

Curriculum für

Certified Professional for
Software Architecture (CPSA)[®]
Advanced Level

**Modul
FLEX**

**Flexible Architekturmodelle - Microservices und Self-
Contained Systems**

2024.1-RC0-DE-20240909



Inhaltsverzeichnis

Verzeichnis der Lernziele	2
Einführung: Allgemeines zum iSAQB Advanced Level	3
Was vermittelt ein Advanced Level Modul?	3
Was können Absolventen des Advanced Level (CPSA-A)?	3
Voraussetzungen zur CPSA-A-Zertifizierung	3
Grundlegendes	4
Was vermittelt das Modul „FLEX“?	4
Struktur des Lehrplans und empfohlene zeitliche Aufteilung	4
Dauer, Didaktik und weitere Details	4
Voraussetzungen	4
Gliederung des Lehrplans	5
Ergänzende Informationen, Begriffe, Übersetzungen	5
1. Motivation	6
1.1. Begriffe und Konzepte	6
1.2. Lernziele	6
1.3. Referenzen	7
2. Modularisierung	8
2.1. Begriffe und Konzepte	8
2.2. Lernziele	8
2.3. Referenzen	10
3. Integration	12
3.1. Begriffe und Konzepte	12
3.2. Lernziele	12
3.3. Referenzen	14
4. Installation und Roll Out	15
4.1. Begriffe und Konzepte	15
4.2. Lernziele	15
4.3. Referenzen	16
5. Betrieb, Überwachung und Fehleranalyse	17
5.1. Begriffe und Konzepte	17
5.2. Lernziele	17
5.3. Referenzen	18
6. Case Study	19
6.1. Begriffe und Konzepte	19
6.2. Lernziele	19
6.3. Referenzen	19
7. Ausblick	20



7.1. Begriffe und Konzepte 20

7.2. Lernziele 20

7.3. Referenzen 21

Referenzen 22

© (Copyright), International Software Architecture Qualification Board e. V. (iSAQB® e. V.) 2023

Die Nutzung des Lehrplans ist nur unter den nachfolgenden Voraussetzungen erlaubt:

1. Sie möchten das Zertifikat zum CPSA Certified Professional for Software Architecture Foundation Level® oder CPSA Certified Professional for Software Architecture Advanced Level® erwerben. Für den Erwerb des Zertifikats ist es gestattet, die Text-Dokumente und/oder Lehrpläne zu nutzen, indem eine Arbeitskopie für den eigenen Rechner erstellt wird. Soll eine darüber hinausgehende Nutzung der Dokumente und/oder Lehrpläne erfolgen, zum Beispiel zur Weiterverbreitung an Dritte, Werbung etc., bitte unter info@isaqb.org nachfragen. Es müsste dann ein eigener Lizenzvertrag geschlossen werden.
2. Sind Sie Trainer oder Trainingsprovider, ist die Nutzung der Dokumente und/oder Lehrpläne nach Erwerb einer Nutzungslizenz möglich. Hierzu bitte unter info@isaqb.org nachfragen. Lizenzverträge, die alles umfassend regeln, sind vorhanden.
3. Falls Sie weder unter die Kategorie 1. noch unter die Kategorie 2. fallen, aber dennoch die Dokumente und/oder Lehrpläne nutzen möchten, nehmen Sie bitte ebenfalls Kontakt unter info@isaqb.org zum iSAQB e. V. auf. Sie werden dort über die Möglichkeit des Erwerbs entsprechender Lizenzen im Rahmen der vorhandenen Lizenzverträge informiert und können die gewünschten Nutzungsgenehmigungen erhalten.

Wichtiger Hinweis

Grundsätzlich weisen wir darauf hin, dass dieser Lehrplan urheberrechtlich geschützt ist. Alle Rechte an diesen Copyrights stehen ausschließlich dem International Software Architecture Qualification Board e. V. (iSAQB® e. V.) zu.

Die Abkürzung "e. V." ist Teil des offiziellen Namens des iSAQB und steht für "eingetragener Verein", der seinen Status als juristische Person nach deutschem Recht beschreibt. Der Einfachheit halber wird iSAQB e. V. im Folgenden ohne die Verwendung dieser Abkürzung als iSAQB bezeichnet.

Verzeichnis der Lernziele

- LZ 1-1: Qualitätsziele von flexiblen Architekturen
- LZ 1-2: Wechselwirkung von Architektur-Typen und Organisation
- LZ 1-3: Kompromisse der vorgestellten Architektur-Typen vermitteln und adaptieren
- LZ 1-4: Verständnis für Rahmenbedingungen von verteilten Systemen
- LZ 1-5: Themen und Buzzwords einordnen
- LZ 2-1: Zerlegung in Bausteine
- LZ 2-2: Modularisierungskonzepte
- LZ 2-3: Modularisierungsstrategien
- LZ 3-1: Strategic Design
- LZ 3-2: Technische Integrations-Strategie wählen
- LZ 3-3: Makroarchitektur erläutern und Kriterien aufstellen können
- LZ 3-4: Domain Events statt RPC für die Integration von Bounded Contexts
- LZ 3-5: Sicherheit
- LZ 4-1: Aspekte bei Rollout und Continuous Delivery
- LZ 5-1: Fehleranalyse in verteilten Systemen
- LZ 5-2: Unterschiede zwischen Metriken, Logs und Traces
- LZ 5-3: Unterschiedliche Betriebsmodelle
- LZ 7-1: Konsistenzmodelle
- LZ 7-2: Resilience Patterns

Einführung: Allgemeines zum iSAQB Advanced Level

Was vermittelt ein Advanced Level Modul?

Das Modul kann unabhängig von einer CPSA-F-Zertifizierung besucht werden.

- Der iSAQB Advanced Level bietet eine modulare Ausbildung in drei Kompetenzbereichen mit flexibel gestaltbaren Ausbildungswegen. Er berücksichtigt individuelle Neigungen und Schwerpunkte.
- Die Zertifizierung erfolgt als Hausarbeit. Die Bewertung und mündliche Prüfung wird durch vom iSAQB benannte Experten vorgenommen.

Was können Absolventen des Advanced Level (CPSA-A)?

