Curriculum für

Certified Professional for Software Architecture (CPSA)®

Foundation Level

2025.1-RC5-DE-20250106

Inhaltsverzeichnis

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 04-06 [ehemaliges LZ 3-07]: Schnittstellen dokumentieren (R1)
- LZ 04-07 [ehemaliges LZ 3-06]: Querschnittsthemen dokumentieren und kommunizieren (R2)
- LZ 04-08 [ehemaliges LZ 3-08]: Architekturentscheidungen erläutern und dokumentieren (R1-R2)
- LZ 04-09 [ehemaliges LZ 3-09]: Weitere Hilfsmittel und Werkzeuge zur Dokumentation kennen (R3)
- LZ 05-01: Gründe für Architekturanalyse kennen (R1)
- LZ 05-02 [ehemaliges LZ 4-3 und 4-4]: Qualitäten eines Softwaresystems analysieren (R1, R3)
- LZ 05-03: Konformität mit Architekturentscheidungen bewerten (R2)
- LZ 06-01: Bezug von Anforderungen und Randbedingungen zur Lösung erfassen (R3)
- LZ 06-02: Technische Umsetzung einer Lösung nachvollziehen (R3)

Einleitung

Was dieser Lehrplan enthält

Dieser Lehrplan für den *Certified Professional for Software Architecture - Foundation Level* (CPSA-F) beinhaltet die Lernziele, die man beherrschen sollte, um die Rolle Softwarearchitekt:in zu übernehmen.

Seine Struktur orientiert sich an den grundlegenden Aktivitäten und Verantwortlichkeiten der Softwarearchitektur als Rolle:

- Anforderungen und Randbedingungen klären
- Entwurf und Entwicklung von Softwarearchitekturen, dabei strukturelle und konzeptionelle Entscheidungen treffen
- Beschreiben und Kommunizieren von Softwarearchitekturen für verschiedene Stakeholder
- Analysieren und Bewerten von Softwarearchitekturen

Was vermittelt eine Foundation-Level-Schulung?

Lizenzierte Schulungen zum *Certified Professional for Software Architecture – Foundation Level* (CPSA-F) vermitteln grundlegende Kenntnisse und Fertigkeiten für den Entwurf einer angemessenen Softwarearchitektur für kleine und mittlere IT-Systeme. Die Teilnehmenden erweitern und vertiefen ihre bestehenden Erfahrungen und Fähigkeiten in der Softwareentwicklung, um relevante Vorgehensweisen, Methoden und Prinzipien für die Entwicklung von Softwarearchitekturen. Durch das Gelernte können sie auf Grundlage angemessen detaillierter Anforderungen und Randbedingungen eine adäquate Softwarearchitektur entwerfen, kommunizieren, analysieren, bewerten und weiterentwickeln. Schulungen zum CPSA-F vermitteln Grundlagenwissen unabhängig von spezifischen Entwurfsmethoden, Vorgehensmodellen, Programmiersprachen oder Werkzeugen. Dadurch können die Teilnehmenden ihre erworbene Fertigkeiten auf ein breites Spektrum von Einsatzfällen anwenden.

Im Mittelpunkt steht der Erwerb folgender Fähigkeiten:

- mit anderen Beteiligten aus den Bereichen Anforderungsmanagement, Projektmanagement, Entwicklung und Test wesentliche Architekturentscheidungen abzustimmen
- die wesentlichen Schritte beim Entwurf von Softwarearchitekturen zu verstehen sowie für kleine und mittlere Systeme selbstständig durchzuführen
- Softwarearchitekturen auf Basis von Sichten, Architekturmustern und technischen Konzepten zu dokumentieren und zu kommunizieren.

Weiterhin behandeln CPSA-F-Schulungen:

- den Begriff und die Bedeutung von Softwarearchitektur
- die Aufgaben und Verantwortung von Softwarearchitekten
- die Rolle von Softwarearchitekt:innen in Entwicklungsvorhaben
- State-of-the-Art-Methoden und -Praktiken zur Entwicklung von Softwarearchitekturen.

Abgrenzung

Dieser Lehrplan reflektiert den aus heutiger Sicht des iSAQB e.V. notwendigen und sinnvollen Inhalt zur Erreichung der Lernziele des CPSA-F. Er stellt keine vollständige Beschreibung des Wissensgebiets „Softwarearchitektur“ dar.

Folgende Themen oder Konzepte sind **nicht Bestandteil des CPSA-F**:

- konkrete Implementierungstechnologien, -frameworks oder -bibliotheken
- Programmierung oder Programmiersprachen
- Spezifische Vorgehensmodelle
- Grundlagen oder Notationen der Modellierung (wie etwa UML)
- Systemanalyse und Requirements Engineering (siehe dazu das Ausbildungs- und Zertifizierungsprogramm des IREB e. V., https://ireb.org, International Requirements Engineering Board)
- Test (siehe dazu das Ausbildungs- und Zertifizierungsprogramm des ISTQB e. V., https://istqb.org, International Software Testing Qualification Board)
- Projekt- oder Produktmanagement
- Einführung in spezifische Werkzeuge.

Ziel des Trainings ist es, die Grundlagen für den Erwerb der für den jeweiligen Einsatzfall notwendigen weiterführenden Kenntnisse und Fertigkeiten zu vermitteln.

Voraussetzungen

Der iSAQB e. V. kann in Zertifizierungsprüfungen die hier genannten Voraussetzungen durch entsprechende Fragen prüfen.

Teilnehmende sollten die im Nachfolgenden genannten Kenntnisse und/oder Erfahrung mitbringen. Insbesondere bilden substanzielle praktischen Erfahrungen aus der Softwareentwicklung im Team eine wichtige Voraussetzung zum Verständnis des vermittelten Lernstoffes und für eine erfolgreiche Zertifizierung.

- mehr als 18 Monate praktische Erfahrung in arbeitsteiliger Softwareentwicklung (d.h. in Teams), erworben durch Programmierung unterschiedlicher Systeme außerhalb der Ausbildung
- Kenntnisse und praktische Erfahrung in mindestens einer höheren Programmiersprache, insbesondere:
  - Konzepte der
    - Modularisierung (Pakete, Namensräume)
    - Parameterübergabe (*Call-by-Value, Call-by-Reference*)
    - Gültigkeit (*scope*), beispielsweise von Typ- oder Variablendeklaration und -definition
  - Grundlagen von Typsystemen (statische und dynamische Typisierung, generische Datentypen)
  - Fehler- und Ausnahmebehandlung in Software
  - Mögliche Probleme von globalem Zustand und globalen Variablen
- Grundlegende Kenntnisse von:
  - Modellierung und Abstraktion
  - Algorithmen und Datenstrukturen (etwa Listen, Bäume, HashTable, Dictionary/Map)
  - UML (Klassen-, Paket-, Komponenten- und Sequenzdiagramme) und deren Bezug zum Quellcode
  - Vorgehen beim Testen von Software (z. B. Unit- und Akzeptanztests)

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Grundbegriffe bzw. Unterschiede von imperativer, deklarativer, objektorientierter und funktionaler Programmierung
- praktische Erfahrung in
  - einer höheren Programmiersprache
  - Konzeption und Implementierung verteilt ablaufender Anwendungen, wie etwa Client/Server-Systeme oder Web-Anwendungen
  - technischer Dokumentation, insbesondere in der Dokumentation von Quellcode, Systementwürfen oder technischen Konzepten

Struktur, Dauer, Didaktik

Die in den nachfolgenden Kapiteln des Lehrplans genannten Zeiten sind lediglich Empfehlungen. Die Dauer einer Schulung sollte mindestens 3 Tage betragen, kann aber durchaus länger sein. Anbieter können sich durch Dauer, Didaktik, Art und Aufbau der Übungen sowie der detaillierten Kursgliederung voneinander unterscheiden. Insbesondere die Art (fachliche und technische Domänen) der Beispiele und Übungen können die jeweiligen Schulungsanbieter individuell festlegen.

Inhalt	Empfohlene Dauer (min)
1. Grundlagen	120
2. Anforderungen und Randbedingungen	180
3. Entwurf und Entwicklung	360
4. Beschreibung und Kommunikation	240
5. Analyse und Bewertung	90
6. Beispiele	90
Summe	1080

Lernziele und Prüfungsrelevanz

Die Kapitel des Lehrplans sind anhand von priorisierten Lernzielen gegliedert. Die Prüfungsrelevanz dieser Lernziele beziehungsweise deren Unterpunkte ist beim jeweiligen Lernziel ausdrücklich gekennzeichnet (durch Angabe der Kennzeichen R1, R2 oder R3, siehe nachstehende Tabelle).

