference on the subject, all while remaining a delightful book to read. In this age where every language incorporates lambdas for the sake of functional programming fashion, this book remains a refreshing experience.

It provides excellent background information in various technical subjects, such as memory management, garbage collection, multiple inheritance, generic programming, type covariance, exception handling, and many other advanced topics. It also includes sections about analysis and style, and even a chapter on "Using inheritance well" and another on object persistence and databases. Needless to say, something many of us would need to read and re-read every so often.

Years later, as the hype of Agile programming methods was getting diluted in the rising reign of DevOps, Monsieur Meyer (still a teacher at ETHZ[6]) published a second gem: Agile! The Good, the Hype and the Ugly[7] (gotta love the exclamation mark, by the way.) This is probably the first serious book ever written about Agile (the second one being the long awaited and recently published Steve McConnell's[8] More Effective Agile[9].)

Ironically enough (or not,) Bertrand Meyer is not a signatory of the Agile Manifesto, resulting in a book free of hype or marketing propaganda. It consists of a short (170 pages) enumeration of "pros and cons" of various methodologies, with discussions about their applicability and suitability for certain projects.

In a time where Scrum seems to have become a *de facto* synonym of Agile, here is a book that actually sheds some light to the subject – and is actually a joy to read (By the way, if you are

[3] https://www.pearson.com/us/higher-education/product/Booch-Object-Oriented-Analysis-and-Design-with-Applications-2nd-Edition/9780805353402.html

[4] https://bertrandmeyer.com/

[5] https://en.wikipedia.org/wiki/Object-Oriented_Software_Construction

[6] https://ethz.ch/en.html

[7] https://www.springer.com/gp/book/9783319051543

[8] https://stevemcconnell.com/

[9] https://moreeffectiveagile.com/

not interested in reading this book, at least watch this ACM Webinar[10] by Bertrand Meyer himself. You will not be disappointed.)

There is humor in the pages of these two books; there is wit, there is irony. But there is also depth, substance, and reflection. Merci beaucoup, monsieur Meyer.

Cover photo by the author.

[10] https://learning.acm.org/techtalks/agile