Curriculum für

Certified Professional for Software Architecture (CPSA)

– Foundation Level –

Version 3.01 (05. Mai 2015)

Inhaltsverzeichnis

Verzeichnis der Lernziele

0 Einleitung

0.1 Was vermittelt eine Foundation Level Schulung?

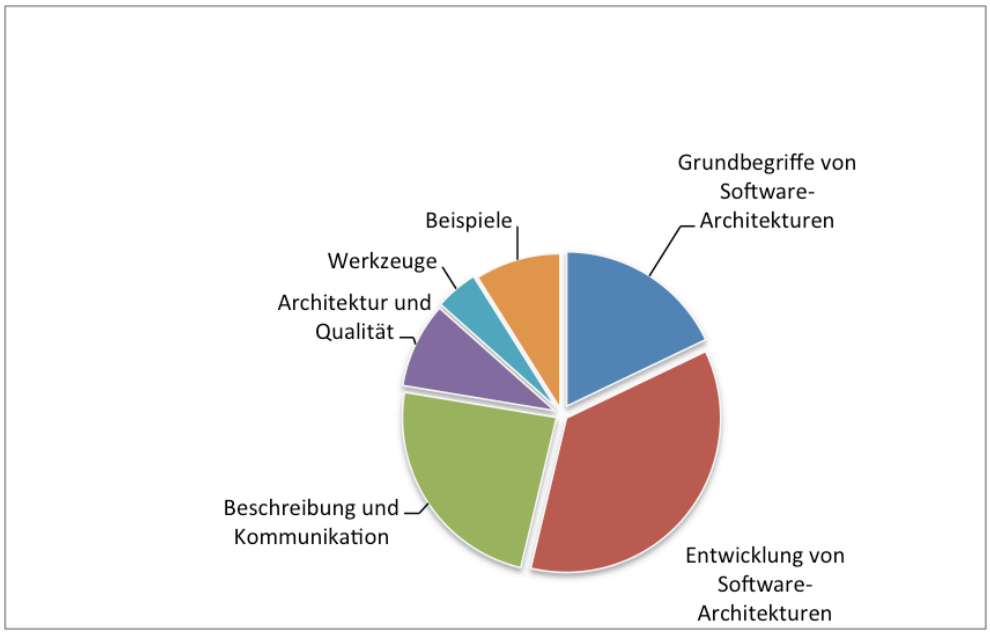
Lizensierte Schulungen zum **Certified Professional for Software Architecture – Foundation Level** (CPSA-F) vermitteln den Teilnehmern das notwendige Wissen und die notwendigen Fähigkeiten, um für kleine und mittlere Systeme ausgehend von einer hinreichend detailliert beschriebene Anforderungsspezifikation eine dem Problem angemessene Softwarearchitektur zu entwerfen und zu dokumentieren. Teilnehmer erhalten das Rüstzeug, um problembezogene Entwurfsentscheidungen auf der Basis ihrer vorab erworbenen Praxiserfahrung zu treffen.

Insbesondere umfassen solche 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, sowie folgende Fähigkeiten:
  - mit anderen Projektbeteiligten, insbesondere aus den Bereichen Anforderungsmanagement, Entwicklung, Projektmanagement und Test wesentliche Softwarearchitekturentscheidungen abzustimmen,
  - Softwarearchitekturen auf Basis von Sichten, querschnittlichen Konzepten, Entscheidungen sowie Architekturmustern und -stilen zu dokumentieren und kommunizieren,
  - die wesentlichen Schritte beim Entwurf von Softwarearchitekturen zu verstehen und für kleine und mittlere Systeme selbständig durchzuführen

Den Teilnehmern einer CPSA-F Schulung wird erfahrbar gemacht, Entwurf und Dokumentation der Architektur sowie die tatsächliche Implementierung konsistent zueinander sind. Die jeweiligen Ergebnisse können sich, abhängig von Vorgehen, Prozesse und Organisation, gegenseitig beeinflussen und im Idealfall stetig aktualisiert und verbessert werden.

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
- Grundlagen der Modellierung und Abstraktion
- 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
- 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.

