Curriculum for

Certified Professional for Software Architecture (CPSA)®

Foundation Level

2025.1-RC5-EN-20250106

Table of Contents

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.

List of Learning Goals

- LG 01-01: Understand Definitions of Software Architecture (R1)
- LG 01-02: Understand and Explain the Goals and Benefits of Software Architecture (R1)
- LG 01-03: Understand Long-term Impact of Software Architecture (R3)
- LG 01-04: Understand the Tasks and Responsibilities of Software Architects (R1)
- LG 01-05 [previously LG 1-09]: Distinction between Software Architecture and other Architectural Domains (R3)
- LG 01-06: Relate the Role of Software Architects to Other Stakeholders (R1)
- LG 01-07: Importance of Data and Data Models (R2)
- LG 02-01: Understand Stakeholder Concerns (R1-R3)
- LG 02-02 [previously LG 2-3]: Clarify and Consider Requirements and Constraints (R1-R3)
- LG 02-03 [previously LG 4-1]: Understand and Explain Qualities of a Software System (R1)
- LG 02-04 [previously LG 04-03]: Formulate Requirements on Qualities (R1-R3)
- LG 02-05 [previously LG 1-08]: Prefer Explicit Statements over Implicit Assumptions (R1)
- LG 03-01 [previously LG 2-8, new content]: Fulfilling Requirements through Architecture (R1)
- LG 03-02 [previously LG 2-02]: Design Software Architectures (R1)
- LG 03-03 [previously LG 2-01]: Select and Use Approaches and Heuristics for Architecture Development (R1,R3)
- LG 03-04 [previously LG 2-06]: Explain and Use Design Principles (R2)
- LG 03-05 [previously LG 1-06]: Correlation between Feedback Loops and Risks (R1, R2)
- LG 03-06 [previously LG 2-07]: Manage Dependencies between Building Blocks (R1)
- LG 03-07 [previously LG 2-09]: Design and Define Interfaces (R1-R3)
- LG 03-08 [previously LG 2-05]: Describe, Explain and Apply Important Architectural Patterns (R1, R3)
- LG 03-09 [previously LG 2-05]: Describe, Explain, and Appropriately Apply Important Design Patterns (R3)
- LG 03-10 [previously LG 2-04]: Identify, Design and Implement Cross-Cutting Concerns (R1)
- LG 03-11 [previously LG 2-10]: Know Fundamental Principles of Software Deployment (R3)
- LG 03-12 [previously LG 1-11]: Know the Challenges of Distributed Systems (R3)
- LG 04-01 [previously LG 3-01]: Explain and Consider the Requirements of Technical Documentation (R1)
- LG 04-02 [previously LG 3-02]: Describe and Communicate Software Architectures (R1-R3)
- LG 04-03 [previously LG 3-03]: Explain and Apply Notations/Models to Describe Software Architecture (R2-R3)
- LG 04-04 [new]: Learning Goal not Found (R3)
- LG 04-05 [previously LG 3-04]: Explain and Use Architectural Views (R1)
- LG 04-06 [previously LG 3-07]: Document Interfaces (R1)

- LG 04-07 [previously LG 3-06]: Document and Communicate Cross-Cutting Concerns (R2)
- LG 04-08 [previously LG 3-08]: Explain and Document Architectural Decisions (R1-R2)
- LG 04-09 [previously LG 3-09]: Know Additional Resources and Tools for Documentation (R3)
- LG 05-01: Know Reasons for Architecture Analysis (R1)
- LG 05-02 [previously LG 4-3 and 4-4]: Analyze the Qualities of a Software System (R1, R3)
- LG 05-03: Evaluate Conformance to Architectural Decisions (R2)
- LG 06-01: Know the Relation between Requirements, Constraints, and Solutions (R3)
- LG 06-02: Understand the technical implementation of a solution (R3)

Introduction

What Does this Curriculum Contain

This curriculum for the Certified Professional for Software Architecture – Foundation Level (CPSA-F) outlines the essential learning goals that should be mastered to take up the role of software architect.

It is structured along the fundamental activities and responsibilities of software architecture as a role:

- Clarifying stakeholder requirements and constraints
- Designing and developing software architectures, thereby taking structural and conceptual decisions
- Communicating and documenting the architecture for various stakeholders
- Analyzing and assessing software architectures

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., https://ireb.org, International Requirements Engineering Board)
- software testing (please refer to the education and certification program by ISTQB e.V., https://istqb.org, International Software Testing Qualification Board)
- project or product management
- introduction to specific software tools.

The aim of the training is to provide the basics for acquiring the advanced knowledge and skills required for the respective application.

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
  - approaches to testing of software (e.g. unit- and acceptance testing)

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

- basics and differences of imperative, declarative, object-oriented and functional programming
- practical experience in
  - a higher level programming language
  - 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.

Content	Recommended Duration (min)
1. Basic Concepts of Software Architecture	120
2. Requirements and Constraints	180
3. Design and Development	360
4. Specification and Communication	240
5. Analysis and Assessment	90
6. Examples	90
Total	1080

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.

Current Version and Public Repository

You find the most current version of this document on the official download page on https://isaqb-org.github.io/.

The document is maintained in a public repository at https://github.com/isaqb-org/curriculum-foundation, all changes and modifications are public.

Please report any issues in our public issue tracker on https://github.com/isaqb-org/curriculum-foundation/issues.

