JDriven Tech Radar Spring 2023

Our view on recent developments in our field

7th edition

Commit to know.

Develop to grow.

Share to show.

jdriven

Introduction

Our experienced specialists participate every day in the development of many different software projects all over the Netherlands, and are involved in worldwide professional communities. Each half year our engineers gather to discuss the latest emerging trends and developments in software engineering. We seek to capture these trends in a technology radar. Each edition of the radar shows the changes in these trends, compared to the previous edition. A shift can indicate that we see a technology is becoming more, or less, relevant for certain use cases. If a trend does not show up in later editions, it signifies that there are no relevant developments or experiences that cause us to shift our assessment. With this document we will discuss the shifts we have observed over the past half year. This analysis guides us in deciding which technologies we use and recommend.

Tech Radar

The idea to create a tech radar was initiated by ThoughtWorks. They periodically showcase their view on new trends and development. In addition, they advise everyone to define their own radar [https://www.thoughtworks.com/radar/byor].

At JDriven we fully support that view. Building a tech radar is an instructive and valuable experience, where we mutually share our knowledge and create a common awareness around technology. We believe that specialists should be able to create and maintain their own toolset to perform their job to their best ability. When composing a radar, you facilitate a broad discussion on technology, so organisations can strike the right balance between the risks and rewards of innovation. We can help you to kick-start these discussions in your organisation. Let your teams innovate and inspire each other. Together they can draw up a set of technologies and techniques that can accelerate your business.

Overview

The radar consists of quadrants and rings, containing blips to indicate technologies of interest.

The quadrants subdivide the different subjects into categories:

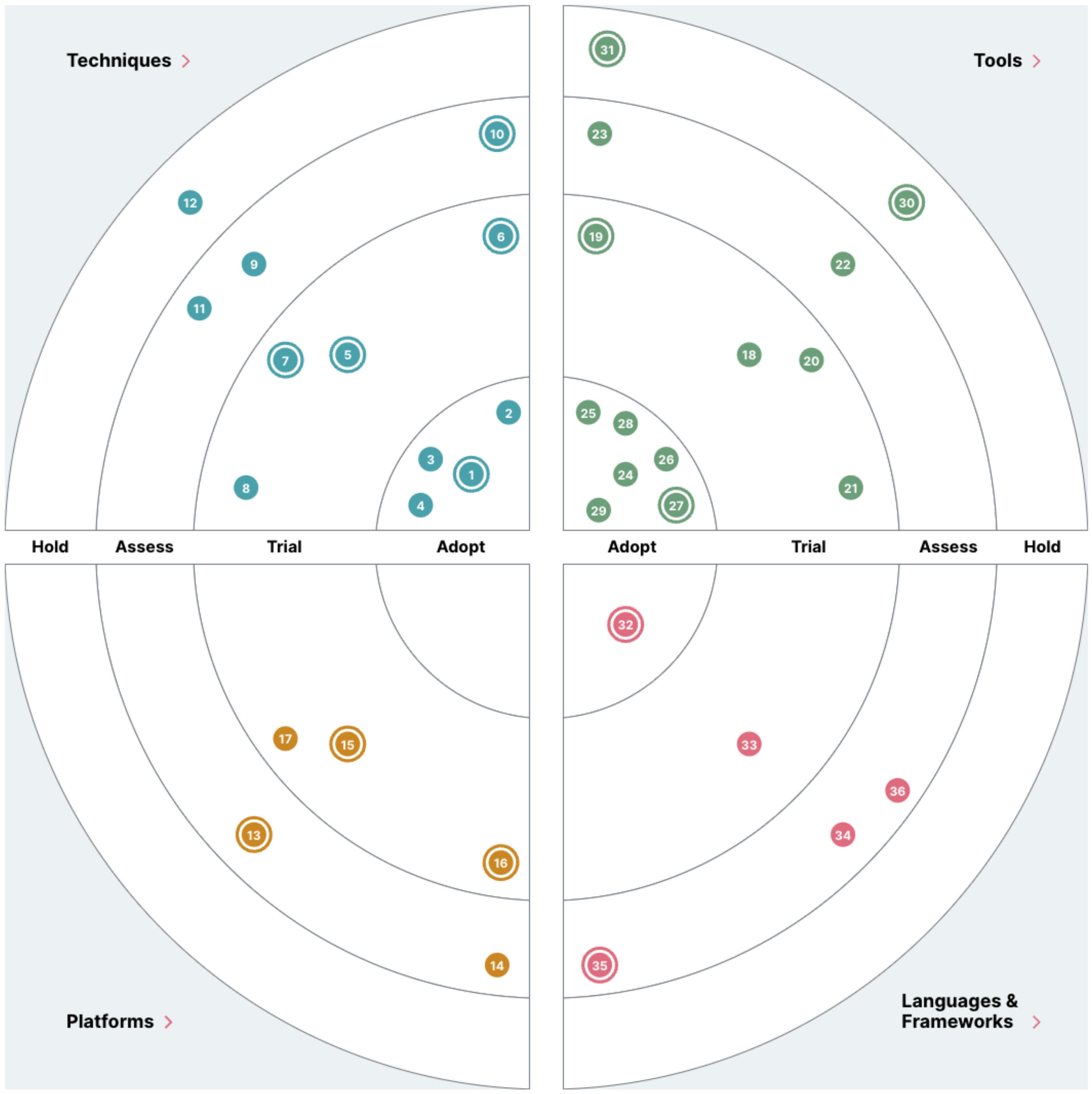
- **Languages and frameworks** that support developers in their daily tasks.
- **Platforms** to deploy and execute programs.
- **Techniques** help developers create better software.
- **Tools** aid in the development and delivery process.

The rings in each quadrant indicate which phase of the adoption cycle we believe a technology is currently at:

- **Adopt**: We recommend to use this technology, wherever it fits the requirements.
- **Trial**: We advise to gain experience with this technology, wherever a project allows for a certain degree of risk.
- **Assess**: Interesting topic to learn more about and assess its future impact; yet too early to use in production.
- **Hold**: Do not deploy in a project not already using this technology.

In the subsequent sections we will delve into our views on recent developments in the field of software engineering more closely.

Commit. Develop. Share.



