Curriculum für

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Modul
FUNAR

Funktionale Softwarearchitektur

2023.1-RC1-DE-20230327

Inhaltsverzeichnis

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 Expert:innen 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

Inhalt	Empfohlene Mindestdauer (min)
1. Grundlagen funktionaler Programmierung	180
2. Technologien	120
3. Funktionale Modellierung	240
4. Funktionale Makro-Architektur	240
5. Beispiel	120
Summe	1080 (18h)

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. Grundlagen funktionaler Programmierung

Dauer: 180 Min.	Übungszeit: 120 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

Werte, Funktionen, Komposition, Typen, Module

1.2. Lernziele

LZ 1-1: Daten

- die Rolle von systematischer Datenanalyse in der Domänenmodellierung verstehen
- Informationen systematisch als Daten repräsentieren können
- den Unterschied zwischen zusammengesetzten und gemischten Daten verstehen
- die Rolle von Selbstbezügen in der Datenrepräsentation verstehen
- bei der Datenanalyse Selbstbezüge finden und nutzen können
- den Unterschied zwischen Objekten und Werten verstehen
- den Unterschied zwischen funktionalen und objektorientierten Datenstrukturen verstehen

LZ 1-2: Funktionen

- Funktionen systematisch anhand von Anforderungen konstruieren können
- den Unterschied zwischen reinen Funktionen und zustandsverändernden Prozeduren beziehungsweise objektorientierten Methoden verstehen
- systematische Abstraktion anwenden können
- Funktionen durch Abstraktion komponieren können

LZ 1-3: Typen

- die Rolle von Typen bei der Datenanalyse verstehen
- Typen benutzen können, um Funktionen systematisch zu konstruieren
- Datenrepräsentationen in sowohl statisch als auch dynamisch getypten Sprachen aus der Datenanalyse herleiten können

LZ 1-4: Modularisierungsmechanismen

- den Modulmechanismus mindestens einer konkreten funktionalen Sprache kennen

- den Unterschied zwischen den Modularisierungsmechanismen in objektorientierter Programmierung und dem behandelten funktionalen Modulmechanismus verstehen
- Organisationsprinzipien für funktionalen Code kennen

1.3. Referenzen

[Felleisen+ 2014], [Hutton 2016], [Sperber+ 2020], [Okasaki 1999]

2. Technologien

Dauer: 60 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 Architekt:innen die Entscheidung für oder gegen eine Technologie zu ermöglichen.

2.1. Begriffe und Konzepte

statisches Typsystem, dynamisches Typsystem, Endrekursion, nicht-strikte Auswertung

2.2. Lernziele

LZ 2-1: Eigenschaften funktionaler Sprachen

- die Auswirkung der Verwendung statischer beziehungsweise dynamischer Typsysteme verstehen
- den Unterschied zwischen strikter und nicht-strikter Auswertung verstehen
- die Bedeutung von Endrekursion als Unterscheidungsmerkmal verstehen

LZ 2-2: Wahl einer funktionalen Sprache

- wichtige funktionale Sprachen und ihre Haupteigenschaften kennen
- die Zuordnung von Programmiersprachen beziehungsweise deren Eigenschaften zu verschiedenen Laufzeitumgebungen kennen
- eine geeignete Programmiersprache für ein Projekt auswählen können

2.3. Referenzen

[Field+ 1988], [Wikipedia 2020]

3. Funktionale Modellierung

Dauer: 180 Min.	Übungszeit: 120 Min.

Dieser Abschnitt behandelt spezifisch funktionale Ansätze zur Domänenmodellierung. Im Zentrum stehen Kombinatormodelle, die besonders flexibel und langlebig sind. Des weiteren geht es um algebraische Strukturen in der Domänenmodellierung, die Aspekte der Domäne zutage fördern können, die ohne Algebra verborgen bleiben. Schließlich behandelt der Abschnitt noch die Beziehung zwischen taktischem Domain-Driven Design und funktionaler Modellierung.