1. Basic Concepts of Software Architecture

Duration: 120 min.	Exercises: none

Purpose

The purpose of this section is to equip training participants with a foundational understanding of key terms and concepts in software architecture. They become familiar with various definitions and their commonalities, understand the essential goals and benefits of software architecture and can communicate these to other stakeholders. Furthermore, they will be able to name and explain the most important tasks and responsibilities of software architects. Additionally, the section explores the role of software architects in the broader architectural context and their interactions with other stakeholders, preparing participants to effectively contribute to diverse software development projects.

Relevant Terms

Software architecture; architecture domains; structure; building blocks; components; interfaces; relationships; cross-cutting concerns; benefits of software architecture; software architects and their responsibilities; tasks and required skills; stakeholders and their concerns; requirements; constraints; influencing factors

Learning Goals

LG 01-01: Understand Definitions of Software Architecture (R1)

Software architects know and understand the commonalities of many definitions of software architecture:

- components/building blocks with interfaces and relationships
- building blocks as a general term, components as a special form thereof
- structures, cross-cutting concerns, principles
- architecture decisions and their consequences on the entire systems and its lifecycle

References

[ISO 42010], [Bass+2021], [Kruchten 2004], [Starke+2023a]

LG 01-02: Understand and Explain the Goals and Benefits of Software Architecture (R1)

Software architects can justify the following essential goals and benefits of software architecture:

- support the design, implementation, maintenance, and operation of systems
- achieve functional requirements or ensure that they can be met
- achieve requirements such as reliability, maintainability, changeability, security, energy efficiency etc.
- ensure that the system's structures and concepts are understood by all relevant stakeholders
- systematically reduce complexity
- specify architecturally relevant guidelines for implementation and operation

LG 01-03: Understand Long-term Impact of Software Architecture (R3)

Software architects can:

- analyze the impact of architectural decisions on the long-term evolution of a system
- explain the relationship between architectural decisions and the future adaptability and maintainability of the system
- assess how changes in requirements, technologies, or system environment affect existing architectural decisions
- recognize the long-term consequences of architectural decisions on various quality characteristics of the system
- explain the interdependencies between IT systems and the supported business and operational processes

References

[Bass+2021], [Lehman 1980], [Wiki-LehmansLaws], [Lilienthal 2024], [Ford 2017], [Rajlich+2000], [Richards+2020]

LG 01-04: Understand the Tasks and Responsibilities of Software Architects (R1)

Software architects are responsible for meeting requirements 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 project responsibility of project management or other roles.

Tasks and responsibilities of software architects:

- clarify and scrutinize the requirements and constraints. Coordinate and agree on any necessary refinements with the corresponding stakeholders.
- decide how to decompose the system into building blocks, while determining dependencies and interfaces between the building blocks
- decide on cross-cutting concerns (for instance persistence, communication, GUI)
- communicate and document software architecture based on views, architectural patterns, cross-cutting concerns, and technical concerns
- 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 that involve meeting the 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 01-05 [previously LG 1-09]: Distinction between Software Architecture and other Architectural Domains (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
- data architecture: semantics and organization of data
- infrastructure or technology architecture: Structure of the technical infrastructure, hardware, networks, etc.
- hardware or processor architecture (for hardware-related systems)
- system architecture (can have various semantics, depending on the definition of "system")

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

LG 01-06: Relate the Role of Software Architects to Other Stakeholders (R1)

Software architects are able to explain their role. They should adapt their contribution to system development depending on the specific context and in relation to other stakeholders and organizational units, in particular to:

- product management and product owners
- project managers
- requirement engineers (requirements- or business analysts, requirements managers, system analysts, business owners, subject-matter experts, etc.)
- developers
- quality assurance and testers
- IT operators and administrators (applies primarily to production environment or data centers for information systems),
- hardware developers and system architects (applies primarily to embedded and hardware-related systems)
- enterprise architects and architecture board members

LG 01-07: Importance of Data and Data Models (R2)

Software architects understand the importance of data and data models for the architecture. They

- can identify data models that have significant impact on the architecture.
- can design such data models systematically.
- understand the difference between products and sums in data modelling.
- understand the importance of decoupling data models from their representation in databases, files, and transmission protocols.
- can explain the impact of data on architecture decisions regarding e.g. storage, security, scalability, reliability, performance etc.

References

[Felleisen+2014], [Sperber+2023], [Sperber+2024], [Kleppmann 2017], [Ford+ 2021]

2. Requirements and Constraints

Duration: 90 min.	Exercises: 90 min.

Purpose

This section deepens participants' understanding of stakeholder concerns, requirements, and qualities of software architecture. Participants learn to identify the influence of stakeholders on architectural decisions and to assess conflicts and synergies in the context of development projects. By exploring diverse requirements and constraints, they gain insight into effectively addressing stakeholders' needs and project constraints. Additionally, they recognize the significance of software system qualities as drivers for architectural design. They can formulate such requirements using scenarios.

