CPSA Certified Professional for Software Architecture®

- Foundation Level Curriculum -

Version V5.0-RC-5-DE, 25. Aug 2019

Inhaltsverzeichnis

Rechtliches

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® erwerben. Für den Erwerb des Zertifikats ist es gestattet, die Textdokumente und/oder Lehrpläne zu nutzen, indem eine Arbeitskopie für den eigenen Rechner erstellt wird. Soll eine darüber hinaus gehende 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 unter keine der vorstehenden Kategorien 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.

Unresolved directive in foundation-curriculum.adoc - include::/home/travis/build/isaqb-org/curriculum-foundation/build/learning-objectives.adoc[tags=FEEDBACK;DE;]

Einleitung

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 selbständig durchzuführen
- Softwarearchitekturen auf Basis von Sichten, Architekturmustern und technischen Konzepten zu dokumentieren und zu kommunizieren.

Darüber hinaus behandeln CPSA-F-Schulungen:

- den Begriff und die Bedeutung von Softwarearchitektur
- die Aufgaben und Verantwortung von Softwarearchitekten
- die Rolle von Softwarearchitekten 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., http://ireb.org, International Requirements Engineering Board)
- Test (siehe dazu das Ausbildungs- und Zertifizierungsprogramm des ISTQB e. V., http://istqb.org, International Software Testing Qualification Board)
- Projekt- oder Produktmanagement
- Einführung in konkrete 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.

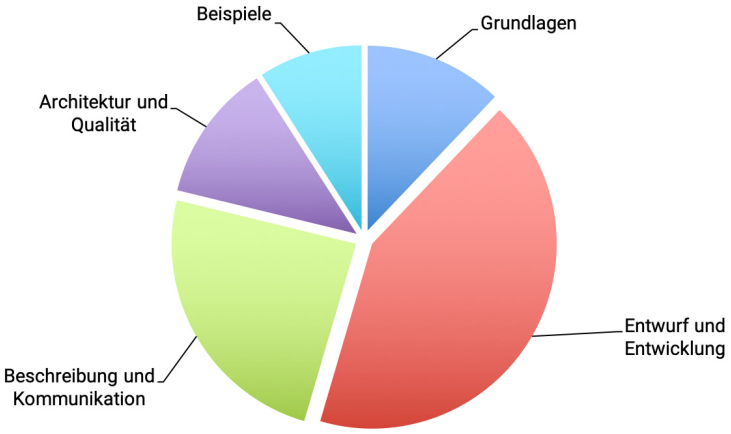
- mehr als 18 Monate praktische Erfahrung in arbeitsteiliger Softwareentwicklung (d.h. in Teams), erworben durch Programmierung unterschiedlicher Systeme außerhalb der Ausbildung
- Kenntnisse und praktische Erfahrung in mindestens einer höheren Programmiersprache, insbesondere:
  - Konzepte der
    - 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

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Grundbegriffe bzw. Unterschiede von imperativer, deklarativer, objektorientierter und funktionaler Programmierung
- praktische Erfahrung in
  - einer objektorientierten Programmiersprache (etwa Java oder C#);
  - 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.



Empfohlene Zeitaufteilung

Inhalt	Empfohlene Dauer (min)
1. Grundlagen	120
2. Entwurf und Entwicklung	420
3. Beschreibung und Kommunikation	240
4. Architektur und Qualität	120
5. Beispiele	90
Summe	990

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 Teilnehmer nach der Schulung selbständig anwenden können. Innerhalb der Schulung werden diese Inhalte durch Übungen und Diskussionen abgedeckt.	Inhalte **werden** geprüft.
R2	Verstehen	Diese Inhalte sollen die Teilnehmer 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.

1. Grundbegriffe

Dauer: 120 Min.	Übungszeit: Keine

Wesentliche Begriffe

Softwarearchitektur; Architekturdomänen; Struktur; Bausteine; Komponenten; Schnittstellen; Beziehungen; Querschnittskonzepte; Nutzen von Softwarearchitektur; Softwarearchitekten und deren Verantwortlichkeiten; Rolle; Aufgaben und benötigte Fähigkeiten; Stakeholder und deren Anliegen; funktionale Anforderungen; Qualitätsanforderungen; Randbedingungen; Einflussfaktoren; Typen von IT-Systemen (eingebettete Systeme; Echtzeitsysteme; Informationssysteme etc.)