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	R-1	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	R-2	Diese Inhalte sollen die Teilnehmer grundsätzlich verstehen. Sie werden in Schulungen i.d.R. nicht durch Übungen vertieft.	Inhalte können geprüft werden
Kennen	R-3	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 Zertfizierungsprogramm

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 zu Advanced-Level
- Optional: Andere Zertifizierungsprogramme

0.8 Änderungshistorie

Datum	Änderer	Beschreibung
2014-07-11	pg	Bugfix aus dem Englischen Dokument übernommen: In LZ 1.7 fünften Spiegelstrich gelöscht, weil Wiederholung von drittem Spiegelstrich

2014-07-31	MISO	• Die Referenzen auf die POSA-Serie in LZ 2-10 korrigiert. Die Referenz selbst ebenfalls korrigiert (4-stellige Jahreszahl) • Die Referenzen mit Leerzeichen, z.B. [Zörner 2012] durch die korrekte Schreibweise ohne Leerzeichen ersetzt.
2015-01-06	AG Foundation	• Div. Fehlerbehebungen • Finalisierung V 3.0 in AG-Sitzung Januar 2015
2015-04-30	AR, GS, PG	• Kleinere Anpassungen

1 Grundbegriffe von Softwarearchitekturen

Dauer: 135 Min	Übungszeit: 45 Min

1.1 Begriffe und Konzepte

Softwarearchitektur, Struktur, Bausteine/Komponenten, Schnittstellen, Beziehungen, übergreifende Konzepte/Aspekte, Architekturziele

Softwarearchitekten und deren Verantwortlichkeit, Rolle, Aufgaben und benötigte Fähigkeiten, Stakeholder,

Funktionale und nichtfunktionale Anforderungen, Architekturziele, 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
- Nutzen und Grenzen der Metapher „Gebäudearchitektur“ für Software (R2)

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)

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, abhän-

gig 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.
- 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)

- Teilnehmer sollen die Rolle von Softwarearchitekten im Zusammenhang zu anderen Stakeholdern kennen und erklären können, insbesondere die möglichen Zusammenhänge zu:
  - Anforderungsanalyse (Systemanalyse, Requirements-Management, Fachbereich)
  - Implementierung
  - Projektleitung und -management
  - Qualitätssicherung
  - IT-Betrieb (Produktion, Rechenzentren), zutreffend primär für Informationssysteme
  - Hardware-Entwicklung

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

- 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)

- Teilnehmer sollen die Bedeutung von Architekturzielen, Randbedingungen und Einflussfaktoren für die Gestaltung von Softwarearchitekturen aufzeigen können
- Erklärung von (langfristigen) Architekturzielen sowie deren Abgrenzung gegen (kurzfristige) Projektziele
- Auf Basis bestehender Anforderungen Architekturziele identifizieren und präzisieren

- Zusammenhang zwischen Anforderungen und Lösungen erläutern

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

- Softwarearchitekten sollten Annahmen oder Voraussetzungen explizit darstellen und implizite Annahmen vermeiden
- Implizite Annahmen bewirken potentielle Missverständnisse zwischen beteiligten Stakeholdern.

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

- Weitere Ebenen von Architektur, z.B. Enterprise-IT-Architektur/Geschäftsarchitektur, Infrastrukturarchitektur (etwa nach [Dern2006]): Es gibt innerhalb der IT von Informationssystemen mehrere Ebenen von Architekturen:
  - Infrastruktur-Architektur: Struktur der technischen Infrastruktur, Hardware, Netze etc.
  - Hardware-Architektur (für hardwarenahe Systeme)
  - Softwarearchitektur: Struktur einzelner Software-Systeme. Dies ist der Fokus von Softwarearchitekten im Sinne des iSAQB und CPSA-F.
  - Unternehmens-IT-Architektur, Enterprise-IT-Architektur: Struktur von Anwendungslandschaften. Nicht inhaltlicher Fokus von CPSA-F.
  - Geschäftsprozess-Architektur, Business-Architektur: Struktur von Geschäftsprozessen. Nicht inhaltlicher Fokus von CPSA-F.

