LZfD-Technologieradar

|  |  |
| --- | --- |
| Dokumentenverantwortlicher | EDV 639 |
| Angestrebter Revisionszyklus | Kontinuierlich |
| Bearbeitungsstatus | Entwurf |
| Version | 0.0.1 |
| Vertraulichkeitsklasse | TLP:GREEN |
| Dokumentenart | Wählen Sie ein Element aus. |

Abkürzungsverzeichnis

|  |  |
| --- | --- |
| Abkürzung | Erläuterung |
| CPSA | Certified Professional for Softwarearchitecture |
| CNCF | Cloud Native Computing Foundation |
| DB | Datenbank |
| DOM | Document Object Model |
| DSL | Domain Specific Language |
| IDE | Integrated Development Environment |
| ISAQB | International Software Architecture Quality Board |
| JDBC | Java Database Connectivity |
| JSON | JavaScript Object Notation |
| JVM | Java Virtuelle Maschine |
| LZfD | Landeszentrum für Datenverarbeitung bei der Oberfinanzdirektion Karlsruhe |
| OWASP | Open Web Application Security Project |
| SLES | Suse Linux Enterprise Server |
| SSH | Secure Shell |
| UI | User Interface |
| VCS | Version Control System |
| VM | Virtuelle Maschine |
| WSDL | Web Services Description Language |
| YAML | YAML Ain’t Markup Language |
| ZAK | Zielarchitektur KONSENS |

Begriffsverzeichnis

Für die Dokumente des ISMS des LZfD wird im LZfD-Intranet auf den Seiten des Informationssicherheitsbeauftragten ein zentrales Begriffsverzeichnis (Glossar) geführt. Diese Begriffsbestimmungen sind verbindlich.

Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

Dokumentenhistorie

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Vorgenommene Änderung | Bearbeiter / Autor |
| 0.01 | 07.08.2020 | Erstellung | Steinbauer |
|  |  |  |  |

Inhaltsverzeichnis

[1 Einleitung 3](#_Toc47710016)

[1.1 Zusammenfassung 3](#_Toc47710017)

[1.2 Ziele 3](#_Toc47710018)

[1.3 Gliederung 4](#_Toc47710019)

[1.4 Aktueller Stand und nächste Schritte 4](#_Toc47710020)

[2 Betrieb und Entwicklungsumgebung 5](#_Toc47710021)

[2.1 Konzepte und Techniken 5](#_Toc47710022)

[2.2 Plattformen 8](#_Toc47710023)

[2.3 Werkzeuge und Libraries 9](#_Toc47710024)

[3 Entwicklung 10](#_Toc47710025)

[3.1 Konzepte und Techniken 10](#_Toc47710026)

[3.2 Sprachen und Frameworks 13](#_Toc47710027)

[3.3 Werkzeuge und Libraries 16](#_Toc47710028)

[4 Requirements Engineering 18](#_Toc47710029)

[4.1 Konzepte und Techniken 18](#_Toc47710030)

# Einleitung

## Überblick

Der LZfD-Technologieradar benennt neue und bereits am LZfD eingesetzte Technologien und ordnet deren Relevanz für die Weiterentwicklung des LZfDs ein. Er orientiert sich dabei grob an der Struktur des Technology Radars der Firma ThoughtWorks.

Der LZfD-Technologieradar fokusiert sich aktuell auf die Themen Entwicklung und Entwicklungsumgebung. Zukünftig sollen die Themen Entwicklung, Client und Data Center abgedeckt werden.

Der LZfD-Technologieradar erhebt keinen Anspruch auf Vollständigkeit. Es handelt sich um eine Momentaufnahme.

## Ziele

Ziele des LZfD-Technologieradars sind:

* Identifikation und gemeinsame Erforschung neuer für das LZfD relevanter Technologien.
* Wissens- und Erfahrungsaustausch zwischen den einzelnen Bereichen.
* Entwicklung eines gemeinsamen Technologiestacks durch Vermeidung unbeabsichtigter Parallelentwicklungen. (Keine Standardisierung! Bewusste Parallelentwicklungen können sinnvoll sein.)
* Unterstützung des Innovations-Managements. (Der LZfD-Technologieradar kann hier nur eine temporäre Lösung sein. Benötigt wird hier ein zukunftsorientierter Emerging Technologies Radar mit breiterem Fokus wie ihn z.B. die Firma Gardner erstellt.)

## Gliederung

Technologien werden in die folgenden drei Phasen eingeordnet:

* Beobachtungs-Phase:

Technologien, deren Entwicklung am LZD beobachtet wird, da sie zukünftig relevant werden könnten.

* Evaluations-Phase:

Technologien, deren Einsatz derzeit aktiv am LZfD geprüft wird.

* Nutzungs-Phase:

Technologien, die derzeit aktiv eingesetzt werden.

Das Dokument ordnet die Technologien analog zum Thought Works Technology Radar zusätzlich in die folgenden Kategorien ein:

* Konzepte und Techniken
* Plattformen
* Sprachen und Frameworks
* Werkzeuge und Libraries

Auf oberster Ebene werden die Technologien schlussendlich thematisch in die folgenden Kategorien einsortiert:

* Betrieb und Entwicklungsumgebung
* Entwicklung
* Requirements Engineering

Insgesamt ergibt sich die folgende Gliederung:

* Thema
  + Typ
    - Phase

Die Kategorien geben eine grobe Orientierung und sind nicht trennscharf.

