JDriven Tech Radar Spring 2024

Our view on recent developments in our field

9th 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:

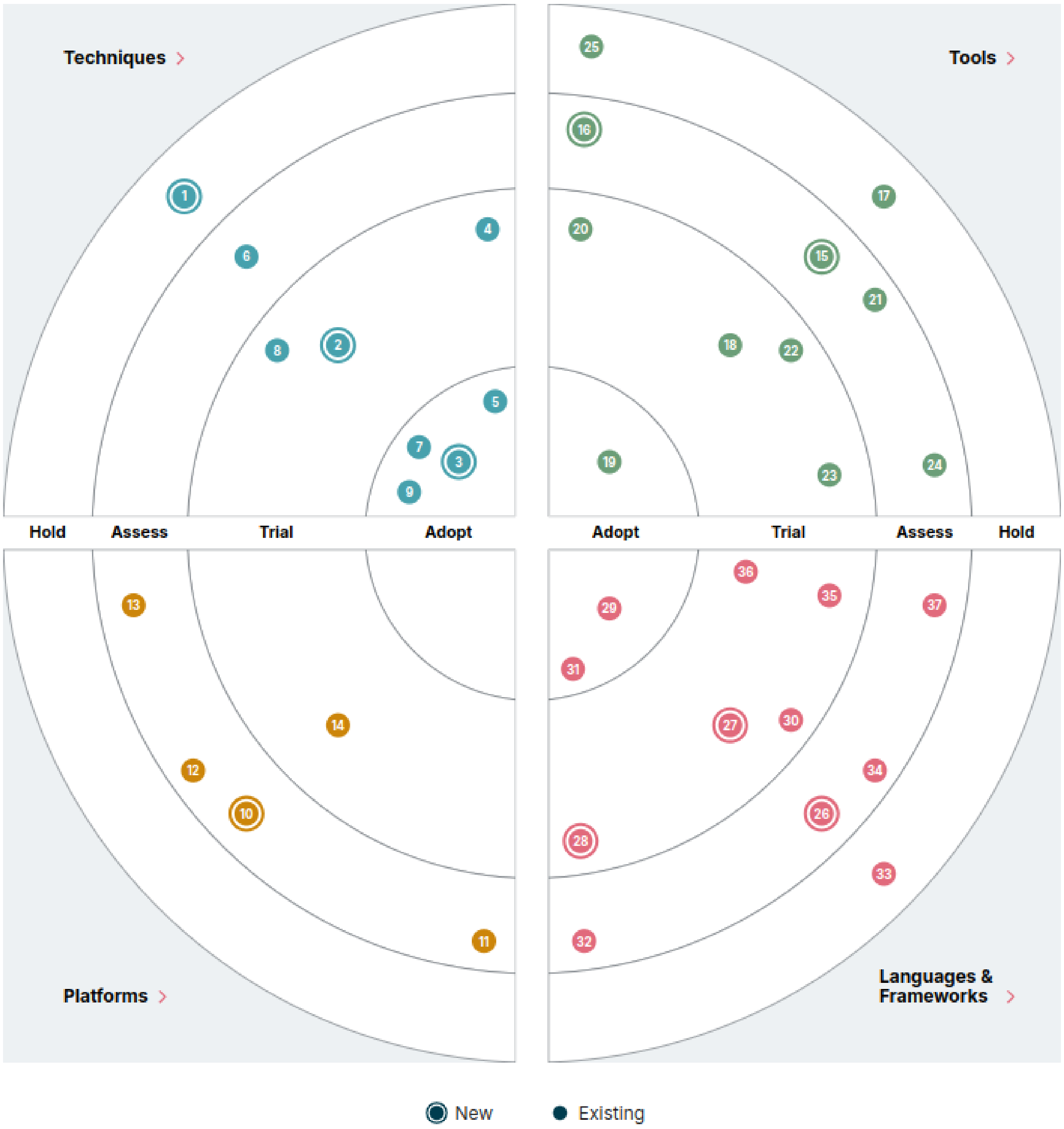
- **Languages and frameworks** that support developers in their daily tasks.
- **Platforms** to deploy and execute programs.
- **Techniques** help developers create better software.
- **Tools** aid in the development and delivery process.

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.

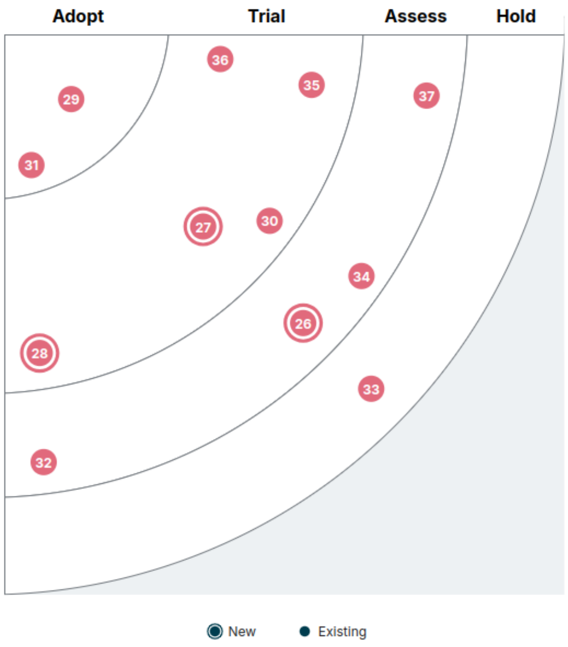


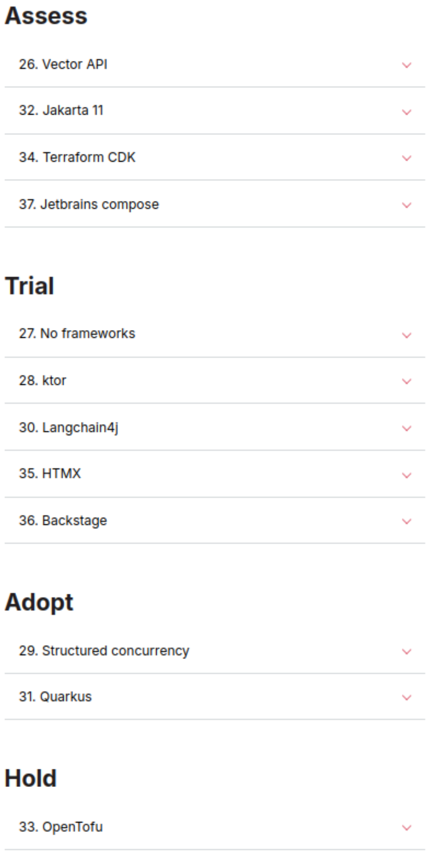
Commit. Develop. Share.