Relevant Terms

Quality; quality characteristics (also called quality attributes); DIN/ISO 25010; Q42; quality scenarios; tradeoffs and interactions between quality characteristics; requirements; constraints; stakeholder concerns

Learning Goals

LG 02-01: Understand Stakeholder Concerns (R1-R3)

Architects can identify stakeholders and their concerns, as well as their impact on the software architecture or the design and development process. (R2)

Examples of stakeholders and concerns (R3):

Stakeholder	Stakeholder Concern
product management and product owners	e.g., required time for the implementation of the requirements
developers	e.g., components and interfaces to be implemented, protocols, technical requirements and constraints
requirement engineers, product owners, business analysts	e.g., fulfillment of the requirements
project managers	e.g., required time and budget for the implementation, associated risks of the chosen architectural approach
quality assurance and testers	e.g., isolated testing of components
operations	e.g., infrastructure requirements related to operating the system

Software architects can identify potential conflicts between short-term and long-term goals (e.g., business and project goals vs. architecture and maintainability goals). They understand that they need to involve the relevant stakeholders in order to resolve these conflicts. (R2)

Architects understand that not all stakeholder concerns can or will be translated into requirements, but still need to be considered. (R3)

Architects can use stakeholder concerns to discover missing or conflicting requirements and/or validate requirements and constraints on the architecture, e.g., in stakeholder interviews. (R3)

LG 02-02 [previously LG 2-3]: Clarify and Consider Requirements and Constraints (R1-R3)

Software architects understand that both requirements and constraints can have an impact on the architecture and the architecture work (R2). They are able to clarify requirements and constraints and take them into account in the architectural design and development process. They understand that their decisions may introduce new requirements or necessitate changes to existing requirements.

They should recognize and account for the impact of:

- product-related requirements such as (R1)
  - functional requirements
  - quality requirements
- technological constraints such as
  - existing or planned hardware and software infrastructure (R1)
  - technological constraints on data structures and interfaces (R2)
  - reference architectures, libraries, components, and frameworks (R1)
  - programming languages (R2)
- organizational constraints such as
  - organizational structure of the development team and of the customer (R1), in particular Conway's law (R2).
  - company and team cultures (R3)
  - partnerships and cooperation agreements (R2)
  - standards, guidelines, and process models (e.g. approval and release processes) (R2)
  - available resources like budget, time, and staff (R1)
  - availability, skill set, and commitment of staff (R1)
- regulatory constraints such as (R2)
  - local and international legal constraints
  - contract and liability issues
  - data protection and privacy laws
  - compliance issues or obligations to provide burden of proof
- trends such as (R3)
  - market trends
  - technology trends (e.g. cloud, microservices, container, generative AI or LLMs))
  - methodology trends (e.g. Agile)

Software architects are able to describe how those factors can influence architecture decisions and can elaborate on the consequences of changing influencing factors by providing examples for some of them (R2).

References

[IREB Foundation], [Bass+2021], [Ghandi+2024], [Starke 2024], [Richards+2020], [Pohl 2025]

LG 02-03 [previously LG 4-1]: Understand and Explain Qualities of a Software System (R1)

Software architects know that the term "quality" is used differently in different contexts:

- referring to "excellence" in the context of quality management, and
- referring to a "specific property (of a software system)" in others.

This learning goal refers to the latter.

Software architects can explain that:

- several taxonomies categorizing qualities of software systems exist
- some categorizations distinguish between functionality and quality, e.g. IREB [IREB Foundation]
- software architecture can impact a software system’s qualities,
- impacting one quality can impact others, necessitating trade-offs, such as:
  - configurability versus reliability
  - memory requirements versus performance efficiency
  - security versus usability
  - runtime flexibility versus maintainability.

They understand that

- some categorizations distinguish between quality requirements and functional requirements, e.g. IREB
- a single requirement might pertain to several qualities

References

[IREB Foundation], [ISO 25010], [Bass+2021], [Q42]

LG 02-04 [previously LG 04-03]: Formulate Requirements on Qualities (R1-R3)

Software architects:

- can formulate scenarios for given qualities with context, stimulus, response, and measurement for a variety of purposes, e.g., to clarify requirements, provide input for architecture assessments, etc. (R1)
- understand that a requirement for a given quality should specify a method of analysis (see LG 05-02 [previously LG 4-3 and 4-4]: Analyze the Qualities of a Software System (R1, R3)) (R1)
- know that the use of a metric as a target can lead to its invalidation (R2), as described, e.g., by Goodhart’s law (R3)

References

[Bass+2021], [Q42], [ISO 25010]

LG 02-05 [previously LG 1-08]: Prefer Explicit Statements over Implicit Assumptions (R1)

Software architects:

- can make assumptions explicit and thus avoid implicit assumptions
- know that implicit assumptions can lead to misunderstandings between stakeholders

3. Design and Development of Software Architectures

Duration: 270 min.	Excercises: 90 min.

Purpose

This section aims to enable participants to take architectural decisions in a way that fulfills stakeholder requirements while respecting the given constraints of the system context. They will learn to develop architectural designs, make informed decisions on system decomposition and shape dependencies between building blocks. To this end, they learn to apply basic approaches and heuristics in architecture development. They recognize the importance of design principles and solution patterns and are able to apply them. In addition, this section addresses the management of cross-cutting concerns, the principles of software deployment and the challenges of distributed systems.