CPSA-A-Absolventen können:

- eigenständig und methodisch fundiert mittlere bis große IT-Systeme entwerfen
- in IT-Systemen mittlerer bis hoher Kritikalität technische und inhaltliche Verantwortung übernehmen
- Maßnahmen zur Erreichung von Qualitätsanforderungen konzeptionieren, entwerfen und dokumentieren sowie Entwicklungsteams bei der Umsetzung dieser Maßnahmen begleiten
- architekturelevante Kommunikation in mittleren bis großen Entwicklungsteams steuern und durchführen

Voraussetzungen zur CPSA-A-Zertifizierung

- erfolgreiche Ausbildung und Zertifizierung zum Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- mindestens drei Jahre Vollzeit-Berufserfahrung in der IT-Branche; dabei Mitarbeit an Entwurf und Entwicklung von mindestens zwei unterschiedlichen IT-Systemen
 - Ausnahmen sind auf Antrag zulässig (etwa: Mitarbeit in Open-Source-Projekten)
- Aus- und Weiterbildung im Rahmen von iSAQB-Advanced-Level-Schulungen im Umfang von mindestens 70 Credit-Points aus mindestens drei unterschiedlichen Kompetenzbereichen
- erfolgreiche Bearbeitung der CPSA-A-Zertifizierungsprüfung



Grundlegendes

Was vermittelt das Modul „FLEX“?

Das Modul Flexible Architekturmodelle - Microservices und Self-Contained Systems zeigt, wie man mit modernen Architektur-Ansätzen besonders flexible Systeme umsetzen kann. Hierbei beginnen wir bei Qualitätszielen, die flexible, verteilte Systeme überhaupt motivieren, betrachten Modularisierungs-Optionen und Architektur-Konzepte. Die Teilnehmer:innen lernen wie Kommunikationsstrukturen zwischen Teams und deren Organisation ihre Architekturentscheidungen entscheidend beeinflussen und wie sie diese Macro-Architektur steuern können.

Am Ende des Moduls kennen die Teilnehmer:innen Grundlagen von Microservices und Self-contained Systems und können Systeme auf Basis dieser Architektur-Konzepte entwerfen. Dabei können sie auch eine sinnvolle fachliche Aufteilung vorschlagen, die Erkenntnisse aus Konzepten wie Continuous Delivery und nachhaltigem Service-Betrieb berücksichtigt.

Struktur des Lehrplans und empfohlene zeitliche Aufteilung

Inhalt	Empfohlene Mindestdauer (min)	Übungszeit (Minuten)
1. Motivation	30	
2. Modularisierung	180	75
3. Integration	210	90
4. Installation und Roll Out	120	30
5. Betrieb, Überwachung und Fehleranalyse	150	45
6. Ausblick	90	
Summe	780 (13 Std)	240 (4 Std)

In diesem Kontext stellt „Dauer“ die Unterrichtszeit ohne Übungszeit dar. Die Zeitaufteilung zwischen Theorie und Übungen ist lediglich eine Empfehlung. Die Ausgestaltung der Beispiele und Übungen sind in diesem Lehrplan nicht vorgegeben.

Dauer, Didaktik und weitere Details

Die unten genannten Zeiten sind Empfehlungen. Die Dauer einer Schulung zum Modul FLEX sollte mindestens 3 Tage betragen, kann aber länger sein. Anbieter können sich durch Dauer, Didaktik, Art und Aufbau der Übungen sowie der detaillierten Kursgliederung voneinander unterscheiden. Insbesondere die Art der Beispiele und Übungen lässt der Lehrplan komplett offen.

Lizenzierte Schulungen zu FLEX tragen zur Zulassung zur abschließenden Advanced-Level-Zertifizierungsprüfung folgende Credit Points bei:

Methodische Kompetenz:	10 Punkte
Technische Kompetenz:	20 Punkte
Kommunikative Kompetenz:	00 Punkte

Voraussetzungen

Teilnehmerinnen und Teilnehmer **sollten** folgende Kenntnisse und/oder Erfahrung mitbringen:

- Grundlagen der Beschreibung von Architekturen mithilfe verschiedener Sichten, übergreifender Konzepte, Entwurfsentscheidungen, Randbedingungen etc., wie es im CPSA-F (Foundation Level) vermittelt wird.
- Erfahrung mit der Implementierung und Architektur in agilen Projekten.
- Erfahrungen aus der Entwicklung und Architektur klassischer Systeme mit den typischen Herausforderungen.

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Verteilte Systeme
 - Probleme und Herausforderungen bei der Implementierung verteilter Systeme
 - Typische verteilte Algorithmen
 - Internet-Protokolle
- Kenntnisse über Modularisierungen
 - Fachliche Modularisierung
 - Technische Umsetzungen wie Pakete oder Bibliotheken
- Klassische Betriebs- und Deployment-Prozesse

Gliederung des Lehrplans

Die einzelnen Abschnitte des Lehrplans sind gemäß folgender Gliederung beschrieben:

- **Begriffe/Konzepte:** wesentliche Kernbegriffe dieses Themas.
- **Unterrichts-/Übungszeit:** Legt die Unterrichts- und Übungszeit fest, die für dieses Thema bzw. dessen Übung in einer akkreditierten Schulung mindestens aufgewendet werden muss.
- **Lernziele:** Beschreibt die zu vermittelnden Inhalte inklusive ihrer Kernbegriffe und -konzepte.

Dieser Abschnitt skizziert damit auch die zu erwerbenden Kenntnisse in entsprechenden Schulungen.

Ergänzende Informationen, Begriffe, Übersetzungen

Soweit für das Verständnis des Lehrplans erforderlich, haben wir Fachbegriffe ins [iSAQB-Glossar](#) aufgenommen, definiert und bei Bedarf durch die Übersetzungen der Originalliteratur ergänzt.

1. Motivation

Dauer: 120 Min.	Übungszeit: 0 Min.
-----------------	--------------------

1.1. Begriffe und Konzepte

Verfügbarkeit, Zuverlässigkeit, Time-to-Market, Flexibilität, Vorhersagbarkeit, Reproduzierbarkeit, Homogenisierung der Stages, Internet/Web-Scale, verteilte Systeme, Parallelisierbarkeit der Feature-Entwicklung, Evolution der Architektur (Build for Replacement), Heterogenität, Automatisierbarkeit.