LZ 1-10: Typen von softwareintensiven Systemen unterscheiden (R2)

- Differenzierung von Softwarearchitektur für unterschiedliche Typen von IT-Systemen verstehen:
  - Zusammenhang mit der System- oder Gesamtarchitektur für eingebettete oder hardwarenahe Systeme.
  - Besonderheiten des Hardware-/Software-Codesigns verstehen (zeitliche und inhaltliche Abhängigkeiten von Hard- und Softwareentwurf)

2 Entwurf und Entwicklung von Softwarearchitekturen

Dauer: 270 Min	Übungszeit: 90 Min

2.1 Begriffe und Konzepte

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, iterativ/inkrementeller Entwurf, Domain-Driven Design, Werkzeug-Material-Ansatz

2.2 Lernziele

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

- Teilnehmer können grundlegende Vorgehensweisen der Architekturentwicklung benennen, erklären und anwenden
- 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
- Einflussfaktoren und Randbedingungen als Beschränkungen beim Architekturentwurf (Global Analysis – vgl. [Hofmeister+2000])

LZ 2-2: Architekturen entwerfen (R1)

- Teilnehmer können Entwurf von Architekturen auf Basis bekannter funktionaler und nichtfunktionaler Anforderungen für nicht sicherheits- oder unternehmenskritische Software-Systeme durchführen 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 (Hierarchisierung) sowie Präzisierung von Bausteinen anwenden
- Entwurf einzelner Architektursichten, insbesondere Verteilungs-, Baustein- und Laufzeitsicht und deren Konsequenzen auf den zugehörigen Quellcode beschreiben
- Abbildung der Architektur auf den Quellcode festlegen und die damit verbundenen Konsequenzen bewerten
- Trennung fachlicher und technischer Bestandteile in Architekturen begründen und anwenden

- Fachliche Strukturen entwerfen und begründen
  - Fachliche Bausteine (Entitäten, Services) identifizieren, begründen und dokumentieren
  - Zusammenwirken fachlicher und technischer Bestandteile von Systemen entwerfen und erläutern
- Den starken Einfluss nichtfunktionaler Anforderungen (wie beispielsweise Änderbarkeit, Robustheit, Effizienz) kennen und in Architektur- und Entwurfsentscheidungen berücksichtigen.

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

- Teilnehmer sollen Randbedingungen und Einflussfaktoren als Einschränkungen der Entwurfsfreiheit erarbeiten und berücksichtigen
- Einfluss von Qualitätsanforderungen auf Architekturen erkennen und berücksichtigen
- Einfluss technischer Entscheidungen und Konzepte auf Architekturen erkennen und berücksichtigen
- (Möglicher) Einfluss organisatorischer Strukturen auf Bausteinstrukturen erkennen und berücksichtigen (R2)

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

- Übergreifende technische / querschnittliche Konzepte entscheiden / entwerfen, beispielsweise Persistenz, Kommunikation, GUI, Fehlerbehandlung, Nebenläufigkeit
- Mögliche gegenseitige Abhängigkeiten dieser Entscheidungen erkennen und beurteilen

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

- Datenfluss- bzw datenzentrierte Architekturstile (R1)
  - Pipes und Filter
- Hierarchische Architekturstile (R1)
  - Schichten, Layer
- Architekturstile für interaktive Systeme (R2)
  - Model-View-Controller
  - Model-View-Presenter
- Heterogene Architekturstile (R1)
- Architekturstile für asynchrone Systeme (R3)
  - nach Hohpe (Messaging, async-Muster)
- Architekturstile für verteilte Systeme (R3)
- Weitere Architekturmuster und –stile (R3), z.B.:
  - Eventbasierte Systems
  - CQRS
  - nach Fowler/PoEAA
  - nach POSA (z.B. Blackboard oder Microkernel)
- Wesentliche Quellen für Architekturmuster kennen, beispielsweise POSA-Literatur und PoEAA (für Informationssysteme). (R3)
- Beispiele weiterer Pattern-Languages kennen (R3)

