CPSA Certified Professional for Software Architecture®

- Foundation Level Curriculum -

Version V5.0-RC-4-EN, 10. Jul 2019

Table of Contents

Legal Notice

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

1. You wish to obtain the CPSA Certified Professional for Software Architecture Foundation 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 contact@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, training provider or training organizer, 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 contact@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 contact@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.

Introduction

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.

Out of scope

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:

- Specific implementation technologies, frameworks or libraries
- Programming or programming languages
- Specific process models
- Fundamentals of 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 Qualification Board)
- Project or product management
- Introduction to specific software tools.

Prerequisites

The iSAQB e. V. may check the following prerequisites in certification examinations via corresponding questions.

Participants should have the following knowledge and/or experience. In particular, substantial practical experience from software development in a team is an important prerequisite for understanding the learning material and successful certification.

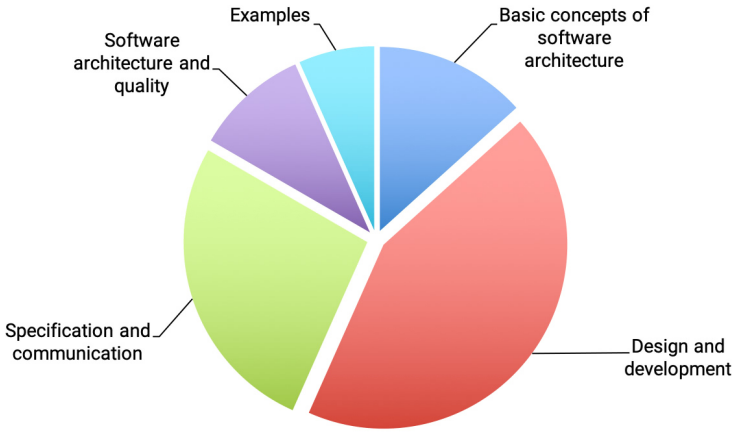
- More than 18 months of practical experience with software development, gained through team-based development of several systems outside of formal education
- Knowledge of and practical experience with at least one higher programming language, especially:
  - Concepts of
    - modularization (packages, namespaces, etc.)
    - parameter-passing (*call-by-value, call-by-reference*)
    - *scope*, i.e. of type- and variable declaration and definition
  - Basics of type systems (static vs. dynamic typing, generic data types)
  - Error- and exception handling in software
  - Potential problems of global state and global variables
- Basic knowledge of:
  - modelling and abstraction
  - algorithms and data structures (i.e. Lists, Trees, HashTable, Dictionary/Map)
  - UML (class, package, component and sequence diagrams) and their relation to source code

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

- Basics and differences of imperative, declarative, object-oriented and functional programming
- Practical experience in
  - an object-oriented programming language (i.e. Java or C#)
  - designing and implementing distributed applications, such as client-server systems or web applications
  - technical documentation, especially documenting source code, system design or technical concepts

Structure, duration and teaching methods

Study times given in the following sections of the curriculum are just 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 training providers.



Recommended time distribution

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 R1, R2, R3 classification, see the table below). 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:

ID	Learning-goal category	Meaning	Relevance for examination
R1	Being able to	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.
R2	Understanding	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.
R3	Knowing	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 words that are associated with the contents of the chapter. Some of them are used in the descriptions of learning goals.

1. Basic concepts of software architecture

Duration: 120 min.	Exercises: none

Relevant terms

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

Learning goals

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

Software architects know several definitions of software architecture (incl. ISO 42010/IEEE 1471, SEI, Booch etc.) and can name their similarities:

- Components/building blocks with interfaces and relationships
- building blocks as a general term, components as a special form thereof
- Structures, crosscutting concepts, principles
- Architecture decisions and their consequences on the entire systems and its lifecycle

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

Essential goals of software architecture are:

- the achievement of quality requirements such as reliability, maintainability, changeability, security etc. in contrast to the implementation of pure functionality
- supporting the creation and maintenance of software, particularly its implementation
- convey the understanding of structures and concepts of the system, related to all relevant stakeholders.

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

Software architects understand their tasks and can integrate their results into the overall lifeycle of IT systems. They can:

- identify the consequences of changes in functional requirements, quality requirements, technologies, or the system environment in relation to software architecture
- elaborate on relationships between IT-systems and the supported business and operational processes.