1.2. Lernziele

LZ 1-1: Qualitätsziele von flexiblen Architekturen

Architekturen können auf unterschiedliche Qualitätsziele hin optimiert werden.

In diesem Modul lernen die Teilnehmer, wie sie Architekturen erstellen, die schnelles Deployment und damit schnelles Feedback aus der Anwendung des Systems erlauben. Solche Architekturen unterstützen die Entwickler:innen dabei, effizient funktionale Angemessenheit zu erreichen und die Usability zu verbessern. Außerdem fördern sie die Modularität und können dazu dienen, hohe Zuverlässigkeit zu erreichen und einzelne Komponenten isoliert zu ersetzen, was der Flexibilität dient.

LZ 1-2: Wechselwirkung von Architektur-Typen und Organisation

Sie haben die Treiber für die Architektur-Typen verstanden, die in diesem Lehrplanmodul vermittelt werden - also das Bedürfnis, Features schnell in Produktion zu bringen und Time-to-Market zu verkürzen. Sie haben außerdem verstanden, welche Konsequenzen die Treiber für die Architekturen haben und wie die Wechselwirkung der Architekturen mit Organisation, Prozessen und Technologien ist.

LZ 1-3: Kompromisse der vorgestellten Architektur-Typen vermitteln und adaptieren

Sie haben die Kompromisse der vorgestellten Architektur-Typen (mindestens Microservices, Self Contained Systems und Deployment Monolithen) verstanden und können diese sowohl vermitteln als auch im Rahmen konkreter Projekte/Systementwicklungen anwenden, um angemessene Architekturentscheidungen zu treffen.

LZ 1-4: Verständnis für Rahmenbedingungen von verteilten Systemen

Teilnehmer sollen Rahmenbedingungen von verteilten Systemen verstehen, wie z. B. folgende:

- Auf die Fähigkeit, neue Features schnell in Produktion bringen zu können, hat die Architektur entscheidenden Einfluss.
- Abhängigkeiten zwischen Komponenten, die von unterschiedlichen Entwicklungsteams verantwortet werden, beeinflussen die Dauer, bis Software in Produktion gebracht werden kann, weil sie die Kommunikationsaufwände erhöhen und sich Verzögerungen fortpflanzen.
- Die Automatisierung von Aktivitäten (wie z. B. Test- und Deployment-Prozesse) erhöht die Reproduzierbarkeit, Vorhersagbarkeit und Ergebnisqualität dieser Prozesse. Das kann zu einer Verbesserung des gesamten Entwicklungsprozesses führen.
- Die Vereinheitlichung der verschiedenen Umgebungen (z. B. Entwicklung, Test, QA, Produktion) reduziert das Risiko von spät entdeckten und (in anderen Umgebungen) nicht reproduzierbaren Fehlern aufgrund unterschiedlicher Konfigurationen.
- Die Vereinheitlichung und Automatisierung sind wesentliche Aspekte von Continuous Delivery.

- Continuous Integration ist eine Voraussetzung für Continuous Delivery.
- Eine geeignete Architektur ist die Voraussetzung für eine Parallelisierbarkeit der Entwicklung sowie die unabhängige Inbetriebnahme von eigenständigen Bausteinen. Das können, müssen aber nicht „Services“ sein.
- Einige Änderungsszenarien lassen sich leichter in monolithischen Architekturen umsetzen. Andere Änderungsszenarien lassen sich leichter in verteilten Service-Architekturen umsetzen. Beide Ansätze können kombiniert werden.
- Es gibt unterschiedliche Arten der Isolation mit jeweils unterschiedlichen Vorteilen. Beispielsweise kann der Ausfall auf eine Komponente begrenzt werden oder Änderungen können auf eine Komponente begrenzt werden.
- Bestimmte Arten der Isolation sind zwischen Prozessen mit Remotekommunikation deutlich einfacher umzusetzen.
- Remotekommunikation hat aber Nachteile – z. B. viele neue Fehlerquellen.

LZ 1-5: Themen und Buzzwords einordnen

- Gesetz von Conway
- Partitionierbarkeit als Qualitätsmerkmal
- Durchlaufzeiten durch die IT-Wertschöpfungskette als Wettbewerbsfaktor
- Aufbau einer Continuous-Delivery-Pipeline
- Die verschiedenen Test-Phasen in einer Continuous-Delivery-Pipeline

1.3. Referenzen

[Humble, et al. 2010], [Wolff 2016], [Humble, et al. 2014]

2. Modularisierung

Dauer: 120 Min.	Übungszeit: 30 Min.
-----------------	---------------------

2.1. Begriffe und Konzepte

- Motivation für die Dekomposition in kleinere Systeme
- Unterschiedliche Arten von Modularisierung, Kopplung
- Systemgrenzen als Mittel für Isolation
- Hierarchische Struktur
- Anwendung, Applikation, Self-Contained System, Microservice
- Domain-Driven Design-Konzepte und „Strategic Design“, Bounded Contexts

2.2. Lernziele

LZ 2-1: Zerlegung in Bausteine

Was sollen die Teilnehmer können?

- Teilnehmer können für eine gegebene Aufgabenstellung eine Zerlegung in einzelne Bausteine entwerfen.
- Die Teilnehmer sollen die Kommunikationsstruktur der Organisation beim Festlegen der Modulgrenzen berücksichtigen (Gesetz von Conway).
- Die Teilnehmer können einen Plan zur Aufteilung eines Deployment-Monolithen in kleine Services aufstellen.

Was sollen die Teilnehmer verstehen?