Lernziele

LZ 1-1: Definitionen von Softwarearchitektur diskutieren (R1)

Softwarearchitekten kennen mehrere Definitionen von Softwarearchitektur (u. a. ISO 42010/IEEE 1471, SEI, Booch etc.) und können deren Gemeinsamkeiten benennen:

- Komponenten/Bausteine mit Schnittstellen und Beziehungen
- Bausteine als allgemeiner Begriff, Komponenten als eine spezielle Ausprägung davon
- Strukturen, Querschnittskonzepte, Prinzipien
- Architekturentscheidungen mit systemweiten oder den gesamten Lebenszyklus betreffenden Konsequenzen

LZ 1-2: Ziele von Softwarearchitektur verstehen und herausstellen (R1)

Wesentliche Ziele von Softwarearchitektur sind:

- Qualitätsanforderungen wie Zuverlässigkeit, Wartbarkeit, Änderbarkeit, Sicherheit sowie funktionale Anforderungen zu erreichen
- Erstellung und Wartung von Software, insbesondere die Implementierung zu unterstützen
- Verständnis für Strukturen und Konzepte des Systems zu vermitteln, bezogen auf sämtliche relevanten Stakeholder.

LZ 1-3: Softwarearchitektur in Software-Lebenszyklus einordnen (R2)

Softwarearchitekten können Ihre Aufgaben und Ergebnisse in den gesamten Lebenszyklus von IT-Systemen einordnen. Sie können:

- Konsequenzen von Änderungen bei funktionalen Anforderungen, Qualitätsanforderungen, Technologien oder der Systemumgebung im Bezug auf die Softwarearchitektur erkennen
- inhaltliche Zusammenhänge zwischen IT-Systemen und den unterstützten Geschäfts- und Betriebsprozessen aufzeigen.

LZ 1-4: Aufgaben und Verantwortung von Softwarearchitekten verstehen (R1)

Softwarearchitekten tragen die Verantwortung für die Erreichung der geforderten oder notwendigen Qualität und die Erstellung des Architekturentwurfs 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 Softwarearchitekten:

- Anforderungen und Randbedingungen klären, hinterfragen und bei Bedarf verfeinern. Hierzu zählen neben den funktionalen Anforderungen (Required Features) insbesondere die geforderten Qualitätsmerkmale (*Required Constraints*)
- Strukturentscheidungen hinsichtlich Systemzerlegung und Bausteinstruktur treffen, dabei Abhängigkeiten und Schnittstellen zwischen den Bausteinen festlegen
- Querschnittskonzepte entscheiden (beispielsweise Persistenz, Kommunikation, GUI) und bei Bedarf umsetzen
- Softwarearchitektur auf Basis von Sichten, Architekturmustern sowie technischen und Querschnittskonzepten 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 geforderten Qualitätsmerkmale
- Die Konsequenzen von Architekturentscheidungen erkennen, aufzeigen und gegenüber anderen Stakeholdern argumentieren

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

LZ 1-5: Rolle von Softwarearchitekten in Beziehung zu anderen Stakeholdern setzen (R1)

Softwarearchitekten können ihre Rolle erklären und ihren Beitrag zur Systementwicklung in Verbindung mit anderen Stakeholdern kontextspezifisch ausgestalten, insbesondere zu:

- Anforderungsanalyse (Systemanalyse, Anforderungsmanagement, Fachbereich)
- Implementierung
- Projektleitung und -management
- Produktmanagement, Product-Owner
- Qualitätssicherung
- IT-Betrieb (Produktion, Rechenzentren), zutreffend primär für Informationssysteme
- Hardwareentwicklung
- Unternehmensarchitektur, Architekturboard.

LZ 1-6: Zusammenhang zwischen Entwicklungsvorgehen und Softwarearchitektur erläutern können (R1)

- Softwarearchitekten können den Einfluss von iterativem Vorgehen auf Architekturentscheidungen erläutern (hinsichtlich Risiken und Prognostizierbarkeit).
- Sie müssen aufgrund inhärenter Unsicherheit oftmals iterativ arbeiten und entscheiden. Dabei müssen sie bei anderen Stakeholdern systematisch Rückmeldung einholen.

LZ 1-7: Kurz- und langfristige Ziele differenzieren (R1)

Softwarearchitekten können:

- langfristige Qualitätsanforderungen sowie deren Abgrenzung gegen (kurzfristige) Projektziele erklären
- Potentielle Zielkonflikte zwischen kurz- und langfristigen Zielen erklären, um eine für alle Beteiligten tragfähige Lösung zu erarbeiten
- Qualitätsanforderungen identifizieren und präzisieren.