## Aktueller Stand und nächste Schritte

In der aktuellen Fassung gibt der LZfD-Technologieradar ausschließlich die gegenwärtige Meinung des Autors sowie des Verfahrens GDA wieder.

In der Sitzung des LZfD-ArcBoards vom 06.07.2020 wurden die Mitglieder gebeten Kollegen zu benennen, die an einer Mitarbeit interessiert sind, damit zukünftig die Bereiche Entwicklung, Client und Data Center abgedeckt werden können.

Im Rahmen einer Ausbildungsaufgabe wird aktuell die Umsetzung des LZfD-Technologieradars als Webanwendung geprüft. Dadurch soll die zukünftige Pflege und Übersichtlichkeit erleichtert werden.

# Betrieb und Entwicklungsumgebung

## Konzepte und Techniken

### Beobachtungs-Phase

#### Chat Ops

* Chat Bots (z.B. Hubot) werden in den Team-Chat (z.B. Rocket-Chat) integriert und können dann genutzt werden, um mit der Entwicklungsumgebung und anderen Infrastrukturkomponenten zu kommunizieren.
* Z.B. kann das Deployment eines neuen Releases auf einen der Server durch eine entsprechende Nachricht an einen Chat Bot getriggert werden. Das Ergebnis des Deployments schreibt der Chat Bot wiederum in den Team-Chat. Tritt ein Fehler auf kann eines der Teammitglieder direkt darauf reagieren, bzw. das Problem transparent für das ganze Team diskutieren.

#### Configuration as Code

* Anstatt Anwendungen wie z.B. den Build-Server mühsam über die Weboberfläche zu konfigurieren, werden diese über im VCS versionierte Konfigurationsdateien konfiguriert.
* Zum einen sind Änderungen an der Konfiguration dadurch nachvollziehbar. Zum anderen können Konfigurationen schneller geändert und wiederverwendet werden.

#### GitOps

* Kombination aus Infrastructure as Code und Continuous Delivery, die häufig im Kubernetes-Umfeld zum Einsatz kommt.
  + Git wird als Single Source of Truth für das gesamte System verwendet.
  + Änderungen am System sind ausschließlich über Pull Requests möglich.
  + Nach dem Mergen eines Pull Requests wird das System automatisch aktualisiert.
* Vorteile: Änderungen am System sind im Nachhinein nachvollziehbar, können im Rahmen einer Continuous Delivery Pipeline getestet werden und jederzeit schnell zurückgerollt werden.

#### Inner Source

* Entwicklungsmethode bei der innerhalb einer Organisation bewährte Praktiken aus großen Open Source Projekten genutzt werden, um eine Art interne Open Source Community aufzubauen.
* Schlüsselpraktiken sind ein offener Zugang zu allen Artefakten (Code, Dokumentation und Issue Tracker), eine offene Kommunikation an der sich alle Interessierte beteiligen können und eine Trennung zwischen contributor und Committer durch Code Reviews.

#### Service Mesh

* Ein Service Mesh übernimmt in einer Microservice Umgebung Dienste wie z.B. Monitoring, Tracing und die Erhöhung der Resilienz durch Circuit Breaker.
* Realisiert wird das Service Mesh transparent für die Microservices durch das Sidecar-Pattern. Jedem Microservice wird ein Sidecar in Form eines Service Proxies zur Seite gestellt. Jegliche Kommunikation des Microservices läuft dann über den Service Proxy.
* Das Service Mesh besteht aus der Data Plane - den dezentralen Service Proxies - und einer zentralen Control Plane, die die gesammelten Daten verarbeitet und die Service Proxies konfiguriert.
* Das Service Mesh entbindet die einzelnen Microservices davon die genannten Dienste selbst zu implementieren. Ohne Service Mesh sind Bibliotheken nötig, die von allen Microservices einheitlich genutzt werden müssen und die technologische Unabhängigkeit und Weiterentwicklung behindern.
* Die Service Proxies erhöhen die Latenz.
* Ein Service Mesh erfordert Konfigurationsaufwand und besitzt eine hohe Lernkurve.

#### Site Reliability Engineering

* Die Grundidee hinter Site Reliability Engineering ist es Softwareentwicklungs-Methoden auf Aspekte des Betriebs und der Infrastrukturprovisionierung anzuwenden.
* Ziel ist die Bereitstellung von Systemen mit angemessener Zuverlässigkeit.
* Entwickelt wurde das Konzept von Google.

### Evaluations-Phase

#### Chaos Engineering

* Im Rahmen kontrollierter Experimente wird Chaos (z.B. Slow Responses oder Ausfälle von Subsystemen) in das zu entwickelnde System injiziert.
* Durch die regelmäßigen zum Teil nicht vorher angekündigten Experimente wird nach und nach die Resilienz des Systems erhöht.
* Chaos Engineering wurde im Rahmen einer Bachelor-Arbeit bei der GDA evaluiert.