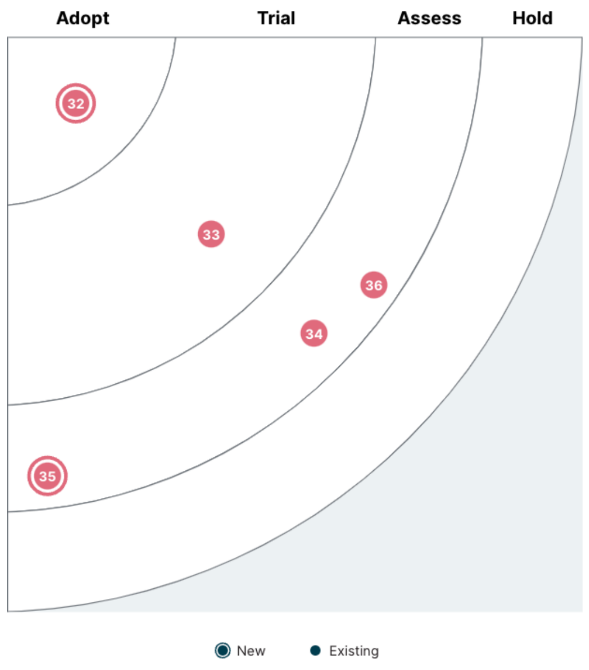
◉ New ● Existing

Commit. Develop. Share.

Tools	Languages & frameworks
ADOPT CNCF Roadmap Datafaker Gradle Kotlin DSL k9s Maven Daemon OpenTelemetry **TRIAL** Error Prone Support Sigrid Spring Boot Migrator Testkube **ASSESS** Diffblue Cover SigNoz **HOLD** Docker 4 Desktop Lombok	**ADOPT** Azure Bicep **TRIAL** Spring Modulith **ASSESS** F# Service Weaver Virtual threads **HOLD** -
Platforms	**Techniques**
ADOPT - **TRIAL** Apache Pulsar Edge Computing GraalVM **ASSESS** Dapr On-premise **HOLD** -	**ADOPT** DDD SBOM Tinkering Weighing developer experience **TRIAL** AI code assist Developer Productivity Engineering Team Topologies WebAuthn **ASSESS** Data Oriented Programming GPU Programming Risk storming **HOLD** SAFe

Commit. Develop. Share.

Languages & frameworks



Adopt
32. Azure Bicep

Trial
33. Spring Modulith

Assess
34. F#
35. Service Weaver
36. Virtual threads

Azure Bicep

ADOPT

Azure Bicep [https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview?tabs=bicep] is a Domain Specific Language (DSL) that enables developers to write infrastructure as code for Azure resources. This technology simplifies the process of creating and deploying Azure resources by allowing developers to write declarative code that defines the desired state of their infrastructure. Azure Bicep is an open-source project developed by Microsoft, and it is built on top of the proven Azure Resource Manager (ARM) infrastructure.

Azure Bicep is used by developers to manage Azure resources such as virtual machines, storage accounts, and network interfaces. By using Bicep, developers can create reusable code modules that can be used across different environments, reducing the time and effort needed for infrastructure deployment. Azure Bicep also enables infrastructure automation, making it easier to manage and maintain Azure resources at scale.

We believe that if you work with Infrastructure as Code, Azure Bicep is the defacto choice when deploying to Azure. Its adoption allows organizations to improve their infrastructure deployment speed, increase reliability, and reduce the risk of human error. Additionally, its integration with Azure Resource Manager ensures that Azure Bicep is fully compatible with existing ARM templates, making it easy to adopt for organizations already using Azure. Finally, the Azure Bicep Visual Studio extension provides developers with a streamlined experience for writing infrastructure as code for Azure resources. With this extension, developers can benefit from IntelliSense and syntax highlighting, making it easier to write, maintain and test Bicep code. As such, we recommend that companies consider adopting Azure Bicep to streamline their Azure infrastructure deployment process and improve their overall infrastructure management.

Commit. Develop. Share.

Spring Modulith

TRIAL

Domain driven design has helped developers understand that a software architecture can and must facilitate the evolution of software systems to follow changes in business requirements and insights. Using microservices is a way of isolating functional modules to that end, but it introduces challenges with regard to service orchestration and HTTP based communication. This has made developers reconsider whether this modularity can be achieved and enforced in monolithic applications.

Spring has been a framework that provides technical stereotypes such as @Controller, @Repository, etc, without regard for how components depend on each other. With Spring Modulith [https://spring.io/projects/spring-modulith], it aims to be more opinionated on the domain-aligned structure of a monolithic Spring Boot application. In practice, it seems to mean that a properly set up, well structured Spring Boot application using Spring Modulith will limit candidates for dependency injection based on package structure convention. It offers ways to deviate from the defaults by customizing module detection.

It builds on ArchUnit in that it tests and restricts dependencies, but it's unclear whether it goes above and beyond in ensuring that the bootstrapping prevents modules depending on each other in undesirable ways at runtime, or that it simply offers annotations that help test and - another great feature - generate C4 architecture documentation in AsciiDoc. Spring Modulith also gives a new way of integration testing modules independently, which will be a lot faster than the old @SpringBootTest because it only instantiates the classes of the module under test.

Furthermore, they recommend using an event pubsub system to further loose coupling. But that shouldn't be the only recommended form of coupling, and again: it's unclear whether this is meant as a help for testing the integration of modules, or to be used at runtime in the actual application.

Taking all this into account, it's worth testing to what degree Spring Modulith can help with modularity in monolithic applications.

F#

ASSESS

Java is getting more and more capabilities for functional programming (FP). In recent years lambdas, records and sealed classes have been added to the language. Pattern matching, a technique for recognizing a specific pattern in data, is currently under development. By also looking at fully functional languages, it becomes easier to understand the FP concepts behind them.

A language ideally suited for this is F# [https://fsharp.org/], a language that runs on the .NET runtime. F# is a strongly typed programming language, where both functional, imperative and object-oriented programming methods can be easily combined. Because the syntax is very straightforward, one can quickly get started with the features offered by the language. FP concepts such as pure (higher order) functions, immutability, currying, pattern matching, recursion, algebraic data types, tail-call optimization, lazy evaluation, tuples and types are basic elements within the language. Since all these features are included in the language itself, you can more quickly understand the underlying concepts.

