Curriculum für

CPSA
Certified Professional for Software Architecture®

- Foundation Level -

Version 4.2 (Juli 2017)

Inhaltsverzeichnis

Verzeichnis der Lernziele

0 Einleitung

0.1 Was vermittelt eine Foundation-Level-Schulung?

Lizenzierte Schulungen zum **Certified Professional for Software Architecture – Foundation Level (CPSA-F)** vermitteln das notwendige Wissen und die notwendigen Fähigkeiten, um für kleine und mittlere Systeme eine der Aufgabenstellung angemessene Softwarearchitektur zu entwerfen.

Auf der Grundlage bestehender Erfahrungen und Fähigkeiten in der Softwareentwicklung lernen Teilnehmer, aus einer vorhandenen Systemidee und angemessen detaillierten Anforderungen eine adäquate Softwarearchitektur abzuleiten. Die Schulung vermittelt methodische Werkzeuge und Prinzipien für Entwurf, Dokumentation und Bewertung von Softwarearchitekturen, unabhängig von spezifischen Vorgehensmodellen.

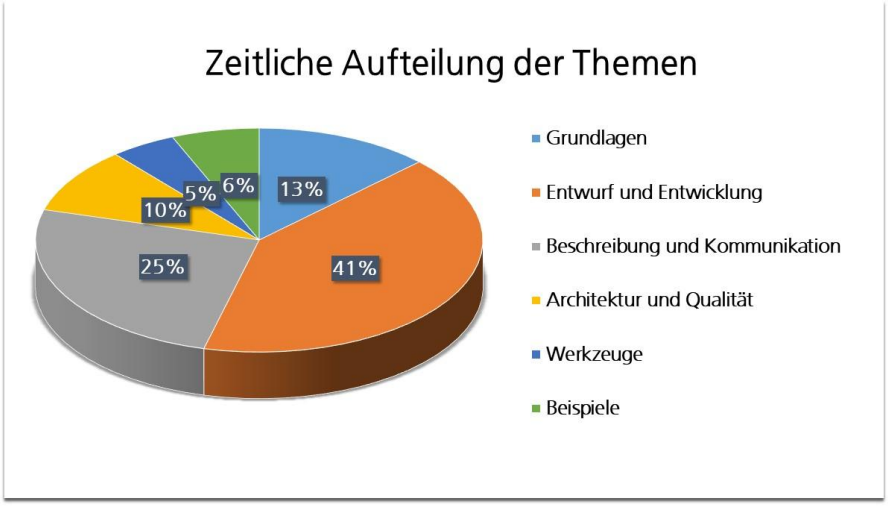
Im Mittelpunkt steht der Erwerb folgender Fähigkeiten:

- 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 Projekten,
- State-of-the-Art-Methoden und -Techniken zur Entwicklung von Softwarearchitekturen.

0.2 Gliederung des Lehrplans und empfohlene zeitliche Aufteilung



0.3 Dauer, Didaktik und weitere Details akkreditierter Schulungen

Die genannten Zeiten sind 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 kann der jeweilige Schulungsanbieter individuell festlegen.

0.4 Voraussetzungen

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

- mehr als 18 Monate praktische Erfahrung in Softwareentwicklung, erworben durch Programmierung unterschiedlicher Projekte oder Systeme außerhalb der Ausbildung
- Kenntnisse und praktische Erfahrung in mindestens einer höheren Programmiersprache, insbesondere:
  - Konzepte der Parameterübergabe (Call-by-Value, Call-by-Reference)
  - Grundlagen von Typsystemen (statische und dynamische Typisierung, parametrisierbare/generische Datentypen)
  - Konzepte der Modularisierung (Pakete, Namensräume)
- Grundlagen der Modellierung und Abstraktion
- Grundlagen von Algorithmen und Datenstrukturen
- Grundlagen von UML (Klassen-, Paket-, Komponenten- und Sequenzdiagramme) und deren Bezug zum Quellcode
- praktische Erfahrung in technischer Dokumentation, insbesondere in der Dokumentation von Quellcode, Systementwürfen oder technischen Konzepten

