I read ThoughtWorks Technology Radar vol.20 so you don't have to

Artur Skowroński Follow
Jun 7, 2019 · 23 min read

ThoughtWorks®

FYI: New edition can be found here. Enjoy!

For those who don't know, ThoughtWorks is a Consulting Company that has Martin Fowler and Rebecca Parson in their team. Due to the type of their business, they have the opportunity to work with different clients from different industries with different problems. The variety of use cases allows them to evaluate different techniques, platforms, tools, frameworks, and languages, suggesting what is really worth considering and which are overhyped and problematic in the long run.

Due to that, they are able to create Technology Radar, their quarterly report that is a highly opinionated view on what is happening in digital business. From my perspective, they are the Zeitgeist of our industry — the sweet spot between 'not yet-worth your time' and 'everybody is already bored with looking at the topic'. Unfortunately, while being full of knowledge, it's also a bit overwhelming due to its structure. Blockchains, Clouds and mobile development are mixed together, so it's really hard to retrieve knowledge specific to the category we're interested in.

That's why I decided to do an analysis of the current edition and present to you the conclusion that emerges from it. I also allowed myself to add a bit of my own analysis on it. I hope it will be an interesting lecture, both for those who have already read it all, as well as for those who have never heard about it.

Have a good time.

> *I missed Vol. 19, sorry. That won't happen again. Let's cover what happened in the 20th edition.*

Blockchain and Distributed Ledger

Last time we met was an especially interesting (in a way, like in a Chinese Curse) time for Distributed Ledgers: the Blockchain Market was just after its big flop at the beginning of 2018, and the outcomes were hard to predict. The community needed to decide which direction was more important to it — crypto-communism or distributed-capitalism. However, the outlook was slightly optimistic : the ecosystem was gaining maturity and tools were flourishing with the goal of resolving the biggest problems of the decentralized protocols. How does it look now?

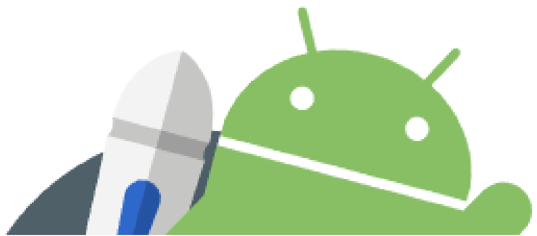
I'd say… a bit depressing. Smart Contracts — the poster child of second-wave blockchains — definitely took the hardest blow. While there was always a lot of doubt about the utility of Smart Contracts, the people from ThoughtWorks currently suggest thinking twice before committing to their usage. They summarised them as an interesting tool, but brittle and hard to control. The biggest problems they found are nested in the concept of "immutable contracts" : due to blocking any changes in business logic, they are far less flexible than real-world, "dumb" contracts. "Holding" is not yet suggested, but dismissing them to the "assess" phase don't express a lot of optimism.

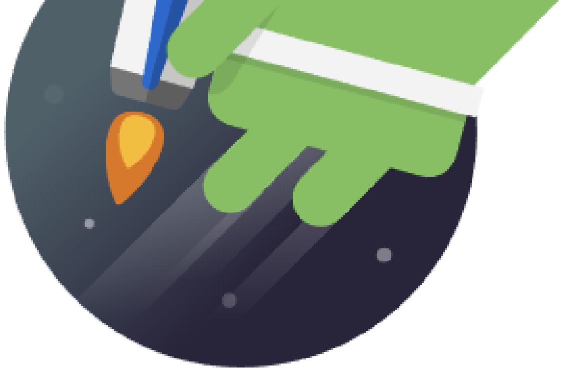
When taking a look at the ecosystem as a whole, the situation seems a little better, but we can also find a dose of skepticism. They are promoting usage of the Ethereum Virtual Machine itself, but simultaneously one of its specific implementations, **Quorum** (an enterprise-targeted fork of "the world-computer", as Ethereum is sometimes called), is shown as including major flaws that were encountered by the ThoughtWork teams that were trying to productize it. I'm really curious to find out if they have some experience with **Burrow**, which is the implementation of EVM created by the **Hyperledger** foundation. Tooling around Hyperledger was an important part of previous editions, but they are strangely absent from this one. Maybe in the next edition we will see more about this project.