- Organizational Patterns
- Reengineering Patterns
- Security Patterns
- Client/Server Patterns
- Patterns für Distributed Systems
- Weitere je nach Schwerpunkt der jeweiligen Schulung

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

- Geheimnisprinzip (*Information Hiding*)
- Kopplung und Kohäsion
- Trennung von Verantwortlichkeiten (*Separation of Concerns*)
- Offen-Geschlossen-Prinzip (*Open-/Closed-Principle*)
- Umkehrung von Abhängigkeiten durch Schnittstellen (*Dependency Inversion*)
- Dependency Injection zur Externalisierung von Abhängigkeiten
- Zusammenhang zwischen Abhängigkeiten im Modell und im Quellcode von Programmiersprachen

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

- Teilnehmer sollen Abhängigkeiten und Kopplung zwischen Bausteinen verstehen und gezielt einsetzen
- Arten der Kopplung (strukturell, zeitlich, über Datentypen, über Hardware etc.) aufzeigen können
- Konsequenzen von Kopplung erkennen
- Möglichkeiten zur Auflösung bzw. Reduktion von Kopplung kennen
- Umsetzung von Beziehungen in (objektorientierten) Programmiersprachen
  - Konstruktoren
  - Factory-Muster
  - Dependency-Injection

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

- Teilnehmer sollen wünschenswerte Eigenschaften (Kapselung, Information Hiding, Geheimnisprinzip) von Bausteinen und Strukturelementen kennen und anwenden
- Blackbox und Whitebox Bausteine
- Arten der Zusammensetzung von Bausteinen (Schachtelung, Benutzung/Delegation, Vererbung)
- UML-Notation für verschiedene Bausteine und deren Zusammensetzung

- Pakete als semantisch schwache Form der Aggregation von Bausteinen
- Komponenten mit fest definierten Schnittstellen als semantisch präzisere Form der Aggregation

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

- Teilnehmer sollen die Bedeutung von Schnittstellen kennen. Sie sollen Schnittstellen zwischen Architekturbausteinen sowie externe Schnittstellen zwischen Architekturbausteinen und Elementen außerhalb des Systems entwerfen bzw. festlegen können.
- Teilnehmer sollen wünschenswerte Eigenschaften von Schnittstellen kennen:
  - Einfach zu erlernen, einfach zu benutzen, einfach zu erweitern
  - Schwer zu missbrauchen
  - Funktional vollständig aus Sicht der Nutzer oder nutzenden Bausteine
- Teilnehmer sollen Schnittstellen beschreiben und dokumentieren können
- Teilnehmer sollen unterschiedliche Betrachtungsweisen von Schnittstellen kennen (R3):
  - ressourcenorientierter Ansatz (wie in REpresentational State Transfer)
  - serviceorientierter Ansatz (wie bei WS-*/SOAP-basierten Webservices)

LZ 2-10: Architekturrelevante Entwurfsmuster verstehen und anwenden (R2)

- Sprachunabhängige Entwurfsmuster verstehen, etwa: Adapter, Wrapper, Gateway, Facade, Registry, Broker
- Weitere Entwurfsmuster (Design Patterns) kennen, die nicht unbedingt Bestandteil einer lizenzierten Schulung sein müssen

Referenzen

[Fowler2003]

[Gharbi+2014]

[Martin2003]

POSA-Serie, insbesondere [Buschmann+1996] und [Buschmann+2007]

[Starke2014]

3 Beschreibung und Kommunikation von Softwarearchitekturen

Dauer: 150 Min	Übungszeit: 90 Min

3.1 Begriffe und Konzepte

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)

- Verständlichkeit, Korrektheit, Effizienz, Angemessenheit, Wartbarkeit
- Orientierung von Form, Inhalt und Detailgrad an Zielgruppe der Dokumentation
- Verständlichkeit technischer Dokumentation kann nur von Lesern beurteilt werden

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

- Beschreibung von Softwarearchitekturen stellt besondere Anforderungen aufgrund der vielseitigen Stakeholder
  - Unterschiedliche Leserkreise: Management, Entwickler, QS sowie andere Softwarearchitekten
  - Unterschiedliche Autoren: Softwarearchitekten, Entwickler und ggfs. weitere.