LZ 1-8: Explizite von impliziten Aussagen unterscheiden (R1)

Softwarearchitekten:

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

LZ 1-9: Zuständigkeit von Softwarearchitekten in organisatorischen Kontext einordnen (R3)

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

Darüber hinaus kennen Softwarearchitekten 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
- Infrastruktur- bzw. Technologiearchitektur: Struktur der technischen Infrastruktur, Hardware, Netze etc.
- Hardware- oder Prozessorarchitektur (für hardwarenahe Systeme).

Diese Architekturdomänen sind nicht inhaltlicher Fokus vom CPSA-F.

LZ 1-10: Typen von IT-Systemen unterscheiden (R3)

Softwarearchitekten kennen unterschiedliche Typen von IT-Systemen, beispielsweise:

- Informationssysteme
- Decision-Support, Data-Warehouse oder Business-Intelligence Systeme
- Mobile Systeme
- Batchprozesse oder -systeme
- hardwarenahe Systeme; hier verstehen sie die Notwendigkeit des Hardware-/Software-Codesigns (zeitliche und inhaltliche Abhängigkeiten von Hard- und Softwareentwurf).

2. Entwurf und Entwicklung von Softwarearchitekturen

Dauer: 330 Min.	Übungszeit: 90 Min.

Wesentliche Begriffe

Entwurf; Vorgehen beim Entwurf; Entwurfsentscheidung; Sichten; Schnittstellen; technische Konzepte und Querschnittskonzepte; Stereotypen; Architekturmuster; Entwurfsmuster; Mustersprachen; Entwurfsprinzipien; Abhängigkeit; Kopplung; Kohäsion; fachliche und technische Architekturen; Top-down- und Bottom-Up-Vorgehen; modellbasierter Entwurf; iterativer/inkrementeller Entwurf; Domain-Driven Design

Lernziele

LZ 2-1: Vorgehen und Heuristiken zur Architekturentwicklung auswählen und anwenden können (R1-R3)

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

- Top-down- und Bottom-Up-Vorgehen beim Entwurf
- Sichtenbasierte Architekturentwicklung
- iterativer und inkrementeller Entwurf
  - Notwendigkeit von Iterationen, insbesondere bei unter Unsicherheit getroffenen Entscheidungen
  - Notwendigkeit von Rückmeldungen zu Entwurfsentscheidungen
- Domain-Driven Design (R3)
- Modellgetriebene Architektur (R3)

LZ 2-2: Softwarearchitekturen entwerfen (R1)

Softwarearchitekten 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
- gegenseitige Abhängigkeiten und Abwägungen bezüglich Entwurfsentscheidungen 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
- die aus diesen Entscheidungen resultierenden Konsequenzen auf den Quellcode erklären
- fachliche und technische Bestandteile in Architekturen trennen und diese Trennung begründen

- Risiken von Entwurfsentscheidungen identifizieren.

LZ 2-3: Einflussfaktoren auf Softwarearchitektur erheben und berücksichtigen können (R1-R2)

Softwarearchitekten können Einflussfaktoren (Randbedingungen) als Einschränkungen der Entwurfsfreiheit erarbeiten und berücksichtigen.

Sie erkennen und berücksichtigen den:

- Einfluss von Qualitätsanforderungen
- Einfluss technischer Entscheidungen und Konzepte
- (möglichen) Einfluss organisatorischer und juristischer Faktoren (R2)
- Einfluss verschiedener Stakeholder (R2)

LZ 2-4: Querschnittskonzepte entwerfen und umsetzen (R1)

Softwarearchitekten können:

- die Bedeutung von Querschnittskonzepten erklären
- Querschnittskonzepte entscheiden und entwerfen, beispielsweise Persistenz, Kommunikation, GUI, Fehlerbehandlung, Nebenläufigkeit
- mögliche wechselseitige Abhängigkeiten dieser Entscheidungen erkennen und beurteilen.

Softwarearchitekten wissen, dass solche Querschnittskonzepte systemübergreifend wiederverwendbar sein können.