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Kenntnisse der Objektorientierung (OO)
- praktische Erfahrung in mindestens einer objektorientierten Programmiersprache
- praktische Erfahrung in der Konzeption und Implementierung verteilt ablaufender Anwendungen, wie etwa Client/Server-Systeme oder Web-Anwendungen

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

0.5 CPSA-F-Lehrplankapitel, 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 R-1, R-2 oder R-3, siehe Tabelle).

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

Lernziel-Kategorie	Kennzeichen	Bedeutung	Relevanz für Prüfung
Können	R1	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.
Verstehen	R2	Diese Inhalte sollen die Teilnehmer grundsätzlich verstehen. Sie werden in Schulungen i. d. R. nicht durch Übungen vertieft.	Inhalte können geprüft werden.
Kennen	R3	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.

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

0.6 Abgrenzung

Dieser Lehrplan reflektiert den aus heutiger Sicht der iSAQB-Mitglieder 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
- 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

0.7 Einführung in das iSAQB-Zertifizierungsprogramm

Dauer: 15 Min.	Übungszeit: keine

Dieser Abschnitt ist nicht prüfungsrelevant.

Die Teilnehmer lernen den Kontext des iSAQB-Zertifizierungsprogrammes und der zugehörigen Prüfungen beziehungsweise Prüfungsmodalitäten kennen:

- iSAQB e. V. als Verein
- Verantwortung des iSAQB e. V. für Ausgestaltung des Lehrplans sowie der zugehörigen Prüfungsfragen
  - organisatorische Trennung zwischen Schulung und Prüfung
  - Ablauf und formale Randbedingungen der CPSA-F-Prüfung
- Foundation Level in Abgrenzung zum Advanced Level
- optional: Andere Zertifizierungsprogramme

1 Grundbegriffe von Softwarearchitekturen

Dauer: 120 Min.	Übungszeit: keine

1.1 Wesentliche Begriffe

Softwarearchitektur, Struktur, Bausteine/Komponenten, Schnittstellen, Beziehungen, übergreifende Konzepte/Aspekte, Architekturziele, Softwarearchitekten und deren Verantwortlichkeiten, Rolle, Aufgaben und benötigte Fähigkeiten, Stakeholder, Funktionale und nichtfunktionale Anforderungen, Qualitätsanforderungen, Randbedingungen, Einflussfaktoren, Typen von IT-Systemen (eingebettete Systeme, Echtzeitsysteme, Informationssysteme etc.)

1.2 Lernziele

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

Vergleich mehrerer Definitionen von Softwarearchitektur (u. a. ISO 42010/IEEE 1471, SEI, Booch etc.) und deren Gemeinsamkeiten benennen:

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

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

- Ziele von Softwarearchitekten und Softwarearchitekturen
  - Fokus von Softwarearchitektur liegt auf Qualitätsmerkmalen wie Langlebigkeit, Wartbarkeit, Änderbarkeit, Robustheit als Differenzierung gegenüber reiner Funktionalität
- Softwarearchitektur unterstützt Erstellung und Wartung von Software, insbesondere die Implementierung
- Softwarearchitektur unterstützt die Erreichung von Qualitätsanforderungen
- Softwarearchitektur unterstützt das Verständnis über das System, bezogen auf sämtliche relevanten Stakeholder

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

- Einordnung von Softwarearchitektur in die gesamte Entwicklung von IT-Systemen
- Zusammenhang mit Geschäfts- und Betriebsprozessen für Informationssysteme
- Zusammenhang mit Geschäfts- und Betriebsprozessen für Entscheidungsunterstützungssysteme (Data Warehouse, Management Information Systems)
- Änderungen von Anforderungen, Qualitätszielen, Technologie oder der Systemumgebung können Änderungen an der Softwarearchitektur erfordern oder hervorrufen.