Relevant Terms

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

Learning Goals

LG 03-01 [previously LG 2-8, new content]: Fulfilling Requirements through Architecture (R1)

Software architects:

- understand that architectural activities should be driven by the need to achieve or improve specific qualities
- can propose an architecture design suitable for fulfilling requirements
- can assess which qualities they improve through specific architectural activities or decisions
- can identify and communicate possible trade-offs between designs and their associated risks

LG 03-02 [previously LG 2-02]: 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 structural decisions regarding system decomposition and building-block structure, thereby defining dependencies between building blocks (see LG 03-06 [previously LG 2-07]: Manage Dependencies between Building Blocks (R1))
- recognize and justify interdependencies and trade-offs between https://public.isaqb.org/glossary/glossary-en.html#term-architecture decisions[architecture decisions]
- explain the terms black box and white box and apply them purposefully
- apply stepwise refinement and specification of building blocks
- design architecture views, especially building-block view, runtime view and deployment view (see LG 04-05 [previously LG 3-04]: Explain and Use Architectural Views (R1))

- explain the consequences of decisions on the corresponding source code
- separate technical and domain-related elements of architectures and justify these decisions
- identify risks related to architecture decisions.

References

[Kruchten 1995], [Rozanski+2011], [Starke+2023a], [arc42], [Brown], [Starke 2024]

LG 03-03 [previously LG 2-01]: Select and Use Approaches and Heuristics for Architecture Development (R1,R3)

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

- top-down and bottom-up approaches to design, see [Gharbi+2024], [Starke 2024] (R1)
- view-based architecture development, see [Rozanski+2011], [Kruchten 1995] (R1)
- domain-driven design, see [Evans 2004] (R3)
- evolutionary architecture, see [Ford 2017] (R3)
- global analysis, see [Hofmeister+1999] (R3)
- model-based design (R3)

LG 03-04 [previously LG 2-06]: Explain and Use Design Principles (R2)

Software architects are able to explain what design principles are. They can outline their general objectives and their application with regard to software architecture. (R2)

Software architects are able to:

- explain the design principles listed below and can illustrate them with examples
- explain how those principles are to be applied
- explain how requirements determine which principles should be applied
- explain the impact of design principles on the implementation
- analyze source code and architecture designs to evaluate whether these design principles have been applied or should be applied

Abstraction (R2)

- in the sense of a means for deriving useful generalizations
- as a design technique, where building blocks are dependent on the abstractions rather than depending on implementations
- interfaces as abstractions

Modularization (R1)

- information hiding and encapsulation
- separation of concerns - SoC

- loose, but functionally sufficient, coupling (R1) of building blocks, see LG 03-06 [previously LG 2-07]: Manage Dependencies between Building Blocks (R1)
- high cohesion
- Open/closed principle
- Dependency inversion principle by means of interfaces or similar abstractions

Conceptual integrity (R2-R3)

- meaning uniformity (homogeneity, consistency) of solutions for similar problems (R2)
- as a means to achieve the principle of least surprise (aka *principle of least astonishment*, POLA) (R3)
- Liskov's substitution principle as a way to achieve consistency and conceptual integrity (R3).

Complexity reduction (R3)

- as the driving factor behind KISS, YAGNI, and CUPID [Terhorst-North 2022]
- DRY (Don't Repeat Yourself) as one option to avoid repetitions

Expect errors (R2-R3)

- as a means to design for robust and resilient systems (R3)
- as a generalization of the robustness principle (*Postel's law*) (R2)

SOLID Principles (R3)

Software architects know the benefits and limitations of the SOLID principles: Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle

References

[Liskov 1994], [SOLID]

LG 03-05 [previously LG 1-06]: Correlation between Feedback Loops and Risks (R1, R2)

Software architects understand the necessity of iterations, especially when decisions are made in the face of uncertainties. They

- are able to explain the influence of iterative approaches on architectural decisions (with regard to risks and predictability). (R1)
- can work and make decisions incrementally and iteratively. (R1)
- understand the necessity for feedback on architecture decisions. (R1)
- can systematically obtain feedback from other stakeholders. (R2)

LG 03-06 [previously LG 2-07]: Manage Dependencies between Building Blocks (R1)

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

- know and understand different types of dependencies of building blocks (e.g. coupling via

use/delegation, messaging/events, composition, creation, inheritance, temporal coupling, coupling via data, data types or hardware)

- understand how dependencies increase coupling
- can differentiate between at least the following categories of coupling:
  - static and dynamic coupling
  - efferent and afferent coupling
- know that forgoing static dependencies in favor of dynamic dependencies does not necessarily reduce the underlying coupling
- can identify coupling and assess its consequences
- can make justified decisions whether a dependency is appropriate or should be removed in view of the requirements and constraints
- know and can apply possibilities to reduce or eliminate coupling, for example:
  - patterns
  - fundamental design principles
  - externalization of dependencies, i.e. defining concrete dependencies at installation- or runtime, for example by using Dependency Injection (R3) (see also LG 03-08 [previously LG 2-05]: Describe, Explain and Apply Important Architectural Patterns (R1, R3)).

References

[Ford+ 2021]

LG 03-07 [previously LG 2-09]: Design and Define Interfaces (R1-R3)

Software architects know the critical importance of interfaces for the interaction between architectural building blocks or between the system and external elements. They can design and specify such interfaces.

They know:

- desired characteristics of interfaces and can achieve them in the design (R1):
  - 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 (R2)
- the distinction between interface and implementation (R1):
  - implementations can be exchanged if required.
- different characteristics of interfaces, for example (R3):
  - Transport channels (for example: TCP/IP as part of the OSI 7-layer model, shared memory)
  - internal or external
  - local or remote
  - synchronous or asynchronous

  - binary (only machine-readable) or textual (also human-readable)
  - stateless or stateful
  - point-to-point or multipoint (broadcast or multicast)
  - Function call (e.g. remote procedure call) or message exchange
  - Batch, request/response or streaming
- Different implementation approaches for interfaces, such as (R3:)
  - resource-oriented (REST, REpresentational State Transfer)
  - graph-oriented (e.g. GraphQL)
  - service-oriented (e.g. WS-*/SOAP-based web services)

See also LG 04-06 [previously LG 3-07]: Document Interfaces (R1).

