Curriculum for

CPSA
Certified Professional for Software Architecture®

– Foundation Level –

Version 4.1.1 (January 2017)

Table of contents

List of learning goals

0 Introduction

0.1 What does a Foundation Level training convey?

Licensed Certified Professional for Software Architecture – Foundation Level (CPSA-F) trainings will provide participants with the knowledge and skills required to design, specify and document a software architecture adequate to fulfil the respective requirements for small and medium-sized systems.

Based upon their individual practical experience and existing skills participants will learn to derive architectural decisions from an existing system vision and adequately detailed requirements. CPSA-F trainings teach methods and principles for design, documentation and evaluation of software architectures, independent of specific development processes.

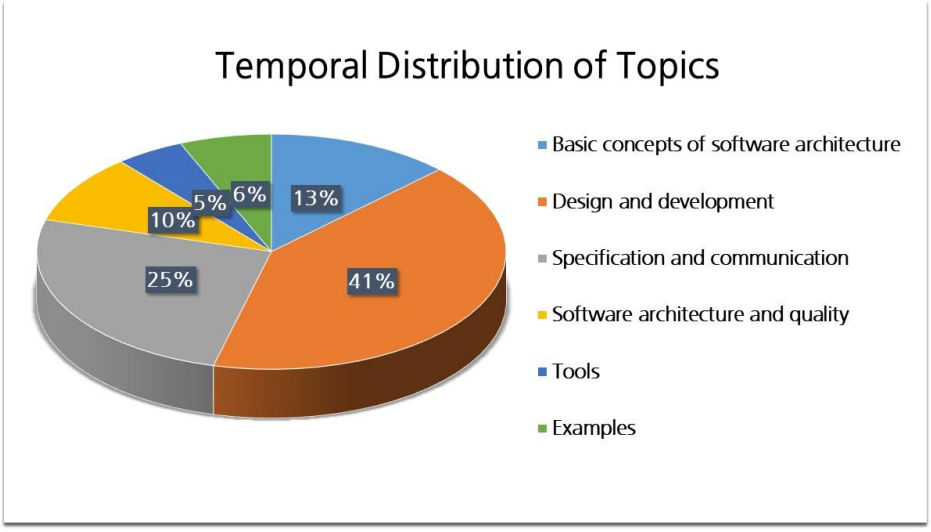
Focus is education and training of the following skills:

- Discuss and reconcile fundamental architectural decisions with stakeholders from requirements, management, development, operations and test,
- understand the essential activities of software architecture, and carry out those for small- to medium sized systems,
- document and communicate software architectures based upon architectural views, architecture patterns and technical concepts

In addition, such trainings cover:

- the term software architecture and its meaning,
- the tasks and responsibilities of software architects,
- the roles of software architects within development projects,
- state-of-the-art methods and techniques for developing software architectures

0.2 Outline of the curriculum and recommended allocation of study times



0.3 Duration, teaching methods and further details on licensed training

Study times given are recommendations. The duration of a training course should be at least three days, but may as well be longer. Providers may vary in their approach to duration, teaching methods, the type and structure of exercises as well as the detailed course outline. The types (domains and technologies) of examples and exercises can be determined individually by the training provider.

0.4 Prerequisites

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

- More than 18 months of practical experience with software development, gained through programming a variety of projects or systems outside of formal education
- Knowledge of and practical experience with at least one higher programming language, especially:
  - Concepts of passing parameters (call-by-value, call-by-reference)
  - Basics of type systems (static vs. dynamic typing, parameterized and generic data types)
  - Concepts of modularization (packages, namespaces, etc.)
- Basics of modelling and abstraction
- Basics of algorithms and data structures
- Basics of UML (class, package, component and sequence diagrams) and their relation to source code
- Practical experience with technical documentation, especially documenting source code, system design or technical concepts

Furthermore, the following will be **useful** for understanding several concepts:

- Knowledge of object-orientation (OO)
- Practical experience in at least one object-oriented programming language
- Practical experience with designing and implementing distributed applications, such as client-server systems or web applications

0.5 CPSA-F curriculum chapters, learning goals and relevance for the examination

The structure of the curriculum's chapters follows a set of prioritized learning goals. For each learning goal, relevance for the examination of this learning goal or its sub-elements is clearly stated (by the R-1/R-2/R-3 classification, see table).