Tools	Languages & frameworks
ADOPT Vector Databases **TRIAL** ChatGPT jlink Maven Build Cache Extension Qodana **ASSESS** Claude 3 GitHub Codespaces Konsist Thunderdome Dev **HOLD** Diffblue Cover JobRunr	**ADOPT** Quarkus Structured concurrency **TRIAL** Backstage HTMX ktor Langchain4j No frameworks **ASSESS** Jakarta 11 Jetbrains compose Terraform CDK Vector API **HOLD** OpenTofu
Platforms	**Techniques**
ADOPT - **TRIAL** Cloud events **ASSESS** Azure container apps eBPF Moderne platform OpenFGA **HOLD** -	**ADOPT** Automatic merging of dependency updates CVE awareness Dev Ex Prompt Engineering **TRIAL** 1 TFB NFC Inner Source Unit testing voor alerting **ASSESS** Sustainable software engineering **HOLD** Distributed Monolith

Commit. Develop. Share.

Languages & frameworks





Assess

- 26. Vector API
- 32. Jakarta 11
- 34. Terraform CDK
- 37. Jetbrains compose

Trial

- 27. No frameworks
- 28. ktor
- 30. Langchain4j
- 35. HTMX
- 36. Backstage

Adopt

- 29. Structured concurrency
- 31. Quarkus

Hold

- 33. OpenTofu

Quarkus

ADOPT

Quarkus [https://quarkus.io/] is a nice framework for developers who are used to Context and Dependency Injection (CDI) and MicroProfile. According to their own website it is: "A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards." It is comparable to other frameworks such as Spring Boot, and is particularly suitable for building server applications.

Quarkus was developed to make life easy for developers. Very little configuration is required to work with it. The framework provides support to easily run in development, test or production mode. Quarkus uses memory efficiently. The functionality can easily be expanded with the help of extensions. It has become a mature product (version 3 has already been released). Quarkus provides standard support for building Docker containers, which can be deployed easily in e.g. Kubernetes or OpenShift.

Using Smallrye Mutiny as a Reactive Streams library allows imperative style to coexist with reactive style. This enables developers to easily make an application completely non-blocking, step by step.

Testing is also extensively supported. With the `@QuarkusTest` annotation you can easily include integration tests in the build pipeline, which also supports test containers. It is also possible to start your application locally in developer mode. And with the tool Skaffold [https://skaffold.dev/] (not a standard part of Quarkus) you can quickly test a new deployment in a locally running Kubernetes cluster (e.g. minikube or Rancher Desktop).

It is very suitable for HTTP RESTful web development and integrates well with common message protocols, authentication protocols, data stores and platforms. Thanks to the richly detailed manuals, it is also relatively easy to expand Quarkus as needed.

Creating native executables is no longer a distinguishing factor for Quarkus, but through its broad support and extensive documentation, Quarkus is one of the major JVM CDI Frameworks. In the past year, Quarkus was successfully used by various customers (including the financial sector). It has proven itself as a stable framework, giving Quarkus the *Adopt* status in this edition.

Structured concurrency

ADOPT

Structured concurrency is a programming paradigm that provides a disciplined and structured approach to managing concurrent tasks in applications. It introduces a set of tools and abstractions to simplify the coordination and execution of parallel operations, aiming to improve code readability, maintainability, and debugging in multithreaded scenarios. Unlike traditional thread management, structured concurrency ensures that all concurrently running tasks are explicitly scoped and managed, making it easier to reason about their lifecycle. This paradigm helps developers avoid common pitfalls associated with thread mismanagement, such as resource leaks and unpredictable behavior, by providing a more organized and deterministic way to handle concurrency.

We recommend using structured concurrency whenever appropriate when developing applications. In Kotlin for example, we recommend using Kotlin Coroutines for code that benefits from parallel execution. In Java 21, Structured concurrency is still in preview. Nevertheless, we recommend trying this new feature in advance and getting familiar with it so that it can be used as soon as it is released.

Backstage

TRIAL

Backstage [https://backstage.io/] is an open source developer portal created by Spotify and donated to the Cloud Native Computing Foundation (CNCF). It was published in 2020, so it is a relatively new platform.

The purpose of Backstage is to offer a central place for software, tooling and documentation. It also offers a way to create templates, with which you can quickly spin up new projects.

In order to reach these goals it offers integrations and many plugins to help with the integrations of all the tools in one place. The strengths of Backstage become more obvious in big projects.

Examples of big companies that have adopted Backstage are the following:

- LinkedIn
- Vodafone
- Lego
- Siemens

Backstage is placed in *Trial* because the usefulness of a developer portal is not always evident to developers having to work with it. Be sure the size of your project warrants it, and it meets the actual needs of developers in your organisation.

HTMX

TRIAL

htmx [https://htmx.org/] is a JavaScript library aimed at building a hypermedia-driven application [https://hypermedia.systems/hypermedia-systems/] (HDA). HDA and htmx are a response to heavy Single Page Application JavaScript frameworks like React and Angular, and refocus on the power of HTTP, HTML, hyperlinks and REST, summarized as *hypermedia architecture*. The rise of SPAs can be explained by the fact that it was hard to have fluid UI transitions and interactivity in web 1.0 applications, due to noticeably slow page reloads. Even now, HTML only offers 2 HTTP methods (GET and POST) and 2 HTML elements (anchor and form) for interacting with server content. The solution that SPA frameworks introduced for these shortcomings in HTML, does have some downsides: it transforms RESTful backend APIs into data APIs that serve JSON, makes JavaScript leading for showing HTML, and introduces a complex frontend architecture based on a lot of JavaScript libraries for seemingly simple tasks like URLs, web content indexing (SEO), the back button, caching, progressive enhancement and loose coupling.

We're seeing an increase in interest in htmx and HDA among full-stack developers, and even frontend developers who started their career applying SPA frameworks and recognise these problems. JDriven advises considering whether HDA and htmx might be a good fit before deciding on an SPA as the de facto standard frontend architecture. Pay attention to the applicability of tools in the frontend ecosystem, like design systems, Tailwind CSS and Playwright. Because htmx is a return to server-side rendering, it's also important to take into consideration developments in the domain of Edge Computing, which is a hot topic in the field of SPA frameworks.

ktor

TRIAL

Ktor [https://ktor.io/] is a Kotlin framework for quickly creating web applications with minimal effort. It is relatively lightweight, as it supports only the more common and contemporary functionalities required for web services, like routing, templating and serialization. By leveraging Kotlin's coroutines at its core, it makes it easy to build responsive web services that should scale relatively well. It also supports multiple HTTP engines out of the box.

Therefore, we have assigned Ktor to the *trial* ring. The framework has existed for quite some time (nine years), and seems to get a decent amount of traction within the community. The number of supported third-party dependencies is somewhat limited, however, compared to a more established framework like Spring. Thus, developers will need to write more glue code to use these dependencies.

Nevertheless, if your project needs can mostly be met with vanilla Ktor, it is a quick way to set up a responsive web service.

