Curriculum for

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Module
FUNAR

Functional Software Architecture

2023.1-RC1-EN-20230327

Table of Contents

Introduction: General information about the iSAQB Advanced Level

What is taught in an Advanced Level module?

- The iSAQB Advanced Level offers modular training in three areas of competence with flexibly designable training paths. It takes individual inclinations and priorities into account.
- The certification is done as an assignment. The assessment and oral exam is conducted by experts appointed by the iSAQB.

What can Advanced Level (CPSA-A) graduates do?

CPSA-A graduates can:

- Independently and methodically design medium to large IT systems
- In IT systems of medium to high criticality, assume technical and content-related responsibility
- Conceptualize, design, and document actions to achieve quality requirements and support development teams in the implementation of these actions
- Control and execute architecture-relevant communication in medium to large development teams

Requirements for CPSA-A certification

- Successful training and certification as a Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- At least three years of full-time professional experience in the IT sector; collaboration on the design and development of at least two different IT systems
  - Exceptions are allowed on application (e.g., collaboration on open source projects)
- Training and further education within the scope of iSAQB Advanced Level training courses with a minimum of 70 credit points from at least three different areas of competence
  - existing certifications (for example: Sun/Oracle Java architect, Microsoft CSA) can be credited upon application
- Successful completion of the CPSA-A certification exam

Essentials

What does the module “FUNAR” convey?

The module presents functional software architecture as an alternative to object-oriented architecture. Compared to OO architecture, functional software architecture relies on immutable data, algebraic abstractions and embedded domain-specific languages. The result are flexible and robust architectures that are less complex and have fewer hidden dependencies than OO.[1]

Unlike OO architectures, FP architectures are entirely code. This module therefore illustrates all architectural principles with concrete code, making them easier to learn.[2]

After completion of the module, participants will know the essential principles of functional architecture and will be able to apply them when designing software systems. They will know the peculiarities of functional programming languages and can use them effectively when implementing software systems. They can convert domain knowledge directly into executable code and systematically use this to develop algebraic abstractions.

Curriculum Structure and Recommended Durations

Content	Recommended minimum duration (minutes)
1. Basics of Functional Programming	300
2. Technologies	60
3. Functional Modeling	300
4. Functional Macro Architecture	300
5. Example	120
Sum	1,080 (18h)

Duration, Teaching Method and Further Details

The times stated below are recommendations. The duration of a training course on the FUNAR module should be at least 3 days, but may be longer. Providers may differ in terms of duration, teaching method, type and structure of the exercises and the detailed course structure. In particular, the curriculum provides no specifications on the nature of the examples and exercises.

Licensed training courses for the FUNAR module contribute the following credit points towards admission to the final Advanced Level certification exam:

Methodical Competence:	10 Points
Technical Competence:	20 Points
Communicative Competence:	0 Points

Prerequisites

Participants **should** have the following prerequisite knowledge:

- Basic knowledge of functional programming

- Experience in modeling architectures

Knowledge in the following areas may be **helpful** for understanding some concepts:

- Basic knowledge of algebra.

Structure of the Curriculum

The individual sections of the curriculum are described according to the following structure:

- **Terms/principles**: Essential core terms of this topic.
- **Teaching/practice time**: Defines the minimum amount of teaching and practice time that must be spent on this topic or its practice in an accredited training course.
- **Learning goals**: Describes the content to be conveyed including its core terms and principles.

This section therefore also outlines the skills to be acquired in corresponding training courses.

Supplementary Information, Terms, Translations

To the extent necessary for understanding the curriculum, we have added definitions of technical terms to the iSAQB glossary and complemented them by references to (translated) literature.

[1] The comparisons to object-oriented software architecture take into account the fact that the authors assume that most potential participants in an advanced training course are familiar with object-oriented software architecture. However, this comparison is not essential for the training course and can be omitted in favor of a more in-depth study of the (already abundant) contents.
[2] A number of languages can be used to accompany such a training course. Haskell and Racket can be particularly suitable, as they directly support all the architectural principles of this curriculum. Racket may be particularly suitable for training courses that follow an introduction to functional programming with the built-in teaching languages.

1. Basics of Functional Programming

Lesson duration: 180 min	Exercises: 120 min

This section introduces the basic elements of the structure of functional software systems: Pure functions operate on immutable values. This is in contrast to the object-oriented model, where methods operate on an encapsulated state. These functions are combined or *composed* into domain-relevant functionality. The design of functions and their composition is driven by their type signatures. In functional architecture, functions are bundled in modules, as opposed to object-oriented programming where they are bundled with classes.