Jedes Lernziel beschreibt die zu vermittelnden Inhalte inklusive ihrer Kernbegriffe und -konzepte. Bezüglich der Prüfungsrelevanz verwendet der Lehrplan folgende Kategorien:

ID	Lernziel-Kategorie	Bedeutung	Relevanz für Prüfung
R1	Können	Diese Inhalte sollen die Teilnehmenden nach der Schulung selbstständig anwenden können. Innerhalb der Schulung werden diese Inhalte durch Übungen und Diskussionen abgedeckt.	Inhalte **werden** geprüft.
R2	Verstehen	Diese Inhalte sollen die Teilnehmenden grundsätzlich verstehen. Sie werden in Schulungen i. d. R. nicht durch Übungen vertieft.	Inhalte **können** geprüft werden.
R3	Kennen	Diese Inhalte (Begriffe, Konzepte, Methoden, Praktiken oder Ähnliches) können das Verständnis unterstützen oder das Thema motivieren. Sie werden in Schulungen bei Bedarf thematisiert.	Inhalte **werden nicht** geprüft.

Bei Bedarf enthalten die Lernziele Verweise auf weiterführende Literatur, Standards oder andere Quellen. Die Abschnitte "Begriffe und Konzepte" zu Beginn jedes Kapitels zeigen Worte, die mit dem Inhalt des Kapitels in Verbindung stehen und z. T. auch in den Lernzielen verwendet werden.

Aktuelle Version sowie öffentliches Repository

Den aktuellen Stand dieses Dokumentes finden Sie auf der offiziellen Download-Seite unter https://isaqb-org.github.io/.

Das Dokument wird in einem öffentlichen Repository unter https://github.com/isaqb-org/curriculum-foundation gepflegt, alle Änderungen sind dort sichtbar.

Bitte melden Sie eventuelle Probleme in unserem öffentlichen Issue Tracker bei https://github.com/isaqb-org/curriculum-foundation/issues.

1. Grundbegriffe von Softwarearchitektur

Dauer: 120 Min.	Übungszeit: Keine

Zielsetzung

Ziel dieses Abschnitts ist es, den Teilnehmenden ein grundlegendes Verständnis der wichtigsten Begriffe und Konzepte von Softwarearchitektur zu vermitteln. Sie lernen verschiedene Definitionen und deren Gemeinsamkeiten kennen, verstehen die wesentlichen Ziele und Vorteile von Softwarearchitektur und können diese anderen Stakeholdern vermitteln. Weiterhin können sie die wichtigsten Aufgaben und Verantwortlichkeiten von Softwarearchitektinnen und Softwarearchitekten benennen und erklären. Außerdem wird deren Rolle im organisatorischen Kontext und ihre Interaktion mit anderen Stakeholdern thematisiert, um es den Teilnehmenden zu ermöglichen, einen effektiven Beitrag zu verschiedenartigen Softwareentwicklungsprojekten zu leisten.

Wesentliche Begriffe

Softwarearchitektur; Architekturdomänen; Struktur; Bausteine; Komponenten; Schnittstellen; Beziehungen; Querschnittsthemen; Nutzen von Softwarearchitektur; Softwarearchitekt:innen und deren Verantwortlichkeiten; Rolle; Aufgaben und benötigte Fähigkeiten; Stakeholder und deren Anliegen; Anforderungen; Randbedingungen; Einflussfaktoren

Lernziele

LZ 01-01: Definitionen von Softwarearchitektur verstehen (R1)

Softwarearchitekt:innen kennen und verstehen die Gemeinsamkeiten vieler Definitionen von Softwarearchitektur:

- Komponenten/Bausteine mit Schnittstellen und Beziehungen
- Bausteine als allgemeiner Begriff, Komponenten als eine spezielle Ausprägung davon
- Strukturen, Querschnittsthemen, Prinzipien
- Architekturentscheidungen und ihre Auswirkungen auf das gesamte System und seinen Lebenszyklus

Referenzen

[ISO 42010], [Bass+2021], [Kruchten 2004], [Starke+2023a]

LZ 01-02: Ziele und Nutzen von Softwarearchitektur verstehen und erläutern (R1)

Softwarearchitekt:innen können die folgenden wesentlichen Ziele und Nutzen der Softwarearchitektur begründen:

- Entwurf, Implementierung, Pflege und Betrieb von Systemen zu unterstützen
- funktionale Anforderungen zu erreichen bzw. deren Erfüllbarkeit sicherzustellen
- Anforderungen wie Zuverlässigkeit, Wartbarkeit, Änderbarkeit, Sicherheit, Energieeffizienz zu erreichen
- Verständnis für Strukturen und Konzepte des Systems bei allen relevanten Stakeholdern sicherstellen
- Komplexität systematisch zu reduzieren
- architekturrelevante Richtlinien für Implementierung und Betrieb zu spezifizieren

LZ 01-03: Langfristige Auswirkungen von Softwarearchitektur verstehen (R3)

Softwarearchitekt:innen können:

- die Auswirkungen von Architekturentscheidungen auf die langfristige Entwicklung eines Systems analysieren
- den Zusammenhang zwischen Architekturentscheidungen und der zukünftigen Anpassbarkeit und Wartbarkeit des Systems erläutern
- einschätzen, wie sich Änderungen von Anforderungen, Technologien oder Systemumgebung auf bestehende Architekturentscheidungen auswirken
- die langfristigen Konsequenzen von Architekturentscheidungen auf verschiedene Qualitätsmerkmale des Systems erkennen
- die Wechselwirkungen zwischen IT-Systemen und den unterstützten Geschäfts- und Betriebsprozessen erläutern

Referenzen

[Bass+2021], [Lehman 1980], [Wiki-LehmansLaws], [Lilienthal 2024], [Ford 2017], [Rajlich+2000], [Richards+2020]

LZ 01-04: Aufgaben und Verantwortung von Softwarearchitekt:innen verstehen (R1)

Softwarearchitekt:innen tragen die Verantwortung für die Erreichung der Anforderungen und die Entwicklung der Architektur der Lösung. Sie müssen diese Verantwortung, abhängig vom jeweiligen Prozess- oder Vorgehensmodell, mit der Gesamtverantwortung der Projektleitung oder anderen Rollen koordinieren.

Aufgaben und Verantwortung von Softwarearchitekt:innen:

- Anforderungen und Randbedingungen klären und hinterfragen. Notwendige Verfeinerungen mit den entsprechenden Stakeholdern koordinieren und abstimmen.
- Strukturentscheidungen hinsichtlich Systemzerlegung und Bausteinstruktur treffen, dabei Abhängigkeiten und Schnittstellen zwischen den Bausteinen festlegen
- Querschnittsthemen entscheiden (beispielsweise Persistenz, Kommunikation, GUI)
- Softwarearchitektur auf Basis von Sichten, Architekturmustern sowie technischen und Querschnittsthemen kommunizieren und dokumentieren
- Umsetzung und Implementierung der Architektur begleiten, Rückmeldungen der beteiligten Stakeholder bei Bedarf in die Architektur einarbeiten, Konsistenz von Quellcode und Softwarearchitektur prüfen und sicherstellen
- Softwarearchitektur analysieren und bewerten, insbesondere hinsichtlich Risiken bezüglich der Erreichung von Anforderungen.
- Die Konsequenzen von Architekturentscheidungen erkennen, aufzeigen und gegenüber anderen Stakeholdern argumentieren

Sie sollen selbstständig die Notwendigkeit von Iterationen bei allen Aufgaben erkennen und Möglichkeiten für entsprechende Rückmeldung aufzeigen.

LZ 01-05 [ehemaliges LZ 1-09]: Abgrenzung zu anderen Architekturdomänen (R3)

Der Fokus des iSAQB CPSA-Foundation Level liegt auf Strukturen und Konzepten einzelner Softwaresysteme.

Darüber hinaus kennen Softwarearchitekt:innen weitere Architekturdomänen, beispielsweise:

- Unternehmens-IT-Architektur (*Enterprise IT Architecture*): Struktur von Anwendungslandschaften
- Geschäfts- bzw. Prozessarchitektur (*Business and Process Architecture*): Struktur von u.a. Geschäftsprozessen
- Informationsarchitektur: systemübergreifende Struktur und Nutzung von Information und Daten
- Datenarchitektur: Semantik und Organization von Daten
- Infrastruktur- bzw. Technologiearchitektur: Struktur der technischen Infrastruktur, Hardware, Netze etc.
- Hardware- oder Prozessorarchitektur (für hardwarenahe Systeme)
- Systemarchitektur (verschiedene Bedeutungen, abhängig von der Definition des Begriffs "System")

Diese Architekturdomänen sind nicht inhaltlicher Fokus vom CPSA-F.