Langchain4j

TRIAL

Langchain4j [https://github.com/langchain4j/langchain4j] is a framework to easily develop applications with AI large language models (LLM).

These LLMs are capable of performing several natural language processing tasks like language translation and content summarization. They are also capable of generative tasks like content creation and are extremely valuable in answering questions.

With this framework, LLM providers can be swapped easily because it uses an abstraction layer. Implementation for OpenAI GPT-4, Azure OpenAI en Google Vertex AI are available, and also Ollama [https://ollama.ai/] for local models. It is the counterpart of Langchain [https://www.langchain.com/] library for Python en Javascript. The Java implementation is actively developed, and new features are added constantly.

If you wish to add AI features to your Java application, then check out this framework, which makes it is possible to easily change and test with different models.

We at JDriven believe that the abstraction layer of this framework leads to more flexible software without vendor lock-in, which is an advantage, because all features and models are constantly changing and getting better. When for example because of privacy or better results the model needs to be swapped then, this can be achieved easily.

No frameworks

TRIAL

Almost everyone uses frameworks for building applications. You can often build an application quickly and relatively easily with a framework, because a framework typically simplifies the following aspects (often using annotations and conventions):

- Loading configuration via properties
- Dependency Injection
- Built-in REST server
- Built-in JSON marshalling/unmarshalling
- Built-in REST client
- Built-in ORM
- Built-in transaction management
- Built-in log library

But frameworks also have disadvantages, often due to their size. You often notice this after some time, for example, when you want to use an additional library (dependency) that conflicts with one of the (transitive) libraries of the framework. Or when you upgrade the framework or such a library to a newer version. Suddenly, inexplicable errors can occur, such as complicated version conflicts of transitive dependencies, annotations no longer working, vague runtime errors causing tests or the application itself to fail to start, etc. It often takes a lot of effort to identify and resolve the cause because the framework is so large and complex and does a lot 'automagically' via annotations and/or conventions (which you don't all know). As a result, all the time saved by using a framework can turn out to be a short-term gain, but a long-term loss.

Why don't we just build an application without frameworks, and only with the lightweight libraries we need (cherry-picking)? This is very feasable with, for example, the following approach:

Commit. Develop. Share.

- Loading configuration via property files can be done with the standard Java `Properties` class. You don't need a library for that.
- Dependency Injection can also be done very well by yourself using constructor injection. You don't need IoC (Inversion of Control) and therefore no framework.
- A lightweight HTTP server like Jetty [https://eclipse.org/jetty/] or Undertow [https://undertow.io/] is sufficient for exchanging text strings over HTTP. And with a library like Jackson [https://github.com/FasterXML/jackson], you can marshal/unmarshal JSON strings yourself.
- A simple HTTP client like OkHttp [https://square.github.io/okhttp/] can be used to build a REST client in a similar way.
- Hibernate [https://hibernate.org/] is also fine to import as a standalone library if you need an ORM.
- And Hibernate also provides transaction management via `EntityManager.getTransaction().begin()` and `EntityManager.getTransaction().commit()`.
- There are several log libraries available for standalone use, for example, SLF4J [https://www.slf4j.org/].

This approach requires a bit more code than with a framework where you get a lot for free via conventions and annotations. But not that much more. The big advantage, however, is that you bring in fewer transitive dependencies and thus less complexity. Also, the code is more explicit, with fewer 'automagic' annotations and conventions. This makes debugging easier. Ultimately, this setup is can be simpler than with a large cumbersome framework that contains a lot more than you actually use (YAGNI - You Aren't Gonna Need It). In other words, a good example of KISS (Keep It Simple Stupid). If a framework is not required, it can certainly be worthwhile to opt for a no-framework approach.

Jakarta 11

ASSESS

Jakarta 11 [https://jakarta.ee/specifications/platform/11/] is the newest (upcoming) version of Jakarta EE. This new version will bring some new features like support for Virtual Threads and Jakarta Data (similar to Spring Data). It will also enforce the use of Java 21 and remove/deprecate some old specifications. This all makes it a very interesting version of Jakarta EE which we believe it makes a version worth assessing.

Jetbrains compose

ASSESS

JetBrains combines the power of Jetpack Compose and Kotlin Multiplatform to allow developing business functionality in one codebase, with a UI tailored to various platforms. It supports servers, Android, iOS, web (through Kotlin/Wasm), and desktop (Linux, macOS, Windows).

For developers who are used to the world of Kotlin and/or Android development, Compose Multiplatform looks very promising. It offers writing business logic and to some extent UI code once, and targeting various platforms. It remains to be seen, whether the effectiveness of working in a single codebase is worth the layers of abstraction required to maintain a big common denominator in code. Also worth noting is that iOS functionality is currently still in alpha phase, and Kotlin/Wasm is deemed experimental. But JetBrains Compose Multiplatform does seem like an option to explore for organisations that want to invest in business logic, a homogenous developer skill set, and a uniform UI across multiple platforms.

Commit. Develop. Share.

Terraform CDK

ASSESS

Terraform uses HashiCorp Configuration Language (HCL) to manage the infrastructure of an application in a cloud-agnostic manner. HCL is a declarative language for describing cloud resources. However, using it means that developers wil need to learn a new language in order to work with this tooling. By leveraging CDK for Terraform (CDKTF), one can now use Java, C#, Python, TypeScript, or Go for infrastructure management.

CDKTF deserves a spot on the Tech Radar because Terraform is already a proven technology, and this addition offers several advantages over the conventional approach. It provides code completion and IDE support, better integrates with existing tooling such as code formatters and linters, and makes it more accessible for developers to use Terraform in their projects.

As of August 21, 2022, HashiCorp designated the tooling as 'production ready,' with the caveat that the tooling is new and therefore not fully mature. Additionally, the documentation on certain topics is subpar, and the CDK introduces a few new concepts which adds to complexity. All in all, an interesting development to keep an eye on, especially if Terraform is already in use or under consideration.

Vector API

ASSESS

The Vector API is an incubator API to predictably leverage SIMD (single instruction, multiple data) instructions on the Java platform. These instructions perform operations on multiple primitive data elements simultaneously, up to 16 elements on modern microarchitectures. This can lead to substantial performance benefits when the same mathematical operation(s) need to be applied to a large number of elements. Originally released as an incubator API in Java 16, it is planned to be merged into the Java mainline when project Valhalla becomes available.

Under some circumstances, the optimizing compiler in e.g. OpenJDK's HotSpot VM can auto-vectorize operations itself. Relying on this behaviour for performance tends to be precarious, though. Only if all implicit conditions are met, the optimizing compiler will perform the auto-vectorization. Furthermore, the Vector API leverages Intel's SMVL library to vectorize common mathematical functions.

We have put the Vector API in the *assess* category. Although it is scheduled to be released under its eighth incubator for the Java 23 release this fall, it is still not finalized. Its main goal is to alleviate performance bottlenecks on parts of the code that depend heavily on numerical calculations. Hence it should be used only after profiling indicates there is a performance issue, as code leveraging the Vector API will be more difficult to read than regular operations on primitive types. Also, further improvements in JIT technology might reduce the need for explicit annotations of vector types.