Every learning goal describes the contents to be taught including their key terms and concepts.

Regarding relevance for the examination, the following categories are used in this curriculum:

Learning-goal category	Identifier	Meaning	Relevance for examination
Being able to ...	R-1	These are the contents participants will be expected to be able to put into practice independently upon completion of the course. Within the course, these contents will be covered through exercises and discussions.	Contents will be part of the examination.
Understanding ...	R-2	These are the contents participants are expected to understand in principle. They will normally not be the primary focus of exercises in training.	Contents may be part of the examination.
Knowing ...	R-3	These contents (terms, concepts, methods, practices or similar) can enhance understanding and motivate the topic. They may be covered in training if required.	Contents will not be part of examination.

If required, the learning goals include references to further reading, standards or other sources.

The sections "Terms and Concepts" of each chapter list word that are associated with the contents of the chapter. Some of them are used in the descriptions of learning goals. In certification examinations, the iSAQB may check these prerequisites by appropriate questions.

0.6 What is and what is not covered by the curriculum

This curriculum reflects the contents currently considered by the iSAQB members to be necessary and useful for achieving the learning goals of CPSA-F. It is not a comprehensive description of the entire domain of 'software architecture'.

The following topics or concepts are **not** part of CPSA-F:

- Concrete implementation technologies, frameworks or libraries
- Programming or programming languages
- Modelling notations (such as UML) or fundamentals of modelling itself
- System analysis and requirements engineering (please refer to the education and certification program by IREB e. V., http://ireb.org, International Requirements Engineering Board)
- Software testing (please refer to the education and certification program by ISTQB e. V., http://istqb.org, International Software Testing Board)
- Project or product management
- Introduction to specific software tools

0.7 Introduction to the iSAQB certification program

Duration: 15 min	Exercises: n/a

This section is not relevant for the examination.

Participants will become familiar with the context of the iSAQB certification program as well as its examinations or examination modalities such as:

- iSAQB as an association
- iSAQB's responsibility for the curriculum as well as corresponding examination questions
  - Organizational separation between training and examination
  - Procedures and formal constraints of the CPSA-F examination
- Foundation Level versus Advanced Level
- Optional: other certification schemes

1 Basic concepts of software architectures

Duration: 120 min	Exercises: none

1.1 Terms and concepts

Software architecture; structure; building blocks/components; interfaces; relationships; cross-cutting concerns/aspects; architecture objectives; software architects and their responsibilities; tasks and required skills; non-functional and functional requirements of systems; constraints; influencing factors; types of IT systems (embedded systems, real-time systems, information systems etc.)

1.2 Learning goals

LG 1-1: Discuss definitions of software architecture (R1)

Comparison of several definitions of software architecture (incl. ISO 42010/IEEE 1471, SEI, Booch etc.) and identification of their similarities:

- Components/building blocks with interfaces and relationships
- Building blocks as a general term, components as specific types of building blocks
- Structures and cross-cutting concerns, principles
- Crosscutting design decisions and their consequences both across the system and concerning the entire life cycle

LG 1-2: Understand and identify the benefits and objectives of software architecture (R1)

Objectives of software architects and software architectures:

- Software architecture focuses on quality attributes such as durability, maintainability, changeability, robustness more than on pure functionality.
- Software architecture supports the creation and maintenance of software, particularly its implementation.
- Software architecture supports the achievement of quality requirements.
- Software architecture supports the understanding of the system, related to all relevant stakeholders.

LG 1-3: Understand software architecture as part of the software life cycle (R2)

Participants shall understand the role that software architecture plays within the overall development of IT systems:

- They understand the correlation with business and operational processes for information systems
- They understand the correlation with business and operational processes for decision support systems (data warehouse, management information systems)
- They understand that changes of requirements, quality goals, technologies, or in the system's environment might require changes to the software architecture.

LG 1-4: Understand software architects' tasks and responsibilities (R1)

Software architects are responsible for achieving the required or necessary quality and functionality of a solution. Depending on the actual approach or process model used, they must align this responsibility with the overall responsibilities for project management and/or other roles.