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

- Teilnehmer sollten mindestens folgende UML Diagramme zur Notation von Architektursichten kennen:
  - Klassen-, Paket- und Komponentendiagramme
  - Verteilungsdiagramme
  - Sequenz- und Aktivitätsdiagramme
  - Zustandsdiagramme
- Nutzen von template-/schablonenbasierter Dokumentation

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

- Baustein- oder Komponentensicht (Aufbau des Systems aus Softwarebausteinen)
- Laufzeitsicht (dynamische Sicht, Zusammenwirken der Softwarebausteine zur Laufzeit, Zustandsmodelle)
- Verteilungs-/Deploymentsicht (Abbildung von Softwarebausteinen auf Hardware oder Ausführungsumgebungen)

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

- Fachlichen und technischen Kontext differenzieren

LZ 3-6: Querschnittliche und technische Architekturkonzepte erläutern und anwenden (R1)

- Bedeutung übergreifender technischer Konzepte (Prinzipien, Architekturaspekte) erklären
- Einige typische Konzepte benennen (z.B. Persistenz, Ablaufsteuerung, GUI, Verteilung/Integration)

LZ 3-7: Schnittstellen beschreiben (R1)

- Beschreibung und Spezifikation von Schnittstellen
- Differenzierung interne/externe Schnittstellen

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

- Systematische Herleitung von Architekturentscheidungen dokumentieren und begründen

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

- Die Beschreibungsmittel für Softwarearchitekturen unterstützen auch bei deren Entwurf und Erstellung
- Sprache und Ausdrucksmittel technischer Dokumentation sollte an den Fähigkeiten und Zielen der Leser ausgerichtet werden

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

- Grundlagen mehrerer der publizierten Frameworks zur Beschreibung von Softwarearchitekturen, beispielsweise
  - TOGAF, RM/ODP, ISO/IEEE-42010 (vormals 1471),
  - arc42, FMC, Tigris
- Ideen und Beispiele von Checklisten für die Erstellung, Dokumentation und Prüfung von Softwarearchitekturen
- Mögliche Werkzeuge zur Erstellung und Pflege von Architekturdokumentation

Referenzen

[Clements+2002]

[Gharbi+2014]

[Starke2014]

[Zörner2012]

4 Softwarearchitekturen und Qualität

Dauer: 60 Min	Übungszeit: 60 Min

4.1 Begriffe und Konzepte

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)

- Begriff der Qualität (angelehnt an DIN/ISO 25010, vormals 9126) und Qualitätsmerkmale erklären
- Qualitätsmodelle (wie etwa DIN/ISO 25010) erklären
- Zusammenhang und Wechselwirkungen von Qualitätsmerkmalen erläutern, beispielsweise:
  - Flexibilität versus Robustheit
  - Speicherplatz versus Laufzeit

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

- Teilnehmer sollen Qualitätsanforderungen an Software und Softwarearchitekturen konkret formulieren und beispielsweise in Form von Szenarien und Qualitätsbäumen darstellen können.
- Erstellung von Szenarien und Qualitätsbäumen erklären und durchführen
- Beispielhafte Qualitätsanforderungen an Software formulieren

LZ 4-3: Softwarearchitekturen qualitativ bewerten (R2)

- Teilnehmer sollen methodische Vorgehen zur Analyse und Bewertung von Softwarearchitekturen kennen und für kleinere Beispiele anwenden können.
- Vorgehen bei qualitativer Bewertung von Softwarearchitekturen nach ATAM
- Zur qualitativen Bewertung von Architekturen können folgende Informationsquellen helfen:
  - Anforderungen, insbesondere in Form von Qualitätsbäumen und –szenarien.
  - Architekturdokumentation
  - Architektur- und Entwurfsmodelle
  - Quellcode
  - Metriken

LZ 4-4: Softwarearchitekturen quantitativ bewerten (R2)

- Quantitative Bewertung (Messung) von Software, insbesondere von Quellcode, kann helfen, kritische Teile innerhalb von Systemen zu identifizieren.