OpenTofu

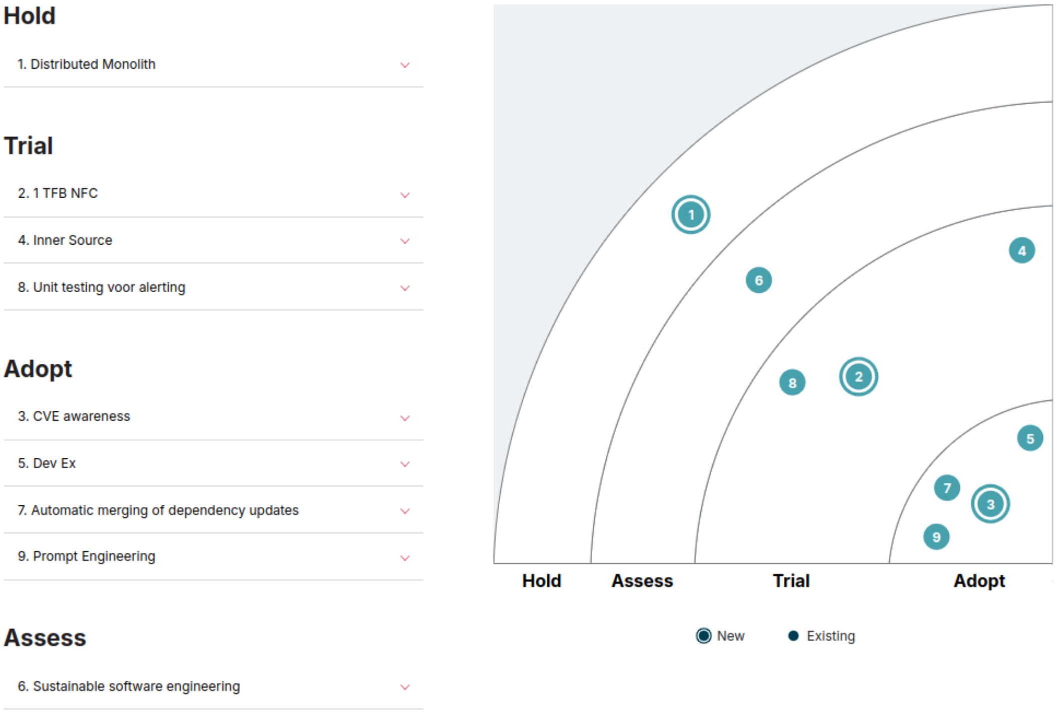
HOLD

HashiCorp has adjusted the source code license of its products, like Terraform, to a Business Source License [https://www.hashicorp.com/bsl]. This has led to concern among Terraform users, who fear the ecosystem of tools will suffer from this. On August 23rd 2023 a fork of the Terraform source code was made, and dubbed OpenTofu. The goal is for this to continue existing as an open source project under control of the Linux Foundation.

However, closer inspection of HashiCorp's explanation [https://www.hashicorp.com/license-faq] of their license shows, that this change to a BSL only affects companies that offer commercial products that use HashiCorp products to compete with HashiCorp products. For the overwhelming majority of Terraform users, that is not the case. We therefore see no reason for them to switch to a Terraform alternative like OpenTofu, which is why we put it on *Hold*.

In April 2024, IBM announced [https://www.prnewswire.com/news-releases/ibm-to-acquire-hashicorp-inc-creating-a-comprehensive-end-to-end-hybrid-cloud-platform-302126646.html] that HashiCorp will continue under the umbrella of IBM by the end of the year. Although this may cause interest in OpenTofu to spike temporarily, we do not anticipate any hindrances for existing Terraform users as a consequence of this acquisition.

Techniques



Hold

1. Distributed Monolith

Trial

2. 1 TFB NFC

4. Inner Source

8. Unit testing voor alerting

Adopt

3. CVE awareness

5. Dev Ex

7. Automatic merging of dependency updates

9. Prompt Engineering

Assess

6. Sustainable software engineering

Automatic merging of dependency updates

ADOPT

Automatic merging of dependency updates is becoming increasingly important these days. Nowadays, it is increasingly common to discover vulnerabilities associated with dependencies in projects. We believe that automatically updating and merging the relevant dependencies is the right way to keep code as up-to-date as possible.

Some tools that enable automatic merging of dependency updates are Renovate [https://github.com/renovatebot/renovate] and Dependabot [https://github.com/dependabot/dependabot-core]. What these essentially do is continuously scan dependencies in the project and create merge requests for them when updates are available. Developers can manually or automatically merge these merge requests.

JDriven chooses to include this technique in *Adopt* quadrant of the Tech Radar because it lets teams work efficiently by (partially) automating the tracking of versions of dependencies. This leads to quality and secure software.

Commit. Develop. Share.

CVE awareness

ADOPT

Within the world of software engineering, MITRE [https://www.mitre.org/focus-areas/cybersecurity] is mostly known for their database of Common Vulnerabilities and Exposures (CVE) [https://www.cve.org/] that they maintain and make public. The MITRE organization has a history with the American department of defense. Many development teams by now outfit their CI/CD pipeline with automated tools for scanning dependencies to see if any CVEs have been found in those.

This is a good practice. Unfortunately, we've observed a lack of awareness among developers of what these tools do, and what findings mean. When notified of a CVE by a tool like OWASP DependencyCheck [https://owasp.org/www-project-dependency-check/], it is still up to the developer to analyse the finding, to determine its impact and decide on possible mitigations.

There are roughly four ways this can go:

1. Regardless of whether the CVE notification is justified or not, there is a simple solution by updating a possibly vulnerable dependency to a newer safe version without any compatibility issues.
2. The CVE finding can be marked as a *false positive*, suppressing future notifications.
3. The CVE finding is correct, but an upgrade to a newer safe version breaks compatibility with existing software and requires a significant investment to apply.
4. The CVE finding is correct, but there is no newer safe version available of the vulnerable dependency, for example because it's no longer actively maintained.

In practice, most developers can be convinced to structurally apply option 1. Option 2 requires further investigation and should only be applied if the CVE indeed does not affect the software (often, a CVE pertains to a specific use of a library; not to all uses). Developers should resist the temptation of marking a CVE finding as a false positive just to stop nagging notifications. That temptation exists, because options 3 and 4 require a significantly higher investment. It may require reimbursing technical debt before upgrading, dropping a dependency altogether, or investing in forking or contributing to maintenance of the vulnerable dependency.

This is why JDriven recommends developers take the time to seriously understand CVEs, their impact, and possible mitigations. Be aware of technical debt, share knowledge, involve security experts, and get acquainted with dependencies and their maintainers, so we together we can keep our software safe.