The applicability of non-JVM languages may be small within the JVM community. However, it will become easier to start using new features in Java such as full pattern matching (JEP 433 [https://openjdk.org/jeps/433]), efficient tail-calls (Project Loom [http://cr.openjdk.java.net/~rpressler/loom/Loom-Proposal.html]), lazy static fields (JEP draft [https://openjdk.org/jeps/8209964]) and Value Objects (JEP draft [https://openjdk.org/jeps/8277163]). Hence, we recommend F# as a language to get to know functional programming.

Commit. Develop. Share.

Service Weaver

ASSESS

Service Weaver [https://opensource.googleblog.com/2023/03/introducing-service-weaver-framework-for-writing-distributed-applications.html] is an framework designed for building distributed applications. It aims to simplify the development process and reduce the complexity involved in designing, implementing, and maintaining distributed systems.

Service Weaver includes a range of abstractions and patterns that enable the definition, communication, and collaboration of services. By adopting a decentralized approach, services can operate autonomously and communicate via asynchronous messages.

Service Weaver is a new framework that offers great potential in development of distributed applications. Therefore Service Weaver is placed in the *Assess* quadrant of this edition of our Tech Radar.

Virtual threads

ASSESS

Threads are valuable, but expensive resources. This principle is heard more and more these days. For example, reactive systems promote a non-blocking way of working, and Javascript is able to perform multiple tasks at the same time, on a single thread. In these cases, the 'threads' executing these tasks are not mapped directly to OS threads under the hood. That is why they are referred to as Virtual Threads, or Fibers (finer-grained threads).

This concept of using threads in a more efficient way by making them execute multiple tasks 'simultaneously' is not new. Early versions of both macOS and Windows were already using so-called cooperative (non-preemptive) multitasking, where multiple processes/tasks could run on a limited number of threads by giving each other room to execute periodically.

In recent years, a number of reactive libraries emerged (like RxJs, RxJava, Project Reactor), that reinvigorated this principle. In addition, some of the big framework developers also embraced these libraries, adding 'reactive' capabilities to their own frameworks (Reactive Spring, Quarkus, etc).

With the Coroutines library, Kotlin also developed an almost native way of executing multiple concurrent tasks on a limited number of threads. Almost native, because this library, like the other reactive Java libraries, is essentially still running on a 'traditional' JVM working with normal OS threads.

With Project Loom (JEP 425, available in Java 19 as a Preview feature), the concept of Virtual Threads is implemented natively in the JVM. It will be interesting to see how the developers of the previously mentioned languages and frameworks will react to this. We expect this to only have a positive impact. Native support for Virtual Threads could especially provide a performance boost, as support for Virtual Threads no longer needs to be built fully on top of the JVM, but is also offered from within the JVM itself, at a lower abstraction level. This might be a reason to start investigating what it could bring to your application.

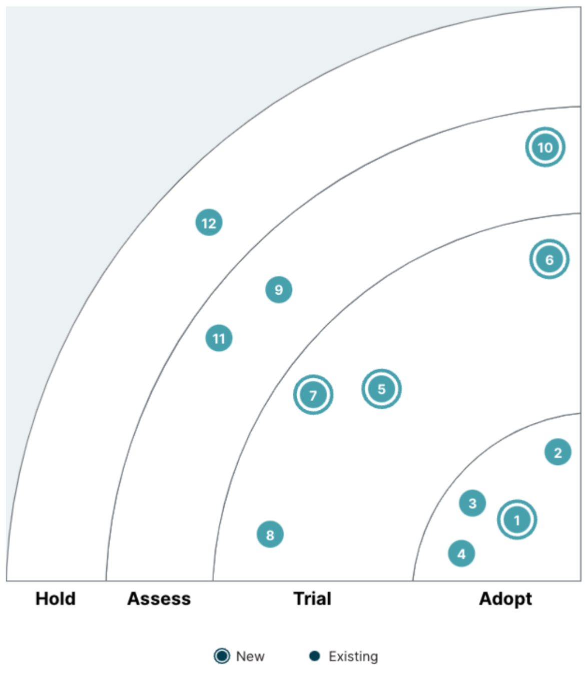
Techniques

Adopt
1. DDD
2. SBOM
3. Tinkering
4. Weighing developer experience

Trial
5. AI code assist
6. Developer Productivity Engineering
7. Team Topologies
8. WebAuth

Assess
9. Data Oriented Programming
10. GPU Programming
11. Risk storming

Hold
12. SAFe



DDD

ADOPT

Domain Driven Design (DDD) is a technique that can be split into strategic and tactical design. For the prior, it offers ways to approach a problem domain and identify bounded contexts, their dependencies and ubiquitous language. For the latter, it offers ways and building blocks to model and implement those bounded contexts.

Although DDD as a concept is now two decades old, we see it has recently been gaining new mainstream traction. Tools for CQRS, Event Sourcing and message driven architectures exist and continue to grow in this space, and techniques like event storming and domain storytelling are finding more adoption. But more importantly, architects and developers are experiencing that going from monoliths to microservices has in some cases aggravated problems, particularly when bounded contexts and their dependencies were insufficiently modelled and understood. They are now reaching for DDD for help.

SBOM

ADOPT

A Software Bill of Materials (SBOM) is a definition of software dependencies documented in one central place. An SBOM specifies all third-party dependencies with their exact version numbers. It helps teams to quickly provide insight into all dependencies. In addition, it gives the ability to quickly update versions of these dependencies.

In the event of major vulnerabilities, it is important to act immediately and adequately. It is necessary quickly identify whether your own software is vulnerable. An SBOM makes that possible.

Thoughtworks has the SBOM on its technology radar under SBOM [https://www.thoughtworks.com/radar/techniques?blipid=202110076].

Commit. Develop. Share.

Tinkering

ADOPT

Tinkering is experimenting with ideas/frameworks in an integrated environment to fully understand their capabilities. Learning through tinkering is based on a talk by Tom Cools [https://tomcools.be/talks/]. Within the IT sector we are required to keep learning as some technologies become obsolete and new technologies are just around the corner. Tinkering allows you to do structured learning, to increase knowledge and sharpen skills. The most important aspect is finding the right thing to learnby using thee zones of proximal development: What you know, what you can learn, and what you cannot learn. Sharing what we have learned is a good practice. We need to have good understanding of the subject, when we need to explain it to other people. It forces us to already think about questions people might have, and have the answers for those questions.