LZ 2-5: Wichtige Architekturmuster beschreiben, erklären und angemessen anwenden (R1-R3)

Softwarearchitekten kennen diverse Architekturmuster und können diese bedarfsgerecht einsetzen:

- Schichten (*Layer*) (R1)
- Pipes und Filter (R1)
- Client/Server (R1)
- Adapter und Facade (R1)
- Proxy (R1)
- Plugin (R1)
- Blackboard (R2)
- Model-View-Controller und Varianten (R2)
- Broker (R2)
- Remote Procedure Call (R2)
- Messaging (R2), z.B. mit Events und Commands

Softwarearchitekten kennen wesentliche Quellen für Architekturmuster, beispielsweise die POSA-Literatur (z.B. [Buschmann+1996])und PoEAA ([Fowler 2003]) (für Informationssysteme) (R3)

LZ 2-6: Entwurfsprinzipien erläutern und anwenden (R1)

Softwarearchitekten können die folgenden Entwurfsprinzipien erläutern und anwenden:

- Abstraktion
- Modularisierung, unter Berücksichtigung von:
  - Geheimnisprinzip (*Information Hiding*) und Kapselung
  - Trennung von Verantwortlichkeiten (*Separation of Concerns*)
  - Hoher Kohäsion
- Single-Responsibility-Prinzip
- Offen-Geschlossen-Prinzip (*Open-/Closed-Principle*)
- möglichst lose, aber funktional ausreichende Kopplung der Bausteine (siehe auch LZ 2-7)
- Umkehrung von Abhängigkeiten durch Schnittstellen oder ähnliche Abstraktionen (*Dependency Inversion*)
- konzeptioneller Integrität zur Erreichung der Gleichförmigkeit (Homogenität, Konsistenz) von Lösungen für ähnliche Probleme (R2)

Softwarearchitekten verstehen die Einflüsse der Entwurfsprinzipien auf Quellcode und können diese gezielt einsetzen.

LZ 2-7: Abhängigkeiten von Bausteinen planen (R1)

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

- kennen unterschiedliche Arten der Abhängigkeiten von Bausteinen (beispielsweise strukturelle Kopplung über Benutzung/Delegation, Schachtelung, Besitz, Erzeugung, Vererbung, zeitliche Kopplung, Kopplung über Datentypen oder über Hardware)
- können solche Arten der Kopplung gezielt einsetzen und die Konsequenzen solcher Abhängigkeiten einschätzen
- kennen Möglichkeiten zur Auflösung bzw. Reduktion von Kopplung und können diese anwenden, beispielsweise:
  - Muster (siehe Lernziel 2-5)
  - Grundlegende Entwurfsprinzipien (siehe Lernziel 2-6)
  - Externalisierung von Abhängigkeiten, d.h. konkrete Abhängigkeiten erst zur Installations- oder Laufzeit festlegen, etwa durch Anwendung von *Dependency Injection*.

LZ 2-8: Qualitätsanforderungen mit passenden Ansätzen und Techniken erreichen (R1)

Softwarearchitekten kennen und berücksichtigen den starken Einfluss von Qualitätsanforderungen in Architektur- und Entwurfsentscheidungen, beispielsweise für:

- Effizienz / Performance
- Verfügbarkeit
- Wartbarkeit, Modifizierbarkeit, Erweiterbarkeit, Adaptierbarkeit

Sie können:

- Lösungsmöglichkeiten, *Design Tactics*, angemessene Praktiken sowie technische Möglichkeiten zur Erreichung wichtiger Qualitätsanforderungen von Softwaresystemen (unterschiedlich für eingebettete Systeme bzw. Informationssysteme) erklären und anwenden
- mögliche Wechselwirkungen zwischen solchen Lösungsmöglichkeiten sowie die entsprechenden Risiken identifizieren und kommunizieren.

LZ 2-9: Schnittstellen entwerfen und festlegen (R1-R3)

Softwarearchitekten kennen die hohe Bedeutung von Schnittstellen. Sie können Schnittstellen zwischen Architekturbausteinen sowie externe Schnittstellen zwischen dem System und Elementen außerhalb des Systems entwerfen bzw. festlegen.

Sie kennen:

- wünschenswerte Eigenschaften von Schnittstellen und können diese beim Entwurf einsetzen:
  - einfach zu erlernen, einfach zu benutzen, einfach zu erweitern
  - schwer zu missbrauchen
  - funktional vollständig aus Sicht der Nutzer oder nutzender Bausteine.
- die Notwendigkeit unterschiedlicher Behandlung interner und externer Schnittstellen
- unterschiedliche Implementierungsansätze von Schnittstellen (R3):
  - ressourcenorientierter Ansatz (REST, REpresentational State Transfer)
  - serviceorientierter Ansatz (wie bei WS-*/SOAP-basierten Webservices).