Tasks and responsibilities of software architects:

- Clarify and question requirements as well as constraints and refine them if required. Together with functional requirements (required features), this includes the required quality attributes (required constraints).
- Decide how to decompose the system into building blocks, while determining dependencies and interfaces between the building blocks.
- Determine and decide cross-cutting technical concerns (for instance persistence, communication, GUI etc.)
- Communicate and document software architecture based on views, architectural patterns and technical concepts
- Accompany the realization and implementation of the architecture; integrate feedback from relevant stakeholders into the architecture if necessary; review and ensure the consistency of source code and software architecture
- Evaluate software architecture, especially with respect to risks involving the required quality characteristics

It is the responsibility of software architects to identify and highlight the consequences of architectural decisions and discuss these with other stakeholders.

Their role involves recognizing the necessity of iterations in all tasks and pointing out possibilities for relevant feedback.

LG 1-5: Relate the role of software architects to other stakeholders (R1)

Software architects are able to explain their role. They should adapt their contribution to a software development in a specific context and in relation to other stakeholders, in particular to:

- Requirements analysis (system analysis, requirements management, specialist field)
- Implementation
- Project lead and management
- Product management/product owner
- Quality assurance
- IT operations (production, data centers), applies primarily to information systems
- Hardware development

LG 1-6: Being able to explain the correlation between development approaches and software architecture (R1)

- Software architects are able to explain the influence of iterative approaches on architectural decisions (with regard to risks and predictability)
- Due to inherent uncertainty, software architects often have to work and decide iteratively. To do so, they must systematically seek feedback from other stakeholders.

LG 1-7: Differentiate between architecture and project objectives (R1)

- Systems can be developed within projects, Scrum-sprints, iterations, releases or other approaches. The term "project" above refers to any of these approaches to work organization.
- Software architects should be able to demonstrate the significance of architectural objectives, constraints and influencing factors for the design of software architectures.
- They can explain the connection between requirements and solutions
- They can identify and specify architectural objectives based upon existing requirements
- They can explain (long-term) architectural objectives and the distinction between those and (short-term) project objectives

LG 1-8: Distinguish explicit statements and implicit assumptions (R1)

- Software architects should explicitly present assumptions or prerequisites and avoid implicit assumptions.
- They know that implicit assumptions can lead to misunderstandings between stakeholders.

LG 1-9: Know roles and responsibilities of software architects in organizational context (R3)

Software architects know about additional architecture levels, for example:

- Infrastructure architecture: the structure of technical infrastructure: hardware, networks etc.
- Hardware architecture (for hardware-related systems)
- Software architecture: the structure of individual software systems. This is what the iSAQB and CPSA-F identify as the focus for software architects.
- Corporate IT architecture, Enterprise IT architecture: the structure of application landscapes. CPSA-F does not focus on this topic.
- Business process architecture, business architecture: the structure of business processes. CPSA-F does not focus on this topic.

LG 1-10: Understanding the differences between types of IT-systems (R3)

Software architects can differentiate software architecture requirements and approaches for different types of IT systems.

- They know about the relation of software architecture to system architecture or overall architecture for embedded or hardware-related systems
- They understand the characteristics of hardware/software design (and code) (dependencies between hardware and software design in relation to time and content)

2 Design and development of software architectures

Duration: 300 min	Exercises: 90 min

2.1 Terms and concepts

Design; design approach; design decision; views, technical and crosscutting concepts; architectural patterns; design principles; domain-related and technical architectures; model-based design; iterative/incremental design; domain-driven design; top-down and bottom-up approaches, pattern-languages, stereotypes, tools-and-material-approach

2.2 Learning goals

LG 2-1: Select and adhere to approaches and heuristics for architecture development (R1)

Software architects are able to name, explain and apply basic methods of architecture development, for example:

- Model- and view-based architecture development
- Domain-driven design
- Iterative and incremental design
  - Necessity of iterations, especially when decision-making is affected by uncertainties
  - Feedback on design decisions based on source code as well as qualitative considerations
- Top-down and bottom-up approaches to design

Software architects are able to consider influencing factors and constraints for architecture design

LG 2-2: Design architectures (R1)

Software architects are able to:

- design and appropriately document a software architecture based upon known functional requirements and quality goals (non-functional requirements) for software systems that are neither safety-critical nor business-critical
- identify and give reasons for interdependencies between design decisions
- make structure-relevant decisions regarding system decomposition and building-block structure and determine dependencies and interfaces between the building blocks
- explain the terms 'black box' and 'white box' and use them purposefully
- apply stepwise refinement and specify building blocks
- design individual architecture views, especially deployment view, building-block view and runtime view, and describe the consequences of these decisions on the corresponding source code
- define how the architecture is mapped to source code and assess or evaluate the consequences
- separate technical and domain-related elements of architectures and justify these decisions

Software architects are able to design and justify domain-related structures in the small, especially they are able to:

- identify, justify and document domain-related building blocks (aggregates, entities, services, value objects)
- develop, discuss, refine and adapt a "ubiquitous language" with stakeholders
- design and explain the interaction between domain-related and technical components of systems

When taking architecture and design decisions software architects know and consider:

- the considerable impact of quality goals (non-functional requirements) – e. g., changeability, robustness, efficiency
- solution options and design tactics for achieving specific quality goals, like changeability, robustness, efficiency
- the trade-offs between such options

Software architects are able to design and justify domain-related structures in-the-large (bounded contexts) (R3):

- Subdomains, especially Core Domain, Supporting Domain, Generic Domain
- Integration of bounded contexts (in the large) with context mapping, e. g., Open-host-services, Published Language, Anticorruption Layer, Separate Ways

LG 2-3: Identify and consider factors influencing software architecture (R1)

Software architects are able to gather and consider constraints and influencing factors that restrict the freedom in design decisions

They should recognize and consider the:

- impact of quality requirements on architectures
- impact of technical decisions and concepts on architectures
- (potential) impact of organizational and legal factors on architectural decisions (R2)

LG 2-4: Decide and design cross-cutting concepts (R1)

- Software architects are able to decide and design cross-cutting/technical concepts, for example persistence, communication, GUI, error handling, concurrency
- Software architects are able to recognize and assess potential interdependencies between these decisions

LG 2-5: Describe, explain and appropriately use important architectural patterns and architectural styles (R1-R3)

Software architects know various architectural patterns and –styles. They are able to apply those appropriately:

- Patterns for dataflow or data-centered architectural styles (R1)
  - Pipes and filters (R1)
  - Blackboard (R2)
- Patterns for hierarchical architectural styles (R1)
  - Layers
  - Master-Slave
  - Virtual Machine
- Hybrid (heterogeneous) architectural styles (R1)
- Language-independent design patterns, such as: adapter, wrapper, gateway, facade, registry, broker, factory (R1)

- Patterns for interactive systems (R2) for example: model view controller (MVC), model view view model (MVVM), model view presenters, presentation-abstraction-control
- Patterns for distributed systems and their integration (R3)
  - Messaging (Publish-Subscribe, Hub-and-Spoke)
  - Event-based Systems (and Event Sourcing)
  - Client/Server (R1)
  - Remote Procedure Call (R2)
  - File or database integration
  - Microservices/Self-Contained Systems
- They know essential sources for architectural patterns, for instance POSA literature and PoEAA (for information systems) (R3)

LG 2-6: Explain and use design principles (R1)

Software architects are able to explain and apply the following design principles:

- Abstraction
- Modularization (cf. LG 2-8) with regard to:
  - information hiding and encapsulation
  - separation of concerns
  - High cohesion
  - Single-Responsibility
  - Open/closed principle
- Loose, but functionally sufficient coupling of building blocks (cf. LG 2-7)
- Dependency inversion by means of interfaces
- Dependency injection in order to externalize dependencies
- Conceptual integrity to achieve uniformity (homogeneity, consistency) of solutions for similar problems (R2)

Software architects understand the impact of design principles onto source code and are able to apply these appropriately.

LG 2-7: Plan dependencies between building blocks (R1-R3)

Software architects understand and can purposefully design dependencies between building blocks (R1).

