JDriven Tech Radar Fall 2022

Our view on recent developments in our field

6th edition

Commit to know.

Develop to grow.

Share to show.

jdriven

Commit. Develop. Share.

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:

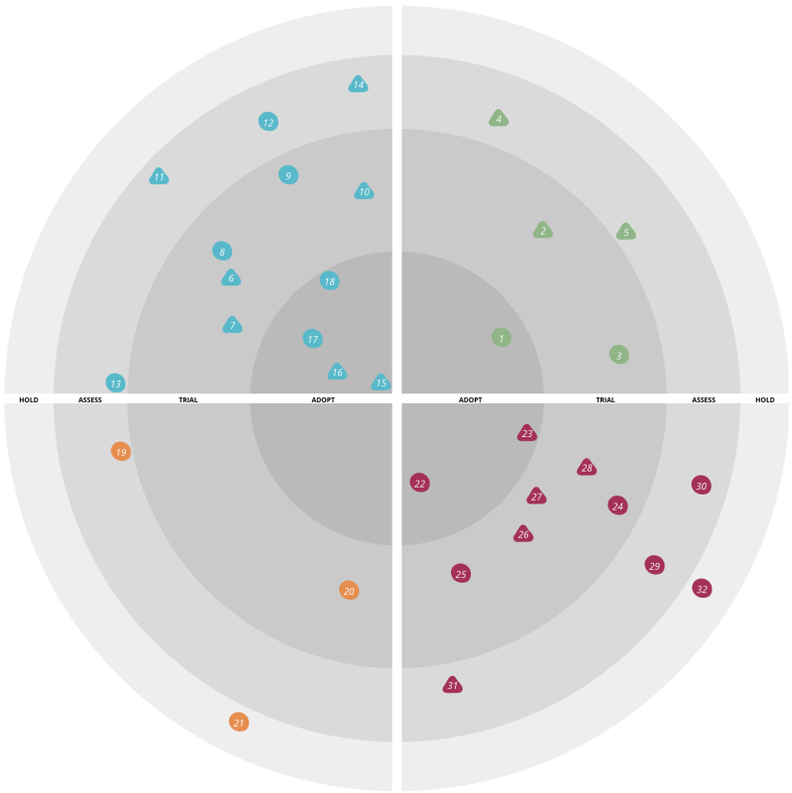
- **Languages and frameworks** that support developers in their daily tasks.
- **Platforms** to deploy and execute programs.
- **Techniques** help developers create better software.
- **Tools** aid in the development and delivery process.

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.

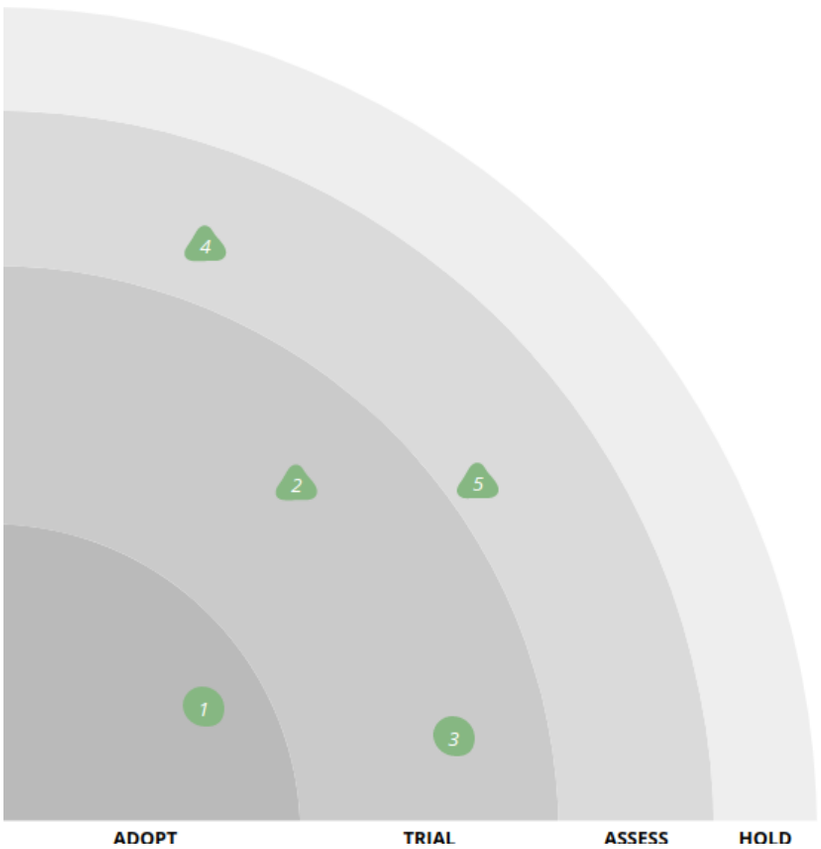
Commit. Develop. Share.