Tinkering is very useful to be focused about what we want to learn, so we gain more knowledge and update our skills and that is why we place it in the *Adopt* quadrant.

Weighing developer experience

ADOPT

Let's first start with a definition of what is meant by developer experience. Developer experience, or DX for short, describes the overall feelings and perceptions a developer has while interacting with a language, tool or technique. The easier it is for a developer to work with the language, tool or technique the higher their sense of DX is. A Software engineer focuses on three aspects of DX: usability, discoverability, and credibility. Usability is how easy your language, tool or technique can be used. Discoverability is how quickly and easily users can find the functionalities they are looking for. Last but not least, credibility, is that your users trust your language, tool or technique to solve their problems.

Why is DX important?

When your language, tool or technique has a high DX the users are more likely to use it in their next projects or new products. For developers of a language, tool or technique it is of the utmost importance to focus their effort on optimizing the DX.

A good DX is all about adopting a developer-first approach and putting yourself in the shoes of your user.

Therefore, Weighing developer experience is placed in the *Adopt* quadrant of this edition of our Tech Radar.

AI code assist

TRIAL

An AI-code assistant is a tool that uses machine learning algorithms and statistical models to predict code. It helps developers write code by making suggestions for completing code fragments, correcting errors and giving recommendations for optimization. It can also use prompt-based engineering, where code is generated using natural language.

As a developer, you can use an AI-code assistant by integrating it as a plugin into your IDE (Integrated Development Environment). This provides you with immediate suggestions and corrections as you type code, making you more productive and efficient. Moreover, an AI-code assistant can help you learn new languages and frameworks faster by offering suggestions based on best practices and commonly used patterns.

Currently, AI-code assistants are still in their early stages. For example, AI code assistants have little knowledge of specific domains, potentially generate faulty code, and ownership can be an issue as well. We believe that, despite all the valid concerns, AI code assistants will become mainstream within software development in about five years. Tools such as Tabnine [https://www.tabnine.com], GitHub Copilot [https://github.com/features/copilot/], Machinet [https://www.machinet.net] and IntelliCode [https://visualstudio.microsoft.com/services/intellicode/] are competing for attention. Therefore, the question is not whether you will use this as a developer, but when. We recommend examining such tooling now, so when AI-code assistants become mature, it can be used immediately for your production code.

Developer Productivity Engineering

TRIAL

Developer Productivity Engineering (DPE) [https://gradle.com/developer-productivity-engineering/] is a branch of software engineering that focuses on improving the efficiency and productivity of developers based on measurable data. DPE is concerned with designing and building tools, processes, and infrastructures that can streamline developers' workflow and accelerate their development cycle. DPE encompasses a wide range of activities, such as designing and implementing automated testing systems and optimizing build and deploy processes to shorten the feedback loop. By optimizing based on measurable data, the DPE team can demonstrate how much time is saved by their activities.

Gradle is currently introducing DPE in the market and we believe this is an interesting trend to follow, especially for larger companies. Because it is based on measurable data and it can be made to fit any company we believe it is suitable to be on *Adopt*.

Team Topologies

TRIAL

Team Topologies [https://teamtopologies.com/key-concepts] is a practical model for organizational design and team interaction based on four fundamental team types and three team interaction patterns. It is a model which treats teams as the fundamental means of delivery, allowing team structures and communication paths to grow with technological and organizational maturity. Team Topologies presents a well-defined way for teams to interact and collaborate with each other to make the resulting software architecture clearer and more sustainable, turning problems between teams into valuable signals for a self-directed organization.

Team Topologies emphasizes optimizing team interactions and complements the strategic patterns from Domain Driven Design to address interactions between software systems. Combined with awareness of Conway's Law and team cognitive load, it helps produce system architectures capable of being sustained by an organization.

WebAuthn

TRIAL

The Web Authentication API (also known as WebAuthn [https://webauthn.guide]) is a specification written by the W3C and FIDO, with the participation of Google, Mozilla, Microsoft, Yubico, and others. The API allows servers to register and authenticate users using public key cryptography instead of a password. WebAuthn uses public key cryptography where the public key is stored on the server that you want to authenticate to. The private key is stored securely on our devices, like a laptop or phone. To decrypt the private key we must use biometrics like a fingerprint or face image supported by our device. This means authentication depends on something we are (fingerprint or face image) and something we have (phone or laptop). All major browsers already have support for WebAuthn, which makes it already usable as authentication option to give users.

The downside is that every website has a different implementation on how to register the public key.

This breaks the user experience for users as there is no standard way. It would be great if in the future this is more streamlined.

Nothing is preventing us from support WebAuthn for authentication for our web applications. We can add it as a two-factor authentication option in the current authentication proces or directly offer a passwordless option for the users and that is why it is part of the *Trial* quadrant.

Data Oriented Programming

ASSESS

Data oriented programming is a paradigm that focuses on data and data transformations. The data oriented programming paradigm is not a new development; it has been practiced for years in languages like C and Clojure. An important note is that it is not an alternative to Object-Oriented programming, but rather an addition since they work quite well together.

Within data oriented programming the key concept is data and data transformations, but what does this actually mean? If you look at applications in a very abstract way, most of them are actually just a data stream with data operations. Data oriented languages embrace this by using generic data structures like Maps instead of types, because they come with a big standard library of functions to transform Maps into other Maps.

Java is getting more features to support the Data Oriented Programming [https://www.infoq.com/articles/data-oriented-programming-java/] paradigm. Types will still be used here, but constructs are being added to make space to also program in a data oriented way. For this reason, records, sealed classes, and pattern matching are becoming available in Java. These constructions make it possible to separate data from functionality, which is key in data oriented programming.

GPU Programming

ASSESS

General purpose programming on graphics processing units (GPU) is a technology to run generic programs on a GPU. This technology is often used for applications that require a lot of processing power, like machine learning, data-analysis or complex scientific and financial simulations. Initially, GPU's were designed to process and render computer graphics. They offer large speedups for any kind of workload, however, that allows a large degree of parallel processing.

We have selected GPU programming for this edition of the Tech radar, as it is gaining traction within various business sectors. Its immense processing capabilities brings certain tasks and workloads, which previously took too long time to process, within reach. The specific architecture of these processors, however, makes it much more complicated to develop software. Moreover, commonly used GPGPU programming languages CUDA and OpenCL are inspired by the C programming language and are thus relatively low-level. To write effective code, a developer needs to have a proper understanding of the intricacies of the GPGPU platform. Hence, we assess that it is still too early for general adoption.