- They are able to apply different types of coupling (structural, temporal, via data types, via hardware etc.) (R1)
- They are able to assess consequences of such dependencies
- They know different types of dependencies of building blocks (nesting, usage/delegation, inheritance)

Software architects understand possibilities of removing or reducing coupling e. g.,

- language-independent patterns like Broker, Plugin, Adapter, Façade (R1)
- creational patterns/Dependency Injection (R1)
- messaging, events, Commands (R2)
- Stability Patterns like Bulkhead, Timeout, and Circuit Breaker (R3)

LG 2-8: Design building blocks/structural elements of software architectures (R1)

Software architects know desirable characteristics (encapsulation, information hiding) of building blocks and structural elements.

They are able to purposefully apply black-box and white-box descriptions of building blocks.

They know UML notation for various building blocks and their compositions (R2)

- Packages as a semantically weak type of building-block aggregation,
- Classes (in OO languages) and components with clearly defined interfaces as a semantically more precise variant.

LG 2-9: Design and define interfaces (R1)

Software architects know about the importance of interfaces. They are able to design or determine interfaces between architectural building blocks as well as external interfaces between the system and elements outside of the system.

- Software architects know desired characteristics of interfaces:
  - Easy to learn, easy to use, easy to expand
  - Hard to misuse
  - Functionally complete from the perspective of users or building blocks using them
- Software architects can describe and document interfaces
- Software architects know different ways of looking at interfaces: (R3)
  - Resource-oriented approach (see Representational State Transfer)
  - Service-oriented approach (see WS-*/SOAP-based web services.

References

[Bass+2012], [Fowler2003], [Gharbi+2014], [Martin2003], POSA series, particularly [Buschmann+1996] and [Buschmann+2007], [Starke2015]

3 Specification and communication of software architectures

Duration: 150 min	Exercises: 90 min

3.1 Terms and concepts

Views; structures; (technical) concepts; documentation; communication; description; meta structures for description and communication; building blocks; building-block view; run-time view; deployment view; node; channel; deployment units; mapping building blocks onto deployment units; description of interfaces

3.2 Learning goals

LG 3-1: Explain and consider quality attributes of technical documentation (R1)

Software architects know the essential quality requirements of technical documentation and can fulfil those when documenting systems:

- Understandability, accuracy, efficiency, adequacy, maintainability
- Form, content and level of detail of documentation chosen in accordance with targeted stake-holders

Readers alone can assess the understandability of technical documentation.

LG 3-2: Describe and communicate software architectures (R1)

Software architects are able to document and communicate software architectures adequately for different stakeholders.

- They are able to address the needs of different readers, e. g., management, developers, quality assurance as well as other software architects.
- They are able to consolidate and harmonize contributions of different author groups stylistically and content-wise.
- They know about the benefits of template-based documentation

LG 3-3: Understand how to explain and apply notations/models to describe software architecture (R2)

Software architects know the following UML diagrams to describe architectural views:

- class, package, component and composition-structure diagrams
- deployment diagrams
- sequence and activity diagrams
- state diagrams

Software architects know alternative notations to UML diagrams, especially for the runtime view, e. g.,

- Flow charts, numbered lists, BPMN

LG 3-4: Explain and use architectural views (R1)

Software architects are able to use the following architectural views:

- Building-block or component view (how the system is composed of software building blocks)
- Run-time view (dynamic view, interaction between software building blocks at run time, state machines)
- Deployment view (hardware and technical infrastructure as well as the mapping of software building blocks onto these)

LG 3-5: Explain and use the system context (R1)

Software architects are able to differentiate between business and technical context

LG 3-6: Document and communicate cross-cutting architectural concepts (R1)

- Software architects are able to adequately document cross-cutting concepts (principles, aspects), e. g., persistence, workflow management, GUI, deployment/integration)
- They are able to explain the significance of such cross-cutting concepts.
- They know that such cross-cutting concepts may be re-used across systems.

LG 3-7: Describe interfaces (R1)

- Software architects are able to describe and specify interfaces
- They are able to differentiate between internal and external interfaces

LG 3-8: Explain and document architectural decisions (R2)

- Software architects are able to systematically derive, reason about and document architectural decisions

LG 3-9: Understand the use of documentation as written communication (R2)

- Software architects use documentation to support design and implementation of systems.
- Software architects are able to choose and adapt wording and notation used in technical documentation to readers' skills and objectives.