Tools	Languages & frameworks
ADOPT CNCF Trailmap Datafaker Renovate Spotless **TRIAL** Error Prone Support Gradle Kotlin DSL Kustomize OpenTelemetry Spring Boot Migrator **ASSESS** Diffblue Cover GitHub copilot Maven Daemon SigNoz	**ADOPT** Kotlin coroutines **TRIAL** Spring Modulith Unity3D **ASSESS** F# Virtual threads
Platforms	**Techniques**
ADOPT - **TRIAL** GraalVM **ASSESS** On-premise **HOLD** CircleCI	**ADOPT** Testing diamond Tinkering **TRIAL** Low-code Mob Programming SBOM WebAuthn Weighing developer experience **ASSESS** Data Oriented Programming Linked data Risk storming **HOLD** SAFe

Commit. Develop. Share.

Languages & frameworks



ADOPT

1. Kotlin co-routines

TRIAL

2. Spring Modulith
3. Unity3D

ASSESS

4. F#
5. Virtual threads

Kotlin coroutines

ADOPT

Kotlin Coroutines [https://kotlinlang.org/docs/coroutines-guide.html] is a first-class Kotlin library to write asynchronous, non-blocking code that is well readable and maintainable. It is developed by JetBrains and builds on the low-level *suspending function* APIs the Kotlin language provides.

If you're writing a Kotlin application which requires asynchronous, non-blocking code, Coroutines is the library to use. It is mature, has great IDE support and makes multithreaded code just a lot easier to write and maintain.

Spring Modulith

TRIAL

Spring Modulith [https://spring.io/projects/spring-modulith]

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, 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.

Unity3D

TRIAL

Unity3D is a market leading game engine. It was first released in 2005 as a macOS exclusive game engine. Over the years it has solidified itself as a developer-friendly game engine, with a fair licensing model and a "build once, deploy anywhere" principle. By now it is used by millions of developers who create thousands of games and other applications on a yearly basis. Unity3D has also branched itself into the automotive, animation, construction and manufacturing industries, creating a more integrated 3D development solution. This expansion has led to a surge of new users of the game engine (or better said, 3D engine) and a newfound interest.

We have noticed that many organizations that are experimenting with VR/AR and/or 3D visualisations of their data are gravitating towards Unity3D. Unity3D being embraced by an increasing number of industry sectors, together with the increasing need to visualize 3D data, has prompted us to add Unity3D to the tech radar this spring. If you are innovating with 3D data, or are prototyping AR/VR solutions, then Unity3D is definitely one of the best tools out there for you to use.

F#

ASSESS

Java is getting more and more options 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. But by looking at a full functional language, it will be easier to understand FP concepts.

F# [https://fsharp.org/] is a language that is particularly suitable for this case. The language does run on the .NET runtime. It is a strongly typed programming language that combines both functional, imperative and object-oriented programming methods. Because the syntax is very straightforward, it's simple to get started with the functions that the language offers. 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. Because all these features are included in the language itself, you will be able to understand the underlying concepts more quickly.