Although there are still some barriers, we believe that, even for Java programmers, GPU programming will only gain traction. A good example is the rapid developments in Large Language Models (LLM's), such as ChatGPT. These kind of applications strongly depend on the GPU processing power to provide users with aptly responses.

Furthermore, GPU programming is a powerful technology that can change the way we process and analyse data.

With the ongoing evolution in GPU hardware design and software tooling, we expect this technology to become more mainstream.

Commit. Develop. Share.

For now, we advise businesses to survey if there are tasks and services that can profit from the massive processing power of the GPU, possibly opening up business opportunities.

Risk storming

ASSESS

While implementing solutions to business requirements, it can happen that developers structurally avoid subjects, architectural matters, or pieces of code. It can be because of technical debt, but often it's a matter of uncertainty; one might call it fear driven development. It's important to recognize such situations and to proceed with a plan to address them.

A word for this activity that is getting traction is Risk Storming. It involves analyzing software systems at various levels of detail - from a single Java class to the entire landscape of interacting services - to assemble an overview of 'hot spots' that developers experience as a risk. Think of places where business requirements are unknown and can only be deduced from the software solution, or a technical implementation that itself is insufficiently understood. But it might also be a concern for aspects like security, compliance, maintainability or the ability to expand and evolve not being properly addressed. So it's wider, or more defined, than technical debt. And it can occur in both greenfield and brownfield projects.

Once identified, it's important to proceed with quantifying the risks of ignoring these hot spots, and with planning when to address them, preferably along the work on the value streams that the team works on. The fact that such a risk storming activity is concerned with various levels of detail of a software solution, fits with how architecture can be expressed with the C4 model of Simon Brown. It may come as no surprise that Simon Brown himself has initiated a technique for risk storming, aptly explained on https://riskstorming.com/ which seems worth exploring as a concrete approach for the aforementioned activities.

SAFe

HOLD

Scaled Agile Framework (SAFe) is a set of organization and workflow patterns intended to guide enterprises in scaling lean and agile practices. Scalability of these principles means giving priority to improving collaboration across all groups within an organization. SAFe is helpful in creating awareness within an enterprise about best practices. However, experience shows that it often results in a lot of new practices that don't contribute to team effectiveness.

An example of a new process is the Program Increment (PI) planning events. Planning events are aimed at encouraging teams to communicate and collaborate. This inadvertently breeds an expectation that PI Events are required to enable effective collaboration and communication, while flexible and spontaneous communication and collaboration can achieve a similar effect.

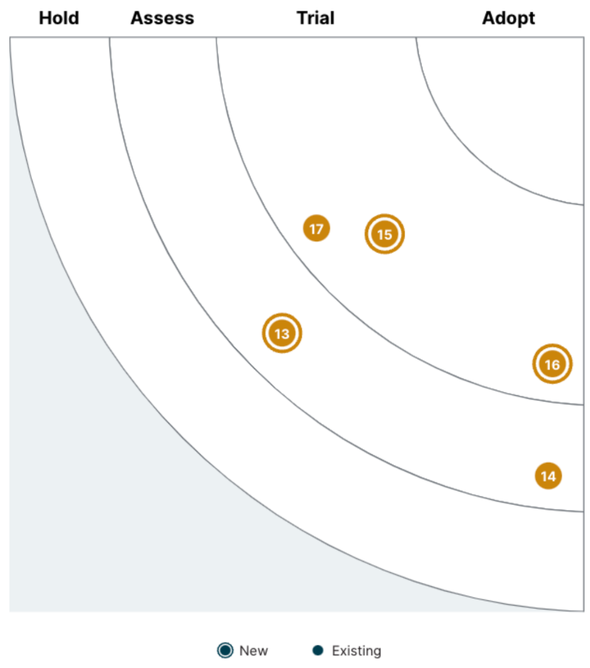
We see that companies embrace SAFe's promise of scalable LEAN and Agile Processes. When SAFe is seen as a cure-all solution, it is easy to forget that SAFe requires significant changes throughout an enterprise, including at the executive level. This often leads to a push to introduce new SAFe processes, even before ensuring that the enterprise is ready to embrace them. This creates a situation of limited team buy-in, while underlying organizational and cultural problems persist.

Commit. Develop. Share.

Platforms

Assess
13. Dapr
14. On-premise

Trial
15. Apache Pulsar
16. Edge Computing
17. GraalVM



Apache Pulsar

TRIAL

Apache Pulsar [https://pulsar.apache.org/] is an open-source distributed pub-sub messaging system that offers a high degree of scalability, durability, and performance. Similar to Apache Kafka and RabbitMQ, Pulsar provides a way for applications to communicate with each other by exchanging messages through a broker.

However, Pulsar has some notable differences when compared to Kafka and RabbitMQ. One of the key advantages of Pulsar is its ability to support both queuing and pub-sub messaging models in a unified system, providing flexibility for different types of messaging patterns. Pulsar also offers advanced features like dynamic partitioning, which allows for better load balancing and resource utilization in distributed environments.

In terms of scalability, Pulsar's architecture is designed to handle millions of messages per second with low latency and high throughput, making it a good choice for real-time streaming applications. Pulsar also has a built-in tiered storage system that allows for efficient data retention and retrieval.

Overall, while Apache Kafka and RabbitMQ have their own strengths and use cases, Apache Pulsar is a compelling option for companies looking for a highly scalable, durable, and flexible messaging system that can handle both queuing and pub-sub messaging models.

At JDriven, we see that more and more customers are experimenting with Apache Pulsar. Given the above benefits we have put Pulsar on *Trial*. For new landscapes, Apache Pulsar can be an interesting alterative to Kafka or RabbitMQ. However, if there are no problems on existing platforms, we see no reason to migrate them to Pulsar yet.

Edge Computing

TRIAL

Edge computing brings computation power and data storage closer to the end user, resulting in faster processing and response times. This is similar to the way content delivery networks improve the performance of static websites. In recent years, various cloud providers have added edge functions to their edge nodes. Edge functions introduce the option to offload certain functionality from the client to these edge nodes.