#### Cloud Native

* Anwendungen, die von Beginn an dafür ausgelegt sind in der Cloud zu laufen
* Cloud Native Anwendungen bestehen häufig aus einer Reihe von Microservices, die in Container laufen, mit Kubernetes orchestriert und per DevOps und GitOps verwaltet und deployed werden.

#### DevOps

* Sammlung von Prinzipien und Praktiken, die der AE, dem Betrieb, dem Sicherheitsteam und dem Test helfen, gemeinsam einen Value Stream aufzubauen, durch den
  + neue Features schnell umgesetzt und in Produktion gebracht werden können.
  + ein sicherer und stabiler Betrieb gewährleistet wird und sich die Qualität der Software stetig verbessert.
  + die Prozesse, Werkzeuge und Fähigkeiten aller Beteiligten sich durch kontinuierliches Lernen stetig weiterentwickeln.
* DevOps und funktional organisierte Organisationen sind kein Widerspruch. Cross-funktionale Teams nicht zwingend erforderlich.
* Security muss ein integraler Bestandteil des DevOps-Ansatzes sein. Dies wird durch den Begriff DevSecOps verdeutlicht. DevSecOps verfolgt zwei Ziele:
  + Erhöhung der Security durch geeignete Maßnahmen in allen Phasen des Value Streams.
  + Nahtlose Integration ohne den Value Stream aufzuhalten oder zu verlangsamen.

#### Security by Design

* Security wird als explizite Anforderung von Beginn an (Shift Left) in allen Phasen des Entwicklungsprozesses berücksichtigt.

### Nutzungs-Phase

#### Continuous Delivery (CD)

* Kontinuierliche Bereitstellung neuer Releases im Rahmen einer Continuous Delivery Pipeline
* Release Kandidaten durchlaufen hochgradig automatisiert die einzelnen Test-Stages der Continuous Delivery Pipeline.
* Continuous Delivery führt zu schnellerer Bereitstellung neuer Features, geringeres Risiko durch die kontinuierliche Bereitstellung vieler kleiner Änderungen, höhere Qualität durch automatisierte Tests und schnelle Bereitstellung von Bug-Fixes, mehr Sicherheit usw..

#### Pair-Programming

* Arbeitstechnik aus der agilen Software-Entwicklung zur Steigerung der Softwarequalität und zur Verbreitung von Wissen.
* Bei der Erstellung von Quellcode arbeiten zwei Entwickler gleichzeitig an einem PC. Einer schreibt Code und der andere kontrolliert diesen und denkt über die Problemstellung nach. Die Rollen werden regelmäßig getauscht.

#### Scrum

* Scrum ist ein agiles Vorgehensmodell für die effektive Zusammenarbeit im Team.

#### Vorgehensmuster für Softwarearchitektur

* Pattern-Katalog zur Architekturarbeit in agilen Projekten nach einem Buch von Stefan Toth, bzw. einem Modul des International Software Architecture Quality Board (ISAQB) Certified Professional for Software Architecture (CPSA) Advanced Level Lehrplans.

## Plattformen

### Beobachtungs-Phase

#### Eclipse Che

* Kombination aus webbasierter IDE und Workspace-Server
* Die einzelnen Komponenten der IDE und die Workspaces wertden Remote in Containern ausgeführt.
* Entwickelt wird im Container (daher quasi Entwicklung auf Linux unter Windows möglich)
* Visual Studio Code-Extensions können genutzt werden.
* Auslieferung vorkonfigurierter IDE und Workspaces möglich.
* Schnelles Onboarding neuer Entwickler möglich

#### EFK Stack

* Der EFK Stack ist eine im Kubernetes-Umfeld häufig eingesetzte Alternative des populären ELK Stacks. Der ELK Stack besteht aus den Komponenten Elastic Search, Logstash und Kibana und ermöglicht ein verteiltes Logging in großen Systemen.
* Beim EFK-Stack wird Logstash durch Fluentd ersetzt.
* Vorteile von Fluentd sind neben der guten Kubernetes-Integration der geringe Hauptspeicherbedarf, die eingebaute persistente Message Queue und ein deklaratives auf Tags basierendes Routing von Events.

#### Knative

* Plattform, die auf Kubernetes aufbaut und das Deployment von Serverless-Architekturen ermöglicht.
* Knative besteht aus den beiden Unterprojekten Serving und Eventing.
* Knative erleichtert das Deployment, Upgrades, Routing, Autoskaling und die Verknüpfung unterschiedlicher Komponenten über Events. Zudem wird für eine hohe Observierbarkeit gesorgt.

#### Kubernetes

* Plattform für Container-Orchestrierung
* Wird unter dem Dach der CNCF entwickelt.

#### Istio

* Service Mesh
* Nutzt Prometheus, Grafana, Jaeger und Envoy.
* Die primär unterstützte Umgebung ist Kubernetes. Istio ist jedoch recht flexibel.