Let's explain this with the following example. The well-known function to calculate Fibonacci numbers can be written in F# as follows:

```
let fib n =
    let rec loop acc1 acc2 n =
        match n with
        | 0 -> acc1
        | 1 -> acc2
        | _ -> loop acc2 (acc1 + acc2) (n - 1)
    loop 0 1 n

printfn "%i" (fib 33) // output: 3524578
```

*The `fib` function defines an inner **rec**ursive `loop` function to do the computation. This inner function uses pattern **match**ing to determine whether it should call itself again. Finally, the compiler uses tail-call optimization to convert the **rec**ursive `loop` function to a for loop. With just a few lines of code, three FP concepts have already been covered!*

The applicability of non-JVM languages may be small within the JVM community. However, it will become easier to implement upcoming new Java features 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.

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).

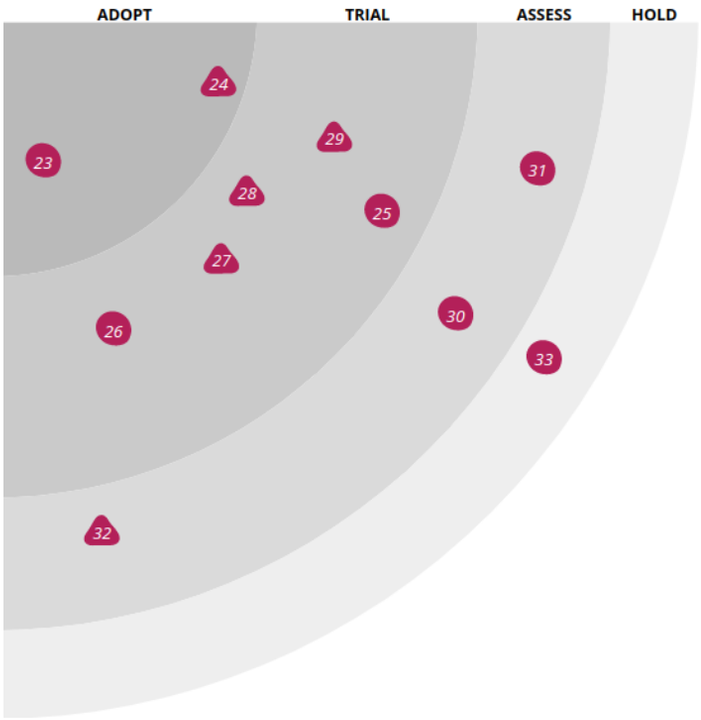
This concept of using threads in a more efficient way by making them execute multiple tasks 'simultaneously' is not new. Early versions of both macOS and Windows were already using so-called cooperative (non-preemptive) multitasking, where multiple processes/tasks could run on a limited number of threads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

23. Testing diamond
24. Tinkering

TRIAL

25. Lowcode
26. Mob Programming
27. SBOM
28. WebAuthn
29. Weighing developer experience

ASSESS

30. Data Oriented Programming
31. Linked data
32. Risk storming

HOLD

33. SAFe

Testing diamond

ADOPT

To take software testing to a higher level, one often chooses to categorize the different types of tests into three groups:

- UI and end-to-end tests
- Integration tests
- Unit tests

In a traditional software landscape the biggest part of testing focuses on the units; the integration and UI tests are considered of lesser value. This is often represented in a pyramid shape. The most important tests form the base of the pyramid, the other tests shape the second layer and the top. Using the 'test pyramid' is a good way to test monolithic applications.

However, in the current software landscape the use of the test pyramid is most likely not sufficient, because multiple microservices are often used. The three test groups are still relevant though, but the proper functioning of mutual relationships between the services is of greater importance than the individual units. Thus, the shape of representation changes from a pyramid to diamond. From that point of view, the integration tests are now the heart of your test suites, the other test groups are secondary. The 'testing diamond' is therefore an excellent method for making complex multi-service architectures testable.

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 learn. When we dive into a new subject it can be very overwhelming, therefore we must restrict ourselves in what we want to learn. We can use the zones of proximal development to help us with it. It consists of three zones: What you know, what you can learn, and what you cannot learn. This means that when we want to learn a new framework written in Java/Kotlin, we might already know Java, but not Kotlin. So what we can learn is how to use the framework in Java. What we cannot learn at the same time is Kotlin. Once we have mastered how to use the framework in Java, you might go to the next step, but don't attempt to learn it all in one go.