Vendors like Netlify, AWS and Vercel are expanding on edge functions and accompanying middleware. From the frontend perspective, there's now a place closer to the client that can do some of the server side rendering. This has the benefit of speed, but carries the security risks inherent with having more public entry points in your architecture and can mean an increase of the complexity of your overall architecture. Consequently, we see potential and risks, which warrants experimenting with these new solutions in architectures between Single Page Application (SPA) frameworks and the backend.

GraalVM

TRIAL

GraalVM [https://www.graalvm.org/] is a high-performance JDK designed to accelerate the execution of applications written in Java and other JVM languages while also providing runtimes for JavaScript, Python, and a number of other popular languages via polyglot technologies. To create native executables it contains the ahead-of-time (AOT) compiler.

It's now part of OpenJDK [https://www.graalvm.org/2022/openjdk-announcement/]. Important to know will be that the polyglot technologies are not part of OpenJDK but probably usable. Nowadays, there is support for native executables for all major frameworks. This has made the threshold to use GraalVM a lot lower, especially with the use of third-party libraries with the Shared Metadata [https://medium.com/graalvm/enhancing-3rd-party-library-support-in-graalvm-native-image-with-shared-metadata-9eeae1651da4].

In trial to use for native executables with a side note that it will not suite every project, because it's a tradeoff between faster startup times and lower memory usage on the one hand and lower throughput on the other.

Dapr

ASSESS

Dapr [https://dapr.io/] (Distributed Application Runtime) is a platform developed by Microsoft that simplifies the connectivity of microservices. It provides a layer of abstraction over techniques that often need to be implemented in each microservice. These include pub/sub messaging, state management, secrets management, configuration, tracing and logging. Dapr runs in a sidecar alongside the microservice. The microservice then communicates with the sidecar via an SDK which is, among others, available for Java, .NET, Python, Javascript and Rust.

We placed Dapr in the *Trial* quadrant of this edition of the Tech Radar because it potentially greatly simplifies the development of a microservice landscape. One drawback we see is that it is likely to be difficult to remove it from an application landscape once implemented.

On-premise

ASSESS

Re-evaluating cloud versus On-premise [https://world.hey.com/dhh/why-we-re-leaving-the-cloud-654b47e0].

It has been just over 15 years since we were introduced to the Cloud. Managers and technicians alike were persuaded to switch by big promises and slick marketing talks. How could one ignore the promises of cutting costs in half? Or the simple use and reduced workload compared to maintaining servers?

Nobody knew what this, mostly unproven, new technology would truly give us in these early days as we migrated vast amounts of infrastructure to the Cloud. Now, over 15 years later, we have a good understanding of the strengths of the Cloud, but we have overall neglected to consider its weaknesses.

High cost

Though the reason for many companies that experience high costs can be explained by poor implementation, there are well-structured, optimised implementations that suffer from the same. This can mostly be observed in implementations where none of the Cloud's strengths are being utilised. Migrating away from the Cloud can reduce costs in these cases. Reductions that can quickly grow, in some cases reducing the costs by more than half. In general, the following scenarios benefit from the Cloud:

- Small applications with limited use. The Cloud is able to scale costs with usage in a way that is impossible to mimic on-premise. Especially if an application is optimised to utilise a serverless infrastructure.
- Applications with huge swings in utilisation. The near infinite available capacity provided by a Cloud provider ensures a level of performance and limits downtime in a way that cannot be achieved in a private datacenter.
- Applications that require high availability. The huge infrastructure capacity of a Cloud provider with failovers configured by default that are capable of operating cross-country and even cross-continent is unmatched by any other technology.

On-premise solutions have their own generic benefits:

- Applications with stable/fixed usage. Any application that serves a stable load, requiring a stable amount of resources with little to no fluctuations will, in general, consume a stable amount of the underlying infrastructure. If the required infrastructure is fixed then there is no benefit in the presence of scalability. Like in the real world, in these cases renting is more expensive than buying.

High complexity

One of the key selling points of the Cloud at the time of its introduction was its ease of use (compared to on-premise). This may have been the case at the time but with the addition of new intermediate services and the feature growth of existing ones, the complexity has increased to a point that it is on par with the (old) on-premise infrastructure.

Dependability of the provider

The number of Cloud infrastructure vendors is small. Within a short time after its introduction, AWS was far ahead of the competition and was dominating the market. Though this gap has been closed since, the total number of Cloud infrastructure providers (used by 90% of Cloud users) can literally be counted on one hand. This has led to a new, internet wide, single point of failure where an interruption at a single Cloud provider can take down a large part of the internet's online services with them.

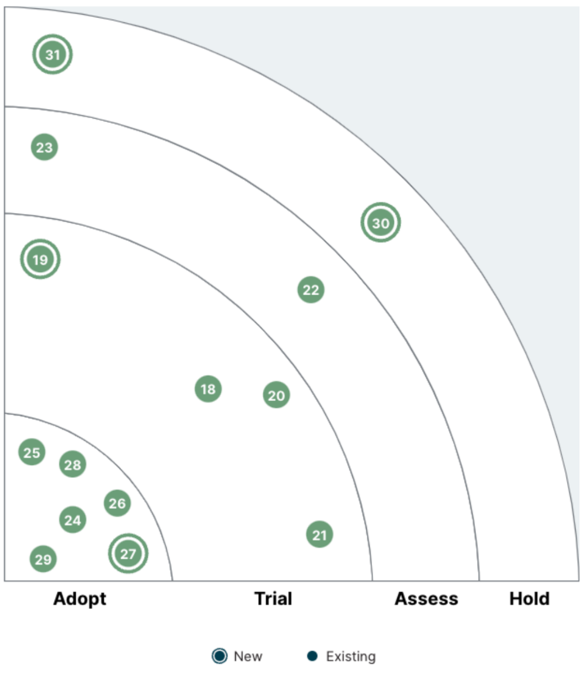
On-premise

The time has come to abandon the default decision to run any new service in the Cloud, and start taking cost and technical considerations. Especially as progress has been made in on-premise applications, just as progress was made in Cloud services. Platforms like OpenShift and Docker have given a similar ease of use when setting up an infrastructure outside of the Cloud.

There is a benefit on any project to take the same level of considerations on what environment will run the applications, as the level of consideration given to the best techniques used in its development.

Commit. Develop. Share.