### Nutzungs-Phase

#### Docker

* Technologie zur Containerisierung von Anwendungen
* Eine containerisierte Anwendung ist isoliert vom Host-System und verhält sich in unterschiedlichen Umgebungen immer gleich.
* Verringert die Komplexität der Installation einer Anwendung auf ihre Konfiguration
* Geringerer Ressourcenverbrauch und performanter als VMs.
* Ermöglicht Technologieunabhängigkeit zwischen Anwendungen.

#### Podman

* Daemon-less Runtime für die Erstellung und Ausführung OCI-kompatibler Container.
* Erlaubt die Ausführung von Containern ohne Root-Rechte und erhöht so die Sicherheit. (Aktuell unter SLES 12 SP3 nicht möglich.)
* Netzwerk-Utilities scheinen noch nicht so gut entwickelt wie bei Docker (z.B. fehlende Out-Of-The-Box-Bridges)

## Werkzeuge und Libraries

### Beobachtungs-Phase

#### Ansible

* Werkzeug für das automatisierte Deployment von Anwendungen und Systemen.
* Auf den Remote Systemen müssen keine Agenten installiert werden. Eine SSH-Verbindung genügt.

#### Clair

* Statischer Vulnerability Scanner für Docker Container

#### HashiCorp Terraform

* Infrastructure as Code Werkzeug mit Fokus auf Cloud-Dienste.

#### Helm

* Paketmanager für die Installation von Anwendungen in Kubernetes.
* CNCF Technology Radar 2020-06: Adopt

#### Jaeger

* Verteilte Tracing Plattform
* Wird unter dem Dach der CNCF entwickelt.

### Evaluationsphase

#### Flux

* Kubernetes Operator für gitOps
* CNCF Technology Radar 2020-06: Adopt

#### Kustomize

* Kubernetes-natives Werkzeug für die Generierung unterschiedlicher Anwendungs-Konfigurationen.
* CNCF Technology Radar 2020-06: Trial

#### Tekton

* Framework für die Definition von Continuous Delivery Pipelines für Kubernetes.
* CNCF Technology Radar 2020-06: Assess

### Nutzungs-Phase

#### arc42

* Template für die Dokumentation einer Softwarearchitektur.

#### Grafana

* Dashboards und Visualisierungen für eine Vielzahl von Quellen wie z.B. Prometheus.

#### Prometheus

* Monitoring System und Time Series Database
* Wird unter dem Dach der CNCF entwickelt.

#### SonarQube

* Werkzeug für statische Code-Analyse zur Verbesserung der Code-Qualität und Sicherheit

# Entwicklung

## Konzepte und Techniken

### Beobachtungs-Phase

#### API Gateway

* Reverse Proxy, mit unterschiedlichen Funktionen wie z.B. Protokollübersetzung, Modellübersetzung, Fassaden für mehrere Microservices, Authorisierung, Rate Limiting, Caching usw..
* Einschätzung: Thought Works Technology Radar warnt vor überambitionierten API Gateways.

#### Blockchain

* Eine Blockchain ist eine bei mehreren Teilnehmern redundant gespeicherte Liste von Blöcken. Die Blöcke sind über kryptographische Verfahren miteinander verkettet. Neue Blöcke werden zwischen den Teilnehmern über ein verteiltes Konsensus-Verfahren ermittelt.
* Die Blöcke können z.B. Transaktionen bei der buchführung oder Audit-Logs enthalten.
* Durch die redundante Speicherung , die Verkettung der Blöcke über kryptographische Verfahren und das verteilte Konsensus-Verfahren ist eine nachträgliche Fälschung der bereits gespeicherten Blöcke nicht möglich.

#### Consumer-Driven Contract Testing

* Vorgehen zum Testen von Schnittstellen
* Consumer spezifiziert Contract in Form von Tests. Frameworks wie Pact ermöglichen zum einen die Ausführung der Tests in der CD-Pipeline des Consumers durch Bereitstellung eines Mocks. Zum anderen werden Tests gegen die tatsächliche Schnittstelle generiert, die in der CD-Pipeline des Providers genutzt werden können.
* Vorteile: Der Consumer erhält genau die benötigte Schnittstelle. Der Provider weiß exakt, welche Teile der Schnittstelle wie genutzt werden.

#### Künstliche Intelligenz (KI)

* Maschinelles Lernen, Deep Learning usw.
* Frameworks: Deeplearning4j, Keras, PyTorch, TensorFlow, scikit-learn usw.

#### Event-getriebene Architekturen

* Verbindet mehrere Softwarekomponenten über Ereignisse. Beim Publizieren von Ereignissen ist nicht bekannt wer diese lesen und verarbeiten wird.
* Führt zu sehr loser Kopplung der einzelnen Systemkomponenten.
* Führt zu hoher Komplexität des Gesamtsystems.
* Ereignisse können als Messages publiziert werden.

#### Micro Frontends

* Micro Frontends sind eine Alternative zu Architekturen mit Microservices im Backend, einem API Gateway und einem Frontend Monolithen. Zu jedem Microservice gibt es entsprechende Micro Frontends.
* Die Integration geschieht im einfachsten Fall über Links. Andere Alternativen sind serverseitige Templates, Build-time Integration, iFrames, Web Components oder eine Integration via Java Script.
* Neben der Integration besteht eine weitere Herausforderung in einem einheitlichen Look and Feel.