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

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

Tasks and responsibilities of software architects:

- Clarify and scrutinize requirements and constraints, and refine them if necessary. Together with functional requirements (required features), this includes the required quality characteristics (*required constraints*).
- Decide how to decompose the system into building blocks, while determining dependencies and interfaces between the building blocks.
- Determine and decide on cross-cutting concerns (for instance persistence, communication, GUI etc.)
- Communicate and document software architecture based on views, architectural patterns, cross-cutting 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
- Analyze and evaluate software architecture, especially with respect to risks involving the quality requirements
- Identify, highlight, and justify the consequences of architectural decisions to other stakeholders

They should independently recognize the necessity of iterations in all tasks and point out possibilities for appropriate and 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
- Enterprise architecture, architecture board.

LG 1-6: Can 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 make decisions iteratively. To do so, they have to systematically obtain feedback from other stakeholders.

LG 1-7: Differentiate between short- and long-term goals (R1)

Software architects can:

- explain long-term quality requirements and their differentiation from (short-term) project goals
- explain potential conflicts between short-term and long-term goals, in order to find a suitable solution for all stakeholders
- identify and specify quality requirements.

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

Software architects:

- should explicitly present assumptions or prerequisites and avoid implicit assumptions
- Tknow that implicit assumptions can lead to potential misunderstandings between stakeholders.

LG 1-9: Responsibilities of software architects within the greater architectural context (R3)

The focus of the iSAQB CPSA Foundation Level is on structures and concepts of individual software systems.

In addition, software architects are familiar with other architectural domains, for example:

- Enterprise IT architecture: Structure of application landscapes
- Business and Process Architecture: Structure of, among other things, business processes
- Information architecture: cross-system structure and use of information and data
- Infrastructure or technology architecture: Structure of the technical infrastructure, hardware, networks, etc.
- Hardware or processor architecture (for hardware-related systems).

These architectural domains are not the content focus of CPSA-F.

LG 1-10: Differentiate types of IT systems (R3)

Software architects know different types of IT systems, for example:

- Information systems
- Decision support, data warehouse or business intelligence systems
- Mobile systems
- Batch processes or systems
- hardware-related systems; here they understand the necessity of hardware/software code design (temporal and content-related dependencies of hardware and software design).

2. Design and development of software architectures

Duration: 330 min.	Excercises: 120 min.

Relevant terms

Design; design approach; design decision; views; interfaces; technical and cross-cutting concerns; stereotypes; architectural patterns; pattern languages; design principles; dependencies; coupling; cohesion; functional and technical architectures; top-down and bottom-up approaches; model-based design; iterative/incremental design; domain-driven design

Learning goals

LG 2-1: Select and use approaches and heuristics for architecture development (R1-R3)

Software architects are able to name, explain, and use the basic approaches of architecture development, for example:

- Top-down and bottom-up approaches to design
- View-based architecture development
- Iterative and incremental design
  - Necessity of iterations, especially when decision-making is affected by uncertainties
  - Necessity of feedback on design decisions
- Domain-driven design (R3)
- Model-driven architecture (R3)

LG 2-2: Design software architectures (R1)

Software architects are able to:

- design and appropriately communicate and document software architectures based upon known functional and quality requirements for software systems that are neither safety- nor business-critical
- make structure-relevant decisions regarding system decomposition and building-block structure and deliberately design dependencies between building blocks
- recognize and justify interdependencies and trade-offs of design decisions
- explain the terms *black box* and *white box* and apply them purposefully
- apply stepwise refinement and specify building blocks
- design architecture views, especially building-block view, runtime view and deployment view
- explain the consequences of these decisions on the corresponding source code
- separate technical and domain-related elements of architectures and justify these decisions
- identify risks related to architecture decisions.

LG 2-3: Identify and consider factors influencing software architecture (R1-R2)

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
- impact of technical decisions and concepts
- (potential) impact of organizational, legal, and other constraints (R2)
- impact of stakeholder concerns (R2)

LG 2-4: Design and implement cross-cutting concerns (R1)

Software architects are able to:

- explain the significance of such cross-cutting concerns
- decide on and design cross-cutting concerns, for example persistence, communication, GUI, error handling, concurrency
- identify and assess potential interdependencies between these decisions.

Software architects know that such cross-cutting concepts may be re-used throughout the system.