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

Softwarearchitekten tragen die Verantwortung für die Erreichung der geforderten oder notwendigen Qualität und Funktionalität 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
- übergreifende technische Konzepte entscheiden (beispielsweise Persistenz, Kommunikation, GUI) und bei Bedarf umsetzen
- Softwarearchitektur auf Basis von Sichten, Architekturmustern und technischen Konzepten 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 bewerten, insbesondere hinsichtlich Risiken bezüglich der geforderten Qualitätsmerkmale

- Softwarearchitekten sollen die Konsequenzen von Architekturentscheidungen erkennen, aufzeigen und gegenüber anderen Stakeholdern argumentieren können
- 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

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

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

LZ 1-7: Architektur- und Projektziele differenzieren (R1)

Systeme können im Rahmen von Projekten entwickelt werden, aber auch im Rahmen von Scrum-Sprints, Iterationen, Releases oder andere Vorgehensweisen. Der Begriff „Projekt" im Titel des Lernzieles bezieht sich auf jede dieser möglichen Formen der Arbeitsorganisation.

- Softwarearchitekten können die Bedeutung von Architekturzielen, Randbedingungen und Einflussfaktoren für die Gestaltung von Softwarearchitekturen aufzeigen;
- sie können den Zusammenhang zwischen Anforderungen und Lösungen erläutern;
- sie können auf Basis bestehender Anforderungen Architekturziele identifizieren und präzisieren;
- sie können (langfristige) Architekturziele sowie deren Abgrenzung gegen (kurzfristige) Projektziele erklären.

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

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

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

Softwarearchitekten kennen weitere Ebenen von Architektur, beispielsweise:

- Infrastruktur-Architektur: Struktur der technischen Infrastruktur, Hardware, Netze etc.
- Hardwarearchitektur (für hardwarenahe Systeme)
- Softwarearchitektur: Struktur einzelner Softwaresysteme. Dies ist der Fokus von Softwarearchitekten im Sinne des iSAQB und des CPSA-F.
- Unternehmens-IT-Architektur, Enterprise-IT-Architektur:
  - Struktur von Anwendungslandschaften. Ist **nicht** inhaltlicher Fokus vom CPSA-F.
- Geschäftsprozess-Architektur, Business-Architektur:
  - Struktur von Geschäftsprozessen. Ist **nicht** inhaltlicher Fokus vom CPSA-F.

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

Softwarearchitekten können zwischen Softwarearchitekturen für unterschiedliche Typen von IT-Systemen differenzieren:

- Sie kennen den Zusammenhang mit der System- oder Gesamtarchitektur für eingebettete oder hardwarenahe Systeme;
- Sie verstehen die Besonderheiten des Hardware-/Software-Co-Designs (zeitliche und inhaltliche Abhängigkeiten von Hard- und Softwareentwurf).

Referenzen

[Bass+2012], [Gharbi+2014], [Starke2015]

2 Entwurf und Entwicklung von Softwarearchitekturen

Dauer: 300 Min.	Übungszeit: 90 Min.

2.1 Wichtige Begriffe

Entwurf, Vorgehen beim Entwurf, Entwurfsentscheidung, Sichten, Schnittstellen, technische und querschnittliche Konzepte, 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, Werkzeug-Material-Ansatz

2.2 Lernziele

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

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

- modell- und sichtenbasierte Architekturentwicklung
- Domain-Driven-Design
- iterativer und inkrementeller Entwurf
  - Notwendigkeit von Iterationen, insbesondere bei unter Unsicherheit getroffenen Entscheidungen
  - Rückmeldungen zu Entwurfsentscheidungen auf Basis von Quellcode sowie qualitativer Betrachtung
- Top-down- und Bottom-Up-Vorgehen beim Entwurf

Softwarearchitekten können Einflussfaktoren und Randbedingungen als Beschränkungen beim Architekturentwurf berücksichtigen

LZ 2-2: Architekturen entwerfen (R1)

