Curriculum for

Certified Professional for Software Architecture (CPSA)® *Advanced Level*

Module FUNAR

Functional Software Architecture

2018.1-EN-20220530

iSAQB®
International Software Architecture Qualification Board

Table of Contents

© (Copyright), International Software Architecture Qualification Board e. V. (iSAQB® e. V.) 2021

The curriculum may only be used subject to the following conditions:

1. You wish to obtain the CPSA Certified Professional for Software Architecture Advanced Level® certificate. For the purpose of obtaining the certificate, it shall be permitted to use these text documents and/or curricula by creating working copies for your own computer. If any other use of documents and/or curricula is intended, for instance for their dissemination to third parties, for advertising etc., please write to info@isaqb.org to enquire whether this is permitted. A separate license agreement would then have to be entered into.
2. If you are a trainer or training provider, it shall be possible for you to use the documents and/or curricula once you have obtained a usage license. Please address any enquiries to info@isaqb.org. License agreements with comprehensive provisions for all aspects exist.
3. If you fall neither into category 1 nor category 2, but would like to use these documents and/or curricula nonetheless, please also contact the iSAQB e. V. by writing to info@isaqb.org. You will then be informed about the possibility of acquiring relevant licenses through existing license agreements, allowing you to obtain your desired usage authorizations.

Important Notice

We stress that, as a matter of principle, this curriculum is protected by copyright. The International Software Architecture Qualification Board e. V. (iSAQB® e. V.) has exclusive entitlement to these copyrights.

The abbreviation "e. V." is part of the iSAQB’s official name and stands for "eingetragener Verein" (registered association), which describes its status as a legal entity according to German law. For the purpose of simplicity, iSAQB e. V. shall hereafter be referred to as iSAQB without the use of said abbreviation.

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

CPSA-A®
Certified Professional for Software Architecture
Advanced Level

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. System structure	180
2. Technologies	120
3. Implementation of functional requirements	240
4. Implementation of non-functional requirements	240
5. Architectural patterns	240
6. Example	60
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. System structure

Lesson duration: 180 min	Exercises: 60 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

Functions and values, Composition, Types, Modules

1.2. Learning Goals

- What should the participants be able to do?
  - Represent information as immutable values
  - Formulate central functionality as functions
  - Construct types for these functions
- What should the participants understand?
  - The difference between the object-oriented and the functional software model
  - The role of types in functional software design
  - Modules as an alternative to classes
- What should the participants know?
  - Examples of functional system architectures

1.3. References

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

2. Technologies

Lesson duration: 120 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 types, dynamic types, tail recursion, strict or non-strict evaluation, runtime environment.

2.2. Learning Goals

- What should the participants be able to do?
  - Select a suitable programming language for a project
  - Select suitable frameworks and libraries
- What should the participants understand?
  - The effect of the use of strict or non-strict evaluation on software architecture
  - The effect of using static or dynamic types
  - The importance of tail recursion as a distinguishing feature
- What should the participants know?
  - Important functional languages and their main characteristics
  - Exemplary frameworks and libraries
  - The assignment of programming languages or their properties to different runtime environments

2.3. References

[Field+ 1988], [Wikipedia 2020]

3. Implementation of functional requirements

Lesson duration: 180 min	Exercises: 60 min

This section discusses inherently functional approaches to translating functional requirements into software architecture. Domain-driven design can be implemented with functional programming instead of object-oriented programming. Combinator models reveal hidden domain knowledge. Embedded domain-specific languages allow fast alignment of requirements and architecture.

3.1. Terms and concepts

functional requirements, DDD, combinator models, embedded domain-specific languages.

3.2. Learning Goals

- What should the participants be able to do?
  - Implement DDD with the aid of functional building blocks
  - Formulate rudimentary combinator models
  - Design small domain-specific languages
- What should the participants understand?
  - The implementation of DDD entities with functional programming
  - The specific properties of combinator models
  - The role of domain-specific languages in development
- What should the participants know?
  - Examples of combinator models
  - The specific advantages of combinator models
  - Examples of embedded domain-specific languages

3.3. References

[Gibbons+ 2003], [Wlaschin 2018]

4. Implementation of non-functional requirements

Lesson duration: 180 min	Exercises: 60 min

This section covers specific functional techniques for implementing non-functional requirements such as data management, distribution and performance.

4.1. Terms and concepts

CQRS, event sourcing, parallelization, distribution

4.2. Learning Goals

- What should the participants be able to do?
  - Design the data management for a specific project with event sourcing
  - Design a query model to match an event sourcing model and user interface requirement
  - Select techniques for effective parallelization
  - Select techniques for effective distribution
- What should the participants understand?
  - The difference between "data warehousing" and event sourcing approaches
  - The role of query models
  - The role of functional data structures in parallelization
  - The role of functional data management in distribution
- What should the participants know?
  - Data parallelism
  - Futures
  - Message passing

4.3. References

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

5. Architectural Patterns

Lesson duration: 180 min	Exercises: 60 min

Functional software architecture has specific architectural patterns. These include the use of high-performance functional data structures, the use of algebraic abstractions and UI development with the Model-View-Update Pattern as an alternative to Model-View-Controller.

5.1. Terms and concepts

functional data structure, monoid, functor, monad, Model-View-Update.

5.2. Learning Goals

- What should the participants be able to do?
  - Utilize the possibilities of functional data structures for the software architecture
  - Recognize algebraic properties in domain objects
  - Reflect algebraic properties in the code
  - Design UI architecture with Model-View-Update
- What should the participants understand?
  - Trade-offs with the use of functional data structures
  - Properties of algebraic abstractions
  - Difference between Model-View-Update and Model-View-Controller
- What should the participants know?
  - Examples of functional data structures (hash tries, finger trees, red-black trees)
  - Properties of algebraic abstractions

5.3. References

[Okasaki 1999], [Chiusano+ 2014]

6. Examples

Lesson duration: XXX min	Exercises: XXX min

6.1. Terms and concepts

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

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

6.2. Learning Goals

- What should the participants be able to do?
  - Identify potential for parallelization and/or distribution
  - Identify possible algebraic abstractions
- What should the participants understand?
  - The effect of using immutable data and pure functions
- What should the participants know?
  - The advantages over traditional object-oriented architecture

References

This section contains references that are cited in the curriculum.

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.