LZ 01-06: Rolle von Softwarearchitekt:innen mit anderen Stakeholdern in Beziehung setzen (R1)

Softwarearchitekt:innen können ihre Rolle erklären. Sie sollten ihren Beitrag zur Systementwicklung in Verbindung mit anderen Stakeholdern und Organisationseinheiten kontextspezifisch ausgestalten, insbesondere zu:

- Produktmanagement, Product-Owner
- Projektleitung und -management
- Anforderungsanalytiker:innen (System-/Businessanalyse, Anforderungsmanagement, Fachbereich)
- Entwicklung
- Qualitätssicherung und Test
- IT-Betrieb (Produktion, Rechenzentren), zutreffend primär für Informationssysteme
- Hardwareentwicklung und Systemarchitektur, zutreffend primär für eingebettete und hardwarenahe Systeme
- Unternehmensarchitektur, Architekturboard.

LZ 01-07: Bedeutung von Daten und Datenmodellen (R2)

Softwarearchitekt:innen verstehen die Bedeutung von Daten und Datenmodellen für die Architektur. Sie

- können Datenmodelle identifizieren, die maßgeblichen Einfluss auf die Architektur haben.
- können solche Datenmodelle systematisch entwerfen.
- verstehen den Unterschied zwischen Produkten und Summen in der Datenmodellierung.
- verstehen die Bedeutung der Entkopplung von Datenmodellen und ihrer Repräsentation in Datenbanken, Dateien und Übertragungsprotokollen.
- können die Auswirkungen von Daten auf Architekturentscheidungen z.B. in Bezug auf Speicherung, Sicherheit, Skalierbarkeit, Zuverlässigkeit, Performance usw. erläutern.

Referenzen

[Felleisen+2014], [Sperber+2023], [Sperber+2024], [Kleppmann 2017], [Ford+ 2021]

2. Anforderungen und Randbedingungen

Dauer: 90 Min.	Übungszeit: 90 Min.

Zielsetzung

Dieser Abschnitt vertieft das Verständnis der Teilnehmenden für Stakeholder-Anliegen, Anforderungen und Qualitäten von Softwaresystemen. Sie lernen, den Einfluss von Stakeholdern auf Architekturentscheidungen zu erkennen sowie Konflikte und Synergien im Kontext von Entwicklungsprojekten einzuschätzen. Durch die Auseinandersetzung mit verschiedenen Anforderungen und Einschränkungen erhalten sie einen Einblick in die effektive Berücksichtigung der Bedürfnisse der Stakeholder und der Projektvorgaben. Weiterhin erkennen sie die Bedeutung von Qualitäten eines Softwaresystems als entscheidende Faktoren für den Architekturentwurf. Sie können solche Anforderungen mit Hilfe von Szenarien formulieren.

Wesentliche Begriffe

Qualität; Qualitätsmerkmale; DIN/ISO 25010; Q42; Qualitätsszenarien; Kompromisse/Wechselwirkungen von Qualitätseigenschaften; Anforderungen; Randbedingungen; Stakeholder-Anliegen

Lernziele

LZ 02-01: Stakeholder-Anliegen verstehen (R1-R3)

Architekten können Stakeholder und deren Anliegen sowie deren Auswirkungen auf die Softwarearchitektur oder den Entwurfs- und Entwicklungsprozess identifizieren. (R2)

Beispiele für Stakeholder und ihre Anliegen (R3):

Interessengruppe	Anliegen der Interessengruppe
Produktmanagement	z. B. benötigte Zeit für die Umsetzung der Anforderungen
Entwicklungsteam	z. B. zu implementierende Komponenten und Schnittstellen, Protokolle, technische Anforderungen und Randbedingungen
Requirements Engineering, Product-Owner, Businss-Analyse, Fachbereiche	z. B. Erfüllung der Anforderungen
Projektmanagement	z. B. benötigte Zeit und Budget für die Umsetzung, verbundene Risiken des gewählten architekturellen Ansatzes
Qualitätssicherung und Test	z. B. isoliertes Testen von Komponenten
Betrieb	z. B. Infrastrukturanforderungen im Zusammenhang mit dem Betrieb des Systems

Softwarearchitekt:innen können potenzielle Konflikte zwischen kurz- und langfristigen Zielen identifizieren (z.B. Geschäfts- und Projektziele vs. Architektur- und Wartbarkeitsziele). Sie verstehen, dass sie die relevanten Stakeholder einbeziehen müssen, um diese Konflikte zu lösen. (R2)

Architekt:innen verstehen, dass nicht alle Anliegen der Stakeholder in Anforderungen umgesetzt werden können oder werden, aber dennoch berücksichtigt werden müssen. (R3)

Architekt:innen können die Anliegen der Stakeholder nutzen, um fehlende oder widersprüchliche Anforderungen zu entdecken und/oder Anforderungen und Einschränkungen an der Architektur zu validieren, z.B. in Stakeholder-Interviews. (R3)

LZ 02-02 [ehemaliges LZ 2-3]: Anforderungen und Randbedingungen klären und berücksichtigen können (R1-R3)

Softwarearchitekt:innen verstehen, dass sowohl Anforderungen als auch Randbedingungen Auswirkungen auf die Architektur und die Architekturarbeit haben können (R2). Sie sind in der Lage, Anforderungen und Randbedingungen zu klären und beim Architekturentwurf und im Entwicklungsprozess zu berücksichtigen. Sie verstehen, dass ihre Entscheidungen zu zusätzlichen Anforderungen führen oder Änderungen an bestehenden Anforderungen erforderlich machen können.

Sie erkennen und berücksichtigen den Einfluss von:

- produktbezogenen Anforderungen wie (R1)
  - funktionale Anforderungen
  - Qualitätsanforderungen
- Technologische Randbedingungen wie
  - bestehende oder geplante Hardware- und Software-Infrastruktur (R1)
  - technologische Beschränkungen für Datenstrukturen und Schnittstellen (R2)
  - Referenzarchitekturen, Bibliotheken, Komponenten und Frameworks (R1)
  - Programmiersprachen (R2)
- Organisatorischen Randbedingungen wie
  - Organisationsstruktur von Entwicklungsteams und Auftraggebenden (R1), insbesondere das Gesetz von Conway (R2)
  - Unternehmens- und Teamkultur (R3)
  - Partnerschaften und Kooperationen (R2)
  - Normen, Richtlinien und Prozessmodelle (z. B. Genehmigungs- und Freigabeprozesse) (R2)
  - Verfügbarkeit von Ressourcen wie Budget, Zeit und Personal (R1)
  - Verfügbarkeit, Qualifikation und Engagement von Mitarbeitenden (R1)
- Regulatorischen Randbedingungen wie (R2)
  - lokale und internationale rechtliche Einschränkungen
  - Vertrags- und Haftungsfragen
  - Datenschutzgesetze und Gesetze zum Schutz der Privatsphäre
  - Fragen der Einhaltung oder Verpflichtungen zur Beweislast
- Trends wie (R3)
  - Markttrends
  - Technologietrends (z. B. Cloud, Microservices, Container, generative KI oder LLMs)
  - Methodik-Trends (z. B. Agilität)

Softwarearchitekt:innen können beschreiben, wie diese Einflüsse auf Architekturentscheidungen wirken,

und sie können die Konsequenzen einer Änderung der Einflussfaktoren anhand von Beispielen erläutern (R2).

Referenzen

[IREB Foundation], [Bass+2021], [Ghandi+2024], [Starke 2024], [Richards+2020], [Pohl 2025]

LZ 02-03 [ehemaliges LZ 4-1]: Qualitäten eines Softwaresystems verstehen und erklären (R1)

Softwarearchitekt:innen wissen dass der Begriff "Qualität" in verschiedenen Kontexten unterschiedlich benutzt wird:

- im Kontext von Qualitätsmanagement im Sinne von "Güte" und
- im Sinne spezifischer Eigenschaften (eines Softwaresystems).

In diesem Lernziel geht es um letzteres.

Softwarearchitekt:innen können erklären dass:

- es unterschiedliche Taxonomien gibt, die Qualitäten von Softwaresystemen kategorisieren
- manche Kategorisierungen zwischen Funktionalität und Qualität unterscheiden, wie zum Beispiel IREB [IREB Foundation]
- Softwarearchitekt:innen die Qualitäten eines Softwaresystems beeinflussen können
- die Veränderung einer Qualität andere so beeinflussen kann, dass Abwägungen notwendig werden, wie z. B.
  - Konfigurierbarkeit versus Zuverlässigkeit
  - Speicherbedarf versus Leistungseffizienz
  - Sicherheit versus Benutzbarkeit
  - Laufzeitflexibilität versus Wartbarkeit.