In our industry, frameworks come and go, but most of the time they are based on common concepts. Learning these concepts is important, because it makes it easier to learn about new frameworks. Concepts will not change as often as technology. Examples of concepts are object-oriented programming, domain driven design and design patterns.

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.

Low-code

TRIAL

Low-code and no-code development platforms allow business users to develop applications without coding the implementation. Programming an application is aided by graphical user interfaces, which allow the user to model simple processes and connect data sources. A no-code platform only exposes a graphical user interface, as opposed to a low-code platform where a user is allowed to write code to accomplish the task.

Low-code and no-code platforms are gaining traction. They provide a solution to automate simple business processes where hiring professional software developers would be too expensive.

The benefits of using these platforms are mainly to relieve scarce resources like software developers to focus on more advanced tasks.

The use of their platforms, however, also comes with certain downsides. Common issues include vendor lock-in to the selected platform, and selective functionality based on the facilities and expressive capabilities of the platform. Integration with external systems can also be challenging when those systems cannot communicate over standardised protocols. In addition, migration to different platforms or solutions will be difficult.

Low-code and no-code platforms are well-suited to quickly prototype an MVP. To do so successfully, it is important that the processes that will be automated are relatively simple, and have few dependencies. If you want to try low-code development yourself, a good place to get started is this article [https://www.thoughtworks.com/insights/articles/making_case_low-code_platforms] by Thoughtworks.

Mob Programming

TRIAL

Mob Programming [https://en.wikipedia.org/wiki/Mob_programming] is a more extreme form of pair programming where the whole team will work on the same feature. During mob programming there is one driver behind the keyboard and the rest of the team will give this driver instructions.

Although this looks inefficient at first, we also see some clear advantages:

- Sharing knowledge; Because the whole team is collaborating knowledge is shared across the whole team
- Training; More junior develops will get a lot of support doing this
- No code reviews needed
- Multiple perspectives

Although this will not work for every team or program, we actually have good experiences in mob programming, and we suggest you give it a try.

SBOM

TRIAL

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].

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.

The main promise of WebAuthn is passwordless authentication. Passwords are considered bad for security as it is a single key that proves who we are. If hackers steal the password it can be used without further restrictions. 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. There is built-in support into operating systems (iOS, Android, Windows, macOS), so you can use platform authenticators like Touch ID sensors on MacBooks and facial recognition and fingerprint sensors on PCs. Authentication providers like Keycloak also support

WebAuthn out of the box.

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.

Weighing developer experience

TRIAL

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.

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.

Linked data

ASSESS

Linked Data is a technique where data is linked to each other and can be interpreted without moving datasets. With Linked Data, you build a structured network of your data. You can use this network for rich analysis of your data, and to easily share your data with context.

In the academic world the term 'Linked Data' has been discussed for years. In the last year or two, there has been an increased interest in the technique, and as a result, it has been rapidly maturing. That is because a lot of standards have been defined that describe basic usages. Such standards describe, for example, how you can define an address and how you can interpret it.

Thanks to such standardisation, Linked Data is easier than ever to implement. For example, the Dutch Cadastre is providing public Linked Data endpoints for general use, where developers can link their datasets with the ones provided by the Cadastre. If you are researching ways to save your data in a way that makes it easily interpreted by machines and simultaneously makes it easy to connect with external datasets, then take a look at Linked Data technologies.

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.