References

[Zimmermann+2022], [Geewax 2021]

LG 03-08 [previously LG 2-05]: Describe, Explain and Apply Important Architectural Patterns (R1, R3)

Software architects can explain and provide examples for the following architectural patterns (R1):

- Layers
- Pipes and Filters
- Microservices

Software architects can explain several of the following architectural patterns, explain their relevance for concrete systems, and provide examples. (R3)

- Blackboard
- Broker
- CQRS (Command-Query-Responsibility-Segregation)
- Event sourcing
- Dependency Injection (see also LG 03-06 [previously LG 2-07]: Manage Dependencies between Building Blocks (R1))
- Integration and messaging patterns (e.g. from [Hohpe+2004])
- MVC (Model View Controller), MVVM (Model View ViewModel), MVU (Model View Update), PAC (Presentation Abstraction Control)
- Plugin
- Ports and Adapters (synonyms: Onion Architecture, Hexagonal Architecture, Clean Architecture)
- SOA (Service-Oriented Architecture)

Software architects know essential sources for architectural patterns, such as POSA (e.g. [Buschmann+1996]) and PoEAA ([Fowler 2002]) (for information systems). (R3)

References

[Buschmann+1996], [Buschmann+2007], [Eilebrecht+2024], [Fowler 2002], [Gamma+ 1994], [Hohpe+2004], [Pethuru 2017]

LG 03-09 [previously LG 2-05]: Describe, Explain, and Appropriately Apply Important Design Patterns (R3)

Software architects know the difference between architectural and design patterns. They can describe several of the following design patterns, explain their relevance for the architecture and specific systems and give examples.

- Combinator
- Interfacing patterns like Adapter, Facade, and Proxy. Architects should know that these patterns can be used independently of a particular programming language or framework.
- Interpreter
- Observer
- Remote procedure call
- Template Method and Strategy
- Visitor

Software architects know essential sources for design patterns, such as GOF and POSA.

They know:

- that patterns are a way of achieving certain qualities for given problems and requirements within given contexts.
- that there are different categories of patterns.
- additional sources of patterns that relate to their specific technical or application domain.

References

[Gamma+ 1994], [Buschmann+1996]

LG 03-10 [previously LG 2-04]: Identify, Design and Implement Cross-Cutting Concerns (R1)

Software architects are able to:

- explain the significance of cross-cutting concerns
- identify cross-cutting concerns
- design cross-cutting concepts, for example persistence, communication, GUI, error handling, concurrency, energy efficiency
- identify and assess potential interdependencies.

Software architects know that such cross-cutting concepts may be re-used across systems.

See also LG 04-07 [previously LG 3-06]: Document and Communicate Cross-Cutting Concerns (R2).

References

[Starke 2024], [Starke+2023a]

LG 03-11 [previously LG 2-10]: Know Fundamental Principles of Software Deployment (R3)

Software architects:

- know that software deployment is the process of making new or updated software available to its users
- are able to name and explain fundamental concepts of software deployment:
  - automated deployments
  - repeatable builds
  - consistent environments (e.g. use immutable and disposable infrastructure)
  - put everything under version-control
  - releases are easy-to-undo

References

[Humble+2010]

LG 03-12 [previously LG 1-11]: Know the Challenges of Distributed Systems (R3)

Software architects are able to:

- identify distribution in a given software architecture
- analyze consistency criteria for a given business problem
- explain causality of events in a distributed system

Software architects know:

- communication may fail in a distributed system
- limitations regarding consistency in real-world databases
- what the "split-brain" problem is and why it is difficult
- that it is impossible to determine the temporal order of events in a distributed system

References

[Tanenbaum+], [Buschmann+2007], [Ford+ 2021], [Miller+]

4. Specification and Communication of Software Architectures

Duration: 180 min.	Exercises: 60 min.

Purpose

The purpose of this section is to enable participants to document and communicate software architectures in a way that meets the needs of important stakeholders and supports the development process. The focus is on understanding the essential requirements for technical documentation, using appropriate models and notations to describe architectures, and applying key architectural views. Additionally, participants will learn to document key architectural decisions, interfaces, and cross-cutting concerns, ensuring clear, correct, and stakeholder-relevant documentation.

Relevant Terms

(Architectural) Views; structures; (technical) concepts; documentation; communication; description; alignment with target-groups and stakeholder concerns; 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; mapping deployment artifacts onto nodes; description of interfaces and design decisions; UML; tools for documentation

Learning Goals

LG 04-01 [previously LG 3-01]: Explain and Consider the Requirements of Technical Documentation (R1)

Software architects know the essential requirements for technical documentation and can consider and fulfil them when documenting systems:

- understandability, correctness, efficiency, appropriateness, maintainability
- form, content, and level of detail tailored to the target group of the documentation

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

References

[Starke+2023a], [Zörner 2021], [Clements+2010]

LG 04-02 [previously LG 3-02]: Describe and Communicate Software Architectures (R1-R3)

Software architects use documentation to support the design, implementation and further development (also called *maintenance* or *evolution*) of systems (R2)

Software architects are able to (R1):

- document and communicate architectures for corresponding stakeholder concerns, 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
- develop and implement measures to support the consistency of written and verbal communication, and balance one against the other appropriately

Software architects know (R1):

- the benefits of template-based documentation
- that various properties of documentation depend on specific properties of the system, its requirements, risks, development process, organization or other factors.