Wrapping up, two more tools found their place in the current edition : the Byzantine-Tolerance Consensus implementation known as **Tendermint** (lack of focus on Proof-of-Whatever and using standard computer science by the community seems to be good for the community itself) and **Embark**, another toolchain that tries to provide a better developer experience for writing Smart Contracts — in a way a bit similar to **Truffle**. I'd rather stick to the latter (as it's vocally supported by Microsoft, which is organizing TruffleCon in August), but a little of diversity didn't kill anybody… at least yet.

Mobile

While that tendency could be observed in previous editions, it's hard not to see the homeopathic amount of new utilities if you are Mobile developers.





If we don't count **Vapor**, a Swift-based server-side framework (and we won't because it's only interesting characteristic is its ability to target mobile developers who want to write server applications), in the current Radar, we only get **Room** — part of the already well-recognized Jetpack, a toolset of off-the-shelf solutions for Android Developers. I like its approach of resolving problems with SQLite and after ten years of dealing with them I hope Room will become de facto standard.

Before we start to create dark scenarios for this part of the industry, it's worth stressing that ThoughtWorks' experiences are unique as they are consultants that work for big companies on high-profile projects. It's just a bit disturbing to notice that mobile is probably not seen as high-profile among big companies anymore.

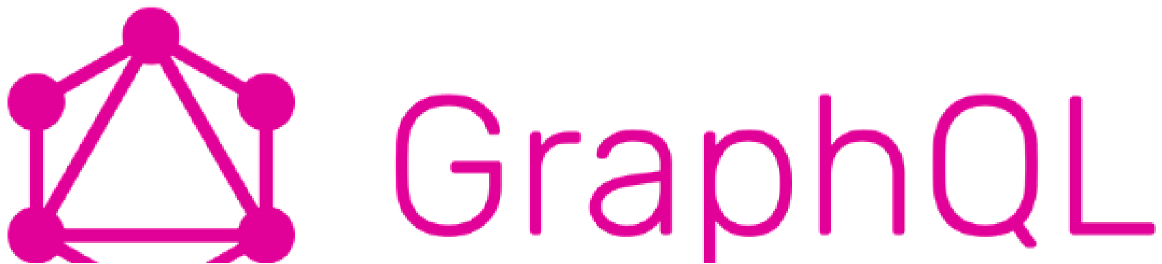
Chatbots

Zilch. Nil. Keine. Null.

Nothing to see there. We will definitely see some sparkling returns to the topic in the future (as everybody is still striving to be "WeChat for the west"), but probably not in the form of the global hysteria of early 2017. I'm not without sin here as — being part of this mob — I still believe in voice assistants. However, they are also nowhere to be found in the current Radar.

UI & Frontend

After two much thinner categories, now we are coming to one that is evergreen as always. New tools, new languages, new runtimes — and that all while excluding testing tools to a different section of this article!



The frontend community is bravely fighting with complexity. Both **Apollo** (GraphQL toolset) and **Micro Frontends** (a technique of splitting code in a manner similar to backend microservices) found their place in the Adopt section, which means they are considered to be great tools when we consider the problems they promise to solve. I'm personally a bit cautious about GraphQL, but the many positive opinions about this approach suggests that it's something worth investigating further, especially as it's not a novelty anymore and we have a lot of post-mortems to learn from.

TypeScript has been with us for a few editions, so we already know that it should be adopted (I have it myself in two commercial projects and — apart from a few hiccups — it has definitely earned its place!). It's also interesting to see **ReasonML** (an OCaml-based language) in Report. It was created byFacebook and has good integration with React and JSX — a necessity for any language to gain popularity. It's such a powerful beast. However, I'm even more interested in **Deno** (anagram for Node), a kind of "Node.js 2.0" designed by the creator of Node, with a focus on resolving its predecessor's problems that have materialized since its debut, like security or the module system. Moreover, it will have out-of-the-box TypeScript support , and it's hard to go wrong with out-of-the-box support for TypeScript. It's a bit unusual for technology at such an early stage to be found on Radar and this says a lot about the size of the expectations and fears (from the more conservative part of the community) that are attached to Deno.