#### NoSQL Datenbanken

* Datenanken, die einen nicht-relationalen Ansatz verfolgen.

#### Progressive Web Apps (PWA)

* PWAs sind Web Apps, die wie native Apps auf dem Desktop oder mobilen Endgerät installiert werden können und ein vergleichbares Erlebnis bieten.
* Damit die PWA auf unterschiedlichen Endgeräten verwendet werden kann, ist ein responsives Design nötig.
* Herzstück einer PWA ist ein Service Worker. Durch den Service Worker kann JavaScript ausgeführt werden, selbst wenn die PWA nicht aktiv ist.
* Service Worker ermöglichen Offline Caching und Push Benachrichtigungen.
* Viele PWAs verfolgen einen Offline First Ansatz.
* Kommunikation über HTTPS ist Voraussetzung für die Nutzung von Service Worker.

#### Reaktive Programmierung

* Programmierparadigma für die Arbeit mit asynchronen Datenströmen.
* Ermöglicht nicht blockierende Server und Client Anwendungen.
* Reaktive Programmierung kann auf Implementierungsebene bei der Umsetzung Reaktiver Systeme eingesetzt werden.

### Evaluations-Phase

#### Message-getriebene Architekturen

* Komponenten des Systems kommunizieren über asynchrone Messages.
* Führt zu sehr loser Kopplung der Komponenten.
* Erhöht die Komplexität des Gesamtsystems.
* Unterschied zu Event-getriebenen Architekturen: Die Messages sind an einen Adressat gerichtet, Events nicht.
* Aktuell Evaluation im Rahmen einer Studienarbeit (Anbindung GeCo NaDien)

#### Reaktive Systeme

* Reaktive Systeme sind nach dem Reaktiven Manifest besser geeignet, den heutigen Ansprüchen an Software gerecht zu werden. Sie haben die folgenden Eigenschaften:
  + Responsiv: Das System antwortet zeitgerecht.
  + Resilient: Das System bleibt auch bei Hard- und Softwarefehlern antwortbereit und kehrt in einen gesunden Ausgangszustand zurück.
  + Elastisch: Das System reagiert auf unterschiedliche Lastanforderungen.
  + Message-Driven: Die Komponenten des Systems kommunizieren asynchron über Nachrichten.

### Nutzungs-Phase

#### Domain Driven Design (DDD)

* DDD ist eine Herangehensweise für die Modellierung komplexer Software. Dabei wird ausgehend von den zentralen Elementen der Problemdomäne das Lösungsmodell entwickelt.
* Wichtige Aspekte von DDD sind:
  + Entwicklung einer Ubiquitären Sprache: Um Missverständnisse zwischen fachlichen Stakeholdern und Entwicklern zu vermeiden müssen im allen Softwareartefakten exakt die gleichen Begriffe wie in der Problemdomäne verwendet werden.
  + Pattern-Katalog: Patterns wie Entities, Value Objects, Aggregates, Factories, Services und Repositories helfen bei der Modellierung.
  + Bounded Context und Strategic DDD: In komplexen Problemdomänen und Softwareprojekten ist die Erstellung eines einzigen allgemeingültigen Modells meist nicht zielführend. Besser ist die Entwicklung mehrerer Modelle, die in ihrem jeweiligen streng abgegrenzten Bounded Context gültig sind. Strategic DDD beschreibt Patterns für die Kommunikation zwischen Bounded Contexts. Dieser Aspekt von DDD ist besonders im Microservice-Umfeld relevant.

#### Evolutionäre Architekturen

* Architekturen müssen auf eine Vielzahl von Änderungen reagieren können:
  + Fachliche Anforderungen und ihre Prioritäten ändern sich.
  + Das Ökosystem, in dem Software entwickelt wird, verändert sich stetig durch neue Technologien und Konzepte.
* Evolutionäre Architekturen ermöglichen daher eine inkrementelle Weiterentwicklung in alle Dimensionen (Komponentenstruktur, Verteilung, Sicherheit usw.), ohne dass die geforderten Qualitätsmerkmale erodieren. Erreicht wird das durch
  + Continuous Delivery Praktiken,
  + eine möglichst lose Kopplung der einzelnen Komponenten
  + und Fitness Funktionen, die die Einhaltung der geforderten Qualitätsmerkmale bei jeder Änderung sicherstellen.

#### Feature Toggles

* Ermöglicht das Aktivieren und Deaktivieren von Features zur Laufzeit.
* Wird im Rahmen von Continuous Delivery verwendet, um noch nicht produktionsreife Features in Produktion komplett zu deaktivieren oder nur bestimmten Anwendergruppen zur Verfügung zu stellen.
* Einfache Feature Toggles werden über simple If-Anweisungen oder Conditional Spring Beans realisiert
* Es existieren aber inzwischen auch eine ganze Reihe von Feature Flag Management Systeme wie ConfigCat oder LaunchDarkly, die produktiv eingesetzt werden, um A/B-Tests oder Feature-Tests für bestimmte Benutzersegmente durchzuführen.