Platforms

TRIAL

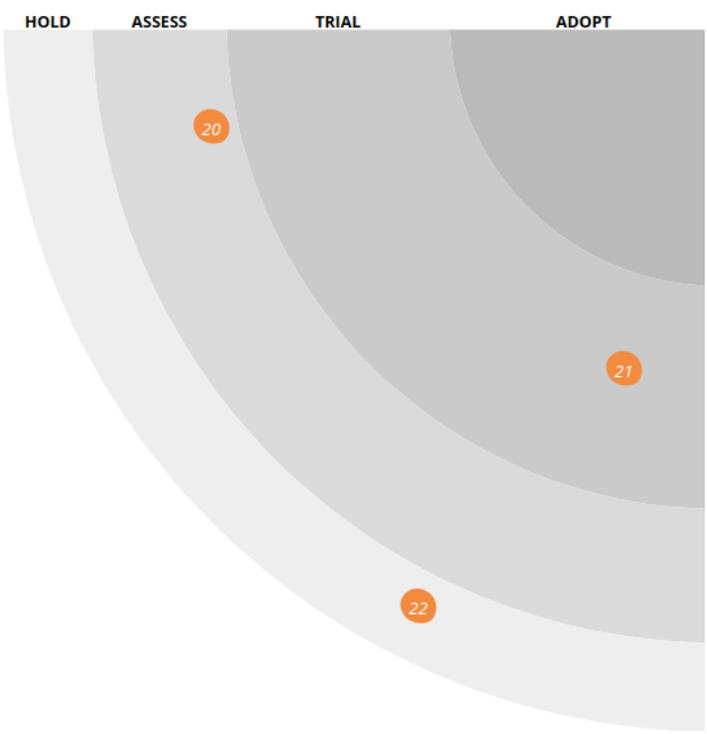
21. GraalVM

ASSESS

20. On-premise

HOLD

22. CircleCI



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.

On-premise

ASSESS

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

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.

CircleCI

HOLD

CircleCI [https://circleci.com/] is a Continuous Integration and Continuous Deployment (CI/CD) cloud platform which is used to automatically build, test and deploy software projects. It supports very complex build pipelines on multiple OSes (such as Linux, macOS and Windows) and hardware platforms (x86_64 and ARM).

After repositories on GitHub or Bitbucket are authorized and added as a project to CircleCI, every commit can trigger a CircleCI job. Builds run in Docker containers and pipelines are created using a YAML configuration file within the Git repository of a software project.

While CircleCI is a very good platform, for most software projects it doesn't offer any added value over tools such as GitHub Actions, GitLab CI or Bitbucket Pipelines; especially because these tools are often already available at reduced or no additional costs. We only recommend using CircleCI when a project requires a complex build pipeline which benefits from extensive parallelization, or when builds need to be run on multiple (hardware) platforms.

Tools

ADOPT

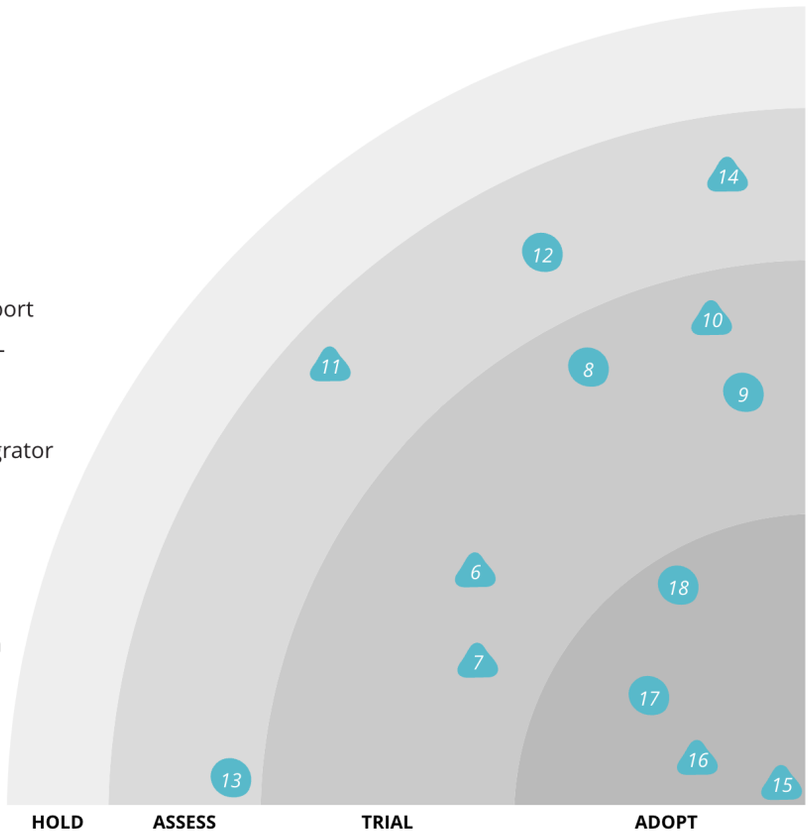
15. CNCF Trailmap
16. Datafaker
17. Renovate
18. Spotless

