Curriculum für

CPSA Certified Professional for Software Architecture®

– Advanced Level –

Modul: FUNAR

Funktionale Softwarearchitektur

Version 1.0 (September 2018)

Inhaltsverzeichnis

0 Einleitung: Allgemeines zum iSAQB-Advanced-Level

0.1 Was vermittelt ein Advanced-Level-Modul?

- 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.

0.2 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 nichtfunktionaler Anforderungen konzeptionieren, entwerfen und dokumentieren. Entwicklungsteams bei der Umsetzung dieser Maßnahmen begleiten.
- Architekturrelevante Kommunikation in mittleren bis großen Entwicklungsteams steuern und durchführen.

0.3 Voraussetzungen zur CPSA-A-Zertifizierung

- Eine erfolgreiche Ausbildung und Zertifizierung zum CPSA-F (Certified Professional for Software Architecture, Foundation Level).
- Mindestens drei Jahre Vollzeit-Berufserfahrung in der IT-Branche, dabei Mitarbeit an Entwurf und Entwicklung von mindestens zwei unterschiedlichen IT-Systemen.
    - Ausnahmen auf Antrag zulässig (etwa: Mitarbeit in OpenSource-Projekten)
- Aus- und Weiterbildung im Rahmen von iSAQB-Advanced-Level-Schulungen im Umfang von mindestens 70 Credit-Points aus allen drei Kompetenzbereichen (Details siehe iSAQB-Website).
    - Bestehende Zertifizierungen können ggfs. auf diese Credit-Points angerechnet werden. Die Liste der aktuellen Zertifikate, für die Credit Points angerechnet werden, ist auf der iSAQB-Homepage zu finden.
- Erfolgreiche Bearbeitung der CPSA-A Zertifizierungsprüfung.

1 Grundlegendes zum Modul Funktionale Softwarearchitektur (FUNAR)

Das Modul präsentiert den Teilnehmern funktionale Softwarearchitektur als Alternative zu objektorientierter Architektur. Im Vergleich zu OO-Architektur setzt die funktionale Softwarearchitektur 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.[1]

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.[2]

Nach Abschluss des Moduls kennen die Teilnehmer 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.

1.1 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
Systemstruktur	180 min
Technologien	120 min
Umsetzung von funktionalen Anforderungen	240 min
Umsetzung von nicht-funktionalen Anforderungen	240 min
Architekturmuster	240 min
Beispiel	60 min
Gesamt (3 Tage à 360 min)	1.080 min = 18 h

1.2 Dauer, Didaktik und weitere Details

Die oben genannten Zeiten sind Empfehlungen. Die Dauer einer Schulung zum Modul Funktionale Softwarearchitektur sollte mindestens drei Tage betragen, kann aber länger sein. Anbieter können sich durch Dauer, Didaktik, Art und Aufbau der Übungen sowie der

[1] 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 Softwarearchitektur vertraut sind. Für die Schulung essenziell ist dieser Vergleich allerdings nicht und kann zugunsten einer Vertiefung der (eh reichlichen) Inhalte auch entfallen.

[2] 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.

detaillierten Kursgliederung voneinander unterscheiden. Insbesondere die Art der Beispiele und Übungen lässt der Lehrplan komplett offen.

Lizenzierte Schulungen zum Modul Funktionale Softwarearchitektur tragen zur Zulassung zur abschließenden Advanced-Level-Zertifizierungsprüfung folgende Punkte (Credit Points) bei:

Methodische Kompetenz:	10 Punkte
Technische Kompetenz:	20 Punkte
Kommunikative Kompetenz:	00 Punkte

1.3 Voraussetzungen für das Modul Funktionale Software-Architektur

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.