#### Funktionale Programmierung

* Programmierparadigma , das auf Funktionen ohne Seiteneffekte setzt.
* Seiteneffektfreie Funktionen liefern für die gleiche Eingabe immer das gleiche Ergebnis. Das erleichtert die Analyse und verhindert Fehler insbesondere in nebenläufigen Prozessen.
* Ein funktionaler Programmierstil führt zu einer höheren Wartbarkeit und weniger Fehlern. Das Programmierparadigma soll daher möglichst häufig genutzt werden.

## Sprachen und Frameworks

### Beobachtungs-Phase

#### BPMN und Workflow-Engines

* BPMN steht für Business Process Model and Notation
* BPMN ist eine grafische Spezifikationssprache zur Darstellung von Geschäftsprozessen und Arbeitsabläufen
* Die Notation umfasst im Wesentlichen Activities, Gateways, Events und Flows
* Workflow-Engines verarbeiten BPMN-Modelle und helfen so bei der Automatisierung von Prozessen und der Orchestrierung von Microservices
* BPMN 2.0 und Camunda als Workflow Engine sind in der KONSENS ZAK 2025 festgelegt.

#### Eclipse Microprofile

* Java Spezifikation, die auf Microservices ausgerichtet ist. Basiert auf der Jakarta EE Spezifikation, bzw. ergänzt diese.
* Ziel ist die Portabilität auf verschiedene Laufzeitumgebungen. Quarkus z.B. ist Microprofile kompatibel.

#### Go

* Programmiersprache mit Fokus auf einfachem und effizientem Code mit guter Lesbarkeit, hoher Performanz und guter Unterstützung für Nebenläufigkeit
* Unterstützt nicht klassenbasierte Objektorientierung (stattdessen werden Typerweiterungen benutzt)
* Intelligente Garbage Collection
* Eignet sich gut für Server- und Cloud-Software

#### gRPC

* Protokoll für Remote Procedure Calls.
* Basiert auf Protocol Buffers und HTTP/2.
* Server und Client Code wird aus einer sprachunabhängigen Schnittstellendeklaration generiert. Java wird unterstützt.
* Vorteile sind eine hohe Performanz durch schnelle Serialisierung und kleine Nachrichten, Support für Streaming und gute Unterstützung für Timeouts.
* gRPC ist gut für die Kommunikation zwischen Microservices geeignet, bietet aber derzeit noch keine Browserunterstützung.

#### GralVM

* Polyglotte virtuelle Maschine von Oracle auf der prinzipiell jede Programmiersprache ausgeführt werden kann. Bisher werden Bytecode erzeugende Programmiersprachen wie Java 8, Scala, Closure usw. sowie R, Ruby und C++ unterstützt. Programmiersprachen können in einem Programm kombiniert werden.
* Oracle stellt eine kostenlose Community und eine Enterprise Edition bereit. Die Enterprise Edition besitzt zusätzliche Features und ist schneller. Beide bieten Performanzvorteile gegenüber der JVM.
* Die GralVM ermöglicht das Erstellen von Native Binaries. Dadurch kann die Startzeit deutlich beschleunigt und der Speicherverbrauch minimiert werden.
* Herzstück der GralVM ist der Gral Compiler. Die GralVM basiert auf dem Open JDK und ersetzt dort den C2 JIT Compiler. Aufgabe des JIT Compilers ist es während der Ausführung des Codes diesen zu optimieren.

#### GraphQL

* Query Sprache sowie ein Runtime Environment für die Verarbeitung der Queries.
* Unterstützt wird das Lesen, Schreiben und Abonnieren von Daten
* Keine Ressourcenorientierung wie in REST. Die Daten werden als Graph repräsentiert und können über einen einzelnen Endpoint gelesen, bzw. geändert werden. Dadurch erhält der Client in einer einzelnen Anfrage genau die benötigten Daten.
* Spring Boot Starter für GraphQL existiert.

#### IBM Equal Access Toolkit

* Das Toolkit besteht aus zwei Teilen:
  + Guidelines: Diese helfen allen Mitgliedern des Teams (Product Owner, Designer, Entwickler Tester usw.) Barrierefreiheit in allen Phasen des Entwicklungszyklus zu berücksichtigen.
  + Accessibility Checker: Browser-Plugins, die bei der automatisierten Überprüfung der Barrierefreiheit und Berichterstellung unterstützen.

#### MicroNaut

* Java Framework mit geringem Speicherverbrauch und kürzerer Startzeit. Erreicht wird beides durch Dependency Injection zur Compile-Zeit.
* Intern wird ein Netty verwendet.
* Reaktive Programmierung wird explizit unterstützt.

PostgreSQL

* Freies objektrelationales Datenbankmanagementsystem für Windows und Linux.
* Weitestgehend konform zum SQL-Standard SQL:2011.
* Vollständig ACID konform.
* Unterstützt Clustering, synchrone und asynchrone Replikation.
* Keine Beschränkungen in Bezug auf DB-Maximalgröße oder maximale Spaltenanzahl.
* Stellt vielfältige JSON Funktionen und Operatoren bereit.
* JDBC-Treiber sind Open Source in den zentralen Maven Repositories verfügbar.
* Es fallen keine Lizenz-Kosten an.
* PostgreSQL wird in der ZAK 2025 neben der Oracle DB als erlaubtes DBMS genannt.