Tools



Trial
18. Error Prone Support
19. Sigrid
20. Spring Boot Migrator
21. Testkube

Assess
22. Diffblue Cover
23. SigNoz

Adopt
24. CNCF Trailmap
25. Datafaker
26. Gradle Kotlin DSL
27. k9s
28. Maven Daemon
29. OpenTelemetry

Hold
30. Docker 4 Desktop
31. Lombok

CNCF Trailmap

ADOPT

The CNCF Trailmap [https://github.com/cncf/trailmap] describes the route that leads from an on-premise monolith to a Cloud Native application.

The trailmap can be used to guide a conversation about the Cloud strategy. The trailmap provides insight into the starting point and the desired destination on the road to a Cloud Native application. The trailmap can be supplemented with the landscape [https://landscape.cncf.io/] that provides an up-to-date overview of tools and techniques. Both tools are maintained by the Cloud Native Computing Foundation (CNCF). The CNCF was founded in 2015 and has since been involved in container- and cloud technology.

The trailmap is an accessible way to check where an organization stands and discuss where an organization would like to be. As a result, the trailmap can be used at various levels in the organization, from development teams to (ICT) management. Together with the landscape, the tool provides guidance to move the organization to the right place. For these reasons, we have placed the CNCF trailmap in the adopt quadrant of this edition of our Tech Radar.

Datafaker

ADOPT

Datafaker [https://www.datafaker.net] is a Java/Kotlin library to generate real looking test data. Using Datafaker we can easily create real looking names, addresses, telephone numbers, credit card numbers, medical data, but also more fun data like characters from Star Wars or quotes from the Back to the Future movie. Using Datafaker has several benefits, like data that cannot be identified as personal data, creation of unlimited set of data and generated data can be specific for a country or language. Datafaker can be extended with so-called custom providers to build new data sets.

Datafaker is in the *Adopt* quadrant, because it is a mature library that adds a lot of value to testing.

Gradle Kotlin DSL

ADOPT

Gradle is the second most widely used build tool after Maven to develop applications and libraries on the JVM. Gradle differs from Maven in that it is faster and, above all, more flexible. Whereas in Maven the build is described with an XML document, Gradle uses a Domain Specific Language (DSL). Traditionally only Groovy was supported, but for several years a Kotlin variant of the DSL has also been available. In fact, as of Gradle 8.2, the Kotlin DSL will be the standard for new projects.

This Kotlin variant offers a number of advantages:

1. First of all, Kotlin is a statically typed language, while Groovy is dynamically typed. Because of this, we get type-safe model accessors when we write our buildscripts in Kotlin. This means we get code completion in our IDE (for now, IntelliJ IDEA and Android Studio support this) when we write our buildscript. We sometimes also get code completion in IntelliJ IDEA for Groovy build scripts, but that is only for known APIs and not for third-party plugins, for example. With Kotlin, the IDE can offer traditional code completion based on classes and does not depend on specific features. As a result, code completion also works for third-party plugins.
2. Restructuring, refactoring, of buildscripts is now also easier from the IDE, because the IDE can already refactor typed code. For the IDE, the buildscript is now a Kotlin source, so everything we can normally do for refactoring Kotlin code we can now also use for our buildscript.
3. We can use Kotlin properties such as delegated properties and null safety. Since the buildscript is written in Kotlin, we can use everything Kotlin has to offer. For example, we can use delegated properties to get a reference to an object that we also want to use later in the buildscript.

The stability of the Kotlin DSL has improved so much in recent years that we have moved the *blip* to *Adopt* in this edition of our Tech Radar. For almost all projects, the Kotlin DSL will offer more value over the old Groovy DSL. The fact that Gradle now also uses the Kotlin DSL as a standard endorses this. Only for existing very complex buildconfigurations a migration is discouraged at this time.

k9s

ADOPT

k9s [https://k9scli.io/] is a tool to manage Kubernetes clusters using the command line. It allows you to both manage and observe your cluster using a simple UI, e.g. viewing logs, changing deployments, and connecting to your pods directly. As of July 2022 the UI based alternative Lens Desktop [https://k8slens.dev/desktop.html] has a paid subscription model, for which its grace period has ended in January 2023. For that reason we place k9s on *Adopt* as we see that it is a good free alternative to Lens Desktop.

Maven Daemon

ADOPT

Maven is a build tool for developing Java applications. To speed up the build we can use Maven Daemon [https://github.com/apache/maven-mvnd]. Maven Daemon provides several features that help running our builds faster.

1. At the first run a background process is started to keep the JVM alive.
2. By default, the build will start in multi thread mode.
3. The executable for Maven Daemon is built with GraalVM and is a native executable that starts fast.

The tool can already be used by developers on their projects and is placed in the *Adopt* quadrant.

OpenTelemetry

ADOPT

OpenTelemetry [https://opentelemetry.io/] is a project for simple, universal, vendor-neutral and loosely coupled solutions for logging, metrics and tracing. It is supported by the Cloud Native Compute Foundation (CNCF) and major players in the observability community. Therefore, it has a good chance to impact our projects in the future. The project consists of tools, APIs and SDKs for different programming languages, which enable the collection and export of instrumentation data. Hence, it is possible to analyse behaviour and performance of an application using any of the supported backends.

We see OpenTelemetry as a good candidate for a language- and vendor-neutral standard that offers a unified framework, that can be used in many scenarios, independent of the local conditions. For Java there exists an impressive set of integrations with existing frameworks, as well as a Java agent to rapidly have a working deployment.

Error Prone Support

TRIAL

Error Prone Support [https://error-prone.picnic.tech/] is a Picnic-opinionated extension of Google's Error Prone. It aims to improve code quality, focusing on maintainability, consistency and avoidance of common pitfalls. Error Prone itself operates as a compiler plugin, providing a fast feedback cycle to highlight common bug patterns. Refaster goes one step further to replace code invariants with a single prescribed code pattern. Error Prone Support provides additional Bug Patterns, as well as Refaster Rules for AssertJ, Guava, Streams and more. The Refaster Rules are particularly welcome, as there's a long-standing issue with Google not releasing their Refaster templates [https://github.com/google/error-prone/issues/649]. The contributions are a reflection of past and current technology choices at Picnic, with clear paths to move from old to new patterns. In the future Error Prone Support also aims to increase performance and make it easier to test Refaster templates.