1.4 Gliederung des Lehrplans für das Modul Funktionale Softwarearchitektur

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. Die Lernziele werden differenziert in folgende Kategorien bzw. Unterkapitel:

- Was sollen die Teilnehmer **können**? Diese Inhalte sollen die Teilnehmer nach der Schulung selbständig anwenden können. Innerhalb der Schulung werden diese Inhalte durch Übungen abgedeckt und sind Bestandteil der Modulprüfung Funktionale Softwarearchitektur und/oder der Abschlussprüfung des iSAQB-Advanced-Levels.
- Was sollen die Teilnehmer **verstehen**? Diese Inhalte können in der Modulprüfung Funktionale Softwarearchitektur geprüft werden.
- Was sollen die Teilnehmer **kennen**? Diese Inhalte (Begriffe, Konzepte, Methoden, Praktiken oder Ähnliches) können das Verständnis unterstützen oder das Thema motivieren. Diese Inhalte sind nicht Bestandteil der Prüfungen, werden in Schulungen thematisiert, aber nicht notwendigerweise ausführlich unterrichtet.

1.5 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.

2 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.

2.1 Begriffe und Konzepte

- Funktionen und Werte
- Komposition
- Typen
- Module

2.2 Lernziele

2.2.1 Was sollen die Teilnehmer können?

- Informationen als unveränderliche Werte repräsentieren
- zentrale Funktionalität als Funktionen formulieren
- für diese Funktionen Typen konstruieren

2.2.2 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

2.2.3 Was sollen die Teilnehmer kennen?

- Beispiele für funktionale System-Architekturen

2.3 Referenzen

- Graham Hutton: Programming in Haskell. Cambridge University Press, 2. Auflage, 2016.
- Herbert Klaeren, Michael Sperber: Schreibe Dein Programm! http://www.deinprogramm.de/
- Simon Brown: Software Architecture for Developers: Volume 1 - Technical Leadership and the Balance with Agility. Leanpub, 2018.

3 Technologien

Dauer: 120 min	Übungszeit: Keine.

Funktionale Softwarearchitektur 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 Architekten die Entscheidung für oder gegen eine Technologie zu ermöglichen.

3.1 Begriffe und Konzepte

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

3.2 Lernziele

3.2.1 Was sollen die Teilnehmer können?

- eine geeignete Programmiersprache für ein Projekt auswählen
- geeignete Frameworks und Bibliotheken auswählen

3.2.2 Was sollen die Teilnehmer verstehen?

- die Auswirkung der Verwendung strikter bzw. nicht-strikter Auswertung auf die Softwarearchitektur
- die Auswirkung der Verwendung statischer bzw. dynamischer Typen
- die Bedeutung von Endrekursion als Unterscheidungsmerkmal

3.2.3 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

3.3 Referenzen

- Field, Harrison: Functional Programming. Addison-Wesley, 1988.
- Comparison of Functional Programming Languages. Wikipedia

4 Umsetzung von funktionalen Anforderungen

Dauer: 180 min	Übungszeit: 60 min

Dieser Abschnitt behandelt inhärent funktionale Ansätze, um funktionale Anforderungen in Softwarearchitektur 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.

4.1 Begriffe und Konzepte

Funktionale Anforderungen, DDD, Kombinatormodelle, eingebettete domänenspezifische Sprachen.

4.2 Lernziele

4.2.1 Was sollen die Teilnehmer können?

- DDD mit Hilfe funktionaler Bausteine umsetzen
- rudimentäre Kombinatormodelle formulieren
- kleine domänenspezifische Sprachen entwerfen

4.2.2 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

4.2.3 Was sollen die Teilnehmer kennen?

- Beispiele für Kombinatormodelle
- die spezifischen Vorteile von Kombinatormodellen
- Beispiele für eingebettete domänenspezifische Sprachen

4.3 Referenzen

- Jeremy Gibbons, Oege de Moor (Herausgeber): The Fun of Programming. Oxford University Press, 2003.
- Scott Wlaschin: Domain Modeling Made Functional. The Pragmatic Bookshelf, 2018.

5 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.

5.1 Begriffe und Konzepte

CQRS, Event Sourcing, Parallelisierung, Verteilung.