A bit of tooling also found its place in the current edition. While highlighting Next.js seems like proposing an alternative just for the sake of having an alternative, **Immer** (Immutable.js 2.0) and **Formiki** (ReduxForm 2.0) seem to be good suggestions, especially for teams that want to bring a bit of fresh air to their 2–3-year-old frontends. They are a good, incremental step to achieving more clarity as both of those libraries have clear advantages over their predecessors.

We will end this section with something for UI creators : **Pose** and **Lottie** for animation (the latter is also accessible for desktop and Mobile. We have also singular technique for more iterative components development — **UI dev environment** — definitely worth investigating and adopting — especially in bigger teams.

APIs

I decided to call this section “APIs”. Previously, it was “Backend” and I’m still not happy with the new name for this highly opinionated discussion of a group of different technologies. Be prepared, maybe next time it will be relabelled again.

Of all the languages highlighted in the current (as well as previous) edition, we can see the “Kubernetes of Modern Backend Development” in **Kotlin**. Although the language itself didn’t find a place in this edition, four different tools strictly related to it did. We have **Detekt** for static analysis; there is the coroutine-first web framework **Ktor**; we’ve included **Arrow** for enthusiasts of more functional approaches; and finally there is a **http4k**,a modern HTTP client. And that’s all without counting **MockK**, a mocking tool which I think should probably be in the testing section (bounded contexts, even when only grouping technologies, is such a hard topic…). Step by step, a world where it’s ‘hard to go wrong’ using Kotlin has started to materialize — a similar situation to TypeScript. It has definitely dethroned Groovy as the support language for JVM Developers.

While talking about JVM developers, it's hard not to see a large proportion of Spring-based tools. **Project Reactor** and **WebFlux** have emerged as the de-facto standard for writing reactive microservices on JVM (sorry RxJava). Furthermore, for those who consider Spring too heavy but still want to find sharing development style with it, we can find **Micronaut**, a Netty-based, reflection-less web framework which is astonishingly lightweight. Being a modern tool for the JVM world, it has first party support for Kotlin — did I already said that "it's hard to go wrong using Kotlin"?

Nevertheless, when talking about backend languages, it's unfair to not mention **Rust**. According to the last StackOverflow Survey, Rust is a favorite tool among developers that is entering mainstream development in places where C++ is too low-level and complex. For example, the previously mentioned Deno is written in Rust, which is motivated by its great concurrency support.

Changing topic from programming languages , for those who need to introduce a bit of low-level networking and home-backed streaming to their technology stack, Aeron seems to be an interesting Protocol-of-Choice. I suppose the number of people interested will be rather small, but those who have niche needs should take a look. For the rest, Kafka Streams, also presented in this Radar, would probably be a better choice.

Wrapping up, there are some other interesting tools: **joi** is a validator for JS schema (I decided to put it there, not in the UI section as I see many use cases outside of UIs); **Resilience4j**, a lightweight circuit breaker for Java languages inspired by Netflix's Hystrix; and, last but not least, Hot Chocolate (a bit of a strange name) , a .NET-based GraphQL Server, which is especially interesting taking into consideration the fact that GraphQL is in *Adopt* Report's section for UI Developers.

A lot of sweet stuff.

Security

In the security section, continuous security testing is most prominent. We can find both **Infrastructure Configuration** as well as **Container Security scanning**, supported by **inSpec** — automatic security tests from Chef. This shows that in more and more projects, automatic pipelines are starting to be augmented with the goal of finding potential security issues before code is deployed to production, and the ecosystem of off-the-shelf tooling is growing.

Of course, that's not all we can find in the current edition. Among techniques, **Secrets as a service** has entered the "adopt" phase. Engineers from ThoughtWorks strongly promote the usage of tools like **Vault** or **KMS** to store application secrets, protected by externalized Identity solution. We can also easily point there a connection with a different technology from this report, modern identity specification called **SPIFFE**.