Dev Ex

ADOPT

DevEx, derived from Developer Experience, is a crucial aspect of software development that encompasses the overall experience of developers while working on a project or in a specific development environment. It focuses on the usability of development tools, efficient workflows, clear documentation, and the satisfaction and productivity of developers.

Within the context of software development, DevEx is the key to optimizing processes and promoting a positive working environment for developers. By ensuring intuitive tools, streamlined workflows, and effective support, it contributes to accelerating development cycles and delivering high-quality software products. It serves as the bridge connecting developers to the capabilities of technology and fosters innovation in the dynamic world of software development.

Prompt Engineering

ADOPT

Prompt engineering is a discipline that revolves around crafting precise, effective prompts for AI-powered models, for example GPT-X (Generative Pre-trained Transformer). These prompts guide the AI model's responses toward the desired outcome. It's a process that involves experimentation, iteration, verification and fine-tuning. Prompt engineering offers several benefits that have the potential to enhance software engineering:

- Accelerated Development
- Enhanced Creativity
- Consistency and Reliability
- Learning and Growth

Prompt engineering encourages a culture of continuous learning and growth. Developers can analyze model responses, adapt their prompts, and learn from their mistakes. Providing these insights to the AI model creators can result in optimized models that provide better and more robust results. While utilizing AI models can provide rapid outputs, achieving high-quality results necessitates an investment of time, effort, and the acquisition of new skills.

1 TFB NFC

TRIAL

1, TFB, NFC [https://estimation.lunarlogic.io#description] is a work estimation technique that tries to solve the issues surrounding more established planning techniques, such as using story points. The idea is that, as tasks (*backlog items* in Scrum) become more complex, time estimates will become more uncertain, often to a point where these estimates hold little value. Too many of these kind of estimates will make sprint plannings unreliable, and hamper steady progress for the team.

Instead, 1, TFB, NFC takes an opposite view to story points, without going so far as to drop estimations entirely. Each story either is *1 point*, *too frighteningly big (TFB)*, or *no faintest clue (NFC)*. The idea is that only stories in the first category can be worked on, as the scope and solution are clear enough and sufficiently small. At the end of the sprint, the team simply counts the number of items completed as a metric for velocity. Stories that are TFB need to be split up, and NFC-type stories should be explored first, using a spike or a small prototype.

We have decided to put the 1, TFB, NFC technique in the *trial* ring. It makes measuring velocity easier, as by definition all tasks should be reduced or split until they simply have work size 1. If teams spend a lot of time on refining, or are constantly missing sprint goals, this technique might help to set more realistic goals.

We do realize that such a classification might be too coarse and could hamper planning, however. Therefore, we encourage teams to explore this idea further, and see where this concept can aid in improving their planning routines.

Inner Source

TRIAL

Inner source refers to applying open source principles within an organization. It is about adopting the collaborative, transparent, and decentralized approach commonly seen in open source projects and applying it to the organization's internal software development. It is an alternative to closed source, where repositories are typically available only to the responsible team.

In an environment where inner source is applied, teams within the company have the ability to openly access code repositories. This promotes collaboration, enhances code analysis, provides the opportunity for contributing code changes via Pull Requests across different projects. It promotes greater transparency, encourages knowledge sharing, and enables developers to work across teams and departments.

This model can improve software quality, accelerate development, and foster a culture of collaboration and innovation within the organization, drawing upon the collective expertise and efforts.

JDriven stands for commit, develop and share with the open source principles in mind. Therefore, we think organisations should try opening up the source code for internal developers.

Unit testing voor alerting

TRIAL

In the realm of software observability and monitoring, establishing robust alert mechanisms is crucial. The unpredictability of events demands accurate and responsive alerting systems, often built upon complex rules. However, the true validation of these rules usually unfolds when real-world scenarios emerge unexpectedly.

Enter "unit testing for alerting," a technique designed to elevate the precision of these alerting rules by proactive evaluation and refinement. This approach empowers teams to thoroughly define and validate rules before they encounter live incidents, significantly strengthening confidence in the rules' efficacy.

The primary goal has a dual purpose: reducing false alarms and ensuring genuine issues are promptly and accurately highlighted. By simulating various scenarios and conditions, teams can assess how well their alerting rules perform, fine-tuning them for optimal responsiveness.

Key tools like Prometheus offer dedicated support for unit testing alerting rules. By preemptively validating alerting rules, we've observed a marked reduction in false positives, enabling swifter and more accurate responses to genuine issues.

JDriven thinks teams and organisations working with alerts, should start adding unit tests for alerts.

Sustainable software engineering

ASSESS

Green software is an emerging discipline at the intersection of climate science, software design, electricity markets, hardware, and data center design. Green software is carbon-efficient software, meaning it emits the least carbon possible. Only three activities reduce the carbon emissions of software; energy efficiency, carbon awareness, and hardware efficiency.

A software architecture that takes into consideration carbon efficiency is one where design and infrastructure choices have been made in order to minimize energy consumption and therefore carbon emissions. The measurement tooling and advice in this space is maturing, making it feasible for teams to consider carbon efficiency alongside other factors such as performance, scalability, financial cost and security.

The cloud now has a larger carbon footprint than the aviation sector. We believe that as software engineers and architects we also have a responsibility to reduce this footprint and improve the climate. By making conscious choices, we can also take these steps. In addition, efficient systems have an additional advantage: fewer resources means lower costs.

Distributed Monolith

HOLD

With the rise of cloud indrastructure and virtualization, developers and operations engineers have found a solution for the problems of their *Big Ball of Mud* monoliths. It allows them to cut up applications into microservices, which can offer several benefits over a monolith:

- With proper separation of functionality, every microservice serves a singular purpose, which lowers the *cognitive load*.
- Every part of an application can be individually changed and deployed, so that a functional change only affects a small part of the operational software.
- Infrastructure for parts of the application, that have more capricious performance requirements than other parts, can be scaled in isolation.
- Code repositories can be maintained by and transferred between teams.

However, microservices do have some downsides:

- Developers need to divide their attention between multiple moving parts in a delivery pipeline and software landscape.
- Interaction between parts is typically over HTTP, which presents additional demands for API management and security.

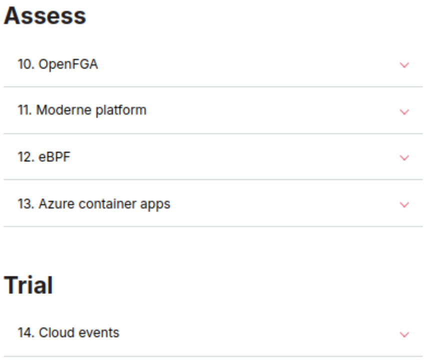
When adopting a microservices-architecture, the benefits won't automatically emerge. There is a risk, that the software succumbs to the downsides of both a *Big Ball of Mud* monolith and microservices, and becomes what is referred to as a *Distributed Monolith*.