Sie verstehen, dass

- manche Kategorisierungen zwischen Qualitätsanforderungen und funktionalen Anforderungen unterscheiden, wie zum Beispiel IREB
- sich eine einzelne Anforderung auf mehrere Qualitäten beziehen kann.

Referenzen

[IREB Foundation], [ISO 25010], [Bass+2021], [Q42]

LZ 02-04 [ehemaliges LZ 04-03]: Anforderungen an Qualitäten formulieren (R1-R3)

Softwarearchitekt:innen:

- können Szenarien für gegebene Qualitäten mit Kontext, Stimulus, Reaktion und Messung für veschiedene Zwecke formulieren, z. B. um Anforderungen zu klären, Input für Architekturbewertungen zu liefern usw. (R1)
- verstehen, dass eine Anforderung für eine gegebene Qualität eine Analysemethode spezifizieren sollte (siehe LZ 05-02 [ehemaliges LZ 4-3 und 4-4]: Qualitäten eines Softwaresystems analysieren (R1, R3)) (R1)

- wissen, dass die Verwendung einer Metrik als Ziel diese invalidieren kann (R2), wie z. B. in Goodhart's Law (R3) beschrieben

Referenzen

[Bass+2021], [Q42], [ISO 25010]

LZ 02-05 [ehemaliges LZ 1-08]: Explizite Aussagen vor impliziten Annahmen bevorzugen (R1)

Softwarearchitekt:innen:

- können Annahmen oder Voraussetzungen explizit darstellen und dadurch implizite Annahmen vermeiden
- wissen, dass implizite Annahmen potenzielle Missverständnisse zwischen beteiligten Stakeholdern bewirken

3. Entwurf und Entwicklung von Softwarearchitekturen

Dauer: 270 Min.	Übungszeit: 90 Min.

Zielsetzung

Dieser Abschnitt zielt darauf ab, die Teilnehmenden in die Lage zu versetzen, Architekturentscheidungen so zu treffen, dass Anforderungen der Stakeholder unter Berücksichtigung der Randbedingungen erfüllt werden können. Sie lernen, Architekturentwürfe zu entwickeln, fundierte Entscheidungen zur Systemzerlegung zu treffen und Abhängigkeiten zwischen Bausteinen zu gestalten. Zu diesem Zweck lernen sie, grundlegende Ansätze und Heuristiken bei der Architekturentwicklung anzuwenden. Sie erkennen die Bedeutung von Entwurfsprinzipien und Lösungsmustern und können diese anwenden. Darüber hinaus werden in diesem Abschnitt der Umgang mit Querschnittsthemen, die Prinzipien des Software-Deployments und die Herausforderungen verteilter Systeme thematisiert.

Wesentliche Begriffe

Entwurf; Vorgehen beim Entwurf; Architekturentscheidung; Sichten; Schnittstellen; technische Konzepte und Querschnittsthemen; Architekturmuster; Entwurfsmuster; Mustersprachen; Entwurfsprinzipien; Abhängigkeiten; Kopplung; Kohäsion; Top-down- und Bottom-up-Vorgehen; modellbasierter Entwurf; iterativer/inkrementeller Entwurf; Domain-Driven Design

Lernziele

LZ 03-01 [ehemaliges LZ 2-8, neue Inhalte]: Anforderungen durch Architektur erreichen (R1)

Softwarearchitekt:innen:

- verstehen, dass architektonische Aktivitäten davon geleitet werden sollten, spezifische Qualitäten zu erreichen oder zu verbessern
- können einen Architekturentwurf vorschlagen, der geeignet ist, Anforderungen zu erfüllen
- können einschätzen, welche Qualitäten sie durch bestimmte Aktivitäten oder Entscheidungen verbessern
- können mögliche Abwägungen zwischen Entwürfen sowie den entsprechenden Risiken identifizieren und kommunizieren

LZ 03-02 [ehemaliges LZ 2-02]: Softwarearchitekturen entwerfen (R1)

Softwarearchitekt:innen können:

- Softwarearchitekturen auf Basis bekannter funktionaler und Qualitätsanforderungen für nicht sicherheits- oder unternehmenskritische Softwaresysteme entwerfen und angemessen kommunizieren und dokumentieren
- Strukturentscheidungen hinsichtlich Systemzerlegung und Bausteinstruktur treffen, dabei Abhängigkeiten zwischen Bausteinen festlegen (siehe LZ 03-06 [ehemaliges LZ 2-07]: Abhängigkeiten von Bausteinen managen (R1))
- gegenseitige Abhängigkeiten und Abwägungen bezüglich Architekturentscheidungen erkennen und begründen
- Begriffe Blackbox und Whitebox erklären und zielgerichtet anwenden

- schrittweise Verfeinerung und Spezifikation von Bausteinen durchführen
- Architektursichten entwerfen, insbesondere Baustein-, Laufzeit- und Verteilungssicht (siehe LZ 04-05 [ehemaliges LZ 3-04]: Architektursichten erläutern und anwenden (R1))
- die aus Entscheidungen resultierenden Konsequenzen auf den Quellcode erklären
- domänenspezifische und technische Bestandteile in Architekturen trennen und diese Trennung begründen
- Risiken von Architekturentscheidungen identifizieren.

Referenzen

[Kruchten 1995], [Rozanski+2011], [Starke+2023a], [arc42], [Brown], [Starke 2024]

LZ 03-03 [ehemaliges LZ 2-01]: Vorgehen und Heuristiken zur Architekturentwicklung auswählen und anwenden können (R1,R3)

Softwarearchitekt:innen können grundlegende Vorgehensweisen der Architekturentwicklung benennen, erklären und anwenden, beispielsweise:

- Top-down- und Bottom-up-Vorgehen beim Entwurf, siehe [Gharbi+2024], [Starke 2024] (R1)
- Sichtenbasierte Architekturentwicklung, siehe [Rozanski+2011], [Kruchten 1995] (R1)
- Domain-Driven Design, siehe [Evans 2004] (R3)
- Evolutionäre Architektur, siehe [Ford 2017] (R3)
- Globale Analyse, siehe [Hofmeister+1999] (R3)
- Modellbasierter Entwurf (R3)

LZ 03-04 [ehemaliges LZ 2-06]: Entwurfsprinzipien erläutern und anwenden (R1-R3)

Softwarearchitekt:innen sind in der Lage zu erklären, was Entwurfsprinzipien sind. Sie können deren grundlegende Ziele und deren Anwendung im Hinblick auf Softwarearchitektur skizzieren. (R2)

Softwarearchitekt:innen sind in der Lage:

- die unten aufgeführten Gestaltungsprinzipien zu erläutern und mit Beispielen zu illustrieren
- zu erklären, wie diese Prinzipien angewendet werden sollen
- darzulegen, wie Anforderungen die Anwendung dieser Prinzipien beeinflussen
- die Auswirkungen der Entwurfsprinzipien auf die Implementierung zu erläutern
- Quellcode und Architektur zu analysieren, um zu beurteilen, ob diese Entwurfsprinzipien angewendet wurden oder angewendet werden sollten

Abstraktion (R2)

- im Sinne eines Vorgehens zur Erarbeitung zweckmäßiger Generalisierungen
- als eine Entwurfstechnik, bei dem die Bausteine von Abstraktionen und nicht von Implementierungen abhängen
- Schnittstellen als Abstraktionen

Modularisierung (R1)

- Geheimnisprinzip (Information Hiding) und Kapselung
- Trennung von Verantwortlichkeiten (Separation of Concerns - SoC)
- Lose, aber funktionell ausreichende Kopplung von Bausteinen, siehe LZ 03-06 [ehemaliges LZ 2-07]: Abhängigkeiten von Bausteinen managen (R1)
- Hohe Kohäsion
- Offen/geschlossen-Prinzip
- Dependency-Inversion-Prinzip - Umkehrung von Abhängigkeiten durch Schnittstellen oder ähnliche Abstraktionen

Konzeptionelle Integrität (R2-R3)

- bedeutet Einheitlichkeit (Homogenität, Konsistenz) von Lösungen für ähnliche Probleme zu erreichen (R2)
- als ein Mittel, um das Prinzip der geringsten Überraschung zu erreichen (*principle of least surprise* oder *principle of least astonishment* (POLA)) (R3)
- Liskov'sches Substitutionsprinzip als eine Möglichkeit, Konsistenz und konzeptionelle Integrität zu erreichen (R3).