For example, software architects can adjust the following documentation characteristics according to the situation (R3):

- the amount and level of detail of documentation needed
- the documentation format
- the accessibility of the documentation
- formality of documentation (e.g. diagrams compliant to a meta model or simple drawings)
- formal reviews and sign-off processes for documentation

LG 04-03 [previously LG 3-03]: Explain and Apply Notations/Models to Describe Software Architecture (R2-R3)

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

- class, package, component (all R2) and composite-structure diagrams (R3)
- deployment diagrams (R2)
- sequence and activity diagrams (R2)
- state machine diagrams (R3)

Software architects know alternative notations to UML diagrams, for example: (R3)

- Archimate
- SysML
- C4, see [Brown]
- Entity-relationship diagrams, see [Chen 1976]
- for runtime views for example flow charts, numbered lists or business-process-modeling-notation (BPMN).

References

[UML], [ArchiMate], [SysML], [Brown], [Chen 1976]

LG 04-04 [new]: Learning Goal not Found (R3)

Software architects are able to gracefully deal with unexpected situations.

References

[IETF HTTP]

LG 04-05 [previously LG 3-04]: Explain and Use Architectural Views (R1)

Software architects are able to use the following architectural views:

- context view
  - contains the external interfaces of systems
  - when appropriate differentiated according to business and technical context
- building block or component view (composition of software building blocks)
- runtime view (dynamic view, interaction between software building blocks at runtime, state machines)
- deployment view (hardware and technical infrastructure as well as the mapping of software building blocks onto the infrastructure)

Additional views might be used as needed to address further stakeholder concerns and requirements, such as functional safety, information view, operational view or user-interface view (R3).

References

[Kruchten 1995], [Rozanski+2011], [Starke+2023a], [arc42], [Brown]

LG 04-06 [previously LG 3-07]: Document Interfaces (R1)

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

See also LG 03-07 [previously LG 2-09]: Design and Define Interfaces (R1-R3).

LG 04-07 [previously LG 3-06]: Document and Communicate Cross-Cutting Concerns (R2)

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

See also LG 03-10 [previously LG 2-04]: Identify, Design and Implement Cross-Cutting Concerns (R1)

LG 04-08 [previously LG 3-08]: Explain and Document Architectural Decisions (R1-R2)

Software architects are able to:

- systematically make, justify, communicate, and document architectural decisions
- identify, communicate, and document interdependencies between architecture decisions

Software architects know about Architecture-Decision-Records (ADR, see [Nygard 2011]) and can apply these to document decisions (R2).

References

[Nygard 2011]

LG 04-09 [previously LG 3-09]: 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/IEC/IEEE 42010,
  - arc42,

    - C4, see [Brown]
- ideas and examples of checklists for the creation, documentation, and review of software architectures
- possible tools for creating and maintaining architectural documentation

References

[ISO 42010], [arc42], [Brown]

5. Analysis and Assessment of Software Architectures

Duration: 60 min.	Exercises: 30 min.

Purpose

The purpose of this section is to equip software architects with the skills and knowledge needed to effectively perform architecture analysis. They learn to identify risks, evaluate conformance to architectural decisions, and assess the overall quality of a system based on its design and implementation. By understanding various analysis methods, such as acceptance testing, architecture metrics, scenario-based analysis, and cost-benefit analysis, architects can ensure that a software architecture meets stakeholder requirements and is aligned with the intended design.

Relevant Terms

Architecture analysis; Risk identification; Quality analysis methods; Scenarios; Scenario-based analysis; Metrics; Tool-supported analysis

Learning Goals

LG 05-01: Know Reasons for Architecture Analysis (R1)

Software architects understand that there are different possible reasons for performing architecture analysis, for example:

- identify risks and possible improvements in the architecture design (before, during, or after implementation)
- determine if the architecture design fulfills, or will fulfill, the requirements
- assess conformance of the implementation to the architecture decisions and design
- verify that architecturally relevant stakeholder concerns are addressed

LG 05-02 [previously LG 4-3 and 4-4]: Analyze the Qualities of a Software System (R1, R3)

Software architects

- understand that, for any given quality, different analysis methods might be available for a particular software system, such as
  - analysis of the results of acceptance testing (R1)
  - quantitative measurement of run-time behaviour (R1)
  - qualitative evaluation via interviews, surveys, penetration tests etc. (R1)
  - scenario-based analysis (R1)
  - architecture metrics for coupling such as the degree of inbound and outbound dependencies (R1)
  - cost-benefit analysis (R3)
  - Architecture Trade-Off Analysis Method [Bass+2021] (R3)
- know sources of information for a quality analysis (R2):
  - requirements documentation (R1)

- architecture documentation (R1)
- architecture and design models (R1)
- source code (R1)
- source-code-related metrics such as lines of code, (cyclomatic) complexity (R1)
- test cases and test results (R1)
- errors and their locations in the source code, especially error clusters (R1)
- other documentation of the system, such as operational or test documentation (R1)
- run-time event logs and metrics (R1)
- revision history, such as the rate of change per component (R3)

See also LG 02-03 [previously LG 4-1]: Understand and Explain Qualities of a Software System (R1), LG 02-04 [previously LG 04-03]: Formulate Requirements on Qualities (R1-R3).