There is a risk of ending up with a Distributed Monolith, when an organization decides to cut up an existing monolith into technical parts, for example because existing teams are divided based on technical expertise. A symptom of such a situation is that every functional change requires changes in parts managed by several teams. We call this anti-pattern *tight coupling*. A different situation, that we see occurring more and more often, is when architectural plans demand that every functionality should indiscriminately be built as a separate microservice. In such cases there is a risk, that even the smallest functional change requires a team to make changes in many small microservices that they maintain, which introduces extra work and an increased risk of making mistakes.

JDriven therefore advises companies to be careful when considering employing microservices. One should first and foremost strive for *low coupling*, *high cohesion*. In this context, the Single Responsibility Principle [https://en.wikipedia.org/wiki/Single_responsibility_principle] can come in handy: "Things that change for the same reasons, should be together". A software design method like *DDD*, which starts with finding distinctions in the functional domain, and then modelling that to *bounded contexts* and *context maps*, can further help to make the right cut. And if all above-mentioned possible benefits of microservices aren't desired, it's worth considering starting with building a *modular monolith* instead of microservices.

Commit. Develop. Share.

Platforms

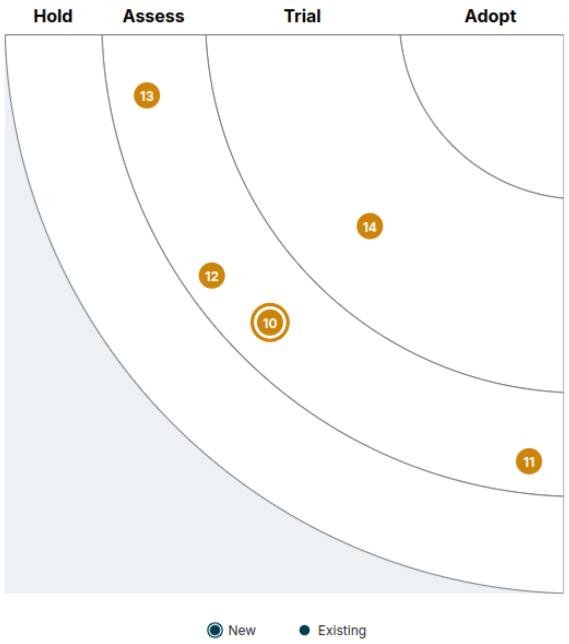


Assess

- 10. OpenFGA
- 11. Moderne platform
- 12. eBPF
- 13. Azure container apps

Trial

- 14. Cloud events



Cloud events

TRIAL

Cloud Events [https://cloudevents.io/] is an open standard developed to enhance compatibility and collaboration between event-driven systems in the cloud. It provides a unified way to specify events, regardless of the cloud platforms, services, or systems involved. This standardization enables organizations to describe events in a format understandable and processable by various cloud providers, reducing the complexity of event processing between systems.

Cloud Events are based on a set of metadata and standard event formats described in JSON or YAML. The metadata of Cloud Events offers a consistent and structured method to capture information about events, including details such as event identification, timestamp, and event source. Additionally, Cloud Events provide the capability to include optional metadata fields alongside the standard metadata fields.

Cloud Events offers flexibility and portability by providing a common language for describing events, irrespective of the underlying technologies, protocols and infrastructure. The benefits of implementing Cloud Events includes enhanced interoperability, increased flexibility, and simplified integration between various cloud services and applications. This enables developers to build event-driven architectures that are easily scalable, flexible, and interoperable across different cloud environments.

JDriven puts Cloud Events in the *Trial* quadrant, because we believe it is worth exploring, especially for large enterprise environments that benefit from a more comprehensive standard.

Azure container apps

ASSESS

Azure Container Apps is a fully managed service offered by Microsoft Azure that simplifies the deployment and scaling of containerized applications in a serverless environment. It is positioned between Azure App Service, which is mainly focused on simple web applications, and Azure Kubernetes Service, which is a managed Kubernetes platform.

Container Apps is built on Kubernetes, KEDA and Dapr. Because of this, it enables event-driven application architectures by supporting advanced autoscaling (including scale to zero and traffic splitting), support for long-running processes and background tasks. There is no direct access to the Kubernetes APIs available.

We think Azure Container Apps is a good alternative for teams who have outgrown Azure App Service but do not want or need to manage their own Kubernetes cluster. That's why we think you should *Assess* it when it fits your needs.

eBPF

ASSESS

eBPF (extended Berkeley Packet Filter) is a technique to run sandboxed programs inside the Linux kernel using a Just-in-Time (JIT) compiler. This allows users to write programs that filter and observe system and network behaviour, or provide additional security measurements, to run with elevated permissions. At the same time the JIT provides the safety and system security boundary limitations associated with programs running in user space. Additionally, eBPF programs need to satisfy certain liveness and safety criteria which the loader will verify before execution. Traditionally, this elevated functionality is either provided by patching the Linux kernel or writing custom kernel modules. This is a cumbersome procedure that is not without dangers to the entire operating system stability. As the in-kernel API is subject to frequent and regular change, such patches typically only work for a limited range of kernel versions. They require both in-depth knowledge of the kernel and constant maintenance.

We have selected this technique for *Assess*, since it can solve a narrow class of problems. These are mainly of specific interest in organisations that operate a lot of infrastructure or have special security requirements, and require specialist knowledge to write the programs. A user account needs to have specific elevated system capabilities to run eBPF modules as well as compiled-in kernel support. Hence, this functionality will typically only be available on systems that are under the control of the client, as public cloud vendors might restrict access to this functionality.

Moderne platform

ASSESS

The Moderne Platform is a SaaS solution built on top of the open-source software OpenRewrite. It offers automated code refactoring and remediation tools that allow developers to quickly and effortlessly migrate complete codebases with a simple click or CLI command. It achieves this using opinionated recipes that, for example, upgrade Spring Boot apps from version 2 to 3.

In our Tech Radar, we have previously mentioned OpenRewrite. Having a SaaS platform that combines the strengths of OpenRewrite with tooling to enhance the developer experience around using OpenRewrite is something we view positively. Competitors such as Snyk, Dependabot, and Sonar only warn developers about software vulnerabilities and findings. Particularly interesting to us is the fact that the Moderne Platform immediately remediates such issues, freeing up developer time.

Due to our limited experience with the Moderne Platform, we're uncertain about its suitability for mission-critical applications. When not using the CLI, a Moderne agent connects to in-house code and artifact repositories, which may pose a hurdle for some organizations. Furthermore, the effectiveness of the platform hinges on the number of recipes and language support it has, an aspect they are currently working hard to expand.

OpenFGA

ASSESS

OpenFGA [https://openfga.dev/] is an authorization solution that offers a flexible approach to access control. OpenFGA supports standard role based acccess control (RBAC), but more interestingly also has the capablity for fine-grained access control.