- Teilnehmer verstehen, dass jede Art von Bausteinen neben einer griffigen Bezeichnung eine Beschreibung benötigt,
 - was Bausteine dieser Art ausmacht,
 - wie ein solcher Baustein zur Laufzeit integriert wird,
 - wie ein solcher Baustein (im Sinne des Build-Systems) gebaut wird,
 - wie ein solcher Baustein deployt wird,
 - wie ein solcher Baustein getestet wird,
 - wie ein solcher Baustein skaliert wird.
- Unterschiedliche Grade an Vorgaben für die Entwicklung eines Bausteins können sinnvoll sein. Einige Vorgaben sollten besser übergeordnet allgemein für die Integration mit anderen Bausteinen dieser Art gelten. Die übergreifenden Entscheidungen, die alle Systeme beeinflussen, können eine Makro-Architektur bilden - dazu zählen beispielsweise die Kommunikationsprotokolle oder Betriebsstandards. Mikro-Architektur kann die Architektur eines einzelnen Systems sein. Es ist weitgehend unabhängig von anderen Systemen. Zu große Einschränkungen auf Ebene der Makro-Architektur führen dazu, dass die Architektur insgesamt auf weniger Probleme angewendet werden kann.

Was sollen die Teilnehmer kennen?

- Die Teilnehmer sollen das Gesetz von Conway kennen.

LZ 2-2: Modularisierungskonzepte

Was sollen die Teilnehmer können?

- Die Teilnehmer sollen technische Modularisierungskonzepte projektspezifisch bewerten und auswählen können.
- Die Teilnehmer sollen die Beziehungen zwischen Modulen sowie zwischen Modulen und Subdomänen veranschaulichen und analysieren können (Context Mapping).
- Die Teilnehmer können ein Konzept erarbeiten, um ein System aus Services aufzubauen.

Was sollen die Teilnehmer verstehen?

- Modularisierung hilft Ziele wie Parallelisierung der Entwicklung, unabhängiges Deployment/Austauschbarkeit zur Laufzeit, Rebuild/Reuse von Modulen und leichtere Verständlichkeit des Gesamtsystems zu erreichen.
- Daher ist beispielsweise Continuous Delivery und die Automatisierung von Test und Deployment ein wichtiger Einfluss auf die Modularisierung.
- Modularisierung bezeichnet Dekomposition eines Systems in kleinere Teile. Diese Teile nach der Dekomposition wieder zu integrieren, verursacht organisatorische und technische Aufwände. Diese Aufwände müssen durch die Vorteile, die durch die Modularisierung erreicht werden, mehr als ausgeglichen werden.
- Je nach gewählter Modularisierungstechnologie besteht Kopplung auf unterschiedlichen Ebenen:
 - Sourcecode (Modularisierung mit Dateien, Klassen, Packages, Namensräumen etc.)
 - Kompilat (Modularisierung mit JARs, Bibliotheken, DLLs etc.)
 - Laufzeitumgebung (Betriebssystem, virtuelle Maschine oder Container)
 - Netzwerkprotokoll (Verteilung auf verschiedene Prozesse)
- Eine Kopplung auf Ebene des Sourcecodes erfordert sehr enge Kooperation sowie gemeinsames SCM. Eine Kopplung auf Ebene des Kompilats bedeutet, dass die Bausteine in der Regel gemeinsam deployt werden müssen. Nur eine Verteilung auf unterschiedliche Anwendungen/Prozesse ist praktikabel im Hinblick auf ein unabhängiges Deployment.
- Teilnehmer verstehen, dass eine vollständige Isolation zwischen Bausteinen nur durch eine Trennung in allen Phasen (Entwicklung, Deployment und Laufzeit) gewährleistet werden kann. Ist das nicht der Fall, können unerwünschte Abhängigkeiten nicht ausgeschlossen werden. Gleichzeitig verstehen die Teilnehmer auch, dass es sinnvoll sein kann, aus Gründen wie effizienter Ressourcennutzung oder Komplexitätsreduktion auf eine vollständige Isolation zu verzichten.
- Teilnehmer verstehen, dass bei der Verteilung auf unterschiedliche Prozesse manche Abhängigkeiten nicht mehr in der Implementierung existieren, sondern erst zur Laufzeit entstehen. Dadurch steigen die Anforderungen an die Überwachung dieser Schnittstellen.
- Microservices sind unabhängige Deployment-Einheiten und damit auch unabhängige Prozesse, die ihre Funktionen über leichtgewichtige Protokolle exponieren, aber auch ein UI haben können. Für die Implementierung jedes einzelnen Microservices können unterschiedliche Technologieentscheidungen getroffen werden.
- Ein Self-Contained System (SCS) stellt ein fachlich eigenständiges System dar. Es beinhaltet üblicherweise UI und Persistenz. Es kann aus mehreren Microservices bestehen. Fachlich deckt ein SCS meist einen Bounded Context ab.

Was sollen die Teilnehmer kennen?

- Die Teilnehmer sollen verschiedene technische Mechanismen kennen, um ein System zur Entwicklungszeit aufzuteilen: Sourcecode-Dateien, JARs, OSGi-Bundles, Module der Programmiersprache.
- Die Teilnehmer sollen verschiedene Möglichkeiten kennen, ein System zur Laufzeit aufzuteilen: Prozesse, Microservices, Self-Contained Systems.
- Die Teilnehmer sollen "The Twelve-Factor App" kennen.

LZ 2-3: Modularisierungsstrategien

Was sollen die Teilnehmer können?

- Die Teilnehmer können die Konsequenzen verschiedener Modularisierungsstrategien bewerten und den mit der Modularisierung verbundenen Aufwand dem zu erwartenden Nutzen gegenüberstellen.
- Die Teilnehmer können die Auswirkungen der Modularisierungsstrategie auf die Autonomie von Bausteinen zur Entwicklungszeit und zur Laufzeit beurteilen.
- Die Teilnehmer können eine geeignete Modularisierung und eine geeignete Granularität der Modularisierung wählen – abhängig von der Organisation und den Qualitätszielen.

Was sollen die Teilnehmer verstehen?