Reduktion von Komplexität (R3)

- als Motiv der Prinzipien KISS, YAGNI und CUPID [Terhorst-North 2022]
- DRY (Don't Repeat Yourself) als eine Möglichkeit, Wiederholungen zu vermeiden

Erwarte Fehler (R2-R3)

- als Mittel für den Entwurf robuster und widerstandsfähiger Systeme (R3)
- als eine Verallgemeinerung des Robustheitsgrundsatzes (*Postel's law*) (R2)

SOLID Prinzipien (R3)

Softwarearchitekt:innen kennen Nutzen und Grenzen der SOLID Prinzipien: Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle

Referenzen

[Liskov 1994], [SOLID]

LZ 03-05 [ehemaliges LZ 1-06]: Zusammenhang zwischen Feedback-Schleifen und Risiko (R1, R2)

Softwarearchitekt:innen verstehen die Notwendigkeit von Iterationen, insbesondere bei unter Unsicherheit getroffenen Entscheidungen. Sie

- können den Einfluss von iterativem Vorgehen auf Architekturentscheidungen erläutern (hinsichtlich Risiken und Prognostizierbarkeit) (R1)
- können inkrementell und iterativ arbeiten und entscheiden (R2)
- verstehen die Notwendigkeit von Rückmeldungen zu Architekturentscheidungen (R1)

- können von anderen Stakeholdern systematisch Rückmeldung einholen. (R2)

LZ 03-06 [ehemaliges LZ 2-07]: Abhängigkeiten von Bausteinen managen (R1)

Softwarearchitekt:innen verstehen Abhängigkeiten und Kopplung zwischen Bausteinen und können diese gezielt einsetzen. Sie:

- kennen und verstehen unterschiedliche Arten der Kopplung von Bausteinen (beispielsweise Kopplung über Benutzung/Delegation, Nachrichten/Ereignisse, Komposition, Erzeugung, Vererbung, zeitliche Kopplung, Kopplung über Daten, Datentypen oder Hardware)
- verstehen, wie Abhängigkeiten die Kopplung vergrößern
- können mindestens die folgenden Arten von Kopplung unterscheiden:
  - statische und dynamische Kopplung
  - efferente und afferente Kopplung
- wissen, dass auf statische Kopplung zu verzichten und stattdessen dynamische Kopplung einzusetzen, die zugrundeliegende Kopplung nicht notwendigerweise reduziert
- können Kopplung erkennen und ihre Auswirkungen bewerten
- können begründete Entscheidungen treffen, ob eine Abhängigkeit in Anbetracht der Anforderungen und Randbedingungen angemessen ist oder entfernt werden sollte
- kennen Möglichkeiten zur Auflösung bzw. Reduktion von Kopplung und können diese anwenden, beispielsweise:
  - Muster
  - Grundlegende Entwurfsprinzipien
  - Externalisierung von Abhängigkeiten, d.h. konkrete Abhängigkeiten erst zur Installations- oder Laufzeit festlegen, etwa durch Anwendung von Dependency Injection (R3) (siehe auch LZ 03-08 [ehemaliges LZ 2-05]: Wichtige Architekturmuster beschreiben, erklären und angemessen anwenden (R1, R3)).

Referenzen

[Ford+ 2021]

LZ 03-07 [ehemaliges LZ 2-09]: Schnittstellen entwerfen und spezifizieren (R1-R3)

Softwarearchitekt:innen kennen die kritische Bedeutung von Schnittstellen für die Interaktion zwischen Architekturbausteinen oder zwischen dem System und externen Elementen. Sie können solche Schnittstellen entwerfen und spezifizieren.

Sie kennen:

- wünschenswerte Eigenschaften von Schnittstellen und können diese beim Entwurf erreichen (R1):
  - einfach zu erlernen, einfach zu benutzen, einfach zu erweitern
  - schwer zu missbrauchen
  - funktional vollständig aus Sicht der Nutzer:innen oder nutzender Bausteine.
- die Notwendigkeit unterschiedlicher Behandlung interner und externer Schnittstellen (R2)
- die Trennung zwischen Schnittstelle und Implementierung (R1):

    - Implementierungen können bei Bedarf ausgetauscht werden.
- unterschiedliche Charakteristika von Schnittstellen, beispielsweise (R3):
    - Transportkanäle (etwa: TCP/IP als Teil des OSI 7-Schichten Modells)
    - intern oder extern
    - lokal oder remote
    - synchron oder asynchron
    - binär (nur maschinenlesbar) oder textuell (auch menschenlesbar)
    - zustandslos oder zustandsbehaftet
    - Punkt-zu-Punkt oder Multipunkt (broadcast oder multicast)
    - Funktionsaufruf (etwa: Remote Procedure Call) oder Nachrichtenaustausch
    - Batch, Request/Response oder Streaming
- unterschiedliche Implementierungsansätze von Schnittstellen, etwa (R3):
    - ressourcenorientiert (REST, REpresentational State Transfer)
    - graphenartig (etwa: GraphQL)
    - serviceorientiert (etwa: WS-*/SOAP-basierte Webservices)

Siehe auch LZ 04-06 [ehemaliges LZ 3-07]: Schnittstellen dokumentieren (R1).

Referenzen

[Zimmermann+2022], [Geewax 2021]

LZ 03-08 [ehemaliges LZ 2-05]: Wichtige Architekturmuster beschreiben, erklären und angemessen anwenden (R1, R3)

Softwarearchitekt:innen können die folgenden Architekturmuster erklären und Beispiele dafür liefern (R1):

- Layer
- Pipes und Filter
- Microservices

Softwarearchitekt:innen können einige der folgendene Muster erklären, ihre Relevanz für konkrete Systeme erläutern und Beispiele dafür liefern (R3):

- Blackboard
- Broker
- CQRS (Command-Query-Responsibility-Segregation)
- Event sourcing
- Dependency Injection (siehe auch LZ 03-06 [ehemaliges LZ 2-07]: Abhängigkeiten von Bausteinen managen (R1))
- Integrations und Messaging-Patterns (z.B. aus [Hohpe+2004])
- MVC (Model View Controller), MVVM (Model View ViewModel), MVU (Model View Update), PAC (Presentation Abstraction Control)

- Plugin
- Ports and Adapters (Synonyme: Onion Architektur, Hexagonal Architektur, Clean Architektur)
- SOA (Service-Oriented Architektur)

Softwarearchitekt:innen kennen wesentliche Quellen für Architekturmuster, beispielsweise die POSA-Literatur (z. B. [Buschmann+1996]) und PoEAA ([Fowler 2002]) (für Informationssysteme). (R3)

Referenzen

[Buschmann+1996], [Buschmann+2007], [Eilebrecht+2024], [Fowler 2002], [Gamma+ 1994], [Hohpe+2004], [Pethuru 2017]

LZ 03-09 [ehemaliges LZ 2-05]: Wichtige Entwurfsmuster beschreiben, erklären und angemessen anwenden (R3)

Softwarearchitekt:innen kennen den Unterschied zwischen Architektur- und Entwurfsmustern. Sie können mehrere der folgenden Entwurfsmuster beschreiben, ihre Relevanz für die Architektur und konkrete Systeme erklären sowie Beispiele nennen.

- Combinator
- Schnittstellenmuster wie Adapter, Facade, und Proxy. Architekt:innen sollten wissen, dass diese Muster unabhängig von bestimmten Programmiersprachen oder Frameworks verwendet werden können.
- Interpreter
- Observer
- Remote procedure call
- Template Method und Strategy
- Visitor

Softwarearchitekt:innen kennen wesentliche Quellen für Entwurfsmuster, wie z.B. GOF und POSA.

Sie wissen:

- dass Muster ein Weg sind, bestimmte Qualitäten für gegebene Probleme und Anforderungen innerhalb gegebener Kontexte zu erreichen.
- dass es verschiedene Kategorien von Mustern gibt.
- zusätzliche Quellen für Muster, die sich auf ihre spezifische technische oder Anwendungsdomäne beziehen.

Referenzen

[Gamma+ 1994], [Buschmann+1996]

LZ 03-10 [ehemaliges LZ 2-04]: Querschnittsthemen identifizieren und Querschnittskonzepte entwerfen und umsetzen (R1)

Softwarearchitekt:innen können:

- die Bedeutung von Querschnittsthemen erklären

- solche Querschnittsthemen identifizieren
- Querschnittskonzepte entwerfen, unter anderem Persistenz, Kommunikation, GUI, Fehlerbehandlung, Nebenläufigkeit, Energieeffizienz
- mögliche wechselseitige Abhängigkeiten erkennen und beurteilen.

Softwarearchitekt:innen wissen, dass solche Querschnittskonzepte systemübergreifend wiederverwendbar sein können.

Siehe auch LZ 04-07 [ehemaliges LZ 3-06]: Querschnittsthemen dokumentieren und kommunizieren (R2).