Wrapping up, two more items. In the post-GDPR world, **Crypto shredding** — a technique that promotes architecting applications in a way that "allows" their authors to lose access to PII on customer demand (by hashing them with a bit of crypto) — seems like a remarkably safe approach to privacy. Meanwhile, **Cillium** looks to be an admirable tool for network security in the service mesh world.

Testing

As we are done with security testing, time to talk about testing altogether.

Let's start (traditionally) with a bit of Chaos. The **Chaos Engineering** ecosystem grows with a variety of tools, two of them — **Chaos Toolkit** and **Gremlin** — are highlighted in the current edition. It's hard to create a more proper description than the one we can find in Radar itself: "Chaos Engineering move from a much talked-about idea to an accepted, mainstream approach to improving and assuring distributed system resilience". I still see a bit of road ahead of it, but I definitely see a future in which the majority of high profile projects will implement Chaos in at least some respect.

Once again, a lot of integration testing tools can be found in the report. However, apart from the **Taiko** automation library that was created by the people from ThoughtWorks itself, there's nothing surprising to see there. We have **Cypress**, we have **Puppeteer**, we have **TestCafe** — all of which have already appeared in a few editions. This could be a sign that this fluid ecosystem has started to solidify. Going one layer deeper in our testing pyramid, we find **AVA** — a test runner for JavaScript tests with much potential. Meanwhile, if we want to run some tests on our test runner, react-testing-library is highlighted as a good substitute for Enzyme, which lags behind React's blazing fast introduction of new features.

Mocking tools also can be seen in the new Radar. **MockK** is a useful tool for creating mocks in Kotlin, which (as we already mentioned) has recently gained more and more popularity. However, while mocking libraries are useful, a tool with even greater potential to make developers' lives easier is **LocalStack**. The capability to mock third-party services like S3 and Lambda is invaluable when we want to achieve stability for integration test suites while making our stub environment as production-like as possible.

For the people from Big-Data stack, there is **HiveRunner** — a first-of-its-kind tool that never appeared on r my own personal "radar". HiveRunner allows you to unit-test Hive queries, which seems like an interesting concept as unit testing any DB is a pain. And while talking about pain, for those who struggle with API testing, **Karate**, with its language-agnostic declarative approach to test definitions, seems like an appealing alternative to existing tools. Meanwhile, if we want to make our already correct API perform well, Taurus looks like an interesting high-level, declarative wrapper for many different performance tools (like Gatling), hidden together under a single interface.

Monitoring

After an enormous section about testing, the monitoring one is rather humble.

AnyStatus provides an at-a-glance view into the status of your production environment from the simple application ; while I don't like tools like this and prefer external quality stations, it looks slick. However, it's good to point that it's currently only accessible for Windows, which potentially limit its popularity ; however, Microsoft's recent moves to win developers' hearts are unprecedented, so maybe it's the best time to introduceWindows-centric development tooling.

honeycomb.io

Another tool, **Honeycomb**, seems like a solution similar to what OverOps or New Relic offer. Their Product Demo looks great though, so I suggest at least taking a look.

Humio is a tool I'll definitely try which handles log management. Knowing how AWS Cloudwatch is subpar in comparison to ex. Splunk (which itself is unfortunately expensive) and how much maintenance effort an ELK stack needs, Humio seems like a tool that potentially can find a niche, as long as it is a viable competitor for Splunk in the SASS Space.

However, if you are in a "build-it-yourself" mindset, the last tool ThoughtWorks wants to present is **OpenAPM** — a tool for choosing and stitching together various Open-Source Monitoring technologies to ideally fit your specific needs. This is probably best suited to bigger teams with requirements that cannot be served by off-the shelf solutions.

Containers

This time the Containers world is a bit underrepresented (especially in comparison to vol. 18).

Containers Security Scanning, previously mentioned in the Security section, is a response to the enormous increase in the usage of Containers in production in mission-critical systems, which makes security a more crucial feature of container-based infrastructures.