- Teilnehmer verstehen, dass eine Integrationsstrategie darüber entscheidet, ob eine Abhängigkeit
 - erst zur Laufzeit entsteht,
 - zur Entwicklungszeit entsteht oder
 - beim Deployment entsteht.
- Der Modulschnitt kann entlang fachlicher oder technischer Grenzen erfolgen. In den meisten Fällen empfiehlt sich ein fachlicher Schnitt, da sich so fachliche Anforderungen klarer einem Modul zuordnen lassen und somit nicht mehrere Module für die Umsetzung einer fachlichen Anforderung angepasst werden müssen. Dabei kann jedes Modul sein eigenes Domänenmodell im Sinne eines Bounded Context und damit unterschiedliche Sichten auf ein Geschäfts- objekt mit eigenen Daten haben.
- Teilnehmer verstehen, dass zur Erreichung von höherer Autonomie der Entwicklungsteams ein Komponentenschnitt besser entlang fachlicher Grenzen anstatt entlang technischer Grenzen erfolgt.
- Transaktionale Konsistenz lässt sich über Prozessgrenzen hinweg nur über zusätzliche Mechanismen erreichen. Wird ein System in mehrere Prozesse aufgeteilt, so stellt die Modulgrenze daher häufig auch die Grenze für transaktionale Konsistenz dar. Daher muss ein DDD-Aggregat in einem Modul verwaltet werden.
- Teilnehmer verstehen, welche Modularisierungskonzepte nicht nur für Transaktions-, sondern auch für Batch- und Datenfluss-orientierte Systeme genutzt werden können.

Was sollen die Teilnehmer kennen?

- Die Teilnehmer kennen die Modularisierungsstrategien "Bounded Context", "Customer Journey" und "Information Hiding" [\[Parnas 1972\]](#).
- Die Teilnehmer sollen folgende Begriffe aus dem Domain-Driven Design kennen: Aggregate Root, Context Mapping, Bounded Contexts und Beziehungen dazwischen (z. B. Anti-Corruption Layer).

2.3. Referenzen

[Eric Evans 2003], [Lewis, Fowler, et al. 2013], [12 Factor App 2012], [Wolff 2018], [Sam Newman 2021],
[Parnas 1972]

3. Integration

Dauer: 90 Min.	Übungszeit: 30 Min.
----------------	---------------------

3.1. Begriffe und Konzepte

Frontend Integration, Legacy Systeme, Authentifizierung, Autorisierung, (lose) Kopplung, Skalierbarkeit, Messaging Patterns, Domain Events, dezentrale Datenhaltung.

3.2. Lernziele

LZ 3-1: Strategic Design

1. Die Teilnehmer sollen ein grundlegendes Verständnis für die Umsetzung von Integrationen mithilfe von Strategic Design aus dem Domain Driven Design und den wesentlichen Pattern (bspw. Customer/Supplier) anwenden können. Ziel ist, dass die Teilnehmer die Auswirkungen von Beziehungen zwischen Systemen anhand einer Context Map darstellen und SOLL/IST-Vergleiche erstellen können, die als Entscheidungskriterien für Team-Setup und Organisations-Architektur nutzbar sind.
2. Für die Integration von Legacy-Systemen muss definiert werden, wie mit alten Datenmodellen umgegangen wird. Dazu kann der Ansatz des Strategic Designs mit wesentlichen Patterns wie beispielsweise Anti-Corruption Layer genutzt werden.

LZ 3-2: Technische Integrations-Strategie wählen

1. Die Teilnehmer sollen die Vor- und Nachteile verschiedener Integrationsmechanismen kennen. Dazu zählen Frontend-Integration mit Mash Ups, Integration auf dem Middle Tier und Integration über Datenbanken oder Datenbank-Replikation.
2. Die Teilnehmer können abhängig von den Qualitätszielen und dem Wissen des Teams eine geeignete Integration vorschlagen.
3. Die Teilnehmer sollen eine Integrationsstrategie wählen, die auf das jeweilige Problem am besten passt. Das kann beispielsweise eine Frontend-Integration, eine Integration über RPC-Mechanismen, mit Message-orientierter Middleware, mit REST oder über die Replikation von Daten sein.
4. Teilnehmer verstehen, dass eine Integrationsstrategie darüber entscheidet, ob eine Abhängigkeit
 - erst zur Laufzeit entsteht,
 - zur Entwicklungszeit entsteht oder
 - beim Deployment entsteht.
5. Die Teilnehmer sollen die Konsequenzen und Einschränkungen verstehen, die sich aus der Integration von Systemen über verschiedenen Technologien und Integrationsmuster z. B. in Bezug auf Sicherheit, Antwortzeit oder Latenz ergeben.
6. Die Teilnehmer sollten verstehen, dass Messaging Kopplung an das Datenformat oder an eine API bedingt. REST macht Vorgaben, die diese Kopplung reduzieren können (Hypermedia, standardisierte API).
7. Die Teilnehmer sollten verstehen, dass synchrone Kommunikation, zum Beispiel durch RPC, zeitliche Kopplung bedingt. Asynchrone Kommunikation ermöglicht, diese Kopplung aufzuweichen und erlaubt unter Umständen, dass ein System Fortschritt macht, ohne dass die Antworten auf alle Anfragen schon vorliegen. Entkopplung lässt sich außerdem durch selektive Kommunikation erreichen.

8. Die Teilnehmer verstehen, dass bei der Verwendung von Messaging gegenüber dem Aufruf lokaler Funktionen Verfügbarkeit und Antwortzeiten des Systems beeinträchtigt werden können.
9. Die Teilnehmer wissen, dass es für Messaging Middleware gibt, die bestimmte Aufgaben erleichtert. Nachrichten können optional persistiert, gefiltert, transformiert etc. werden. Es gibt verschiedene Messaging-Patterns wie Request/Reply, Publish/Subscribe oder Broadcast.
10. Eine Integration über Daten ermöglicht hohe Autonomie, die allerdings über die Notwendigkeit zur redundanten Datenhaltung und die damit notwendige Synchronisation erkauft wird. Es darf nicht angenommen werden, dass andere Systeme dieselben Schemata nutzen, weil das eine unabhängige Weiterentwicklung der Schemata verhindert. Daher muss für die Integration eine angemessene Transformation vorgesehen werden.
11. Die Teilnehmer sollen verschiedene Ansätze für die Frontend-Integration sowie deren Vor- und Nachteile und Anwendbarkeit abhängig von Kontextkriterien benennen können.
12. Die Teilnehmer sollen Techniken für die Integration von Services: REST, RPC, Message-orientierte Middleware kennen.
13. Die Teilnehmer sollen Datenbank-Replikationsmechanismen mit ETL-Tools oder anderen Ansätzen kennen.
14. Die Teilnehmer sollen Messaging Patterns (Request/Reply, Publish/Subscribe etc.) kennen.
15. Die Teilnehmer sollen Messaging Systeme (RabbitMQ, Kafka, etc.), Protokolle (AMQP, MQTT, STOMP, etc.) und APIs (JMS) kennen.