LG 3-10: Know additional resources and tools for documentation (R3)

Software architects know about the fundamentals of several of the published frameworks for describing software architectures, such as:

  - ISO/IEEE-42010 (previously 1471)
  - TOGAF, RM/ODP,
  - arc42, FMC
- They know ideas and examples of checklists for the creation, documentation and evaluation of software architectures
- They know potential tools for creating and maintaining architectural documentation

References

[Clements+2002], [Gharbi+2014], [Starke2015], [Zörner2012]

4 Software architectures and quality

Duration: 60 min	Exercises: 60 min

4.1 Terms and concepts

Quality; quality attributes; DIN/ISO 9126 resp. 25010; ATAM; scenarios; quality tree; compromises (when implementing quality attributes); evaluation of software architectures (qualitative and quantitative)

4.2 Learning goals

LG 4-1: Discuss quality models and quality characteristics (R1)

- Software architects are able to explain the concepts of quality (following ISO/IEC 25010, previously 9126) and quality attributes
- They are able to explain generic quality models (such as ISO/IEC 25010)
- They are able to explain relationships and interactions between quality attributes, for instance:
  - Configurability versus robustness
  - Memory usage versus run time consumption
  - Security versus usability
  - Efficiency versus changeability

LG 4-2: Define quality requirements for software architectures (R1)

Software architects are able to specific and document quality requirements for the software under construction and its software architectures in concrete terms, for example using scenarios and quality trees.

They are able to explain and create scenarios and quality trees.

They are able to express exemplary quality requirements for software.

LG 4-3: Perform qualitative evaluation of software architectures (R2)

Software architects know about methodical approaches to analyzing and evaluating software architectures and be able to apply it to smaller examples.

They know the ATAM approach to qualitative evaluation of software architectures.

They know that the following sources of information may be helpful for the qualitative evaluation of architectures:

- Requirements, in particular in the form of quality trees and scenarios
- Architecture documentation
- Architecture and design models
- Source code
- Metrics

LG 4-4: Understanding the quantitative evaluation of software architectures (R2)

- The quantitative evaluation (metrics) of software, in particular of source code, can help to identify critical parts within systems.
- Further information can be consulted for assessment and evaluation of architectures, such as:
  - Source code (e. g., metrics like lines of code, cyclomatic complexity, incoming and outgoing dependencies, instability, abstractness, distance)
  - Known errors in source code, in particular error clusters
  - Test cases and test results
  - Requirement and solution models

LG 4-5: Understanding how quality objectives can be achieved (R2)

- Software architects are able to explain and apply tactics, appropriate practices and technical possibilities for achieving important quality objectives of software systems (different for embedded systems and information systems), for instance for:
  - Efficiency/performance
  - Maintainability, changeability, expandability, flexibility
- They are able to identify corresponding risks

References

[Bass+2003]

[Clements+2002]

[Gharbi+2014]

[Martin2003]

[Starke2015]

5 Tools for software architects

Duration: 45 min	Exercises: n/a

5.1 Terms and concepts

Modelling tools, static analysis tools, dynamic analysis tools, generation tools, requirements-management tools, build systems/tools, configuration management tools

5.2 Learning goals

LG 5-1: Name and explain important tool categories (R1)

Software architects are able to name and explain the most important categories of tools in relation to their work – for instance tools for:

- requirements specification, documentation, and management
- software development, debugging, versioning and configuration
- static and dynamic analysis of software and systems in operation
- modelling and documentation of software architectures
- build and deployment systems
- team collaboration
- planning and documentation of architecture and development

LG 5-2: Select tools according to requirements (R2)

The working environment and tools of software architects depend on the relevant constraints and influencing factors.

6 Examples of software architectures

Duration: 60 min	Exercises: n/a

This section is not examination relevant.

LG 6-1: Know how the relation between requirements and solutions is established (R3)

Software architects are expected to recognize and comprehend the correlation between requirements/architectural objectives and chosen solutions using at least one example.

LG 6-2: Know the rationale of a solution's technical implementation (R3)

Software architects understand the technical realization (implementation, technical concepts, products used, architectural decisions, solution strategies) of at least one software system.

7 Sources and References related to Software Architecture

Several learning goals or sections contain references to books, papers or other resources. These references can be found free of charge in the documents section of the iSAQB website (http://www.isaqb.org/documents).