Curriculum for

CPSA Certified Professional for Software Architecture®

– Advanced Level –

Module: FUNAR

Functional Software Architecture

Version 1.0 (September 2018)

Table of contents

0 Introduction: General information about the iSAQB Advanced Level

0.1 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.

0.2 What can Advanced Level graduates (CPSA-A) 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 non-functional requirements. Support development teams in the implementation of these actions.
- Control and execute architecture-relevant communication in medium to large development teams.

0.3 Requirements for CPSA-A certification

- Successful training and certification as CPSA-F (Certified Professional for Software Architecture, Foundation Level).
- 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 allowed on application (e.g.: collaboration in OpenSource projects)
- Training and further education within the scope of iSAQB Advanced Level training courses with a minimum of 70 credit points from all three areas of competence (for details see iSAQB website).
    - Existing certifications can be credited to these credit points if necessary. The list of current certificates for which credit points are credited is available on the iSAQB homepage.
- Successful completion of the CPSA-A certification exam.

1 Basics of the Functional Software Architecture (FUNAR) module

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.

1.1 Structure of the Functional Software Architecture curriculum and recommended schedule

The structure of the curriculum is based on the structure of the software architecture tasks in Simon Brown: Software Architecture for Developers: Volume 1 - Technical Leadership and the Balance with Agility. Leanpub, 2018.

Content	Recommended minimum duration
System structure	180 min
Technologies	120 min
Implementation of functional requirements	240 min
Implementation of non-functional requirements	240 min
Architectural patterns	240 min
Example	60 min
Total (3 days of 360 min)	1,080 min = 18 h

1.2 Duration, teaching method and further details

The above times are recommendations. The duration of a training course on the Functional Software Architecture module should be at least three 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.

[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.

Licensed training courses for the Functional Software Architecture 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:	00 points

1.3 Prerequisites for the Functional Software Architecture module

Participants **should** have the following knowledge and/or experience:

- Basic knowledge of functional programming
- Experience in modeling architectures

Helpful for the understanding of some principles are further:

- Basic knowledge of algebra.

1.4 Structure of the curriculum for the Functional Software Architecture module

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. The learning goals are differentiated into the following categories or sub-chapters:

- What should the participants be **able to do**? The participants should be able to apply these contents independently after the training course. These contents are covered with exercises during the training course and are part of the Functional Software Architecture module exam and/or the final exam of the iSAQB Advanced Level.
- What should the participants **understand**? These contents can be assessed in the Functional Software Architecture module exam.
- What should the participants **know**? These contents (terms, principles, methods, practices or similar) can support the understanding or motivate the topic. These contents are not part of the exams, are discussed in training courses, but are not necessarily taught in detail.

1.5 Supplementary information, terms, translations

Where necessary for the understanding of the curriculum, we have included and defined technical terms in the iSAQB glossary and supplemented them with translations of the original literature as required.

2 System structure

Duration: 180 min	Practice time: 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.

2.1 Terms and principles

- Functions and values
- Composition
- Types
- Modules

2.2 Learning goals

2.2.1 What should the participants be able to do?

- Represent information as immutable values
- Formulate central functionality as functions
- Construct types for these functions

2.2.2 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

2.2.3 What should the participants know?

- Examples of functional system architectures

2.3 References

- Graham Hutton: Programming in Haskell. Cambridge University Press, second edition, 2016.
- Matthias Felleisen et al.: How to Design Programs. MIT Press, 2014. https://htdp.org/
- 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 Technologies

Duration: 120 min	Practice time: 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.

3.1 Terms and principles

Static types, dynamic types, tail recursion, strict or non-strict evaluation, runtime environment.

3.2 Learning goals

3.2.1 What should the participants be able to do?

- Select a suitable programming language for a project
- Select suitable frameworks and libraries

3.2.2 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

3.2.3 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

3.3 References

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

4 Implementation of functional requirements

Duration: 180 min	Practice time: 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.

4.1 Terms and principles

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

4.2 Learning goals

4.2.1 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

4.2.2 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

4.2.3 What should the participants know?

- Examples of combinator models
- The specific advantages of combinator models
- Examples of embedded domain-specific languages

4.3 References

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

5 Implementation of non-functional requirements

Duration: 180 min	Practice time: 60 min

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

5.1 Terms and principles

CQRS, event sourcing, parallelization, distribution.

5.2 Learning goals

5.2.1 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

5.2.2 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

5.2.3 What should the participants know?

- Data parallelism
- Futures
- Message passing

5.3 References

- 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 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 Architectural patterns

Duration: 180 min	Practice time: 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.

6.1 Terms and principles

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

6.2 Learning goals

6.2.1 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

6.2.2 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

6.2.3 What should the participants know?

- Examples of functional data structures (hash tries, finger trees, red-black trees)
- Properties of algebraic abstractions

6.3 References

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

7 Example of functional software architecture

Duration: 60 min	Practice time: None.

7.1 Terms and principles

At least one example of functional software architecture must be presented within each licensed training course.

Type and characteristics of the presented examples can depend on the training course or the interests of the participants and are not specified by the iSAQB.

7.2 Learning targets

Description of what the participants should learn about functional software architecture when they see the example.

7.2.1 What should the participants be able to do?

- Identify potential for parallelization and/or distribution
- Identify possible algebraic abstractions

7.2.2 What should the participants understand?

- The effect of using immutable data and pure functions

7.2.3 What should the participants know?

- The advantages over traditional object-oriented architecture

7.3 References

None. Training providers are responsible for the selection and description of examples.

8 Sources and references on functional software architecture

This section contains sources that are referenced in the curriculum in whole or in part.

- 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.
- Matthias Felleisen et al.: How to Design Programs. MIT Press, 2014. https://htdp.org/
- Field, Harrison: Functional Programming. Addison-Wesley, 1988.
- Jeremy Gibbons, Oege de Moor (editor): The Fun of Programming. Oxford University Press, 2003.
- Graham Hutton: Programming in Haskell. Cambridge University Press, second edition, 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 Press, 2008.
- Comparison of functional programming languages. Wikipedia
- Scott Wlaschin: Domain Modeling Made Functional. The Pragmatic Bookshelf, 2018.