Project Reactor

* Framework für Reaktive Programmierung
* Basis für den Reaktive Stack in Spring mit Spring Web Flux usw.
* Einschätzung: Baldige Evaluation im Rahmen der Anbindung von NaDien.

#### Python

* Beliebte, dynamisch typisierte, leicht zu erlernende Programmiersprache.
* Schwächen von Python sind die Performanz und die Unterstützung mehrerer Prozessoren.
* Python ist im KI-Umfeld weit verbreitet.

#### Quarkus

* Java Framework für Cloud Native Anwendungen.
* Geringer Speicherverbrauch und schnelle Startzeit.
* Optimiert für die GralVM.
* Native Integration mit Docker und Kubernetes.
* Unterstützt Imperative und Reaktive Programmierung.

#### RxJava

* Bibliothek für Reaktive Programmierung
* Basiert auf den von Microsoft entwickelten Reactive Extensions (Rx).

#### Web Assembly

* Standardisiertes Binärformat, das von einer Stack-basierten VM ausgeführt und von den meisten Browsern unterstützt wird.
* Erlaubt es z.B. C, C++ und Rust Code im Browser auszuführen.
* Verspricht bessere Performanz als JavaScript.
* Bisherige APIs noch sehr eingeschränkt. Erweiterungen sind geplant.

#### Web Components

* Webkomponenten sind eine Gruppe von Web-Technologien, die es ermöglichen, benutzerdefinierte, wiederverwendbare HTML Elemente zu erstellen. Deren Funktionalität ist gekapselt und vom Rest der Seite isoliert.
* Basiert auf Custom Elements, Shadow DOM und HTML Templates.
* Einschätzung: Aktuell uninteressant

### Evaluations-Phase

#### Spring Data JDBC

* Spring Data basiertes ORM Framework
* Fokusiert auf Einfachheit und verzichtet dafür auf einige von der Java Persistence API (JPA) bekannte Features wie Caching, Lazy Loading usw..
* Aktuell keine offizielle Unterstützung für Oracle DBs.

### Nutzungs-Phase

#### React

* JavaScript-Library für das Erstellen von User Interfaces.

#### RESTful HTTP APIs

* REST Prinzipien sind u.a. Ressourcenorientierung, Zustandslosigkeit, Nutzung der http Verben und HATEOAS.
* Einfache Nutzung in SPAs per JavaScript möglich.
* Einschätzung:
  + Starke Fokusierung auf die Einhaltung der REST-Prinzipien, insbesondere wenn es um HATEOAS geht eher hinderlich.
  + http APIs sind in fast jeder Hinsicht eine bessere Alternative zu SOAP

#### Spring Boot

* Java Application Framework

#### SCSS

* CSS-Präprozessor, der die Erstellung und Pflege von CSS-Stylesheets vereinfacht

#### TypeScript

* Auf JavaScript basierende Programmiersprache mit Unterstützung für statische Typisierung.
* Die statische Typisierung führt zu besser dokumentiertem Code, hilft Fehler bereits zur Compile-Zeit zu entdecken und sorgt für bessere IDE-Unterstützung.
* TypeScript-Code wird durch den TypeScript-Compiler in JavaScript-Code übersetzt.

## Werkzeuge und Libraries

### Beobachtungs-Phase

#### Cypress Frontend Testing

* Unterstützt System-, Integrations- und Unit-Tests während des Entwicklungsprozesses, wobei der Fokus eher auf den System Frontend-Tests liegt.
* Die Evaluation des JavaScript Test-Codes findet innerhalb des Browsers statt
* Ermöglicht die Aufzeichnung von automatisierten Tests
* Hat an Popularität in den letzten Jahren deutlich hinzugewonnen, da die Tests einfacher, schneller und fehlerfreier als in Selenium laufen (z.B. keine expliziten Sleeps und Waits notwendig)

#### Gatling

* Werkzeug für Last- und Performanztests
* Scala-basierte DSL für Lasttest-Skripte
* Maven- und Gradle-Plugins für Verwendung in CD-Pipeline vorhanden

#### Material UI

* Komponenten-Bibliothek für React basierend auf dem Google Material Design.
* Alternative zu React Bootstrap

#### springdoc-openapi

* Generiert automatisiert OpenAPI Dokumentation für eine Spring Boot Anwendung.
* Integriert Swagger UI
* Kein offizielles Spring Projekt

#### Thymeleaf

* Serverseitige JAVA-template Engine
* Es können Vorlagen für HTML-, XHTML- oder HTML5-Seiten geschrieben werden
* Später können diese Serverseitig mit Daten/Subseiten gefüllt werden
* Die GDA verwendet aktuell noch Freemarker.

### Evaluations-Phase

#### Postman

* Werkzeug zum Testen und Entwerfen von SOAP, REST und GraphQL APIs
* Automatisierte Ausführung innerhalb einer CD-Pipeline möglich

### Nutzungs-Phase

#### Antlr

* Ein objektorientierter Parsergenerator, der aus einer Grammatik ausführbaren Code erzeugt.