3.1. Begriffe und Konzepte

Kombinatormodelle, Algebra, Halbgruppe, Monoid, Funktor, applikativer Funktor, DDD, closure of operations

3.2. Lernziele

LZ 3-1: Kombinatormodelle

- die Struktur von Kombinatormodellen mit Selbstbezüge verstehen, also Modelle mit Konstruktoren, die Objekte eines Typs T aus anderen Objekten des Typs T herstellen
- die Vorteile von Kombinatormodellen gegenüber "normalen" Modellen verstehen
- Kombinatoren in Domänenmodellen entdecken können
- Kombinatormodelle konstruieren können

LZ 3-2: Algebraische Strukturen

- die Natur und den Wert algebraischer Struktur in der Domänenmodellierung verstehen
- die Konzepte Halbgruppe, Monoid, Funktor und applikativer Funktor kennen
- Halbgruppen und Monoide in Kombinatormodellen finden können

LZ 3-3: Domain-Driven Design

- die Unterschiede und Gemeinsamkeiten zwischen dem DDD- und dem funktionalen Ansatz zur Domänenmodellierung verstehen
- die Beziehung zwischen Kombinatormodellen und dem Konzept "closure of operations" in DDD verstehen
- Teile einer DDD-Architektur mit funktionaler Programmierung umsetzen können

3.3. Referenzen

[Gibbons+ 2003], [Wlaschin 2018], [Yorgey 2012], [Maguire 2021]

4. Funktionale Makro-Architektur

Dauer: 180 Min.	Übungszeit: 120 Min.

Dieser Abschnitt behandelt spezifisch funktionale Konzepte für Makro-Architektur. Speziell geht es um die Organisation von Systemen in Abläufe sowie funktionale Frontend-Architektur.

4.1. Begriffe und Konzepte

Monaden, Dependency Injection, Model View Update, Events

4.2. Lernziele

LZ 4-1: Monaden

- verstehen, wie Monaden sequenzielle Prozesse mit Datenbabhängigkeiten abbilden + mehrere konkrete Beispiele für Monaden in gängigen Abstraktionen kennen
- verstehen, wie Monaden über den Kontrollfluss abstrahieren
- Monaden einsetzen können, um Domänenprozesse zu modellieren
- Monaden einsetzen können, um Dependency Injection umzusetzen

LZ 4-2: Events

- verstehen, wie Events in den Architektur-Entwurfsprozess passen
- verstehen, wie funktionale Programmierung verwendet werden kann, um event-basierte Architektur umzusetzen

LZ 4-3: Model View Update

- die grundlegende Struktur des Patterns Model View Update verstehen
- den Unterschied zwischen Model View Update und objekt-orientierten Model-View-Patterns verstehen
- eine einfache Beispielapplikation auf Grundlage von Model View Update entwickeln können

4.3. Referenzen

[Hutton 2016], [Chiusano+ 2014], [Betts+ 2013], [Marlow 2013]

5. Beispiele

Dauer: 60 Min.	Übungszeit: 60

5.1. Begriffe und Konzepte

Innerhalb jeder lizenzierten Schulung muss mindestens ein Beispiel für funktionale Architektur vorgestellt werden.

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

5.2. Lernziele

LZ 5-1: Anwendung funktionaler Architektur

- die Verwendung unveränderlicher Daten und reiner Funktionen verstehen
- die Verwendung von funktionaler Modellierung verstehen
- die funktionale Makroarchitektur verstehen
- die Vorteile gegenüber einer traditionell objektorientierten Architektur verstehen
- das Beispiel selbst ändern und weiterentwickeln können

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. Technical leadership and the balance with agility. Leanpub, 2018.

C

- [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

- [Maguire 2021] Sandy Maguire: Algebra-Driven Design. Leanpub, 2021.

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.

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.

Y

- [Yorgey 2012] Brent A. Yorgey, Monoids: Theme and Variations. Proceedings of the 2012 Haskell Symposium, September 2012 https://doi.org/10.1145/2364506.2364520