- Zur Bewertung von Architekturen können weitere Informationen zu Rate gezogen werden, etwa:
  - Quellcode (insbesondere Metriken wie Lines-of-Code, zyklomatische Komplexität, ein- und ausgehende Abhängigkeiten, Instabilität, Abstraktheit, Distanz)
  - bekannte Fehler in Quellcode, insbesondere Fehlercluster
  - Testfälle und Testergebnisse
  - Anforderungs- und Lösungsmodelle

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

- Teilnehmer sollen Taktiken, Praktiken sowie technische Möglichkeiten zur Erreichung wichtiger Qualitätsziele von Software-Systemen (unterschiedlich für eingebettete Systeme bzw. Informationssysteme) erklären und anwenden, beispielsweise:
  - Effizienz/Performance
  - Wartbarkeit, Änderbarkeit, Erweiterbarkeit, Flexibilität
  - Identifikation entsprechender Risiken

Referenzen

[Bass+2003]

[Clements+2002]

[Gharbi+2014]

[Martin2003]

[Starke2014]

5 Werkzeuge für Softwarearchitekten

Dauer: 45 Min	Übungszeit: keine

5.1 Begriffe und Konzepte

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

5.2 Lernziele

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

- Teilnehmer sollen die wichtigsten Kategorien von Werkzeugen sowie deren Nutzen für die Arbeit von Softwarearchitekten benennen und erläutern können.

LZ 5-2: Werkzeuge bedarfsgerecht auswählen (R2)

- Die Arbeitsumgebung und die Arbeitsmittel von Softwarearchitekten hängen von den jeweiligen Randbedingungen und Einflussfaktoren ab.

6 Beispiele für Softwarearchitekturen

Dauer: 60 Min	Übungszeit: keine

Dieser Abschnitt ist nicht prüfungsrelevant.

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

- Teilnehmer sollen an mindestens einem Beispiel den Bezug von Anforderungen und Architekturzielen zu Lösungsentscheidungen erkennen und nachvollziehen.

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

- Teilnehmer sollen für ein Beispiel die technische Umsetzung (Implementierung, technische Konzepte, eingesetzte Produkte, Lösungsstrategien) einer Lösung nachvollziehen können.

7 Quellen und Referenzen zu Softwarearchitektur

Dieser Abschnitt enthält Quellenangaben, die für den Aufbau des Lehrplans verwendet wurden.

B

[Bachmann2000]
Felix Bachmann, Len Bass, Jeromy Carriere, Paul Clements, David Garlan, James Ivers, Robert Nord, Reed Little: Software Architecture Documentation in Practice: Documenting Architectural Layers. Special Report CMU/SEI-2000-SR-004 March 2000, CMU, 2000,

[Bass+2012]
Bass, L., Clements, P. und Kazman, R.: Software Architecture in Practice. 3rd Edition, Addison-Wesley, Reading, Mass., p. 640, 2012, ISBN10 978-0-3218-1573-6

[Berns+2010]
Berns K., Schürmann B., Trapp, M.: Eingebettete Systeme, Systemgrundlagen und Entwicklung eingebetteter Software. 1. Ausgabe, Springer, p. 270, ISBN13 978-3-8348-9661-2

[Bosch2000]
Jan Bosch (Autor): Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach (ACM Press). 07. June 2000, Addison-Wesley Longman, Amsterdam, p. 368, 2000,
ISBN10 201674947, ISBN13 978-0201674941

[Buschmann+1996]
Frank Buschmann (Autor), Regine Meunier (Autor), Hans Rohnert (Autor), Peter Sommerlad (Autor): A System of Patterns: Pattern-Oriented Software Architecture: 1 (Wiley Software Patterns). 1. Auflage (12. July 1996), John Wiley & Sons, p. 476, 1996, ISBN10 471958697, ISBN13 978-0471958697

[Buschmann+2007]
Frank Buschmann (Autor), Kevlin Henney (Autor), Douglas C. Schmidt (Autor): Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing. Volume 4: Pattern Language for Distributed Object Computing v. 4. 1. Auflage (16. März 2007), John Wiley & Sons, p. 636, 2007, ISBN10 470059028, ISBN13 978-0470059029