Referenzen

[Bass+2012], [Fowler 2003], [Gharbi+2017], [Gamma+94], [Martin 2003], [Buschmann+1996] and [Buschmann+2007], [Starke 2017], [Lilienthal 2017]

3. Beschreibung und Kommunikation von Softwarearchitekturen

Dauer: 180 Min.	Übungszeit: 60 Min.

Wesentliche Begriffe

Architektur-)Sichten; Strukturen; (technische) Konzepte; Dokumentation; Kommunikation; Beschreibung; zielgruppen- oder stakeholdergerecht; Meta-Strukturen und Templates zur Beschreibung und Kommunikation; Kontextabgrenzung; Bausteine; Bausteinsicht; Laufzeitsicht; Verteilungssicht; Knoten; Kanal; Verteilungsartefakte; Mapping von Bausteinen auf Verteilungsartefakte; Beschreibung von Schnittstellen und Entwurfsentscheidungen; UML; Werkzeuge zur Dokumentation

Lernziele

LZ 3-1: Qualitätsmerkmale technischer Dokumentation erläutern und berücksichtigen (R1)

Softwarearchitekten kennen die wesentlichen Qualitätsmerkmale technischer 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.

LZ 3-2: Softwarearchitekturen beschreiben und kommunizieren (R1)

Softwarearchitekten:

- können Architekturen stakeholdergerecht dokumentieren und kommunizieren und dadurch unterschiedliche Zielgruppen adressieren, z. B. Management, Entwicklungsteams, QS, andere Softwarearchitekten sowie möglicherweise zusätzliche Stakeholder
- sind in der Lage, die Beiträge unterschiedlicher Autorengruppen stilistisch und inhaltlich zu konsolidieren und harmonisieren
- kennen den Nutzen von Template-basierter Dokumentation.

LZ 3-3: Notations-/Modellierungsmittel für Beschreibung von Softwarearchitektur erläutern und anwenden (R2)

Softwarearchitekten kennen mindestens folgende UML-Diagramme zur Notation von Architektursichten:

- Klassen-, Paket-, Komponenten- und Kompositionsstrukturdiagramme
- Verteilungsdiagramme
- Sequenz- und Aktivitätsdiagramme
- Zustandsdiagramme

Softwarearchitekten kennen Alternativen zu UML-Diagrammen. Insbesondere für die Laufzeitsicht eignen sich z. B. Flussdiagramme, nummerierte Listen, BPMN.

LZ 3-4: Architektursichten erläutern und anwenden (R1)

Softwarearchitekten können folgende Architektursichten anwenden:

- Kontextsicht (auch genannt Kontextabgrenzung)
- 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)

LZ 3-5: Kontextabgrenzung von Systemen erläutern und anwenden (R1)

Softwarearchitekten können:

- Kontext von Systemen z.B. in Form von Kontextdiagramm mit Erläuterungen darstellen
- externe Schnittstellen von Systemen in der Kontextabgrenzung darstellen
- fachlichen und technischen Kontext differenzieren.

LZ 3-6: Querschnittskonzepte dokumentieren und kommunizieren (R1)

Softwarearchitekten können typische Querschnittskonzepte (auch genannt *Prinzipien* oder *Aspekte*) adäquat dokumentieren und kommunizieren, z. B. Persistenz, Ablaufsteuerung, UI, Verteilung/Integration, Protokollierung.

LZ 3-7: Schnittstellen beschreiben (R1)

Softwarearchitekten können sowohl interne als auch externe Schnittstellen beschreiben und spezifizieren.

LZ 3-8: Architekturentscheidungen erläutern und dokumentieren (R2)

Softwarearchitekten können:

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

LZ 3-9: Dokumentation als schriftliche Kommunikation nutzen (R2)

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

LZ 3-10: Weitere Hilfsmittel und Werkzeuge zur Dokumentation kennen (R3)

Softwarearchitekten kennen:

- Grundlagen mehrerer publizierter Frameworks zur Beschreibung von Softwarearchitekturen, beispielsweise:
  - ISO/IEEE-42010 (vormals 1471)

    - arc42
    - C4
    - RM/ODP
    - FMC
- Ideen und Beispiele von Checklisten für die Erstellung, Dokumentation und Prüfung von Softwarearchitekturen
- mögliche Werkzeuge zur Erstellung und Pflege von Architekturdokumentation

Referenzen

[Bass+2012], [Clements+2010], [Gharbi+2017], [Starke 2017], [Zörner 2015]

4. Softwarearchitektur und Qualität

Dauer: 60 Min.	Übungszeit: 60 Min.