LZ 3-3: Makroarchitektur erläutern und Kriterien aufstellen können

1. Anhand dieser Ansätze sollen die Teilnehmer eine Makroarchitektur entwerfen können, die zumindest Kommunikation und Sicherheit abdeckt.
2. Die Teilnehmer können klare Kriterien benennen anhand derer Entwickler selbständig herausfinden, ob eine (Architektur-)Entscheidung ein Mikro- oder eine Makroarchitektur Entscheidung ist resp. ob sie die Entscheidung allein treffen können oder team-externe Stakeholder in den Entscheidungsprozess einbeziehen müssen

LZ 3-4: Domain Events statt RPC für die Integration von Bounded Contexts

1. Bei einer Event-Driven Architecture (EDA) werden Remote Procedure Calls (RPC) vermieden oder reduziert, indem Domain Events (vgl. DDD Curriculum) publiziert werden. Domain Events beschreiben Zustandsänderungen. Interessierte Systeme können diese Nachrichten verarbeiten (Publish/Subscribe). Dieses Vorgehen hat Auswirkungen darauf, wie der Zustand gespeichert wird. Während bei einer RPC-basierten Integration der Server die Daten speichern muss, liegt diese Verantwortung bei EDA beim Subscriber. Somit entstehen Replikate (dezentrale Datenhaltung). Die Subscriber fungieren dadurch als Cache für den Publisher. Es können weitere Subscriber hinzugefügt werden, ohne den Publisher zu beeinflussen (außer durch Polling). Monitoring der Event-Flüsse ist wichtig.
2. Domain Events können via Messaging publiziert werden. Dabei pusht der Publisher die Nachrichten in ein Messaging System. Alternativ können die Nachrichten auch beim Publisher gepollt werden (bspw. Atom/RSS). Beim Einsatz eines Messaging Systems können Subscriber die Nachrichten per Push oder Pull erhalten. Dies hat Auswirkungen auf den Umgang mit Backpressure.
3. Die Teilnehmer sollen Herausforderungen bei der Nutzung gemeinsamer Daten kennen.

LZ 3-5: Sicherheit

1. Die Teilnehmer sollen einen geeigneten Ansatz für das Umsetzen von Sicherheit (Autorisierung/Authentifizierung) in einem verteilten System konzeptionieren können.
2. Die Teilnehmer sollen typische verteilte Sicherheitsmechanismen wie OAuth2, OpenID Connect, SAML oder Kerberos kennen und deren Flows erläutern können.

3.3. Referenzen

[\[Eric Evans 2003\]](#), [\[Parecki et al. 2006\]](#), [\[Hohpe, Woolf 2003\]](#)

4. Installation und Roll Out

Dauer: 90 Min.	Übungszeit: 30 Min.
----------------	---------------------

4.1. Begriffe und Konzepte

Moderner Betrieb, DevOps, Infrastructure as Code, Configuration Management.

4.2. Lernziele

LZ 4-1: Aspekte bei Rollout und Continuous Delivery

Was sollen die Teilnehmer können?

- Die Teilnehmer sollen ein Konzept grob skizzieren und verstehen können, wie ein System möglichst einfach automatisiert deployt wird. Sie müssen zwischen den verschiedenen technologischen Ansätzen abwägen können.
- Die Teilnehmer sollen Deployment-Automatisierung konzeptionieren und bewerten können. Beispielsweise müssen sie die Qualität und das Testen dieser Ansätze bewerten können. Sie müssen einen geeigneten Ansatz für ein Projekt-Szenario auswählen können.
- Die Teilnehmer sollen auf Basis des DevOps-Organisationsmodells einen Team-Aufbau skizzieren können.

Was sollen die Teilnehmer verstehen?

- Basis für die Automatisierung eines Deployments sind Virtualisierung oder Cloud mit Infrastructure as a Service (IaaS). Eine leichtgewichtige Alternative sind Linux-Container, wie sie Docker umsetzt.
- Ohne eine Deployment-Automatisierung ist das Deployment einer großen Anzahl von Servern und Services praktisch nicht möglich.
- Moderne Deployment-Werkzeuge ermöglichen es, auf Rechnern Software automatisiert zu installieren. Neben der Anwendung selbst kann dabei auch die vollständige Infrastruktur automatisiert aufgebaut werden. Dabei sind die Installationen idempotent, d. h. sie führen unabhängig vom Ausgangszustand des Systems immer zum gleichen Ergebnis.
- Immutable Server werden grundsätzlich nie geändert. Muss eine neue Version der Software in Betrieb genommen werden, wird der Server komplett neu aufgebaut. Das kann einfacher und zuverlässiger sein, als sich auf idempotente Werkzeuge zu verlassen.
- PaaS (Platform as a Service) stellt eine vollständige Plattform bereit, in die Anwendungen deployt werden können. Da in diesem Fall die Infrastruktur nicht selber aufgebaut wird, ist der Ansatz einfacher, aber auch weniger flexibel.
- Konzepte wie Tests oder Code Reviews sind auch für die Deployment-Automatisierung unabdingbar. Infrastruktur wird zu Code, der denselben Anforderungen wie produktiver Code gerecht werden muss (Infrastructure as Code).
- Um das Deployment zu unterstützen, kann das Ergebnis eines Build-Prozesses Packages für ein Betriebssystem oder gar Images für virtuelle Maschinen sein.
- Die Umgebungen eines Entwicklers sollten idealerweise mit den Umgebungen in Produktion übereinstimmen. Mit modernen Werkzeugen ist es möglich, eine solche Umgebung auf Knopfdruck zu erzeugen und aktuell zu halten.
- Die Komplexität des Deployments wird zu einem weiteren Qualitätsmerkmal des Systems und beeinflusst Architektur-Werkzeuge.