Nomad

A variety of different deployment environments and orchestration tooling pops up in this edition. The most interesting from my perspective, **Cage**, seems a great alternative for the direction in which the Docker ecosystem has evolved. It used the Docker Compose format in its second version, which had some interesting features that were removed in the third one , like the ability to provide “health checks”, which was really helpful in some use cases, but was removed due to it's incompatibility with Swarm. This was not a big waste as Swarm is probably *#OfficiallyDead* by now. Apart from this, we have **AWS Fargate** (one from a variety of Amazon’s containers orchestration services) and **Nomad**, HashiCorp’s response to the popularity of Kubernetes. While Kubernetes seems to be the current gold standard for orchestration, HashiCorp solutions are nearly always at least interesting (spoiler: we see this in the current edition in the Cloud & Infrastructure Section).

Probably nobody will be surprised to see Kubernetes tools in this edition. However, wit the exception of **KNative** (which better suits the Serverless Section), there are only two of them. First, **Helm** is already a well-known package manager for Kubernetes; it has polarized users for some time, and has as many fans as detractors (check our previous article). Still, it has reached the Trial phase in new the Radar. The second tool is **Kubernetes Operator** from the people behind CoreOs. Kubernetes Operator is an opinionated Lifecycle manager for Kubernetes applications that attempts to streamline testing and the deployment process.

Data Analytics

The motto of this edition — "**Use Jupyter Network, but don't try to productise it**" — seems like a nice shortcut, but the limitations of the notebook format will soon annoy you and you will need to hack around it. However, it's still a wonderful tool for ad hoc data analysis (I recently started to use it myself and can confirm — while not being a fan of Python — that Jupyter is really awesome and has great support from tools like Visual Studio Code). What's more, for those who already like Jupyter, **Dask**, which can be described as a parallel scheduler for the Python data science community, allows you to use all of your machine cores while working with libraries like pandas or Numpy — also inside your notebooks.

In Radar vol.19 there are two additional tools for Data Analysts. The first is **Prophet**, a forecasting platform created by Facebook which can be programmed in both Python and R. Additionally, **Apache Beam**, a high-level data programming model, has graduated from Assess to Trial. Its biggest asset is unified abstraction over runners like Apache Samza, Flink or Spark, as well as Google Cloud Dataflow. Beam's standardization approach is currently gaining popularity and its future seems to be bright.

Machine Learning

Due to a popular trend in community, I decided to split Data Analytics and Machine Learning tooling to two different sections. In this edition, it means I was able to put in Machine Learning one three different items — one tool, two techniques.

If we treat success in software development as "Google, Amazon and Microsoft are using your library in their services", **fastAI** has hit the jackpot. While I'm not data scientist myself, I've already seen a lot of buzz about fastAI due to targeting those who are beginners in neural networks (it seems strange to talk about "beginners" in this context). Bonus points for its great synergy with Jupyter Notebook.

Machine Learning has started to use more sophisticated techniques from "classic" development. In this issue **Continuous delivery for machine learning (CD4ML) models** can be found around techniques worth to try. Generating reproducible artifacts for every change of model, as well as automatic deployment of them to production systems looks like streamlining of the whole process , especially from the perspective of a developer with a previously mentioned "classic" background. ThoughtWorks has always focused on sound process, and I'm happy to see the introduction of this philosophy to the world of Machine Learning. While talking about models, if you are using ElasticSearch, you should look at the **LTR** plugin, which helps to use ML for better "ranking" of results.

Last but not least, we have an Academic Paper that **covers the transfer of layers between NLP models**. I haven't read it yet (but it's on my to-do list), but 31 citations in the first year since publishing looks like a rather promising start, even while chatbots — the most consumer-facing example of NLP usage — seems to be a bit sluggish in this edition.

Cloud and Infrastructure

Let's start with Terraform, which in this Radar is promoted as a go-to method for infrastructure creation. While Terraform itself is rather well known, it's mentioned in a variety of other items, and two additional tools that appear in this edition. **Terratest** is a tool that could probably also find itself in both the CI/CD and Testing sections, but I decided to put it there for article cohesion. Terratest allows you to test your Terraformed infrastructure in an automatic way : create infrastructure, deploy an application, and then tear machines down. Stale scripts for infrastructure creation are a problem that has followed me throughout my engineering career, so I'll definitely try it in the next project using Terraform. To show how important Terraform is for ThoughtWorks, they also promote using their syntax to create pipelines in their own Continuous Deployment tool , GoCD.