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

Software architects know various architectural patterns and can use them as required:

- Layers (R1)
- Pipes and filters (R1)
- Client/Server (R1)
- Adapter and Facade (R1)
- Proxy (R1)
- Plugin (R1)
- Blackboard (R2)
- Model View Controller and Variants (R2)
- Broker (R2)
- Remote Procedure Call (R2)
- Messaging (R2), e.g. with events and commands

Software architects know essential sources for architectural patterns, such as 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, 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 or similar abstractions
- 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: Planning dependencies between building blocks (R1)

Software architects understand dependencies and coupling between building blocks and can use them in a targeted manner. They:

- know different types of dependencies of building blocks (e.g. structural coupling via use/delegation, nesting, ownership, generation, inheritance, temporal coupling, coupling via data types or via hardware)
- can use such types of coupling in a targeted manner and can assess the consequences of such dependencies
- know and can apply possibilities to reduce or eliminate coupling, for example:
  - Patterns (see learning goal 2-5)
  - Basic design principles (see learning goal 2-6)
  - Externalization of dependencies, i.e. defining concrete dependencies at installation- or runtime, for example by using *Dependency Injection*.

LG 2-8: Achieve quality requirements with appropriate approaches and techniques (R1)

Software architects understand and consider the considerable influence of quality requirements in architecture and design decisions, e.g. for:

- Efficiency / Performance
- Availability
- Maintainability, modifiability, extensibility, adaptability

They can:

- explain and apply solution options, *Design Tactics*, suitable practices as well as technical possibilities to achieve important quality requirements of software systems (different for embedded systems or information systems)

- identify and communicate possible trade-offs between such solutions and their associated risks.

LG 2-9: Design and define interfaces (R1-R3)

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

They know:

- desired characteristics of interfaces and can use them in the design:
  - easy to learn, easy to use, easy to extend
  - hard to misuse
  - functionally complete from the perspective of users or building blocks using them.
- the necessity to treat internal and external interfaces differently
- different approaches for implementing interfaces (R3):
  - Resource-oriented approach (REST, Representational State Transfer)
  - Service-oriented approach (see WS-*/SOAP-based web services.

References

[Bass+2012], [Fowler 2003], [Gharbi+2017], [Gamma+94], [Martin 2003], [POSA-Series], esp. [Buschmann+1996] and [Buschmann+2007], [Starke 2017], [Lilienthal 2017]

3. Specification and communication of software architectures

Duration: 180 min.	Exercises: 120 min.

Relevant terms

(Architectural) Views; structures; (technical) concepts; documentation; communication; description; stakeholder-oriented, meta structures and templates for description and communication; system context; building blocks; building-block view; runtime view; deployment view; node; channel; deployment artifacts; mapping building blocks onto deployment artifacts; description of interfaces and design decisions; UML, tools for documentation

Learning goals

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

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

- Understandability, correctness, efficiency, appropriateness, maintainability
- Form, content, and level of detail tailored to the stakeholders

They know that only the target audiences can assess the understandability of technical documentation.

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

Software architects are able to:

- document and communicate architectures for corresponding stakeholders, thereby addressing different target groups, e.g. management, development teams, QA, other software architects, and possibly additional stakeholders
- consolidate and harmonise the style and content of contributions from different groups of authors
- know the benefits of template-based documentation.

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

Software architects know at least 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:

- Context view
- Building-block or component view (composition 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 the infrastructure)

LG 3-5: Explain and apply context view of systems (R1)

Software architects can:

- Depict the context of systems, e.g. in the form of context diagrams with explanations
- represent external interfaces of systems in the context view
- differentiate functional and technical context.

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

Software architects are able to adequately document and communicate typical cross-cutting concerns, e. g., persistence, workflow management, UI, deployment/integration, logging.

LG 3 7: Describe interfaces (R1)

Software architects are able to describe and specify both internal and external interfaces.

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

Software architects are able to:

- systematically take, justify, communicate, and document architectural decisions
- Identify, communicate, and document interdependencies between design decisions

LG 3-9: Use documentation as written communication (R2)

Software architects use documentation to support the design, implementation and further development (also called *maintenance* or *evolution*) of systems.