- Bei DevOps sind Entwicklung und Betrieb in einem Team integriert. Deshalb muss der Betrieb bei der Entwicklung bereits berücksichtigt werden. Neben dem Provisioning hat das auch Auswirkungen auf Continuous Delivery (siehe Kapitel „Continuous Delivery“).

Was sollen die Teilnehmer kennen?

- Grundlegendes Konzept von moderner Infrastruktur wie IaaS, PaaS und Virtualisierung
- Konzepte von Deployment-Werkzeuge wie Chef, Puppet, Ansible oder Salt
- Organisationsformen für DevOps
- Konzept von Deployments mit Package-Managern oder Linux Containern
- Verschiedene PaaS-Plattformen und ihre Konzepte

4.3. Referenzen

[\[Vossen, Haselmann, Hoeren 2012\]](#), [\[Wolff, Müller, Löwenstein 2013\]](#), [\[Humble, et al. 2010\]](#), [\[Wolff 2016\]](#)

5. Betrieb, Überwachung und Fehleranalyse

Dauer: 90 Min.	Übungszeit: 30 Min.
----------------	---------------------

5.1. Begriffe und Konzepte

Monitoring, Operations, Logging, Tracing, Metrics, Alerting.

5.2. Lernziele

LZ 5-1: Fehleranalyse in verteilten Systemen

Was sollen die Teilnehmer können?

- Die Teilnehmer sollen ein Konzept grob skizzieren und verstehen können, auf dessen Basis ein System überwacht werden kann, d.h. den aktuellen Status zu beurteilen, Fehler und Abweichungen vom normalen Betrieb möglichst zu vermeiden oder zumindest so früh wie möglich zu erkennen und zu behandeln.

Was sollen die Teilnehmer verstehen?

- Die Teilnehmer sollten die Unterschiede zwischen Logging, Monitoring und Metrik-Sammlung verstehen und die damit einhergehenden Unterschiede bei den verwendeten Werkzeugen.
- Die richtige Auswahl von Daten ist zentral für ein zuverlässiges und sinnvolles Monitoring und Logging.
- Damit Systeme, insbesondere solche, die sich aus vielen einzelnen Teilsystemen zusammensetzen, betreibbar sind, muss die Unterstützung des Betriebs mit hoher Priorität Bestandteil der Architekturkonzepte sein.
- Damit eine möglichst hohe Transparenz erreicht wird, müssen sehr viele Daten erfasst, aber auch zielgruppengerecht voraggregiert und auswertbar gemacht werden.
- Die Teilnehmer sollen verstehen, welche Informationen sie aus Log-Daten und welche sie (besser) durch Instrumentierung des Codes mit Metrik-Sonden beziehen können.

LZ 5-2: Unterschiede zwischen Metriken, Logs und Traces

Was sollen die Teilnehmer verstehen?

- Logging und Monitoring kann sowohl fachliche als auch technische Daten enthalten.
- Logs sind Events, Metriken sind Zustände zu einem bestimmten Zeitpunkt

Was sollen die Teilnehmer kennen?

- Werkzeuge für zentralistische Logdaten-Verwaltung
- Werkzeuge für zentralistische Metriken-Verarbeitung
- Unterscheidung zwischen Geschäfts-, Anwendungs- und Systemmetriken
- Bedeutung wichtiger, werkzeugunabhängiger System- und Anwendungsmetriken

LZ 5-3: Unterschiedliche Betriebsmodelle

Was sollen die Teilnehmer können?

- Anhand des Kontextes - wie Organisation und Skillset - für einen zentralen oder dezentralen Anwendungsbetrieb entscheiden.

- Dabei können sie abhängig vom konkreten Projekt-Szenario den Fokus im Konzept auf Logging, Monitoring und die dazu notwendigen Daten legen.
- Die Teilnehmer sollen Architekturvorgaben so treffen können, dass der Einsatz geeigneter Werkzeuge bestmöglich unterstützt wird, dabei jedoch angemessen mit Systemressourcen umgegangen wird.

Was sollen die Teilnehmer verstehen?

- Die Teilnehmer sollen verstehen, wie eine typische zentralistische Logdaten-Verwaltung aufgebaut ist und welche Auswirkungen sie auf die Architektur hat.
- Die Teilnehmer sollen verstehen, wie eine typische zentralistische Metriken-Pipeline aufgebaut ist (Erfassen, Sammeln & Samplen, Persistieren, Abfragen, Visualisieren) und welche Auswirkungen sie auf die Architektur hat (Performance-Overhead, Speicherverbrauch,...).

5.3. Referenzen

[Wolff 2016], [Nygard 2018]

6. Case Study

Dauer: 90 Min.	Übungszeit: 60 Min.
----------------	---------------------

Im Rahmen einer Lehrplan-konformen Schulung muss eine Fallstudie die Konzepte praktisch erläutern.

6.1. Begriffe und Konzepte

Die Case Study führt keine neuen Begriffe, Prinzipien und Konzepte ein.

6.2. Lernziele

Die Case Study soll keine neuen Lernziele vermitteln, sondern die Themen durch praktische Übungen vertiefen und die Praxis verdeutlichen.

6.3. Referenzen

Keine. Schulungsanbieter sind für die Auswahl und Beschreibung von Beispielen verantwortlich.

7. Ausblick

Dauer: 120 Min.	Übungszeit: 0 Min.
-----------------	--------------------

Der Ausblick stellt fortgeschrittene Themen dar, in die sich Teilnehmer vertiefen können. So erreichen sie ein tieferes Verständnis für die Herausforderungen bei der Umsetzung flexibler Systeme. Außerdem lernen sie weitere Einflussfaktoren auf die Auswahl von Technologien kennen.