Referenzen

[Starke 2024], [Starke+2023a]

LZ 03-11 [ehemaliges LZ 2-10]: Grundlegende Prinzipien von Software-Deployments kennen (R3)

Softwarearchitekt:innen:

- wissen, dass Software-Deployment der Prozess ist, durch den neue oder aktualisierte Software zur Benutzung bereitgestellt wird
- können grundlegende Konzepte des Deployments von Software benennen und erklären:
  - Automatisierung von Deployments
  - Wiederholbare Builds
  - Konsistente Umgebungen (z. B. durch Nutzung von unveränderlicher (*immutable*) Infrastruktur)
  - Alles liegt unter Versionskontrolle
  - Releases sind einfach zurückzunehmen

Referenzen

[Humble+2010]

LZ 03-12 [ehemaliges LZ 1-11]: Herausforderungen verteilter Systeme kennen (R3)

Softwarearchitekt:innen können:

- die Verteilung in einer gegebenen Software-Architektur identifizieren
- Konsistenzkriterien für ein gegebenes fachliches Problem analysieren
- Kausalität von Ereignissen in einem verteilten System erklären

Softwarearchitekt:innen wissen:

- dass Kommunikation in einem verteilten System fehlschlagen kann
- dass es bei verteilten Systemen Einschränkungen hinsichtlich der Konsistenz in Datenbanken gibt
- was das "Split-Brain"-Problem ist und warum es schwierig zu lösen ist
- dass es unmöglich ist, die exakte zeitliche Reihenfolge der Ereignisse in einem verteilten System zu bestimmen

Referenzen

[Tanenbaum+], [Buschmann+2007], [Ford+ 2021], [Miller+]

4. Beschreibung und Kommunikation von Softwarearchitekturen

Dauer: 180 Min.	Übungszeit: 60 Min.

Zielsetzung

Ziel dieses Abschnitts ist es, die Teilnehmer:innen in die Lage zu versetzen, Softwarearchitekturen so zu dokumentieren und zu kommunizieren, dass sie den Bedürfnissen wichtiger Stakeholder gerecht werden und den Entwicklungsprozess unterstützen. Der Schwerpunkt liegt auf dem Verständnis der grundlegenden Anforderungen an technische Dokumentationen, der Verwendung geeigneter Modelle und Notationen zur Beschreibung von Architekturen und der Anwendung wichtiger Architekturansichten. Außerdem lernen die Teilnehmer:innen, wichtige Architekturentscheidungen, Schnittstellen und Querschnittskonzepte zu dokumentieren, um eine klare, korrekte und für die Stakeholder relevante Dokumentation zu gewährleisten.

Wesentliche Begriffe

(Architektur-)Sichten; Strukturen; (technische) Konzepte; Dokumentation; Kommunikation; Beschreibung; Ausrichtung an Zielgruppen und Stakeholder-Anliegen; Meta-Strukturen und Templates zur Beschreibung und Kommunikation; Kontextabgrenzung; Bausteine; Bausteinsicht; Laufzeitsicht; Verteilungssicht; Knoten; Kanal; Verteilungsartefakte; Mapping von Bausteinen auf Verteilungsartefakte; Mapping von Verteilungsartefakten auf Knoten; Beschreibung von Schnittstellen und Architekturentscheidungen; UML; Werkzeuge zur Dokumentation

Lernziele

LZ 04-01 [ehemaliges LZ 3-01]: Anforderungen an technische Dokumentation erläutern und berücksichtigen (R1)

Softwarearchitekt:innen kennen die wesentlichen Anforderungen an technische Dokumentation und können diese bei der Dokumentation von Systemen berücksichtigen bzw. erfüllen:

- Verständlichkeit, Korrektheit, Effizienz, Angemessenheit, Wartbarkeit
- Orientierung von Form, Inhalt und Detailgrad an Zielgruppe der Dokumentation

Sie wissen, dass Verständlichkeit technischer Dokumentation nur von deren Zielgruppen beurteilt werden kann.

Referenzen

[Starke+2023a], [Zörner 2021], [Clements+2010]

LZ 04-02 [ehemaliges LZ 3-02]: Softwarearchitekturen beschreiben und kommunizieren (R1-R3)

Softwarearchitekt:innen nutzen Dokumentation zur Unterstützung bei Entwurf, Implementierung und Weiterentwicklung (auch genannt *Wartung* oder *Evolution*) von Systemen. (R2)

Softwarearchitekt:innen (R1):

- können Architekturen entsprechend der Anliegen der Stakeholder dokumentieren und kommunizieren und dadurch unterschiedliche Zielgruppen adressieren, z. B. Management, Entwicklungsteams, QS, andere Softwarearchitekt:innen sowie möglicherweise zusätzliche Stakeholder

- können die Beiträge unterschiedlicher Autorengruppen stilistisch und inhaltlich konsolidieren und harmonisieren
- können Maßnahmen entwickeln und umsetzen, die mündliche und schriftliche Kommunikation in Einklang miteinander halten und miteinander angemessen ausbalancieren
- kennen den Nutzen von Template-basierter Dokumentation
- wissen, dass verschiedene Eigenschaften der Dokumentation von Spezifika des Systems, seinen Anforderungen, Risiken, dem Entwicklungsvorgehen, der Organisation oder anderen Faktoren abhängen.

Sie können beispielsweise die folgenden Merkmale von Dokumentation je nach Situation anpassen (R3):

- Umfang und Detaillierungsgrad der benötigten Dokumentation
- das Dokumentationsformat
- die Zugänglichkeit der Dokumentation
- Formalitäten der Dokumentation (z. B. Diagramme, die einem Metamodell entsprechen, oder einfache Zeichnungen)
- formale Überprüfungen und Freigabeprozesse für die Dokumentation

LZ 04-03 [ehemaliges LZ 3-03]: Notations-/Modellierungsmittel für Beschreibung von Softwarearchitektur erläutern und anwenden (R2-R3)

Softwarearchitekt:innen kennen mindestens folgende UML-Diagramme zur Notation von Architektursichten:

- Klassen-, Paket-, Komponenten- (jeweils R2) und Kompositionsstrukturdiagramme (R3)
- Verteilungsdiagramme (R2)
- Sequenz- und Aktivitätsdiagramme (R2)
- Zustandsdiagramme (R3)

Softwarearchitekt:innen kennen Alternativen zu UML, beispielsweise (R3)

- ArchiMate
- SysML
- C4, siehe [Brown]
- Entity-Relationship Diagramme, siehe [Chen 1976]
- für Laufzeitsichten beispielsweise Flussdiagramme, nummerierte Listen oder Business-Process-Modelling-Notation (BPMN).

Referenzen

[UML], [ArchiMate], [SysML], [Brown], [Chen 1976]

LZ 04-04 [new]: Lernziel nicht gefunden (R3)

Softwarearchitekt:innen können unerwartete Situationen geschickt bewältigen.

Referenzen

[IETF HTTP]

LZ 04-05 [ehemaliges LZ 3-04]: Architektursichten erläutern und anwenden (R1)

Softwarearchitekt:innen können folgende Architektursichten anwenden:

- Kontextsicht (auch genannt Kontextabgrenzung)
  - enthält die externen Schnittstellen von Systemen
  - bei Bedarf differenziert nach fachlichem und technischem Kontext
- Baustein- oder Komponentensicht (Aufbau des Systems aus Softwarebausteinen)
- Laufzeitsicht (dynamische Sicht, Zusammenwirken der Softwarebausteine zur Laufzeit, Zustandsmodelle)
- Verteilungs-/Deploymentsicht (Hardware und technische Infrastruktur sowie Abbildung von Softwarebausteinen auf diese Infrastruktur)

Zusätzliche Sichten können nach Bedarf verwendet werden, um weitere Anliegen oder Anforderungen von Stakeholden zu berücksichtigen, z. B. Funktionale Sicherheit, Informations-, Betriebs- oder User-Interface Sicht (R3).

Referenzen

[Kruchten 1995], [Rozanski+2011], [Starke+2023a], [arc42], [Brown]

LZ 04-06 [ehemaliges LZ 3-07]: Schnittstellen dokumentieren (R1)

Softwarearchitekt:innen können sowohl interne als auch externe Schnittstellen dokumentieren.

Siehe auch LZ 03-07 [ehemaliges LZ 2-09]: Schnittstellen entwerfen und spezifizieren (R1-R3).

LZ 04-07 [ehemaliges LZ 3-06]: Querschnittsthemen dokumentieren und kommunizieren (R2)

Softwarearchitekt:innen können typische Querschnittsthemen und die dazugehörigen Lösungskonzepte (Querschnittskonzepte) adäquat dokumentieren und kommunizieren, z. B. Persistenz, Ablaufsteuerung, UI, Verteilung/Integration, Protokollierung.

