Curriculum für

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Modul
FUNAR

Funktionale Softwarearchitektur

2018.1-DE-20210113

Inhaltsverzeichnis

Einführung: Allgemeines zum iSAQB Advanced Level . . . 2
Was vermittelt ein Advanced Level Modul? . . . 2
Was können Absolventen des Advanced Level (CPSA-A)? . . . 2
Voraussetzungen zur CPSA-A-Zertifizierung . . . 2
Grundlegendes . . . 3
Was vermittelt das Modul „FUNAR“? . . . 3
Gliederung des Lehrplans für Funktionale Softwarearchitektur und empfohlene zeitliche Aufteilung . . . 3
Dauer, Didaktik und weitere Details . . . 4
Voraussetzungen . . . 4
Gliederung des Lehrplans . . . 4
Ergänzende Informationen, Begriffe, Übersetzungen . . . 5
1. Systemstruktur . . . 6
1.1. Begriffe und Konzepte . . . 6
1.2. Lernziele . . . 6
1.3. Referenzen . . . 6
2. Technologien . . . 7
2.1. Begriffe und Konzepte . . . 7
2.2. Lernziele . . . 7
2.3. Referenzen . . . 7
3. Umsetzung von funktionalen Anforderungen . . . 8
3.1. Begriffe und Konzepte . . . 8
3.2. Lernziele . . . 8
3.3. Referenzen . . . 8
4. Umsetzung von nicht-funktionalen Anforderungen . . . 9
4.1. Begriffe und Konzepte . . . 9
4.2. Lernziele . . . 9
4.3. Referenzen . . . 9
5. Architekturmuster . . . 10
5.1. Begriffe und Konzepte . . . 10
5.2. Lernziele . . . 10
5.3. Referenzen . . . 10
6. Beispiele . . . 11
6.1. Begriffe und Konzepte . . . 11
6.2. Lernziele . . . 11
Referenzen . . . 12

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

Einführung: Allgemeines zum iSAQB Advanced Level

Was vermittelt ein Advanced Level Modul?

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

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

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

CPSA-A-Absolventen können:

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

Voraussetzungen zur CPSA-A-Zertifizierung

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

Grundlegendes

Was vermittelt das Modul „FUNAR“?

Das Modul präsentiert den Teilnehmerinnen funktionale Software-Architektur als Alternative zu objektorientierter Architektur. Im Vergleich zu OO-Architektur setzt die funktionale Software-Architektur auf unveränderliche Daten, algebraische Abstraktionen und eingebettete domänenspezifische Sprachen. Das Resultat sind flexible und robuste Architekturen, die gegenüber OO weniger komplex sind und weniger versteckte Abhängigkeiten mit sich bringen. (*Die Vergleiche zu objektorientierter Softwarearchitektur tragen dem Umstand Rechnung, dass die Autoren davon ausgehen, dass die meisten potenziellen Teilnehmer an einer Advanced-Schulung mit objektorientierter Software-Architektur vertraut sind. Für die Schulung essenziell ist dieser Vergleich allerdings nicht und kann zugunsten einer Vertiefung der Inhalte auch entfallen.*)

Anders als bei OO-Architekturen sind FP-Architekturen direkt Code. In diesem Modul können deshalb alle Architekturprinzipien durch konkreten Code illustriert werden und sind damit anschaulich erlernbar. (*Eine Reihe von Sprachen kommt für die Begleitung einer solchen Schulung in Betracht. Besonders geeignet können Haskell und Racket sein, die alle Architekturprinzipien dieses Lehrplans direkt unterstützen. Racket mag sich besonders gut für Schulungen eignen, denen eine Einführung in die funktionale Programmierung mit den dort eingebauten Lehrsprachen vorangeht.*)