References

[Bass+2021], [Starke+2023a], [Clements+2002], [ISO 25019], [Lilienthal 2019]

LG 05-03: Evaluate Conformance to Architectural Decisions (R2)

Software architects are able to assess whether the system's implementation aligns with the architectural design and decisions, using methods such as code and architecture reviews or tool-supported analysis.

6. Examples of Software Architectures

Duration: 90 min.	Exercises: none

This section is not relevant for the exam.

Learning Goals

LG 06-01: 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.

References

[arc42], [Starke+2023b], [Hruschka+2021], [Zörner 2021], [AOSA]

LG 06-02: Understand the technical implementation of a solution (R3)

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

References

[arc42], [Starke+2023b], [Hruschka+2021], [Zörner 2021], [AOSA]

References

- [AOSA] The Architecture of Open Source Applications. Edited by Amy Brown and Greg Wilson. Online: https://aosabook.org/en/.
- [arc42] arc42, the open-source template for software architecture communication, online: https://arc42.org. Maintained on https://github.com/arc42
- [ArchiMate] The ArchiMate® Enterprise Architecture Modeling Language, online: https://www.opengroup.org/archimate-forum/archimate-overview
- [Bass+2021] Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. 4th Edition, Addison Wesley 2021.
- [Brown] Simon Brown: The C4 model for visualising software architecture. https://c4model.com https://www.infoq.com/articles/C4-architecture-model.
- [IREB Foundation] Stan Bühne, Martin Glinz, Hans van Loen, Stefan Staal: Certified Professional for Requirements Engineering - Foundation Level - Syllabus - Version 3.2.0, IREB, 2024.
- [Burns 2018] Brendan Burns: Designing Distributed Systems, Patterns and Paradigms for Scalable, Reliable Services, OReilly 2018.
- [Buschmann+1996] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal: Pattern-Oriented Software Architecture (POSA): A System of Patterns. Wiley, 1996.
- [Buschmann+2007] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt: Pattern-Oriented Software Architecture (POSA): A Pattern Language for Distributed Computing, Wiley, 2007.
- [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
- [CloudNative] The Cloud Native Computing Foundation, online: https://www.cncf.io/
- [Eilebrecht+2024] Karl Eilebrecht, Gernot Starke: Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung (in German). 6th Edition Springer Verlag 2024.
- [Chen 1976] Chen, Peter (March 1976): *The Entity-Relationship Model - Toward a Unified View of Data*. ACM Transactions on Database Systems. 1 (1): 9–36..
- [Evans 2004] Eric Evans: *Domain-Driven Design: Tackling Complexity in the Heart of Software,* Addison-Wesley, 2004.
- [Felleisen+2014] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi: How to Design Programs. Second Edition. MIT Press, 2014. https://htdp.org/
- [Ford 2017] Neil Ford, Rebecca Parsons, Patrick Kua: Building Evolutionary Architectures: Support Constant Change. OReilly 2017.
- [Ford+ 2021] Neal Ford, Mark Richards, Pramod Sadalage und Zhamak Dehghani: Software Architecture: The Hard Parts. Modern Trade-Off Analyses for Distributed Architectures. OReilly 2021.
- [Fowler 2002] Martin Fowler: Patterns of Enterprise Application Architecture. (PoEAA) Addison-Wesley, 2002.

- [Ghandi+2024] Raju Gandhi, Mark Richards and Neal Ford. Head-First Software Architecture. OReilly 2024.
- [Gamma+ 1994] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
- [Geewax 2021] J. Geewax. API Design Patterns. Manning, 2021. This book lays out a set of design principles for building internal and public-facing APIs.
- [Geirhos 2015] Matthias Geirhos. Entwurfsmuster: Das umfassende Handbuch (in German). Rheinwerk Computing Verlag. 2015
- [Gharbi+2024] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 5. Auflage, dpunkt Verlag, Heidelberg 2024.
- [Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java (in German). Springer-Vieweg Verlag, 2. Auflage 2014.
- [Hofmeister+1999] Christine Hofmeister, Robert Nord, Dilip Soni: *Applied Software Architecture*, Addison-Wesley, 1999
- [Hohpe+2004] Hohpe, G. and WOOLF, B.A.: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, 2004
- [Hombergs 2024] Hombergs, Tom: Get Your Hands Dirty on Clean Architecture, Packt, 2nd edition 2024.
- [Humble+2010] Jez Humble, David Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Pearson International, 2010
- [Hruschka+2021] Peter Hruschka, Ivan Kostov and Wolfgang Reimesch: arc42-by-Example Vol 2: Architecture Documentation for Embedded Systems and IoT. Leanpub, 2021. https://leanpub.com/arc42byexample-volume2
- [IETF HTTP] Internet Engineering Task Force: RFC 9110, HTTP Semantics. Online: https://www.rfc-editor.org/rfc/rfc9110.html
- [iSAQB Downloads] iSAQB public download site. https://public.isaqb.org. Contains curricula and mock-examination.
- [iSAQB Glossary] Gernot Starke et. al. iSAQB Glossary of Software Architecture Terminology. Freely available from https://leanpub.com/isaqbglossary or its source repository https://github.com/isaqb-org/glossary/releases
- [iSAQB References] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Freely available https://leanpub.com/isaqbreferences.
- [ISO 25010] ISO/IEC 25010:2023(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. Terms and definitions online: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-2:v1:en
- [ISO 25019] ISO/IEC 25019:2023(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality-in-use model. Terms and definitions online: https://www.iso.org/obp/ui/#iso:std:iso-iec:25019:ed-1:v1:en
- [ISO 42010] ISO/IEC/IEEE 42010:2022, Software, systems and enterprise Architecture description, online: https://www.iso.org/standard/74393.html
- [Keeling 2017] Michael Keeling. Design It!: From Programmer to Software Architect. Pragmatic Programmer.