Wesentliche Begriffe

Qualität; Qualitätsmerkmale; DIN/ISO 25010; Qualitätsszenarien; Qualitätsbaum; Kompromisse/Wechselwirkungen von Qualitätseigenschaften; qualitative Architekturbewertung; Metriken und quantitative Bewertung

Lernziele

LZ 4-1: Qualitätsmodelle und Qualitätsmerkmale diskutieren (R1)

Softwarearchitekten können:

- den Begriff der Qualität (angelehnt an DIN/ISO 25010, vormals 9126) und der Qualitätsmerkmale erklären
- generische Qualitätsmodelle (wie etwa DIN/ISO 25010) erklären
- Zusammenhänge und Wechselwirkungen von Qualitätsmerkmalen erläutern, beispielsweise:
  - Konfigurierbarkeit versus Zuverlässigkeit
  - Speicherbedarf versus Leistungseffizienz
  - Sicherheit versus Benutzbarkeit
  - Laufzeitflexibilität versus Wartbarkeit.

LZ 4-2: Qualitätsanforderungen an Softwarearchitekturen klären (R1)

Softwarearchitekten können:

- spezifische Qualitätsanforderungen an die zu entwickelnde Software und deren Architekturen klären und konkret formulieren, beispielsweise in Form von Szenarien und Qualitätsbäumen
- Szenarien und Qualitätsbäume erklären und anwenden.

LZ 4-3: Softwarearchitekturen qualitativ analysieren und bewerten (R2-R3)

Softwarearchitekten:

- kennen methodische Vorgehensweisen zur qualitativen Analyse und Bewertung von Softwarearchitekturen (R2), beispielsweise nach ATAM (R3)
- können kleinere Systeme qualitativ analysieren und bewerten (R2)
- wissen, dass zur qualitativen Analyse und Bewertung von Architekturen folgende Informationsquellen helfen können (R2):
  - Qualitätsanforderungen, beispielsweise in Form von Qualitätsbäumen und -szenarien
  - Architekturdokumentation
  - Architektur- und Entwurfsmodelle

- Quellcode
- Metriken
- Sonstige Dokumentationen des Systems, etwa Anforderungs-, Betriebs- oder Testdokumentation.

LZ 4-4: Softwarearchitekturen quantitativ bewerten (R2)

Softwarearchitekten kennen Ansätze zur quantitativen Analyse und Bewertung (Messung) von Software.

Sie wissen, dass:

- quantitative Bewertung helfen kann, kritische Teile innerhalb von Systemen zu identifizieren
- zur Bewertung von Architekturen weitere Informationen hilfreich sein können, etwa:
  - Anforderungs- und Architekturdokumentation
  - Quellcode und diesbezügliche Metriken wie Lines-of-Code, (zyklomatische) Komplexität, ein- und ausgehende Abhängigkeiten
  - bekannte Fehler in Quellcode, insbesondere Fehlercluster
  - Testfälle und Testergebnisse.

Referenzen

[Bass2003], [Clements+2002], [Gharbi+2017], [Martin 2003], [Starke 2017]

5. Beispiele für Softwarearchitekturen

Dauer: 90 Min.	Übungszeit: Keine

Dieser Abschnitt ist nicht prüfungsrelevant.

Lernziele

LZ 5-1: Bezug von Anforderungen und Randbedingungen zu Lösung erfassen (R3)

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

LZ 5-2: Technische Umsetzung einer Lösung nachvollziehen (R3)

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

Referenzen

- [iSAQB Working Groups] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Freely available https://leanpub.com/isaqbreferences.
- [Bass+2012] Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. 3rd Edition, Addison Wesley 2012.
- [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
- [Gamma+94] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
- [Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java. Springer-Vieweg Verlag, 2. Auflage 2014.
- [Fowler 2003] Martin Fowler: Patterns of Enterprise Application Architecture. (PoEAA) Addison-Wesley, 2003.
- [Gharbi+2017] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 3. Auflage, dpunkt Verlag, Heidelberg 2017.
- [Martin 2003] Robert Martin: Agile Software Development. Principles, Patterns, and Practices. Prentice Hall, 2003.
- [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.
- [Starke 2017] Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden (in German). 8. Auflage, Carl Hanser Verlag 2017. Website: https://esabuch.de
- [Lilienthal 2017] Carola Lilienthal: Langlebige Softwarearchitekuren. 2. Auflage, dpunkt Verlag 2018.
- [Zörner 2015] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. 2. Auflage, Carl Hanser Verlag 2015.