Softwarearchitekten können:

- Architekturen auf Basis bekannter funktionaler und Qualitätsanforderungen (nichtfunktionaler Anforderungen) für nicht sicherheits- oder unternehmenskritische Softwaresysteme entwerfen und angemessen dokumentieren;
- gegenseitige Abhängigkeiten von Entwurfsentscheidungen erkennen und begründen;
- Strukturentscheidungen hinsichtlich Systemzerlegung und Bausteinstruktur treffen, dabei Abhängigkeiten und Schnittstellen zwischen Bausteinen festlegen;
- Blackbox- und Whitebox-Begriff erklären und zielgerichtet anwenden;
- schrittweise Verfeinerung und Spezifikation von Bausteinen durchführen;
- Architektursichten entwerfen, insbesondere Verteilungs-, Baustein- und Laufzeitsicht, sowie Konsequenzen dieser Entscheidungen auf den zugehörigen Quellcode beschreiben;
- Abbildung der Architektur auf den Quellcode festlegen und die damit verbundenen Konsequenzen bewerten;
- fachliche und technische Bestandteile in Architekturen trennen und diese Trennung begründen.

Softwarearchitekten können fachliche Strukturen im Kleinen entwerfen und begründen, insbesondere können sie:

- fachliche Bausteine (Aggregate, Entitäten, Services, Wertobjekte) identifizieren, begründen und dokumentieren;
- eine „Ubiquitous Language" mit den Stakeholdern entwickeln, abstimmen, verfeinern und im Verlauf der Entwicklung anpassen;
- die Interaktion fachlicher und technischer Bestandteile von Systemen entwerfen und erläutern.

Softwarearchitekten kennen und berücksichtigen in Architektur- und Entwurfsentscheidungen:

- den starken Einfluss von Qualitätsanforderungen (nichtfunktionaler Anforderungen) – wie beispielsweise Änderbarkeit, Robustheit, Effizienz;
- Lösungsmöglichkeiten und „Design Tactics" für wesentliche Qualitätsanforderungen, beispielsweise für Änderbarkeit, Robustheit oder Effizienz;
- mögliche Wechselwirkungen zwischen solchen Lösungsmöglichkeiten.

Softwarearchitekten können fachliche Strukturen im Großen (Bounded Contexts) entwerfen und begründen (R3):

- Core Domain, Supporting Domain, Generic Domain, Subdomain
- Zusammenwirken fachlicher Bestandteile im Großen (Context Mapping), beispielsweise:
  - Open-Host-Service, Published-Language, Anticorruption-Layer, Separate Ways

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

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

Sie erkennen und berücksichtigen:

- den Einfluss von Qualitätsanforderungen auf Architekturen
- den Einfluss technischer Entscheidungen und Konzepte auf Architekturen
- den (möglichen) Einfluss organisatorischer und juristischer Faktoren auf Architekturentscheidungen (R2)

LZ 2-4: Übergreifende Konzepte entwerfen und umsetzen (R1)

- Softwarearchitekten können übergreifende technische/querschnittliche Konzepte entscheiden und entwerfen, beispielsweise Persistenz, Kommunikation, GUI, Fehlerbehandlung, Nebenläufigkeit.
- Sie können mögliche wechselseitige Abhängigkeiten dieser Entscheidungen erkennen und beurteilen.

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

Softwarearchitekten kennen diverse Architekturmuster und Architekturstile und können diese bedarfsgerecht einsetzen:

- Muster für datenfluss- bzw. datenzentrierte Architekturstile (R1)
  - Pipes und Filter (R1)
  - Blackboard (R2)
- Muster für hierarchische Architekturstile (R1)
  - Schichten, Layer
  - Master-Slave
  - Virtual Machine
- Hybride (Heterogene) Architekturstile (R1)
- sprachunabhängige Entwurfsmuster, etwa: Adapter, Wrapper, Gateway, Facade, Registry, Broker, Factory (R1)
- Muster für interaktionsorientierte Systeme (R2), beispielsweise: Model-View-Controller, Model-View-ViewModel, Model-View-Presenter, Presentation-Abstraction-Control
- Muster für verteilte Systeme und deren Integration (R3)
  - Messaging (Publish-Subscribe, Hub-and-Spoke)
  - ereignisbasierte Systeme (und Event Sourcing)
  - Client/Server (R1)
  - Remote Procedure Call (R2)
  - datei- oder datenbankbasierte Integration
  - Microservices/Self-Contained Systems
- Softwarearchitekten kennen wesentliche Quellen für Architekturmuster, beispielsweise die POSA-Literatur und PoEAA (für Informationssysteme) (R3)

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

Softwarearchitekten können die folgenden Entwurfsprinzipien erläutern und anwenden:

- Abstraktion
- Modularisierung (siehe auch LZ 2-8), 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 loser, aber funktional ausreichender Kopplung der Bausteine (siehe auch LZ 2-7)
- Umkehrung von Abhängigkeiten durch Schnittstellen (*Dependency Inversion*)
- Dependency Injection zur Externalisierung von Abhängigkeiten
- 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-R3)

- Softwarearchitekten verstehen Abhängigkeiten und Kopplung zwischen Bausteinen und können diese gezielt einsetzen (R1);
- Sie können unterschiedliche Arten der Kopplung (strukturell, zeitlich, über Datentypen, über Hardware etc.) gezielt einsetzen (R1) und die Konsequenzen solcher Abhängigkeiten einschätzen (R1);
- Sie kennen unterschiedliche Arten der Abhängigkeiten von Bausteinen (Schachtelung, Benutzung/Delegation, Vererbung).
- Softwarearchitekten kennen Möglichkeiten zur Auflösung bzw. Reduktion von Kopplung und können diese anwenden, beispielsweise:
  - sprachunabhängige Muster wie Broker, Plugin, Adapter, Facade (R1)
  - Erzeugungsmuster/Dependency-Injection (R1)
  - Messaging, Events, Commands (R2)
  - Stability-Patterns wie Bulkhead, Timeout, Circuit-Breaker (R3)

LZ 2-8: Bausteine/Strukturelemente von Softwarearchitekturen entwerfen (R1)

Softwarearchitekten

- können wünschenswerte Eigenschaften (Kapselung, Information Hiding, Geheimnisprinzip) von Bausteinen und Strukturelementen zielgerichtet anwenden;
- können Blackbox- und Whitebox-Beschreibung von Bausteinen zielgerichtet verwenden;
- kennen UML-Notationen für Bausteine und deren Zusammensetzung (R2);
  - Pakete als semantisch schwache Form der Aggregation von Bausteinen,
  - Klassen (in OO-Sprachen) und Komponenten mit fest definierten Schnittstellen als semantisch präzisere Variante.

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

Softwarearchitekten kennen

- die Bedeutung von Schnittstellen. Sie können Schnittstellen zwischen Architekturbausteinen sowie externe Schnittstellen zwischen dem System und Elementen außerhalb des Systems entwerfen bzw. festlegen.
- 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
- unterschiedliche Betrachtungsweisen von Schnittstellen (R3):
  - ressourcenorientierter Ansatz (wie in REpresentational State Transfer)
  - serviceorientierter Ansatz (wie bei WS-*/SOAP-basierten Webservices)

Softwarearchitekten können Schnittstellen beschreiben und dokumentieren.

Referenzen

[Bass+2012], [Fowler2003], [Gharbi+2014], [Martin2003], POSA-Serie, insbesondere [Buschmann+1996] und [Buschmann+2007], [Starke2015]

3 Beschreibung und Kommunikation von Softwarearchitekturen

Dauer: 150 Min.	Übungszeit: 90 Min.

3.1 Wichtige Begriffe

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, Werkzeuge zur Dokumentation

3.2 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 Lesern beurteilt werden kann.