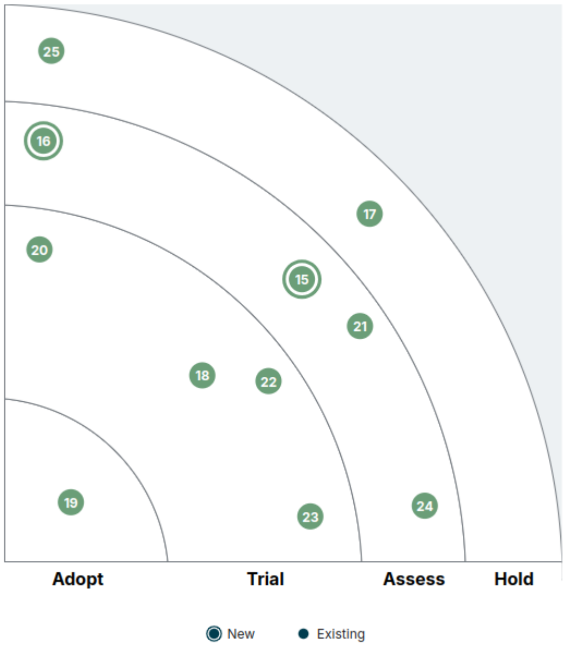
OpenFGA makes use of a relationship-based access control (ReBAC) model, allowing authorization definitions to be based on a user's relationship with a resource, not just their role. This enables fine-grained authorization, which offers a major advantage over RBAC which can become cumbersome for intricate scenarios. These relationship models can be deployed automatically and managed through code instead of manually. The modeling language that is use is expressive enough to handle a wide range of authorization needs while supposedly remaining understandable for non-programmers. However, OpenFGA's rule modeling language introduces a new concept for developers, which could result in a learning curve when switching to OpenFGA's approach.

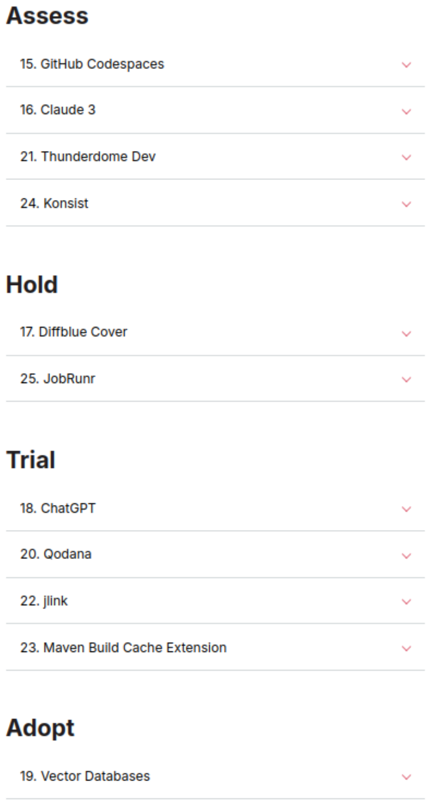
Due to its backing by Cloud Native Foundation it has good support and multiple SDKs for integration. While actively maintained, there might be less documentation or community support available due to it being a relatively new project compared to some established access control solutions.

OpenFGA was designed for performance and to efficiently handle authorization checks for large-scale applications with many users and resources. Implementing OpenFGA adds another layer to the application architecture and it is unknown how its performace compaires with tradtional access control platforms. If the authorization needs are simple, the added complexity might outweigh the benefits.

The combination of a new autorization concept as well as a comprehensive platform warrants OpenFGA being in the *assess* ring.

Tools





Assess

15. GitHub Codespaces
16. Claude 3
21. Thunderdome Dev
24. Konsist

Hold

17. Diffblue Cover
25. JobRunr

Trial

18. ChatGPT
20. Qodana
22. jlink
23. Maven Build Cache Extension

Adopt

19. Vector Databases

Vector Databases

ADOPT

Vector databases are specialized databases that efficiently store, index, and retrieve high-dimensional vectors. These vectors are used for semantically searching text, images, and more, enabling applications like ChatGPT and GitHub Copilot to function.

At JDriven, we believe that companies should consider adopting vector databases when exploring the possibilities of semantic search or venturing into the AI space.

While there are numerous vector database vendors available, based on our experience, most of them are similar, with no big differentiating factor. Hence, we have chosen not to list each of these databases separately in our Tech Radar. The most popular databases include Pinecone, Chroma, pgVector, and Weaviate. Additionally, companies such as Elasticsearch, Redis, and Neo4j have incorporated vector capabilities into their toolsets.

Commit. Develop. Share.

ChatGPT

TRIAL

A well-known language model capable of having complex conversations with its users, is now more than a year old. ChatGPT can help developers solve problems, refactor code, detect bugs, perform static code analysis and much more.

However, we cannot rely on ChatGPT blindly. It is important that due diligence is exercised for a couple of reasons. First and foremost, code will be presented with confidence by the tool, even if it contains bugs, is unoptimized or uses code constructs for which better alternatives exist. The free version has served generated code containing divide by zero problems, n+1 selects problems and refactors which reduced the quality of the code. The generated code by the paid version is of higher quality, but it's still not spotless.

Other than functional problems, the tool can pose a security or privacy risk. Chat history is logged and fed to the model. Therefore, it is important to only expose code which is generic of nature, and does not contain company-specific details or sensitive information. There is no guarantee your data is safe.

All in all, ChatGPT is a useful and capable tool, especially for learning a new language, concept or framework. Use it to get pointers on how to solve problems, but do not rely on the solutions without knowing exactly what the generated code does. Use it for refactoring, but keep in mind that it might not apply fitting patterns or approaches.

jlink

TRIAL

jlink is a tool that lets you build a custom Java runtime environment (JRE) tailored to your application. It does this by removing components of the JRE that your application does not require, such as superfluous Java modules, documentation and peripheral tooling. The benefits of a tailored JRE are twofold; it results in a smaller size of the deployable payload (e.g. Docker images), and reduces the potential attack surface. The smaller size leads to faster deploy times and possibly reduced costs for hosting images. By excluding modules that are superfluous, a malicious party cannot exploit vulnerabilities in these modules. In the worst case, these could lead to privilege escalation and data breaches.

We put jlink under *Trial*, as there are clear benefits to be reaped. As with all tools, however, there are also some drawbacks. First of all, while the accompanying jdeps tool can scan which modules are required for a given application archive, in our experience it often still takes some manual tinkering to include all the correct modules. Worse, missing modules often lead to cryptic error messages or changed application behaviour with no tell-tale sign of the underlying problem. This makes custom JREs a bit of a risk that not all environments are willing to tolerate.

Maven Build Cache Extension

TRIAL

Maven Build Cache [https://maven.apache.org/extensions/maven-build-cache-extension/] is an extension for Maven (available from version 3.9) that makes your Maven builds more efficient by applying smart caching mechanisms (comparable to the Gradle build cache).