C

[Clements+2002]
Paul Clements (Autor), Rick Kazman (Autor), Mark Klein (Autor): Evaluating Software Architectures Methods and Case Studies. Addison-Wesley Longman, Amsterdam, p. 368, 2001, ISBN10 020170482X, ISBN13 978-0201704822

[Clements+2010]
Paul Clements (Autor), Felix Bachmann (Autor), Len Bass (Autor), David Garlan (Autor), James Ivers (Autor), Reed Little (Autor), Paulo Merson (Autor), Robert L. Nord (Autor): Documenting Software Archictectures Views and Beyond. 2nd revised edition. (5. Oktober 2010), Addison Wesley, p. 608, 2010, ISBN10 321552687, ISBN13 978-0321552686

D

[Dern2006]
Gernot Dern (Autor): Management von IT-Architekturen: Leitlinien für die Ausrichtung, Planung und Gestaltung von Informationssystemen (Edition CIO), Edition 3., durchges. Aufl. 2009 (12. März 2009), Vieweg+Teubner Verlag, p. 343, 2009, ISBN10 3834807184, ISBN13 9783834807182

E

[Evans2004]
Eric J. Evans (Autor): Domain-Driven Design: Tackling Complexity in the Heart of Software. 1. Ausgabe (20. August 2003), Addison-Wesley Longman, Amsterdam, p. 529, 2003, ISBN10 321125215, ISBN13 978-0321125217

F

[Fowler2003]
Martin Fowler (Autor): Patterns of Enterprise Application Architecture. 1. Ausgabe (5. November 2002), Addison-Wesley Longman, Amsterdam, p. 560, 2002, ISBN10 321127420, ISBN13 978-0321127426

G

[Gamma+1995]
Erich Gamma (Author), Richard Helm (Author), Ralph Johnson (Author), John M. Vlissides (Author): Design Patterns: Elements of Reusable Object-Oriented Software. edition (November 10, 1994), Addison-Wesley Professional, p. 416, 1994, ISBN10 201633612, ISBN13 978-0201633610

[Gharbi+2014]
Gharbi M., Koschel, A., Rausch, A., Starke, G.: Basiswissen für Softwarearchitekten. 2. Auflage, Dpunkt Verlag, 2014, ISBN10 3864901650

[Gorton2011]
Gorton, I.: Essential Software Architecture. 2nd edition (2011), Springer, p. 242, 2011, ISBN13 978-3-642-19176-3

H

[Hargis+2004]
Gretchen Hargis (Autor), Michelle Carey (Autor), Ann Hernandez (Autor): Developing Quality Technical Information: A Handbook for Writers and Editors (IBM Press Series-Information Management). 2nd ed. (6. April 2004), Prentice Hall, p. 432, 2004, ISBN10 131477498, ISBN13 978-0131477490

[Hofmeister+2000]
Christine Hofmeister (Author), Robert Nord (Author), Dilip Soni (Author): Applied Software Architecture. 1 edition (November 14, 1999), Addison-Wesley Professional, p. 432, 1999, ISBN10 321643348, ISBN13 978-0321643346

[Hofmeister+2005]
C. Hofmeister, R. Nord, D. Soni.: Global Analysis: Moving from Software Requirements Specification to Structural Views of the Software Architecture. Vol. 152, Issue 4, pp. 187-197, August 2005.,
IEE Proceedings Software, p. 11, 2005

J

[Josuttis2008]
Josuttis, N.M.: SOA in der Praxis - System-Design für verteilte Geschäftsprozesse. , Dpunkt Verlag,
p. 408, 2008, ISBN13 978-3-89864-476-1

K

[Koschel+2008]
Dunkel J., Eberhart A., Fischer S., Kleiner C, Koschel A.: Systemarchitekturen für Verteilte Anwendungen - Client-Server, Multi-Tier, SOA, Event-Driven Architectures, P2P, Grid, Web 2.0., 1. Ausgabe, Carl Hanser Verlag München, p. 3005, 2008, ISBN13 978-3-446-41321-4