7.1. Begriffe und Konzepte

- Konsistenzmodelle: ACID, BASE, Partitionierung, CAP
- Resilience: Resilient Software Design, Stabilität, Verfügbarkeit, Graceful Degradation, Circuit Breaker, Bulkhead

7.2. Lernziele

LZ 7-1: Konsistenzmodelle

- Was sollen die Teilnehmer können?
 - Die Teilnehmer sollen verschiedene Konsistenzmodelle kennen. Die Tradeoffs der verschiedenen Konsistenzmodelle sollten sie grundlegend kennen.
 - Abhängig von den Anforderungen und Rahmenbedingungen sollen sie entscheiden können, ob traditionelle Stabilitätsansätze hinreichend sind oder ob Resilient Software Design erforderlich ist.
- Was sollen die Teilnehmer verstehen?
 - Die Notwendigkeit für ACID Transaktionen ist wesentlich geringer, als häufig angenommen wird.
 - Unterschiedliche Skalierungs-, Verteilungs- und Verfügbarkeitsanforderungen erfordern unterschiedliche Konsistenzmodelle.
 - Das CAP-Theorem beschreibt ein Spektrum, in dem man abhängig von den gegebenen Anforderungen sehr feingranular ein geeignetes Konsistenzmodell wählen kann.
 - BASE-Transaktionen garantieren Konsistenz, sie sind nur nicht unbedingt atomar und isoliert wie ACID-Transaktionen, weshalb vorübergehend Inkonsistenzen sichtbar werden können.
- Was sollen die Teilnehmer kennen?
 - Eigenschaften von und Unterschiede zwischen ACID- und BASE-Transaktionen
 - Einige Produktbeispiele aus unterschiedlichen Kategorien (z. B. NoSQL, Konfigurationswerkzeuge, Service Discovery)
 - CAP zur Beschreibung und Erklärung von Konsistenzmodellen

LZ 7-2: Resilience Patterns

- Was sollen die Teilnehmer können?
 - Abhängig von den Anforderungen und Rahmenbedingungen sollen sie entscheiden können, ob traditionelle Stabilitätsansätze hinreichend sind oder ob Resilient Software Design erforderlich ist.
- Was sollen die Teilnehmer verstehen?
 - Traditionelle Stabilitätsansätze (Fehlervermeidungsstrategien) auf Infrastrukturebene sind für heutige verteilte, hochvernetzte Systemlandschaften in der Regel nicht mehr hinreichend.

- Es gibt keine Silver Bullet für Resilient Software Design, d. h. die relevanten Maßnahmen und eingesetzten Muster und Prinzipien hängen von den Anforderungen, den Rahmenbedingungen und den beteiligten Personen ab.
- Was sollen die Teilnehmer kennen?
 - Die Formel für Verfügbarkeit und die unterschiedlichen Ansätze, die Verfügbarkeit zu maximieren (Maximierung von MTBF, Minimierung von MTTR)
 - Isolation und Latenzüberwachung als sinnvolle Einstiegsprinzipien von Resilient Software Design
 - Grundlegende Resilience-Muster wie Bulkhead, Circuit Breaker, Redundanz, Failover

7.3. Referenzen

[Tanenbaum, van Steen 2006], [Lamport 1998], [Brewer 2000], [Takada 2013], [Nygard 2018], [Hanmer 2007], [Hamilton 2007]

Referenzen

Dieser Abschnitt enthält Quellenangaben, die ganz oder teilweise im Curriculum referenziert werden.

B

- [\[Brewer 2000\]](#) Eric Brewer, Towards Robust Distributed Systems, PODC Keynote, July-19-2000

E

- [\[Eric Evans 2003\]](#) Eric Evans: Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison- Wesley Professional, 2003

H

- [\[Hamilton 2007\]](#) James Hamilton, On Designing and Deploying Internet-Scale Services, 21st LISA Conference 2007
- [\[Hanmer 2007\]](#) Robert S. Hanmer, Patterns for Fault Tolerant Software, Wiley, 2007
- [\[Hohpe, Woolf 2003\]](#) Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley, 2003, ISBN 978-0-32120-068-6
- [\[Humble, et al. 2010\]](#) Jez Humble, David Farley: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010, ISBN 978-0-32160-191-9
- [\[Humble, et al. 2014\]](#) Jez Humble, Barry O'Reilly, Joanne Molesky: Lean Enterprise: Adopting Continuous Delivery, DevOps, and Lean Startup at Scale, O'Reilly 2014, ISBN 978-1-44936-842-5

L

- [\[Lewis, Fowler, et al. 2013\]](#) James Lewis, Martin Fowler, et al.: Microservices - <http://martinfowler.com/articles/microservices.html>, 2013
- [\[Lamport 1998\]](#) Leslie Lamport, The Part-Time Parliament, ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169

N

- [\[Sam Newman 2021\]](#) Sam Newman: Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2nd ed. edition, 2021, ISBN 978-1-49203-402-5
- [\[Nygard 2018\]](#) Michael T. Nygard, Release It!, 2. Auflage, Pragmatic Bookshelf, 2018

O

- [\[Parecki et al. 2006\]](#) Aaron Parecki et al., Explaining OAuth 2.0 <http://oauth.net/>

P

- [\[Parnas 1972\]](#) David Parnas, On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM 15(12):1053–1058, 1972.

T

- [Takada 2013] Mikito Takada, Distributed Systems for Fun and Profit, <http://book.mixu.net/distsys/> (Guter Einstieg und Überblick)
- [Tanenbaum, van Steen 2006] Andrew Tanenbaum, Marten van Steen, Distributed Systems – Principles and Paradigms, Prentice Hall, 2nd Edition, 2006

V

- [Vossen, Haselmann, Hoeren 2012] Gottfried Vossen, Till Haselmann, Thomas Hoeren: Cloud-Computing für Unternehmen: Technische, wirtschaftliche, rechtliche und organisatorische Aspekte, dpunkt, 2012, ISBN 978-3- 89864-808-0

W

- [Wolff 2016] Eberhard Wolff: Continuous Delivery: Der pragmatische Einstieg, 2. Auflage, dpunkt, 2016, ISBN 978-3-86490-371-7
- [Wolff 2018] Eberhard Wolff: Microservices - Grundlagen flexibler Software Architekturen, 2. Auflage, dpunkt, 2018, ISBN 978-3-86490-555-1
- [Wolff, Müller, Löwenstein 2013] Eberhard Wolff, Stephan Müller, Bernhard Löwenstein: PaaS - Die wichtigsten Java Clouds auf einen Blick, entwickler.press, 2013

1

- [12 Factor App 2012] 12 Factor App, Guidelines for building apps on Heroku <http://12factor.net/>