TRIAL

6. Error Prone Support
7. Gradle Kotlin DSL
8. Kustomize
9. OpenTelemetry
10. Spring Boot Migrator

ASSESS

11. Diffblue Cover
12. GitHub co-pilot
13. Maven Daemon
14. SigNoz



CNCF Trailmap

ADOPT

The Cloud Native Computing Foundation (CNCF) is an organisation which strives for better container and cloud technology since 2015. Projects like Kubernetes and Helm are part of the CNCF. Besides the many projects of the CNCF the organisation also develops tools to help projects become more Cloud Native, organises events and offers courses and certifications.

The CNCF maintains a landscape [https://landscape.cncf.io/] of tools related to Cloud Native Development sorted by subject. The CNCF also developed the CNCF Trailmap [https://github.com/cncf/trailmap] [https://github.com/cncf/trailmap/blob/master/CNCF_TrailMap_latest.pdf]. The Trailmap (combined with the landscape) can help in developing a Cloud Native Strategy.

The Trailmap can be used to plot a route from an on-prem monolith to a Cloud Native Application. Like all trails it's not about the destination, it's the journey that matters. The Trailmap offers an easy way to discuss the Cloud strategy. Within teams, it can be used to discuss where a project currently stands and where it should be. It can help you plot the journey and offers hints on tools to help you take the next step. Within the organisation it can help selecting the appropriate tools to support the Cloud projects.

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:

1. There are no issues that data can result into personal identifiable data. As the data is generated and not based on any real data it cannot contain real person data. The alternative of anonymized datasets could still have issues when the process of anonymizing the data is not done correctly.
2. It is easy to generate an unlimited set of data. We only have to define the number of elements we want to have generated and Datafaker will create it for us.
3. The generated data can be localized. Datafaker supports generation of data based on a locale. For example names can be generated based on a Dutch locale, but also using Arabic. This makes it possible to test your code with values we might not come up with ourselves.

We can use Datafaker in our unit test code when we want to have random values to test our production code and we want the random values to make sense. For example when we have a method that accepts a `String` argument as a name value, we could randomly generate a value where each character is randomly generated. But it makes more sense to have a randomly generated name value and that is where Datafaker can help. Not only in unit tests we can use Datafaker, but we can also use Datafaker to generate CSV, JSON, SQL, XML and YAML files.

Datafaker can be extended with so-called custom providers. These providers are responsible for returning random data from a set of predefined values. So if we are missing a provider that is already supported by Datafaker we can extend Datafaker with our own provider.

Renovate

ADOPT

Renovate [https://docs.renovatebot.com/] is an open source tool to automate dependency updates, similar to Dependabot which we featured in Spring 2020. On top of what we've seen from Dependabot, Renovate has a nice onboarding flow for new projects, with minimal required configuration. But what really sets Renovate apart is the official support for multiple platforms such as Azure, BitBucket & GitLab. This makes it an attractive option for anyone not using GitHub, after Dependabot was acquired and integrated there.

We wanted to highlight Renovate as an option for anyone not using GitHub in particular, to get the same benefits when using other platforms. Given what we've seen over the past half year with high impact CVEs there's enough reason to stay up to date, and automation makes that easier.

Spotless

ADOPT

Spotless [https://github.com/diffplug/spotless] is a tool to check and apply code formatting rules. What sets Spotless apart is that it is a build tool plugin and supports multiple languages. Because of this it integrates well with projects and build pipelines.

This is why we see Spotless as a good solution to remove the discussions about code style and formatting from the teams, which is especially relevant with the rise of more diverse develop environments.

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.