[Kruchten1995]
Philippe Kruchten: The 4+1 View Model of Architecture. [[edition::vol. 12 [6], pp. 45-50, 1995. DOI: 10.1109/52.469759]], IEEE Software, p. 16, 1995,

M

[Martin2003]
Martin, R. C.: Agile Software Development, Principles, Patterns and Practices., Prentice Hall, 2002,
p. 529, 2002, ISBN10 135974445, ISBN13 978-0135974445

[MVP]
http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter

P

[Parnas1972]
David Parnas: On the criteria to be used in decomposing systems into modules. Volume 15 Issue 12, Dec. 1972 Pages 1053-1058, Communications of the ACM, CACM Homepage archive, 1972,

Q

[Quian+2010]
Qian, K., Fu, X., Tao, L., Xu Ch., Herrera, J.: Software Architecture and Design Illuminated. 1st edition, Jones and Bartlett, p. 387, 2010, ISBN13 9780763754204

R

[Reussner+2008]
Ralf Reussner (Hrsg.) (Autor), Wilhelm Hasselbring (Hrsg.) (Autor): Handbuch der Software-Architektur (Gebundene Ausgabe). 2. überarb. und erw. Auflage (15. Dezember 2008), dpunkt Verlag, p. 575, 2008, ISBN10 3898645592, ISBN13 978-3898645591

[Rupp+2012]
Rupp, Ch., Queins, S., SOPHISTen: UML 2 glasklar. 4. aktualisierte u. erweiterte Auflage, Carl Hanser Verlag München, p. XX, 2012, ISBN13 978-3-446-43057-0

S

[Schmidt+2000]
Douglas C. Schmidt (Autor), Michael Stal (Autor), Hans Rohnert (Autor), Frank Buschmann (Autor): Pattern-Oriented Software Architecture: Volume 2: Patterns for Concurrent and Networked Objects. 1. Auflage (15. August 2000), John Wiley & Sons, p. 666, 2000, ISBN10 471606952, ISBN13 978-0471606956

[Shaw+1996]
Mary Shaw (Autor), Shaw (Autor), David Garlan (Autor): Software Architecture: Perspectives on an Emerging Discipline. (Oktober 1996), Prentice Hall, p. 242, 1996, ISBN10 131829572, ISBN13 978-0131829572

[Siedersleben2004]
Johannes Siedersleben (Autor): Moderne Software-Architektur: Umsichtig planen, robust bauen mit Quasar. 1., Aufl. (Juli 2004), Dpunkt Verlag, p. 281, 2004, ISBN10 3898642925, ISBN13 978-3898642927

[Starke+2011]
Starke, Gernot und Peter Hruschka: Software-Architektur kompakt. 2. Auflage, Spektrum Akademischer Verlag, p. 121, 2011, ISBN13 978-3-8274-2835-6

[Starke2014]
Gernot Starke (Autor): Effektive Softwarearchitekturen. 6. überarbeitete Auflage (01/2014), Carl Hanser Verlag GmbH & Co. KG, p. 409, 2014, ISBN13 978-3-446-43614-5

T

[Tilkov2011]
Tilkov, S.: REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien. 2. Auflage (vor. April 2014), Dpunkt Verlag, p. 2011, ISBN13 978-3-86490-120-1

[Toth2013]
Toth, S.: Vorgehensmuster für Softwarearchitektur - Kombinierbare Praktiken in Zeiten von Agil und Lean. 1. Ausgabe, Carl Hanser Verlag München, p. 249, 2013, ISBN13 978-3-446-43615-2

V

[Vernon2013]
Vernon, V.: Implementing Domain Driven Design. 1st edition (Febr. 2013), Addison Wesley, p. 656, 2013, ISBN13 978-0-3218-3457-7

Z

[Zörner2012]
Zörner, St.: Softwarearchitekturen dokumentieren und kommunizieren. 1. Auflage, Carl Hanser Verlag München, p. 280, 2012, ISBN13 978-3-446-42924-6