5.2 Lernziele

5.2.1 Was sollen die Teilnehmer können?

- für ein 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

5.2.2 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

5.2.3 Was sollen die Teilnehmer kennen?

- Datenparallelität
- Futures
- Message Passing

5.3 Referenzen

- Simon Marlow: Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming. O’Reilly, 2013.
- John H. Reppy: Concurrent Programming in ML. Cambridge University University Press, 2008.
- Francesco Cesarini, Steve Vinoski: Designing for Scalability with Erlang/OTP: Implement Robust, Fault-Tolerant Systems. O’Reilly, 2016.
- Betts et al.: Exploring CQRS and Event Sourcing. Microsoft, 2013.

6 Architekturmuster

Dauer: 180 min	Übungszeit: 60 min

In der funktionalen Softwarearchitektur 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.

6.1 Begriffe und Konzepte

Funktionale Datenstruktur, Monoid, Funktor, Monade, Model-View-Update.

6.2 Lernziele

6.2.1 Was sollen die Teilnehmer können?

- die Möglichkeiten funktionaler Datenstrukturen für die Softwarearchitektur ausnutzen
- algebraische Eigenschaften in Domänenobjekten erkennen
- algebraische Eigenschaften im Code reflektieren
- UI-Architektur mit Model-View-Update entwerfen

6.2.2 Was sollen die Teilnehmer verstehen?

- Trade-Offs der Verwendung funktionaler Datenstrukturen
- Eigenschaften algebraischer Abstraktionen
- Unterschied zwischen Model-View-Update und Model-View-Controller

6.2.3 Was sollen die Teilnehmer kennen?

- Beispiele für funktionale Datenstrukturen (hash tries, finger trees, red-black trees)
- Eigenschaften algebraischer Abstraktionen

6.3 Referenzen

- Chris Okasaki: Purely functional data structures. Cambridge University Press, 1999.
- Paul Chiusano, Runar Bjarnason: Functional Programming in Scala. September, 2014.

7 Beispiel für funktionale Softwarearchitektur

Dauer: 60 min	Übungszeit: Keine.

7.1 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.

7.2 Lernziele

Beschreibung dessen, was die Teilnehmer über funktionale Softwarearchitektur lernen sollen, wenn sie das Beispiel sehen.

7.2.1 Was sollen die Teilnehmer können?

- Potential für Parallelisierung und/oder Verteilung erkennen
- mögliche algebraische Abstraktionen identifizieren

7.2.2 Was sollen die Teilnehmer verstehen?

- die Auswirkung der Verwendung unveränderlicher Daten und reiner Funktionen

7.2.3 Was sollen die Teilnehmer kennen?

- die Vorteile gegenüber einer traditionell objektorientierten Architektur

7.3 Referenzen

Keine. Schulungsanbieter sind für die Auswahl und Beschreibung von Beispielen verantwortlich.

8 Quellen und Referenzen zu Funktionale Softwarearchitektur

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

- Betts et al.: Exploring CQRS and Event Sourcing. Microsoft, 2013.
- Simon Brown: Software Architecture for Developers: Volume 1 - Technical leadership and the balance with agility. Leanpub, 2018.
- Francesco Cesarini, Steve Vinoski: Designing for Scalability with Erlang/OTP: Implement Robust, Fault-Tolerant Systems. O'Reilly, 2016.
- Paul Chiusano, Runar Bjarnason: Functional Programming in Scala. September, 2014.
- Field, Harrison: Functional Programming. Addison-Wesley, 1988.
- Jeremy Gibbons, Oege de Moor (Herausgeber): The Fun of Programming. Oxford University Press, 2003.
- Graham Hutton: Programming in Haskell. Cambridge University Press, 2. Auflage, 2016.
- Herbert Klaeren, Michael Sperber: Schreibe Dein Programm! http://www.deinprogramm.de/
- Simon Marlow: Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming. O'Reilly, 2013.
- Chris Okasaki: Purely functional data structures. Cambridge University Press, 1999.
- John H. Reppy: Concurrent Programming in ML. Cambridge University University Press, 2008.
- Comparison of functional programming languages. Wikipedia
- Scott Wlaschin: Domain Modeling Made Functional. The Pragmatic Bookshelf, 2018.