Siehe auch LZ 03-10 [ehemaliges LZ 2-04]: Querschnittsthemen identifizieren und Querschnittskonzepte entwerfen und umsetzen (R1).

LZ 04-08 [ehemaliges LZ 3-08]: Architekturentscheidungen erläutern und dokumentieren (R1-R2)

Softwarearchitekt:innen können:

- Architekturentscheidungen systematisch herbeiführen, begründen, kommunizieren und dokumentieren
- gegenseitige Abhängigkeiten solcher Entscheidungen erkennen, kommunizieren und dokumentieren

Softwarearchitekt:innen kennen Architecture-Decision-Records (ADR, siehe [Nygard 2011]) und können diese zur Dokumentation von Entscheidungen einsetzen (R2).

Referenzen

[Nygard 2011]

LZ 04-09 [ehemaliges LZ 3-09]: Weitere Hilfsmittel und Werkzeuge zur Dokumentation kennen (R3)

Softwarearchitekt:innen kennen:

- Grundlagen mehrerer publizierter Frameworks zur Beschreibung von Softwarearchitekturen, beispielsweise:
  - ISO/IEC/IEEE 42010,
  - arc42,
  - C4, siehe [Brown]
- Ideen und Beispiele von Checklisten für die Erstellung, Dokumentation und Prüfung von Softwarearchitekturen
- mögliche Werkzeuge zur Erstellung und Pflege von Architekturdokumentation

Referenzen

[ISO 42010], [arc42], [Brown]

5. Analyse und Bewertung von Softwarearchitekturen

Dauer: 60 Min.	Übungszeit: 30 Min.

Zielsetzung

In diesem Abschnitt sollen Softwarearchitekt:innen die Fähigkeiten und Kenntnisse vermittelt werden, die sie für eine effektive Architekturanalyse benötigen. Sie lernen, Risiken zu erkennen, die Konformität mit Architekturentscheidungen zu bewerten und die Gesamtqualität eines Systems auf der Grundlage seines Designs und seiner Implementierung zu beurteilen. Durch das Verständnis verschiedener Analysemethoden wie Abnahmetests, Architekturmetriken, szenariobasierte Analyse und Kosten-Nutzen-Analysen können Architekt:innen sicherstellen, dass eine Softwarearchitektur den Anforderungen der Stakeholder entspricht und mit dem beabsichtigten Design übereinstimmt.

Wesentliche Begriffe

Architekturanalyse; Risikoidentifikation; Qualitätsanalysemethoden; Szenarien; szenariobasierte Analyse; Metriken; Tool-gestützte Analyse

Lernziele

LZ 05-01: Gründe für Architekturanalyse kennen (R1)

Softwarearchitekt:innen verstehen, dass es verschiedene mögliche Gründe gibt, um eine Architekturanalyse durchzuführen, zum Beispiel:

- Risiken und mögliche Verbesserungen im Architekturentwurf identifizieren (vor, während und nach der Implementierung)
- feststellen, ob der Architekturentwurf die Anforderungen erfüllt oder erfüllen wird
- erheben, inwiefern die Implementierung zu Architekturentscheidungen und Architekturentwurf passt
- überprüfen, inwiefern architekturrelevante Anliegen der Stakeholder berücksichtigt sind

LZ 05-02 [ehemaliges LZ 4-3 und 4-4]: Qualitäten eines Softwaresystems analysieren (R1, R3)

Softwarearchitekt:innen

- verstehen, dass für eine einzelne Qualität eines Softwaresystems verschiedene Analysemethoden zur Verfügung stehen können, wie z. B.:
  - Analyse der Ergebnisse von Akzeptanztests (R1)
  - quantitative Messung von Laufzeitverhalten (R1)
  - qualitative Auswertung durch Interviews, Umfragen, Penetrationstests etc. (R1)
  - szenariobasierte Analyse (R1)
  - Architektur-Metriken für Kopplung wie der Grad eingehender und ausgehender Abhängigkeiten (R1)
  - Kosten-Nutzen-Analyse (R3)
  - Architecture Trade-Off Analysis Method [Bass+2021] (R3)
- kennen Informationsquellen für Qualitätsanalyse:

- Anforderungsdokumentation (R1)
- Architekturdokumentation (R1)
- Architektur- und Entwurfsmodelle (R1)
- Quelltext (R1)
- Quelltext-bezogene Metriken wie z. B: Lines-of-Code, (zyklomatische) Komplexität (R1)
- Testfälle und ihre Testresultate (R1)
- Fehler und ihre Position im Quelltext, besonders Fehlercluster (R1)
- andere Dokumentation des Systems, wie z. B. Betriebs- und Testdokumentation (R1)
- Laufzeit-Logs und Metriken (R1)
- Revisionshistorie, wie z. B. wie Änderungsrate pro Komponente (R3)

Siehe auch LZ 02-03 [ehemaliges LZ 4-1]: Qualitäten eines Softwaresystems verstehen und erklären (R1), LZ 02-04 [ehemaliges LZ 04-03]: Anforderungen an Qualitäten formulieren (R1-R3).

Referenzen

[Bass+2021], [Starke+2023a], [Clements+2002], [ISO 25019], [Lilienthal 2019]

LZ 05-03: Konformität mit Architekturentscheidungen bewerten (R2)

Softwarearchitekt:innen können beurteilen, ob die Systemimplementierung mit dem Architekturentwurf und den Entscheidungen übereinstimmt, indem sie Methoden wie Code- und Architekturreviews oder toolgestützte Analysen einsetzen.

6. Beispiele für Softwarearchitekturen

Dauer: 90 Min.	Übungszeit: Keine

Dieser Abschnitt ist nicht prüfungsrelevant.

Lernziele

LZ 06-01: Bezug von Anforderungen und Randbedingungen zur Lösung erfassen (R3)

Softwarearchitekt:innen haben an mindestens einem Beispiel den Bezug von Anforderungen und Randbedingungen zu Lösungsentscheidungen erkannt und nachvollzogen.

Referenzen

[arc42], [Starke+2023b], [Hruschka+2021], [Zörner 2021], [AOSA]

LZ 06-02: Technische Umsetzung einer Lösung nachvollziehen (R3)

Softwarearchitekt:innen können anhand mindestens eines Beispiels die technische Umsetzung (Implementierung, technische Konzepte, eingesetzte Produkte, Lösungsstrategien) einer Lösung nachvollziehen.

Referenzen

[arc42], [Starke+2023b], [Hruschka+2021], [Zörner 2021], [AOSA]

Referenzen

- [AOSA] The Architecture of Open Source Applications. Edited by Amy Brown and Greg Wilson. Online: https://aosabook.org/en/.
- [arc42] arc42, the open-source template for software architecture communication, online: https://arc42.org. Maintained on https://github.com/arc42
- [ArchiMate] The ArchiMate® Enterprise Architecture Modeling Language, online: https://www.opengroup.org/archimate-forum/archimate-overview
- [Bass+2021] Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. 4th Edition, Addison Wesley 2021.
- [Brown] Simon Brown: The C4 model for visualising software architecture. https://c4model.com https://www.infoq.com/articles/C4-architecture-model.
- [IREB Foundation] Stan Bühne, Martin Glinz, Hans van Loen, Stefan Staal: Certified Professional for Requirements Engineering - Foundation Level - Syllabus - Version 3.2.0, IREB, 2024.
- [Burns 2018] Brendan Burns: Designing Distributed Systems, Patterns and Paradigms for Scalable, Reliable Services, OReilly 2018.
- [Buschmann+1996] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal: Pattern-Oriented Software Architecture (POSA): A System of Patterns. Wiley, 1996.
- [Buschmann+2007] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt: Pattern-Oriented Software Architecture (POSA): A Pattern Language for Distributed Computing, Wiley, 2007.
- [Clements+2002] Paul Clements, Rick Kazman, Mark Klein: Evaluating Software Architectures. Methods and Case Studies. Addison Wesley, 2002.
- [Clements+2010] Paul Clements, Felix Bachmann, Len Bass, David Garlan, David, James Ivers, Reed Little, Paulo Merson and Robert Nord: *Documenting Software Architectures: Views and Beyond*, 2nd edition, Addison Wesley, 2010
- [CloudNative] The Cloud Native Computing Foundation, online: https://www.cncf.io/
- [Eilebrecht+2024] Karl Eilebrecht, Gernot Starke: Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung (in German). 6th Edition Springer Verlag 2024.
- [Chen 1976] Chen, Peter (March 1976): *The Entity-Relationship Model - Toward a Unified View of Data*. ACM Transactions on Database Systems. 1 (1): 9–36..
- [Evans 2004] Eric Evans: *Domain-Driven Design: Tackling Complexity in the Heart of Software,* Addison-Wesley, 2004.
- [Felleisen+2014] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi: How to Design Programs. Second Edition. MIT Press, 2014. https://htdp.org/
- [Ford 2017] Neil Ford, Rebecca Parsons, Patrick Kua: Building Evolutionary Architectures: Support Constant Change. OReilly 2017.
- [Ford+ 2021] Neal Ford, Mark Richards, Pramod Sadalage und Zhamak Dehghani: Software Architecture: The Hard Parts. Modern Trade-Off Analyses for Distributed Architectures. OReilly 2021.
- [Fowler 2002] Martin Fowler: Patterns of Enterprise Application Architecture. (PoEAA) Addison-Wesley, 2002.