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

Software architects know:

- basics of several published frameworks for the description of software architectures, for example:
  - ISO/IEEE-42010 (formerly 1471)
  - arc42
  - C4

    - RM/ODP
    - FMC
- ideas and examples of checklists for the creation, documentation, and testing of software architectures
- possible tools for creating and maintaining architectural documentation

References

[Bass+2012], [Clements+2010], [Gharbi+2017], [Starke 2017], [Zörner 2015]

4. Software architecture and quality

Duration: 60 min.	Exercises: 60 min.

Relevant terms

Quality; quality characteristics (also called quality attributes); DIN/ISO 25010; quality scenarios; quality tree; trade-offs between quality characteristics; qualitative architecture assessment; metrics and quantitative assessment

Learning goals

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

Software architects can:

- explain the concept of quality (based on DIN/ISO 25010, formerly 9126) and quality characteristics
- Explain generic quality models (such as DIN/ISO 25010)
- Explain correlations and trade-offs of quality characteristics, for example:
  - Configurability versus reliability
  - Memory requirements versus performance efficiency
  - Security versus usability
  - Runtime flexibility versus maintainability.

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

Software architects can:

- clarify and formulate specific quality requirements for the software to be developed and its architectures, for example in the form of scenarios and quality trees
- explain and apply scenarios and quality trees.

LG 4-3: Qualitative analysis and asessment of software architectures (R2-R3)

Software Architects:

- know methodical approaches for the qualitative analysis and assessment of software architectures (R2), for example, as specified by ATAM (R3);
- can qualitatively analyze and assess smaller systems (R2)
- know that the following sources of information can help in the qualitative analysis and assessment of architectures (R2):
  - quality requirements, e.g. in the form of quality trees and scenarios
  - architecture documentation
  - architecture and design models
  - source code

- metrics
- other documentation of the system, such as requirements, operational or test documentation.

LG 4-4: Quantitative evaluation of software architectures (R2)

Software architects know approaches for the quantitative analysis and evaluation (measurement) of software.

They know that:

- quantitative evaluation can help to identify critical parts within systems
- further information can be helpful for the evaluation of architectures, for example:
  - Requirements and architecture documentation
  - source code and related metrics such as lines of code, (cyclomatic) complexity, inbound and outbound dependencies
  - known errors in source code, especially error clusters
  - Test cases and test results.

References

[Bass2003], [Clements+2002], [Gharbi+2017], [Martin 2003], [Starke 2017]

5. Examples of Software architectures

Duration: 90 min.	Exercises: none

This section is not relevant for for the exam.

Learning goals

LG 5-1: Know the relation between requirements, constraints, and solutions (R3)

Software architects are expected to recognize and comprehend the correlation between requirements and constraints, and the chosen solutions using at least one example. Software architects have used at least one example to identify and understand the relationship between requirements and constraints, and the solution decisions.

LG 5-2: 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 solution.

References

- [iSAQB Working Groups] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Freely available https://leanpub.com/isaqbreferences.
- [Bass+2012] Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. 3rd Edition, Addison Wesley 2012.
- [Clements+2002] Paul Clements, Rick Kazman, Mark Klein: Evaluating Software Architectures. Methods and Case Studies. Addison Wesley, 2002.
- [Clements+2010] Paul Clements, Felix Bachmann, Len Bass, David Garlan, David, James Ivers, Reed Little, Paulo Merson and Robert Nord. *Documenting Software Architectures: Views and Beyond*, 2nd edition, Addison Wesley, 2010
- [Gamma+94] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
- [Fowler 2003] Martin Fowler: Patterns of Enterprise Application Architecture. Addison-Wesley, 2003.
- [Gharbi+2017] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 3. Auflage, dpunkt Verlag, Heidelberg 2017.
- [Martin 2003] Robert Martin: Agile Software Development. Principles, Patterns, and Practices. Prentice Hall, 2003.
- [POSA-Series] Pattern-Oriented Software Architecture (POSA), Prentice Hall International.
- [Buschmann+1996] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal: Pattern-Oriented Software Architecture: A System of Patterns. Wiley, 1996.
- [Buschmann+2007] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt: Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, Wiley, 2007.
- [Starke 2017] Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden (in German). 8. Auflage, Carl Hanser Verlag 2017. Website: https://esabuch.de
- [Lilienthal 2017] Carola Lilienthal: Langlebige Softwarearchitekuren. 2. Auflage, dpunkt Verlag 2018.
- [Zörner 2015] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. 2. Auflage, Carl Hanser Verlag 2015.