1.1. Terms and concepts

Values, Functions, Composition, Types, Modules

1.2. Learning Goals

LG 1-1: Data

- understand the role of systematic data analysis in domain modeling
- be able to represent information as data systematically
- understand the difference between compound and mixed data
- understand the role of self-references in data representation
- be able to find and use self-refencences during data analysis
- understand the difference between objects and values
- understand the difference between functional and object-oriented data structures

LG 1-2: Functions

- be able to design functions systematically to requirements
- understand the difference between pure functions and state-mutating procedures viz. object-oriented methods
- be able to apply abstraction systematically
- be able to compose functions via abstractions

LG 1-3: Types

- understand the role of types in data analysis
- be able to use types to design functions systematically
- be able to derive data representations from data analysis in both statically and dynamically typed languages

LG 1-4: Modularisation mechanisms

- know the module mechanism in at least one concrete functional language
- understand the difference between modularisation mechanisms in object-oriented programming and

the covered module mechanism

- know organisation principles for functional code

1.3. References

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

2. Technologies

Lesson duration: 60 min	Exercises: None

Functional software architecture is usually associated with the use of functional programming languages and frameworks. This section provides an overview of the key features and limitations of available technologies to enable architects to decide for or against a technology.

2.1. Terms and concepts

static type system, dynamic type system, tail recursion, lazy evaluation

2.2. Learning Goals

LG 2-1: Properties of functional languages

- understand the consequences of using static and dynamic type systems, respectively
- understand the difference between strict and lazy evaluation
- understand the relevance of proper tail recursion

LG 2-2: Choice of functional language

- know important functional languages and their main characteristics
- know the relationship between languages, their characteristics and the underlying run-time environment
- be able to choose an appropriate language for a project

2.3. References

[Field+ 1988], [Wikipedia 2020]

3. Functional Modeling

Lesson duration: 180 min	Exercises: 120 min

This section discusses specifically functional approaches to domain modeling. Combinator models are the most important aspect, which are particularly flexible and long-lived. Another theme is algebraic structure in domain modeling, which can reveal aspects of the domain that would remain hidden without algebra. Finally, the section discusses the relationship between tactical Domain-Driven Design and functional modeling.

3.1. Terms and concepts

combinator models, algebra, semigroup, monoid, functor, applicative functor, DDD, closure of operations

3.2. Learning Goals

LG 3-1: Combinator Models

- understand the structure of combinator models with self-references, i.e. models with constructors that build objects of a type T from other objects of type T
- understand the advantages of combinator models over "plain" models
- be able to discover combinators in domain models
- be able to construct combinator models

LG 3-2: Algebraic Structures

- understand the nature and value of algebraic structures in domain modeling
- know the concepts semigroup, monoid, functor, and applicative functor
- be able to identify semigroups and monoids in combinator models

LG 3-3: Domain-Driven Design

- understand the differences and similarities between the DDD and the functional approaches to domain modeling
- understand the relationship between combinator models and the concept of "closure of operations" in DDD
- be able to construct parts of DDD architecture using functional programming

3.3. References

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

4. Functional Macro Architecture

Lesson duration: 180 min	Exercises: 120 min

This section is about specifically functional concepts for macro architecture. Specifically, it convers the organisation of a system into workflows and functional frontend architecture.

4.1. Terms and concepts

monads, dependency injection, Model View Update, events

4.2. Learning Goals

LG 4-1: Monads

- understand how monads express sequential processes with data dependencies
- know several concrete examples of monads in common abstractions
- understand how monads abstract over control flow
- be able to use monads to model domain processes
- be able to use monads for dependency injection

LG 4-2: Events

- understand how events fit into the architecture design process
- understand how functional programming can express event-based architecture

LG 4-3: Model View Update

- understand the basic structure of the Model View Update pattern
- understand the difference between Model View Update and object-oriented Model View patterns
- be able to develop a simple application frontend based on Model View Update

4.3. References

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

5. Examples

Lesson duration: 60 min	Exercises: 60

5.1. Terms and concepts

In every licensed training session, at least one example for functional architecture must be presented.

Type and structure of the examples presented may depend on the training and participants' interests. The are not prescribed by iSAQB.

5.2. Learning Goals

LG 5-1: Application of functional architecture

- understand the usage of immutable data and pure functions
- understand the use of functional modeling
- understand the macro architecture
- understand the advantages over a traditional object-oriented architecture
- be able to change and extend the application

References

This section contains references that are cited in the curriculum.

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