#### Gradle

* Build-Werkzeug
* Performanter als Maven durch inkrementelle Builds und Build Caches.
* Builds sind durch die DSL und den Task-basierten Ansatz leichter an individuelle Bedürfnisse anpassbar.
* Die GDA stellt aktuell von Maven auf Gradle um.

#### Jest

* Von Facebook entwickeltes und gut dokumentiertes Frontend Testtool für jede JavaScript Code-Basis.
* Wird gerne als Test-Runner zusammen mit Enzyme oder React Testing Library zum Testen für React-Anwendungen verwendet.

#### JMeter

* Werkzeug für Last- und Performanztests
* Maven-Plugin ermöglicht die Nutzung im Rahmen einer CD-Pipeline.
* Ausblick: Unübersichtliche XML-Konfigurationsdateien und unkomfortable API führen zu schlechter Wartbarkeit der Tests. Mittelfristig sollten evtl. Alternativen wie Gatling evaluiert werden.

#### Jsonnet

* Open-Source-Daten-Template-Sprache
* Erlaubt die Generierung, Manipulation und Konfiguration von Konfigurationsdateien wie z.B.:
  + YAML
  + JSON

#### OWASP Dependency Check

* Scannt verwendete Fremd-Bibliotheken auf bekannte Schwachstellen
* Automatisiert im Rahmen eines Maven-Builds nutzbar

#### Project Lombok

* Java Library, die zur Compile-Zeit automatisch Getter, Setter, Builder usw. generiert.
* Eclipse Plugin existiert.

#### React Bootstrap

* Auf dem Bootstrap Frontend Framework aufbauende Komponenten-Library für React

#### Resilience4j

* Funktional orientierte Java Library für Fehlertoleranz
* Circuit Breaker, Bulk Heads, Timeouts usw.

#### SoapUI

* Werkzeug u.a. zum Testen von SOAP APIs
* Nutzung innerhalb der GDA für WSDL-basierte automatisierte Funktionstests und zum Nachstellen von Fehlersituationen im Support-Bereich.
* Ausblick: Wartbarkeit der automatisierten Tests sehr schlecht. Aktuell wird Postman als Alternative evaluiert.

#### Vavr

* Java Library für Funktionale Programmierung

# Requirements Engineering

## Konzepte und Techniken

### Beobachtungs-Phase

#### Behaviour Driven Development

* Methode bei der Anforderungen bereits während der Anforderungs-Analyse in einer strukturierten Sprache als Akzeptanztests festgehalten werden. Diese werden dann während der Entwicklung in ausführbare, automatisierte Tests überführt.

#### Design Thinking

* Systematische Herangehensweise an komplexe Problemstellungen
* Nicht die technische Lösbarkeit, sondern Nutzerwünsche und nutzerorientiertes Erfinden stehen im Zentrum des Prozesses.

#### Design Studio

* Iterativer Prozess für die Ideenfindung im Team

#### Domain Story Telling

* Interview- und Modellierungstechnik für das Verstehen einer Domäne und Erlernen einer Fachsprache

#### Event Storming

* Methode für das gemeinsame Erforschen einer Domäne im Rahmen eines Workshops.
* Die Domäne wird dabei rückwärts beginnend bei den finalen Events erforscht.

#### Lean Startup

* Ansatz für einen Produkt-Launch mit sehr kurzer Planungs- und Entwicklungsphase und möglichst schnellem messbarem Feedback, das für die nächste Iteration verwendet wird.

#### User Story Mapping

* Flache Backlogs werden schnell unübersichtlich.
* User Story Maps visualisieren die Zusammenhänge zwischen User Stories, der Reihenfolge ihrer Verwendung durch den Anwender sowie den Epics und Features.

#### Wireframes

* Wireframes zeigen die grobe Anordnung der Elemente einer graphischen Oberfläche und demonstrieren die Benutzerführung. Auf das konkrete Design der Oberfläche wird noch kein Wert gelegt.
* Wireframes sind schnell erstellt und können leicht verändert werden. Sie helfen bei der Abstimmung mit fachlichen Stakeholdern und der Diskussion im Team.
* Z.T. erlauben Wireframe-Werkzeuge die Generierung von Prototypen.
* Bekannte Werkzeuge sind z.B.: Adobe XD oder Balsamiq

#### Goal Question Metric (GQM)

* Methode, um zielgerichtet einen Mess- und Verbesserungsprozess durchzuführen
* Qualitätsmodell als Baumstruktur durch Definition von Zielen und das Stellen von Fragen

#### Reifegradmodell CMMI

* systematische Aufbereitung bewährter Praktiken, um die Verbesserung einer Organisation zu unterstützen
* konkretes Verbesserungsziel zwingend notwendig

### Nutzungs-Phase

#### Qualitätsszenarien

* Qualitätsszenarien sind kurze, beispielhafte, aktiv formulierte und für fachliche Stakeholder verständliche Beschreibungen der geforderten Qualitätsmerkmale (Nichtfunktionale Anforderungen).
* Qualitätsszenarien helfen bei der Diskussion, der Priorisierung von Anforderungen und der Bewertung von Architekturen.