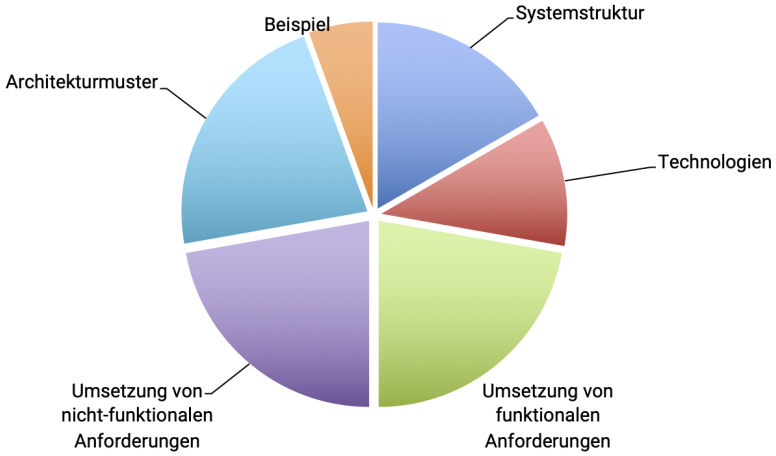
Nach Abschluss des Moduls kennen die Teilnehmerinnen die wesentlichen Prinzipien funktionaler Architektur und können diese beim Entwurf von Software-Systemen anwenden. Sie kennen die Eigenheiten funktionaler Programmiersprachen und können diese bei der Implementierung von Software-Systemen effektiv ausnutzen. Sie können Domänenwissen direkt in ausführbaren Code umwandeln und daraus systematisch algebraische Abstraktionen entwickeln.

Gliederung des Lehrplans für Funktionale Softwarearchitektur und empfohlene zeitliche Aufteilung

Die Gliederung des Lehrplans richtet sich nach der Gliederung der Aufgaben von Software-Architektur in Simon Brown: Software Architecture for Developers: Volume 1 - Technical Leadership and the Balance with Agility. Leanpub, 2018.

Inhalt	Empfohlene Mindestdauer (min)
1. Systemstruktur	180
2. Technologien	120
3. Umsetzung von funktionalen Anforderungen	240
4. Umsetzung von nicht-funktionalen Anforderungen	240
5. Architekturmuster	240
6. Beispiel	60
Summe	1080 (18h)

Zeitliche Aufteilung der Themengebiete



Dauer, Didaktik und weitere Details

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

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

Methodische Kompetenz:	10 Punkte
Technische Kompetenz:	20 Punkte
Kommunikative Kompetenz:	0 Punkte

Voraussetzungen

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

- Grundkenntnisse in funktionaler Programmierung
- Erfahrung bei der Modellierung von Architekturen

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Grundkenntnisse Algebra

Gliederung des Lehrplans

Die einzelnen Abschnitte des Lehrplans sind gemäß folgender Gliederung beschrieben:

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

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

Ergänzende Informationen, Begriffe, Übersetzungen

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

1. Systemstruktur

Dauer: 180 Min.	Übungszeit: 60 Min.

Diese Lerneinheit führt in die grundlegenden Elemente der Struktur funktionaler Software-Systeme ein: Reine Funktionen operieren auf unveränderlichen Werten. Dies steht im Gegensatz zum objektorientierten Modell, in dem Methoden auf gekapseltem Zustand operieren. Diese Funktionen werden zu domänenrelevanter Funktionalität miteinander kombiniert beziehungsweise *komponiert.* Der Entwurf von Funktionen und ihre Komposition wird von ihren Typsignaturen getrieben. Anders als in der objektorientierten Architektur sind Funktionen nicht an Klassen gebunden, sondern werden in Modulen gebündelt.

1.1. Begriffe und Konzepte

Funktionen und Werte, Komposition, Typen, Module

1.2. Lernziele

- Was sollen die Teilnehmer können?
  - Informationen als unveränderliche Werte repräsentieren
  - zentrale Funktionalität als Funktionen formulieren
  - für diese Funktionen Typen konstruieren
- Was sollen die Teilnehmer verstehen?
  - den Unterschied zwischen dem objektorientierten und dem funktionalen Software-Modell
  - die Rolle von Typen beim funktionalen Software-Design
  - Module als Alternative zu Klassen
- Was sollen die Teilnehmer kennen?
  - Beispiele für funktionale System-Architekturen

1.3. Referenzen

[Brown 2018], [Felleisen+ 2014], [Hutton 2016], [Sperber+ 2020]

2. Technologien

Dauer: 120 Min.	Übungszeit: Keine.