We appreciate it whenever companies like Picnic contribute back to Open Source Software. Their additional bug patterns and Refaster templates allow anyone to further improve their code quality, provided they use a similar technology stack. We encourage users to try out Error Prone Support on a low risk project, to discover the benefits and limitations to further adoption. Therefore, Error Prone Support is placed in the *Assess* quadrant of this edition of our Tech Radar.

Sigrid

TRIAL

Sigrid [https://www.softwareimprovementgroup.com/solutions/sigrid-software-assurance-platform/] is a code quality benchmarking tool, aiming to give insights, and improving the quality of your codebase using static code analysis. It provides benchmarks based on best practices and patterns collected from various sources of code taken from, among others, open source projects. It assesses how the scanned code compares against it. Sigrid gives valuable insights in code quality to both developers and management, with a slight focus on the latter. Analysis is done after code is pushed to a repository and a pull-request is created. Sigrid does not provide a plugin for an IDE, and it won't be developed either. The Sigrid group suggests developers use tooling like SonarLint in combination with SonarQube alongside Sigrid. We use it together with a select group of our customers to improve insights in the pros and cons. Therefore, we've added Sigrid to the *Trial* quadrant of this edition of our Tech Radar.

Spring Boot Migrator

TRIAL

Spring Boot Migrator [https://github.com/spring-projects-experimental/spring-boot-migrator] (SBM) aims to help developers migrate applications to Spring Boot, upgrade existing Spring Boot applications or migrate an application to use (new) Spring Boot features.

It is an experimental Spring project created by the developers of Spring. With the expected release of Spring Boot 3 at the end of 2022 the Spring Boot Migrator could be extremely useful for applications. The project uses OpenRewrite [https://docs.openrewrite.org] recipes for migrating and updating the code.

The project is under development and has some limitations. Have a closer look at the GitHub project to see the actual supported build tools.

For now the project is looking for early adopters. Since it is supported by the Spring developers, it could be a trial for development teams. They would appreciate feedback on the project.

Testkube

TRIAL

Testkube [https://testkube.io] is a framework to execute and coordinate tests on Kubernetes. Testkube defines tests as Kubernetes Custom Resources (CRD) and provides support for tools like Maven and Gradle. With TestKube it is possible to run integration tests when for example a new version of a microservice is deployed. Tests are managed using a dashboard, command-line tools or configuration files. Testkube is part of the Cloud Native Interactive Landscape (CNCF). The product takes a different approach for running tests by running them on Kubernetes instead of in a Continuous Integration (CI) pipeline. Therefore we place it in the *Trial* quadrant so more experience can be gained with running tests this way.

Diffblue Cover

ASSESS

Diffblue Cover [https://www.diffblue.com/products/] creates suites of unit tests that run in your continuous integration pipeline between versions and protect against regressions, so you can catch errors faster and earlier in the software development lifecycle. Diffblue Cover's AI engine can quickly write a suite of unit tests that are derived from existing code, so they reflect the current functionality of the program. Because unit regression tests will only be created en masse, the individual tests themselves matter less than their collective capability as a net for catching regressions. The breadth and depth of tests that can be generated by Diffblue Cover quickly encompasses a wide variety of scenarios, including edge cases, corner cases and simpler boilerplate code that might have been missed (either intentionally due to cost, or accidentally) by a human.

We believe tools like Diffblue Cover have great potential to complement the work of a developer. By taking away some of the toil of invariant testing, developers can focus on more interesting test cases. The regression test capabilities can be invaluable when faced with a large otherwise poorly tested code base, or a need to move fast and continuously deploy to production. Compared to other AI completion tools we like the fact that Diffblue Cover runs locally. But the installation weighs in at 5 GB combined with a pricy per developer subscription that comes at a cost. Therefore, Diffblue Cover is placed in the *Assess* quadrant of this edition of our Tech Radar.

Commit. Develop. Share.

SigNoz

ASSESS

SigNoz [https://signoz.io/] is an open source APM (Application Performance Monitoring) platform for cloud-native applications that enables users to monitor, troubleshoot, and resolve performance issues in real-time. It provides an end-to-end solution for distributed tracing, log management, and metric visualization. With its clear display, underlying relationships can be more easily identified in the event of problems. Additionally, SigNoz supports OpenTelemetry, allowing a wide range of languages to be used.

SigNoz is a potential alternative to DataDog, NewRelic, and Prometheus in combination with Jaeger, and is therefore placed in the *Assess* quadrant of this edition of our Tech Radar.

Docker 4 Desktop

HOLD

Docker for desktop [https://www.docker.com/products/docker-desktop/] is a tool to work with containers on desktop computers or laptops and is mainly used on Microsoft Windows and MacOS operating systems. The licenses for Docker for desktop have changed in 2022 and could have consequences for companies where they have to pay for licenses. In 2023 the grace period of one year before companies had to pay for licenses has passed. Therefore we place Docker for desktop on *Hold* and advise to think about the implications of using Docker for desktop and also look for alternative tools that could be used.

Lombok

HOLD

Lombok [https://projectlombok.org/] is a code generator used to generate boilerplate code such as getters, setters, and constructors in Java code. The code is mainly generated based on annotations. The large number of annotations, the need for expertise in the Lombok implementation, and the emergence of newer Java language features like records are factors that make JDriven cautious about using Lombok for new projects.

There are many Lombok annotations, of which not every developer is aware of the consequences. Lombok generates code that a developer should be aware of but cannot actually see, as it is invisibly integrated into the bytecode during the code generation process. Reading code takes up ten times more of a developer's time than writing it. Thus, improving readability becomes crucial as it not only makes the code easier to read but also easier to write. This has already been discussed in Robert C. Martin's book "Clean Code".

Before incorporating annotations such as @Getter, @Setter, @NoArgsConstructor, and @Builder, it's important to assess whether they enhance the code's readability or pose difficulties when reading it. Overlapping functionality can occur when using several Lombok annotations, leading to code that contains redundant annotations that do not enhance its readability. In some cases, it is preferable to improve code readability by writing out the Java code entirely in a POJO (Plain Old Java Object) without relying on Lombok annotations.

JDriven suggests that simple examples which commonly implement Lombok annotations should not be directly used as project models, without a deliberate evaluation of the suitability of Lombok annotations for the given project. Therefore Lombok is placed in the Hold quadrant of this edition of our Tech Radar.

Commit. Develop. Share.

Commit. Develop. Share.

Commit.
Develop.
Share.