At several moments during the build process, the build cache will place different outputs, like generated code or compiled classes, into a cache. If at a later stage the build is executed again with (partially) unmodified inputs, the earlier cached outputs can be reused. This way, parts of the build, like compiling code and executing tests, can be skipped, which decreases the overall build time. An additional benefit is that build caches can be shared by using a remote cache.

Introducing the Maven Build Cache extension will in many cases significantly reduce the build times of your Maven projects. Because it is a relatively new Maven feature, it may not yet interact properly with all third-party Maven plugins. Additionally, every Maven project is different, so it will take some research time to configure the build cache properly. A wrongly configured build cache could potentially lead to unstable, non-reproducible builds. Therefore, we place it in the *Trial* quadrant.

Qodana

TRIAL

Qodana is JetBrains' Static Code Analysis Tool. It is bundled with IntelliJ IDE, but has to be activated separately. Besides IntelliJ, it has many available CI/CD and IDE integrations and supports Java, Kotlin, JavaScript and TypeScript code, among other languages. A Qodana Cloud dashboard is available for giving an overview of individually scanned projects.

Qodana was released in 2023, after having been available in preview mode for a couple of years. There is a free community model, but for JavaScript, TypeScript and Spring support, you'll have to opt for a paid model. The pricing, however, is also what makes it interesting when compared to SonarQube, Qodana doesn't charge or limit you for your amount of lines of code (LoC), but for the amount of active contributors. Next to that CI/CD integration is part of the free version of Qodana. All this means that, depending on the nature of your project and how it is maintained, that Qodana could turn out to be a lot cheaper in use. It is worth making a comparison and trying out the tool.

Claude 3

ASSESS

Claude 3 [https://en.wikipedia.org/wiki/Claude_(language_model)] is a foundational AI service from Antrophic, an offspring of OpenAI, with a strong focus on user safety. The main driver behind their model is Constitutional AI. This system tries to optimize helpfulness and minimize harmfulness in responses. Constitutional AI uses a set of rules to prevent harmful responses that are used to train a metamodel. The metamodel is then used to interfere with the training of the regular model, so that it learns not to generate content that is considered harmful. The metamodel does this by critiquing the regular model's output according to its constitution iteratively, until the outcome contains no more violations. To validate, the model is also checked against a large corpus of malicious input prompts.

We choose to put the use of Claude in the *assess* ring. Currently, the web UI cannot be used from inside the EU, making it difficult to easily experiment with the technology. Also, the choice of its constitutional rules, while overall sensible, still introduce a bias in the type of responses it generates. These responses, while universal, might not always align with specific cultural patterns. Furthermore, there is a fine line between harmlessness and evasiveness in responses. In Anthrophic's research papers [https://www.anthropic.com/research], this seemed to be handled reasonably well, but users are advised to draw their own conclusion.

GitHub Codespaces

ASSESS

GitHub Codespaces [https://github.com/features/codespaces] is a Cloud Development Environment (CDE) that provides virtual machines for developers to work on. It is built on top of devcontainers [https://containers.dev/], which uses a Docker image that contains the required development toolchain. Developers can use their own local IDE, or a web-based variant. It prevents reproducibility issues between developers because of diverging developer environments. Using a common toolchain, configuration and platform, onboarding new team members should become less time-consuming.

GitHub Codespaces has first-class support for Visual Studio Code, but support for the JetBrains IDEs is still lagging, and requires JetBrains Gateway [https://www.jetbrains.com/remote-development/gateway/].

We place GitHub Codespaces in the *assess* ring, as it augments, rather than replaces current work processes. That makes trying it out a low-risk exercise. We believe that standardizing development environments can aid in developer productivity. A minor drawback is that a stable internet connection is required for a satisfying developer experience.

As it is a GitHub service, it works best with code repositories already hosted on GitHub itself. There are also costs associated with using the virtual machine time, as well as storage costs for a codespace while it exists. Additionally, there might be difficulties with integrating on-premise services for e.g. testing purposes, requiring a VPN to do so.

Lastly, there are also similar offerings from other companies, that might provide a better match for certain use cases.

Konsist

ASSESS

Konsist [https://docs.konsist.lemonappdev.com/getting-started/readme] is a static code analyzer library for Kotlin that helps to ensure coding conventions and can check defined architectural layer boundaries. Checks are executed in unit tests and as such can run during pull request reviews to safeguard a project's architecture. Konsist supports Kotlin projects, including Android, Spring, and Kotlin Multiplatform whereas ArchUnit, a similar tool, only supports (JVM) byte-code analysis.

We at JDriven believe in code quality and clear defined software architecture. Konsist is very useful tool in our toolbox to help formalize code conventions and architectural constraints, and verify them. Since Konsist is still in its infancy, we keep an eye out for its development before we endorse to adopt it.

Thunderdome Dev

ASSESS

Thunderdome.dev [https://thunderdome.dev/] is an open source collaboration tool. One of its distinguishing features is as an online planning poker tool. It can also be applied for other processes, such as retrospectives, team checkins, and feature/story mapping. The process of planning poker works well in a context where a team collaborates in person. In a hybrid collaboration context, this is not always the case. Collaborators will have to adjust their planning poker process around this context. Doing planning poker with Thunderdome.dev provides a basic structure that simplifies these adjustments.

On the one hand, it makes for an accessible tool, on the other hand, effective application of this tool strongly depends on a Development Team's Way of Working and its subjective needs. This is why we place Thunderdome.dev in *Assess*. Consider all of this tool's features when making your assessment. In particular, consider the possibility of using it alongside, or even instead of established tools like Jira.

Diffblue Cover

HOLD

Diffblue Cover [https://www.diffblue.com/products/], an automated unit test generation tool for Java code, demonstrates strengths in understanding the code being tested and creating appropriate mocks. However, its ability to generate complex mock data is limited, and its opinionated approach to test generation may not align with existing project styles.

While Diffblue Cover can be useful for generating boilerplate code and setting up initial tests, it cannot replace manual testing entirely. Additionally, its compatibility is restricted to specific versions of Spring, Java, and JUnit, and it struggles with methods that have no parameters or classes without constructors. Furthermore, Diffblue Cover cannot be run on an entire project or even on packages, limiting its applicability.

Therefore, Diffblue Cover is placed in the *Hold* quadrant of this edition of our Tech Radar.

JobRunr

HOLD

JobRunr [https://www.jobrunr.io/] is a Java library for running and managing background processes. It allows you to schedule jobs, run batches, or create workflows using Java Lambda functions, with builtin retry mechanisms for failed jobs. Its builtin web interface enables you to monitor, control, and debug your jobs. Although the free version is very interesting and should be considered for usage, we put JobRunr on *Hold* because of the pro version. As JobRunr Pro is not open source and only has a few people working on it, that poses some risk that your organisation might not want to take. Therefore, we would advise to proceed with caution.

Commit. Develop. Share.

Commit.
Develop.
Share.