Funktionale Software-Architektur ist in der Regel verbunden mit der Verwendung funktionaler Programmiersprachen und Frameworks. Dieser Abschnitt gibt einen Überblick über die wichtigsten Eigenschaften und Beschränkungen verfügbarer Technologien, um Architektinnen die Entscheidung für oder gegen eine Technologie zu ermöglichen.

2.1. Begriffe und Konzepte

statische Typen, dynamische Typen, Endrekursion, strikte bzw. nicht-strikte Auswertung, Laufzeitumgebung

2.2. Lernziele

- Was sollen die Teilnehmer können?
  - eine geeignete Programmiersprache für ein Projekt auswählen
  - geeignete Frameworks und Bibliotheken auswählen
- Was sollen die Teilnehmer verstehen?
  - die Auswirkung der Verwendung strikter bzw. nicht-strikter Auswertung auf die Software-Architektur
  - die Auswirkung der Verwendung statischer bzw. dynamischer Typen
  - die Bedeutung von Endrekursion als Unterscheidungsmerkmal
- Was sollen die Teilnehmer kennen?
  - wichtige funktionale Sprachen und ihre Haupteigenschaften
  - exemplarische Frameworks und Libraries
  - die Zuordnung von Programmiersprachen bzw. deren Eigenschaften zu verschiedenen Laufzeitumgebungen

2.3. Referenzen

[Field+ 1988], [Wikipedia 2020]

3. Umsetzung von funktionalen Anforderungen

Dauer: 180 Min.	Übungszeit: 60 Min.

Dieser Abschnitt behandelt inhärent funktionale Ansätze, um funktionale Anforderungen in Software-Architektur umzusetzen. Domain-driven Design kann mit funktionaler Programmierung statt objektorientierter Programmierung umgesetzt werden. Kombinatormodelle decken verstecktes Domänenwissen auf. Eingebettete domänenspezifische Sprachen ermöglichen einen schnellen Abgleich von Anforderungen und Architektur.

3.1. Begriffe und Konzepte

funktionale Anforderungen, DDD, Kombinatormodelle, eingebettete domänenspezifische Sprachen

3.2. Lernziele

- Was sollen die Teilnehmer können?
  - DDD mit Hilfe funktionaler Bausteine umsetzen
  - rudimentäre Kombinatormodelle formulieren
  - kleine domänenspezifische Sprachen entwerfen
- Was sollen die Teilnehmer verstehen?
  - die Umsetzung von DDD-Entitäten mit funktionaler Programmierung
  - die spezifischen Eigenschaften von Kombinatormodellen
  - die Rolle domänenspezifischer Sprachen bei der Entwicklung
- Was sollen die Teilnehmer kennen?
  - Beispiele für Kombinatormodelle
  - die spezifischen Vorteile von Kombinatormodellen
  - Beispiele für eingebettete domänenspezifische Sprachen

3.3. Referenzen

[Gibbons+ 2003], [Wlaschin 2018]

4. Umsetzung von nicht-funktionalen Anforderungen

Dauer: 180 Min.	Übungszeit: 60 Min.

Dieser Abschnitt behandelt spezifische funktionale Techniken zur Umsetzung nicht-funktionaler Anforderungen wie Datenhaltung, Verteilung und Performance.

4.1. Begriffe und Konzepte

CQRS, Event Sourcing, Parallelisierung, Verteilung

4.2. Lernziele

- Was sollen die Teilnehmer können?
  - für spezifisches Projekt die Datenhaltung mit Event Sourcing entwerfen
  - ein Query-Modell passend zu einem Event-Sourcing-Modell und einer Oberflächen-Anforderung entwerfen
  - Techniken für effektive Parallelisierung auswählen
  - Techniken für effektive Verteilung auswählen
- Was sollen die Teilnehmer verstehen?
  - den Unterschied zwischen „Data Warehousing“ und Event-Sourcing-Ansätzen
  - die Rolle von Query-Modellen
  - die Rolle funktionaler Datenstrukturen bei der Parallelisierung
  - die Rolle funktionaler Datenhaltung bei der Verteilung
- Was sollen die Teilnehmer kennen?
  - Datenparallelität
  - Futures
  - Message Passing

4.3. Referenzen

[Betts+ 2013], [Cesarini+ 2016], [Marlow 2013], [Reppy 2008]

5. Architekturmuster

Dauer: 180 Min.	Übungszeit: 60 Min.