- [Ghandi+2024] Raju Gandhi, Mark Richards and Neal Ford. Head-First Software Architecture. OReilly 2024.
- [Gamma+ 1994] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
- [Geewax 2021] J. Geewax. API Design Patterns. Manning, 2021. This book lays out a set of design principles for building internal and public-facing APIs.
- [Geirhos 2015] Matthias Geirhos. Entwurfsmuster: Das umfassende Handbuch (in German). Rheinwerk Computing Verlag. 2015
- [Gharbi+2024] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 5. Auflage, dpunkt Verlag, Heidelberg 2024.
- [Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java (in German). Springer-Vieweg Verlag, 2. Auflage 2014.
- [Hofmeister+1999] Christine Hofmeister, Robert Nord, Dilip Soni: *Applied Software Architecture*, Addison-Wesley, 1999
- [Hohpe+2004] Hohpe, G. and WOOLF, B.A.: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, 2004
- [Hombergs 2024] Hombergs, Tom: Get Your Hands Dirty on Clean Architecture, Packt, 2nd edition 2024.
- [Humble+2010] Jez Humble, David Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Pearson International, 2010
- [Hruschka+2021] Peter Hruschka, Ivan Kostov and Wolfgang Reimesch: arc42-by-Example Vol 2: Architecture Documentation for Embedded Systems and IoT. Leanpub, 2021. https://leanpub.com/arc42byexample-volume2
- [IETF HTTP] Internet Engineering Task Force: RFC 9110, HTTP Semantics. Online: https://www.rfc-editor.org/rfc/rfc9110.html
- [iSAQB Downloads] iSAQB public download site. https://public.isaqb.org. Contains curricula and mock-examination.
- [iSAQB Glossary] Gernot Starke et. al. iSAQB Glossary of Software Architecture Terminology. Freely available from https://leanpub.com/isaqbglossary or its source repository https://github.com/isaqb-org/glossary/releases
- [iSAQB References] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Freely available https://leanpub.com/isaqbreferences.
- [ISO 25010] ISO/IEC 25010:2023(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. Terms and definitions online: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-2:v1:en
- [ISO 25019] ISO/IEC 25019:2023(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality-in-use model. Terms and definitions online: https://www.iso.org/obp/ui/#iso:std:iso-iec:25019:ed-1:v1:en
- [ISO 42010] ISO/IEC/IEEE 42010:2022, Software, systems and enterprise Architecture description, online: https://www.iso.org/standard/74393.html
- [Keeling 2017] Michael Keeling. Design It!: From Programmer to Software Architect. Pragmatic Programmer.

- [Kleppmann 2017] Martin Kleppmann: Designing Data-Intensive Applications. O'Reilly 2017.
- [Kruchten 1995] Philippe Kruchten: Architectural Blueprints—The "4+1" View Model of Software Architecture, IEEE Software 12 (6), November 1995, pp. 42-50
- [Kruchten 2004] Philippe Kruchten: The Rational Unified Process: An Introduction. 3rd edition. Addison-Wesley Professional 2004.
- [Lange 2021] Kenneth Lange: The Functional Core, Imperative Shell Pattern, online: https://www.kennethlange.com/functional-core-imperative-shell/
- [Lehman 1980] Meir M. Lehman: Programs, Life Cycles, and Laws of Software Evolution. Proceedings of the IEEE, 68(9), 1060-1076, 1980.
- [Wiki-LehmansLaws] Laws of Software Evolution. https://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution
- [Lilienthal 2024] Carola Lilienthal: Langlebige Softwarearchitekuren. 4. Auflage, dpunkt Verlag 2024.
- [Lilienthal 2019] Carola Lilienthal: Sustainable Software Architecture: Analyze and Reduce Technical Debt. dpunkt Verlag 2019.
- [Liskov 1994] Barbara H. Liskov, Jeannette M. Wing: A behavioral notion of subtyping. ACM Transactions on Programming Languages and Systems, Volume 16, Issue 6, 1994. <doi:10.1145/197320.197383>
- [Maguire 2019] Sandy Maguire: Algebra-Driven Design - Elegant Solutions from Simple Building Blocks. Leanpub, 2019.
- [Miller+] Heather Miller, Nat Dempkowski, James Larisch, Christopher Meiklejohn: Distributed Programming (to appear, but content-complete) https://github.com/heathermiller/dist-prog-book.
- [Newman 2021] Sam Newman. Building Microservices - Designing Fine-Grained Systems. O'Reilly 2nd edition 2021.
- [Terhorst-North 2022] Daniel Terhorst-North: CUPID - for joyful coding. See https://dannorth.net/2022/02/10/cupid-for-joyful-coding/.
- [Nygard 2011] Michael Nygard: Documenting Architecture Decision. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions. See also https://adr.github.io/
- [Pethuru 2017] Raj Pethuru et. al: Architectural Patterns. Packt 2017.
- [Pohl 2025] Klaus Pohl: Requirements Engineering - Fundamentals, Principles and Techniques. Springer 2025
- [Q42] arc42 Quality Model, online: https://quality.arc42.org.
- [Rajlich+2000] Václav T. Rajlich, Keith H. Bennett: A Staged Model for the Software Life Cycle. IEEE Computer 33(7): 66-71, 2000.
- [Read 2023] Jacqui Read: Communication Patterns - An Engineering Approach. A Guide for Developers and Architects. OReilly 2023.
- [Richards+2020] Mark Richards, Neal Ford: Fundamentals of Software Architecture - An Engineering Approach. OReilly 2020.
- [Rozanski+2011] Nick Rozanski, Eoin Woods: Software Systems Architecture - Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley, 2nd edition 2011.

- [SOLID] Samuel Oloruntoba and Anish Singh Walia: SOLID: The First 5 Principles of Object Oriented Design, https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.
- [Sperber+2023] Michael Sperber, Herber Klaeren: Schreibe Dein Programm! Tübingen University Press, 2023. https://www.deinprogramm.de/sdp/.
- [Sperber+2024] Michael Sperber, Stefan Wehr: Datenmodellierung mit Summen und Produkten, 2024. https://funktionale-programmierung.de/2024/11/25/sums-products.html. (English translation: Data Modeling with Sums and Products, 2024. https://funktionale-programmierung.de/2024/11/25/sums-products-english.html)
- [Starke 2024] Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden (in German). 10. Auflage, Carl Hanser Verlag 2024. Website: https://esabuch.de
- [Starke+2023a] Gernot Starke, Alexander Lorz: Software Architecture Foundation, CPSA Foundation® Exam Preparation. Van Haaren Publishing, 2nd edition, 2023.
- [Starke+2023b] Gernot Starke, Michael Simons, Stefan Zörner, Ralf D. Müller, and Hendrik Lösch: arc42-by-Example - Software Architecture Documentation in Practice. Leanpub, 3rd edition 2023. https://leanpub.com/arc42byexample
- [SysML] What is SysML https://sysml.org/. For diagrams, see also https://sysml.org/tutorials/sysml-diagram-tutorial/.
- [Tanenbaum+] Andrew Tanenbaum, Maarten van Steen: Distributed Systems, Principles and Paradigms. https://www.distributed-systems.net/.
- [UML] The UML reading room, collection of UML resources https://www.omg.org/technology/readingroom/UML.htm. See also https://www.uml-diagrams.org/.
- [Yorgey 2012] Brent A. Yorgey, Monoids: Theme and Variations. Proceedings of the 2012 Haskell Symposium, September 2012 https://doi.org/10.1145/2364506.2364520
- [Zimmermann+2022] Olaf Zimmermann, Mirko Stocker, Daniel Lübke, Uwe Zdun, Cesare Pautasso: Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges. Addison-Wesley, 2022.
- [Zörner 2021] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. 3. Auflage, Carl Hanser Verlag, 2021.