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

- Softwarearchitekten können Architekturen Stakeholder-gerecht dokumentieren und kommunizieren;
- sie können unterschiedliche Leserkreise adressieren, z. B. Management, Entwickler, QS, andere Softwarearchitekten sowie möglicherweise zusätzliche Stakeholder;
- sie sind in der Lage, die Beiträge unterschiedlicher Autorengruppen stilistisch und inhaltlich zu konsolidieren und harmonisieren;
- sie kennen den Nutzen von Template-/schablonenbasierter 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-/Deployment-Sicht (Hardware und technische Infrastruktur sowie Abbildung von Softwarebausteinen auf diese Infrastruktur)

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

Softwarearchitekten können fachlichen und technischen Kontext differenzieren.

LZ 3-6: Querschnittliche Architekturkonzepte dokumentieren und kommunizieren (R1)

- Softwarearchitekten können typische Konzepte (Prinzipien, Aspekte) adäquat dokumentieren, z. B. Persistenz, Ablaufsteuerung, GUI, Verteilung/Integration;
- sie können die Bedeutung solch übergreifender, technischer Konzepte erklären;
- sie wissen, dass solche querschnittlichen Konzepte systemübergreifend wiederverwendbar sein können.

LZ 3-7: Schnittstellen beschreiben (R1)

- Softwarearchitekten können Schnittstellen beschreiben und spezifizieren
- sie können interne und externe Schnittstellen differenzieren

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

- Softwarearchitekten können Architekturentscheidungen systematisch herleiten, begründen und dokumentieren

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

- Softwarearchitekten nutzen Dokumentation zur Unterstützung bei Entwurf und Implementierung von Systemen;
- Softwarearchitekten können Sprache und Ausdrucksmittel technischer Dokumentation an den Fähigkeiten und Zielen der Leser ausrichten bzw. aus technischer Dokumentation entsprechende zielgruppenorientierte Artefakte erstellen.

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

- Softwarearchitekten kennen die Grundlagen mehrerer publizierter Frameworks zur Beschreibung von Softwarearchitekturen, beispielsweise:
  - ISO/IEEE-42010 (vormals 1471)
  - TOGAF, RM/ODP
  - arc42, FMC
- sie kennen Ideen und Beispiele von Checklisten für die Erstellung, Dokumentation und Prüfung von Softwarearchitekturen
- sie kennen mögliche Werkzeuge zur Erstellung und Pflege von Architekturdokumentation

Referenzen

[Clements+2002], [Gharbi+2014], [Starke2015], [Zörner2015]

4 Softwarearchitekturen und Qualität

Dauer: 60 Min.	Übungszeit: 60 Min.

4.1 Wichtige Begriffe

Qualität, Qualitätsmerkmale, DIN/ISO 9126 bzw. 25010, ATAM, Szenarien, Qualitätsbaum, Kompromisse/Wechselwirkungen von Qualitätseigenschaften, qualitative Architekturbewertung, Metriken und quantitative Bewertung

4.2 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 Robustheit
  - Speicherplatz versus Laufzeit
  - Sicherheit versus Benutzbarkeit
  - Effizienz versus Wartbarkeit/Änderbarkeit

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

Softwarearchitekten können

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

LZ 4-3: Softwarearchitekturen qualitativ bewerten (R2)

Softwarearchitekten

- kennen methodische Vorgehen zur Analyse und Bewertung von Softwarearchitekturen und können diese für kleinere Beispiele anwenden;
- kennen das Vorgehen bei qualitativer Bewertung von Softwarearchitekturen nach ATAM;
- wissen, dass zur qualitativen Bewertung von Architekturen folgende Informationsquellen helfen können:
  - Anforderungen, insbesondere in Form von Qualitätsbäumen und -szenarien
  - Architekturdokumentation
  - Architektur- und Entwurfsmodelle
  - Quellcode
  - Metriken

LZ 4-4: Softwarearchitekturen quantitativ bewerten (R2)