When talking about Terraform, there is a trend that is especially put "on-hold" by ThoughtWork specialists , who recommend changing the way in which you are writing your infrastructure script : get rid-of YAML and JSON for infrastructure templating. They write that due to the restriction of that format, a lot of hacks are necessary to do things like conditionals or loops. They promote specialized formats like Jsonnet, or just using general-usage languages like Python — anything that does not hide templating "logic" behind some opinionated protocol. Additional hint : if you're thinking about writing your own Cloud Formation tools , **stop** and use an existing tool like the previously mentioned Terraform.

Istio

Orchestration is another important part of this category. Service Mesh is gaining popularity, probably thanks to the fact that the ecosystem around it is maturing, which makes this approach more accessible for "normal" developers without a big infrastructure team behind them. **Istio** has become a standard tool for those who want to overcome service mesh chaos, bringing the whole package of service-discovery, security, and observability. It is one of the fastest growing projects in Github and can be considered one level above classic Envoy proxies (which it is compatible with) , but with a lot of additional capabilities. However, for those who need only a bit of networking and routing, not a fully-fledged service mesh, **Traeffik** seems like a really comprehensive tool (we are using it in one of our projects and it's definitely serving our needs).

MINIO

Wrapping up, while in the previous edition there was a bit of cautious against generic cloud usage, this time we have two tools that allow exactly that. **CloudEvents** is a standard that allows intercommunication between different clouds (if you are interested in this, I suggest this great podcast), and **MinIO** allows abstraction over data storage from different providers. While they are a bit inconsistent, it's good the world is not black and white for the Radar's creators and present tool sets that can be useful in different use cases.

Database

This is a tricky section. From one perspective, it should be merged with Data Analytics (which itself should be merged with Machine Learning). From the other, in IT nearly everything is connected in some way, so I decided to put more “low-level” tooling here.

In the current edition, we have two time series databases (which are related more to monitoring than Data Analytics… my hurts now). InfluxDB is currently the most popular solution in this category — battle tested, part of the popular TICK stack that was highlighted in the previous edition. However, if you are already using PostgreSQL, you can think about using **TimescaleDB** : administration can be easier, and the querying language is more natural. Unfortunately, if you use managed Postgres from one of the clouds, you might have a bit of a problem as only Azure and Digital Ocean support TimescaleDB out of the box. However, if you decide you want TimescaleDB so much that you want to create your own Kubernetes Cluster with it, the current report has **Patroni** for people like you : it is a template which contains high-quality configuration for HA Postgres, created by Zalando.

We have also one technique that developers should “hold” using. To protect data encapsulation, ThoughtWorks engineers suggest stopping using **Change data capture** — automatically synchronizing data between different datastores using database triggers. While sometimes helpful, it can potentially make your integration totally miserable by killing the observability of how changes are introduced to your data.

Serverless

As always, exciting stuff can be found in this section.

With every Serverless implementation, there always comes a decision : who will be responsible for this “server” part that is invisible for cloud users. While talking about containers, I already mentioned **Knative**, and now is the time for a bit more emphasis on it. While Patroni is a build-your-own PSQL cluster, Knative can be described as a build-your-own serverless cluster. Created by Google, it is one of the building blocks of their infrastructure which you can install on your own Kubernetes Cluster. If you are not sure if you want to be “as” self-serviced, there is a bit of a unique solution from Cloudflare. On their **Workers**, you are able to run JavaScript code with the promise of big performance gains: as Cloudflare is a CDN provider, they are really near your end users , which should assure really tiny latency.

After this slightly unusual approach to serverless, Laconia looks like a somewhat standard framework for applications deployed on AWS Lambda. Its biggest assets are the nice dependency injection and the hiding of AWS API requests and responses under a fairly elegant abstraction. Nothing really crucial, but in a world where more and more applications are deployed on serverless platforms, it can be handy to protect you from edge cases that you never even thought would affect you.