Gradle Kotlin DSL

TRIAL

Gradle is next to Maven a major build tool for developing applications and libraries. To describe a build in Gradle in a build script we use a Domain Specific Language (DSL) which is implemented on top of Gradle's Java API. We can choose to write the DSL using Groovy or Kotlin. The script itself is compiled and then executed by Gradle. Although traditionally only Groovy was supported we get a lot of benefits when we use the Kotlin variant:

1. First of all, Kotlin is statically typed, where Groovy is dynamically typed. We get type-safe model accessors when we write our build script with Kotlin. This means that we can get code completion in our IDE (at the moment only Intellij IDEA and Android Studio) when we write our build script. Groovy build scripts also support some code completion in IntelliJ IDEA, but that was only for known APIs and not for third-party plugins. Now with Kotlin the IDE only has to support traditional code completion and doesn't depend on specific IDE features, so also extensions from third-party plugins work with code completion.
2. Refactoring is easier from the IDE, because the IDE knows how to refactor typed code already. For the IDE it is now just a Kotlin source file, so everything we would normally do for refactoring also applies to our build script.
3. Usage of Kotlin language features like delegated properties and null safety. As the script is written in Kotlin we have access to everything Kotlin has to offer. For example we can use delegated properties in our build script which can be useful if want to get hold of an object and reference it in our build script.

The Kotlin DSL is mature and can be used in projects. It improves the developer experience a lot, especially when IntelliJ IDEA or Android Studio is used.

Kustomize

TRIAL

Kustomize [https://kustomize.io/] is a template engine for Kubernetes configurations. It allows for a single declarative system to specify the full configuration and deployment conditions for multiple targets, such as *develop*, *staging* and *production* by reusing common parts of the configuration. It works by providing different overlays of templates, each filling in or modifying the parent layer manifest

templates. Kustomize unifies this description for the entire application context, both for in-house developed applications and off-the-shelf tools. Kustomize is natively integrated with the Kubernetes `kubectl` utility since version 1.14. This makes it easy to integrate it into existing deployment pipelines. Being a template system, the syntax is highly reminiscent of the regular Kubernetes configuration files.

While the templating syntax for Kustomize can sometimes be a bit confusing, its layering approach makes it easy to extend a plain Kubernetes configuration step-by-step. This also makes the configuration differences between environments easier to use and understand when many deployment targets are used. All this makes for virtually no downsides in experimenting with Kustomize.

OpenTelemetry

TRIAL

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.

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.

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.

GitHub copilot

ASSESS

GitHub Copilot [https://copilot.github.com/] is powered by Codex, the new AI system created by OpenAI. GitHub Copilot understands significantly more context than most code assistants. So, whether it's in a docstring, comment, function name, or the code itself, GitHub Copilot uses the context you've provided and synthesizes code to match.

GitHub Copilot sends surrounding parts of text in the current file to the server, where it is automatically processed and code completion suggestions are generated by the AI. The requested text is saved for future predictions and training of the backing AI. We recommend caution when working with tools like these and privacy concerns surrounding it, since your data is sent and kept for training purposes.

For the time being there is no license and users are admitted on the basis of a waiting list.

Maven Daemon

ASSESS

Maven is a popular 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. This means that on subsequent runs the JVM doesn't have to be started, but an already running JVM is used. Also, Just-In-Time (JIT) optimizations are kept and can be re-used by new build executions. The classloaders of plugins are also cached, so they only have to read once.
2. By default, the build will start in multi thread mode. Normally we can start Maven with the `-T` or `--threads` argument to start a build with parallel threads, but Maven Daemon uses this as the default. This is a big advantage for multimodule projects, because it means modules that are built in parallel. So the final build time is ony determined by the module that takes the longest and no longer the sum of the build time of all modules. To keep the logging tight and neat the Maven Daemon captures the build output of each module and displays the progress on one line. In case of an error we can still the complete logging.
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 individual developers on their projects.

SigNoz

ASSESS

SigNoz [https://signoz.io/] is an open source APM (Application Performance Monitoring) for metrics, traces and logs. Signoz is an alternative for DataDog, NewRelic and Prometheus in combination with Jaeger. There are many advantages to combining metrics, traces and logs into a single pane. It creates an overview and helps determining underlying relations when problem solving. SigNoz is also available for self hosting, and therefore suitable for systems where data is not allowed to leave the private network. Because SigNoz supports the standard OpenTelemetry, a broad variaty of languages can be used.

Commit.
Develop.
Share.