- Softwarearchitekten kennen Ansätze zur quantitativen Bewertung (Messung) von Software. Sie wissen, dass quantitative Bewertung helfen kann, kritische Teile innerhalb von Systemen zu identifizieren;
- sie wissen, dass zur Bewertung von Architekturen weitere Informationen hilfreich sein können, etwa:
  - Quellcode und diesbezügliche Metriken wie Lines-of-Code, zyklomatische Komplexität, ein- und ausgehende Abhängigkeiten
  - bekannte Fehler in Quellcode, insbesondere Fehlercluster
  - Testfälle und Testergebnisse
  - Anforderungs- und Lösungsmodelle

LZ 4-5: Qualitätsziele mit passenden Ansätzen und Techniken erreichen (R2)

- Softwarearchitekten können Taktiken, angemessene Praktiken sowie technische Möglichkeiten zur Erreichung wichtiger Qualitätsziele von Softwaresystemen (unterschiedlich für eingebettete Systeme bzw. Informationssysteme) erklären und anwenden, beispielsweise für:
  - Effizienz/Performance
  - Wartbarkeit, Änderbarkeit, Erweiterbarkeit, Flexibilität
- sie können entsprechende Risiken identifizieren

Referenzen

[Bass+2003], [Clements+2002], [Gharbi+2014], [Martin2003], [Starke2015]

5 Werkzeuge für Softwarearchitekten

Dauer: 45 Min.	Übungszeit: keine

5.1 Wichtige Begriffe

Werkzeuge und deren Kategorien, Modellierungswerkzeuge, Werkzeuge zur statischen und dynamischen Analyse, Generierungswerkzeuge, Anforderungsmanagement-Werkzeuge, Build-Systeme/-Werkzeuge, Konfigurationsmanagement

5.2 Lernziele

LZ 5-1: Wichtige Werkzeugkategorien benennen und erläutern (R1)

Softwarearchitekten können die wichtigsten Kategorien von Werkzeugen sowie deren Nutzen für die Arbeit von Softwarearchitekten benennen und erläutern, beispielsweise Werkzeuge zu:

- Erfassung, Modellierung und Management von Anforderungen
- Entwicklung, Debugging, Versionierung und Konfiguration von Quellcode
- statische und dynamische Analyse von Quellcode und laufenden Systemen
- Analyse von Architekturen
- Modellierung, Dokumentation und Kommunikation von Architekturen
- Build- und Deployment
- verteiltem Arbeiten in Teams
- Planung und Dokumentation von Architektur- und Entwicklungstätigkeiten

LZ 5-2: Werkzeuge bedarfsgerecht auswählen (R2)

- Softwarearchitekten haben verstanden, dass ihre Arbeitsumgebung und ihre Arbeitsmittel von den jeweiligen Randbedingungen und Einflussfaktoren abhängen;
- sie können bedarfsgerecht Kriterien zur Auswahl von Werkzeugen aufstellen.

6 Beispiele für Softwarearchitekturen

Dauer: 60 Min.	Übungszeit: keine

Dieser Abschnitt ist nicht prüfungsrelevant.

LZ 6-1: Bezug von Anforderungen zu Lösung erfassen (R3)

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

LZ 6-2: Technische Umsetzung einer Lösung nachvollziehen (R3)

Softwarearchitekten können für mindestens ein Beispiel die technische Umsetzung (Implementierung, technische Konzepte, eingesetzte Produkte, Lösungsstrategien) einer Lösung nachvollziehen.

7 Quellen und Referenzen zu Softwarearchitektur

Quellen und Referenzen werden ein einem eigenständigen Dokument gepflegt, das im Download-Bereich der iSAQB-Website frei zur Verfügung steht (http://www.isaqb.org/documents).

Begriffserklärungen und -definitionen finden Sie im frei verfügbaren iSAQB-Glossar, ebenfalls im Download-Bereich der iSAQB-Website.