CI/CD

As continuous delivery for machine learning (CD4ML) models were described in the Machine Learning section, let’s cover two other things that can be put under the CI/CD umbrella. **Flagr** is… feature flag as a service.

I have to say that I’m intrigued. It’s a tool that can be compared with the rule engines of the distant past (hello Drolls, you are still haunting me in my dreams!), giving businesses easy access to changing feature flags on the fly. What’s more, it also supports more complex use cases like canary releases and A/B Tests. I will definitely test it, as while I love feature flags, the question “how do we implement it?” never has an easy answer. Maybe Flagr will deliver what it promises, without becoming Drolls 2.0.

I need those feature flags because I hate the **Release Train** process. I know that feature flags are only a solution for a tiny subset of problems related to it, but I want to put every code in production as soon it’s passing tests, without coordination with business people (“turn on the feature whenever you want, I’m deploying it now”), and without waiting for a specific time slot to be able to deploy. Release Train can be useful while a company transitions to Continuous Delivery, but needs to be stopped as soon as enough maturity is achieved. I’m happy that the current edition of Radar highlights that we should put Release Train on hold as a technique because it promotes bad engineering values.

IOT

Nothing. There was no single item that can be put in this section (apart from TimeseriesDB, which is often related to the Internet of Things). Maybe in the next edition we will find something interesting.

Organizational Engineering

> *“There are only two hard things in Computer Science: cache invalidation and naming things”*

“There are only two hard things in Computer Science: cache invalidation and naming things”.

I think this is the name “Organizational Engineering” greatly describes that category of loose coupled practices and tools connected to process how we create software.

Many of us work in organizations that need a bit of evolution. If we are in a position that can introduce process changes, it is worth reading about **Wardley Mappings** : if you want to change or correct something, you need to first understand it and this technique was created to help you visualize how your business works, especially on the value chain and customer level. While talking about understanding how the system works, if you love diagrams, **Systems** (I assume it’s difficult to find name harder to google for) seems like an interesting tool for creating diagrams and exporting them to Jupyter Notebook.

When you’ve educated yourself on what’s important to your business, you can continue to the team level. I fell in love with **four key metrics** from the Accelerate book that ThoughtWorks is promoting (with a bit of GoCD product placement) as a health check of the engineering team — lead time (how long it takes to deliver a feature to production from the moment development started), deployment frequency, change fail frequency and mean time to restore. Together, those metrics show the condition of your process and are control metrics for each other. Recently, we started experimenting with all of them in one of my projects, and I’m really curious to see how the experiment will work out.

After a few scandals in 2018, in 2019 technology companies have started to be the media's "villain of the story". This is a topic for deeper analysis than I can do here, but important commenters in the industry have started to realize we have a problem as an industry. That's why it's exciting to see **EthicalOS** in the current Radar. EthicalOS is a remarkably interesting group of potential risks (from an ethical point of view) you should take a look at even before you start developing a new product. It provides a reasonable and easy-to-use checklist (Checklist manifesto, anybody?) that will guide you through the potential harmful impact your application can have on society. As I said, the topic is much broader, but I think EthicalOS is at least worth a look.

We have also some more "ground level" advice. While there has been a bit of a backlash in recent years, **Polyglot Programming** still finds a highlight in the current Report. It's a powerful technique, especially if have a really good case for it. However, while Polyglot Programming is fairly controversial, it is really hard to show the drawbacks of using Opinionated and automated code formatting. It brings clarity for the codebase, streamlines the review process, and helps new developers join a project thanks to specific guidelines. But you know what is even better than guidelines on your first day? A fully working development environment on your first day. **batec** is a tool created by ThoughtWorks itself that can help orchestrate this.

Other

Wrapping up, there are two items I didn't find a place for.

Contentful was already described in my previous article ; if you're looking for a CMS, it looks like an attractive solution.

Anka looks like a bit of a niche tool for those who need to virtualize the MacOS system. If you are among those people (for example if you need to create a CI environment for iOS builds), Anka looks like something worth a look.

Thanks to Szymon Gąsienica-Kotelnicki and Hubert Słojewski.

Kotlin Kubernetes Typescript Cloud Tech