In der funktionalen Software-Architektur gibt es spezifische Architekturmuster. Dazu gehört die Verwendung performanter funktionaler Datenstrukturen, die Verwendung algebraischer Abstraktionen und die UI-Entwicklung mit dem Model-View-Update-Pattern als Gegenmodell zu Model-View-Controller.

5.1. Begriffe und Konzepte

funktionale Datenstruktur, Monoid, Funktor, Monade, Model-View-Update

5.2. Lernziele

- Was sollen die Teilnehmer können?
  - die Möglichkeiten funktionaler Datenstrukturen für die Software-Architektur ausnutzen
  - algebraische Eigenschaften in Domänenobjekten erkennen
  - algebraische Eigenschaften im Code reflektieren
  - UI-Architektur mit Model-View-Update entwerfen
- Was sollen die Teilnehmer verstehen?
  - Trade-Offs der Verwendung funktionaler Datenstrukturen
  - Eigenschaften algebraischer Abstraktionen
  - Unterschied zwischen Model-View-Update und Model-View-Controller
- Was sollen die Teilnehmer kennen?
  - Beispiele für funktionale Datenstrukturen (hash tries, finger trees, red-black trees)
  - Eigenschaften algebraischer Abstraktionen

5.3. Referenzen

[Okasaki 1999], [Chiusano+ 2014]

6. Beispiele

Dauer: 60 Min.	Übungszeit: keine

6.1. Begriffe und Konzepte

Innerhalb jeder lizenzierten Schulung muss mindestens ein Beispiel für Funktionale Softwarearchitektur vorgestellt werden.

Art und Ausprägung der vorgestellten Beispiele können von der Schulung bzw. den Interessen der Teilnehmer abhängen und werden seitens iSAQB nicht vorgegeben.

6.2. Lernziele

- Was sollen die Teilnehmer können?
  - Potential für Parallelisierung und/oder Verteilung erkennen
  - mögliche algebraische Abstraktionen identifizieren
- Was sollen die Teilnehmer verstehen?
  - die Auswirkung der Verwendung unveränderlicher Daten und reiner Funktionen
- Was sollen die Teilnehmer kennen?
  - die Vorteile gegenüber einer traditionell objektorientierten Architektur

Referenzen

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

B

- [Betts+ 2013] Betts et al.: Exploring CQRS and Event Sourcing. Microsoft, 2013.
- [Brown 2018] Simon Brown: Software Architecture for Developers: Volume 1 - Technical leadership and the balance with agility. Leanpub, 2018.

C

- [Cesarini+ 2016] Francesco Cesarini, Steve Vinoski: Designing for Scalability with Erlang/OTP: Implement Robust, Fault-Tolerant Systems. O'Reilly, 2016.
- [Chiusano+ 2014] Paul Chiusano, Runar Bjarnason: Functional Programming in Scala. September, 2014.

F

- [Felleisen+ 2014] Matthias Felleisen et al.: How to Design Programs. MIT Press, 2014. https://htdp.org/
- [Field+ 1988] Field, Harrison: Functional Programming. Addison-Wesley, 1988.

G

- [Gibbons+ 2003] Jeremy Gibbons, Oege de Moor (Herausgeber): The Fun of Programming. Oxford University Press, 2003.

H

- [Hutton 2016] Graham Hutton: Programming in Haskell. Cambridge University Press, 2. Auflage, 2016.

M

- [Marlow 2013] Simon Marlow: Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming. O'Reilly, 2013.

O

- [Okasaki 1999] Chris Okasaki: Purely functional data structures. Cambridge University Press, 1999.

R

- [Reppy 2008] John H. Reppy: Concurrent Programming in ML. Cambridge University University Press, 2008.

S

- [Sperber+ 2020] Michael Sperber, Herbert Klaeren: Schreibe Dein Programm! http://www.deinprogramm.de/

W

- [Wikipedia 2020] Comparison of functional programming languages. Wikipedia
- [Wlaschin 2018] Scott Wlaschin: Domain Modeling Made Functional. The Pragmatic Bookshelf, 2018.