- [Kleppmann 2017] Martin Kleppmann: Designing Data-Intensive Applications. O'Reilly 2017.
- [Kruchten 1995] Philippe Kruchten: Architectural Blueprints—The "4+1" View Model of Software Architecture, IEEE Software 12 (6), November 1995, pp. 42-50
- [Kruchten 2004] Philippe Kruchten: The Rational Unified Process: An Introduction. 3rd edition. Addison-Wesley Professional 2004.
- [Lange 2021] Kenneth Lange: The Functional Core, Imperative Shell Pattern, online: https://www.kennethlange.com/functional-core-imperative-shell/
- [Lehman 1980] Meir M. Lehman: Programs, Life Cycles, and Laws of Software Evolution. Proceedings of the IEEE, 68(9), 1060-1076, 1980.
- [Wiki-LehmansLaws] Laws of Software Evolution. https://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution
- [Lilienthal 2024] Carola Lilienthal: Langlebige Softwarearchitekuren. 4. Auflage, dpunkt Verlag 2024.
- [Lilienthal 2019] Carola Lilienthal: Sustainable Software Architecture: Analyze and Reduce Technical Debt. dpunkt Verlag 2019.
- [Liskov 1994] Barbara H. Liskov, Jeannette M. Wing: A behavioral notion of subtyping. ACM Transactions on Programming Languages and Systems, Volume 16, Issue 6, 1994. <doi:10.1145/197320.197383>
- [Maguire 2019] Sandy Maguire: Algebra-Driven Design - Elegant Solutions from Simple Building Blocks. Leanpub, 2019.
- [Miller+] Heather Miller, Nat Dempkowski, James Larisch, Christopher Meiklejohn: Distributed Programming (to appear, but content-complete) https://github.com/heathermiller/dist-prog-book.
- [Newman 2021] Sam Newman. Building Microservices - Designing Fine-Grained Systems. O'Reilly 2nd edition 2021.
- [Terhorst-North 2022] Daniel Terhorst-North: CUPID - for joyful coding. See https://dannorth.net/2022/02/10/cupid-for-joyful-coding/.
- [Nygard 2011] Michael Nygard: Documenting Architecture Decision. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions. See also https://adr.github.io/
- [Pethuru 2017] Raj Pethuru et. al: Architectural Patterns. Packt 2017.
- [Pohl 2025] Klaus Pohl: Requirements Engineering - Fundamentals, Principles and Techniques. Springer 2025
- [Q42] arc42 Quality Model, online: https://quality.arc42.org.
- [Rajlich+2000] Václav T. Rajlich, Keith H. Bennett: A Staged Model for the Software Life Cycle. IEEE Computer 33(7): 66-71, 2000.
- [Read 2023] Jacqui Read: Communication Patterns - An Engineering Approach. A Guide for Developers and Architects. OReilly 2023.
- [Richards+2020] Mark Richards, Neal Ford: Fundamentals of Software Architecture - An Engineering Approach. OReilly 2020.
- [Rozanski+2011] Nick Rozanski, Eoin Woods: Software Systems Architecture - Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley, 2nd edition 2011.

- [SOLID] Samuel Oloruntoba and Anish Singh Walia: SOLID: The First 5 Principles of Object Oriented Design, https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.
- [Sperber+2023] Michael Sperber, Herber Klaeren: Schreibe Dein Programm! Tübingen University Press, 2023. https://www.deinprogramm.de/sdp/.
- [Sperber+2024] Michael Sperber, Stefan Wehr: Datenmodellierung mit Summen und Produkten, 2024. https://funktionale-programmierung.de/2024/11/25/sums-products.html. (English translation: Data Modeling with Sums and Products, 2024. https://funktionale-programmierung.de/2024/11/25/sums-products-english.html)
- [Starke 2024] Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden (in German). 10. Auflage, Carl Hanser Verlag 2024. Website: https://esabuch.de
- [Starke+2023a] Gernot Starke, Alexander Lorz: Software Architecture Foundation, CPSA Foundation® Exam Preparation. Van Haaren Publishing, 2nd edition, 2023.
- [Starke+2023b] Gernot Starke, Michael Simons, Stefan Zörner, Ralf D. Müller, and Hendrik Lösch: arc42-by-Example - Software Architecture Documentation in Practice. Leanpub, 3rd edition 2023. https://leanpub.com/arc42byexample
- [SysML] What is SysML https://sysml.org/. For diagrams, see also https://sysml.org/tutorials/sysml-diagram-tutorial/.
- [Tanenbaum+] Andrew Tanenbaum, Maarten van Steen: Distributed Systems, Principles and Paradigms. https://www.distributed-systems.net/.
- [UML] The UML reading room, collection of UML resources https://www.omg.org/technology/readingroom/UML.htm. See also https://www.uml-diagrams.org/.
- [Yorgey 2012] Brent A. Yorgey, Monoids: Theme and Variations. Proceedings of the 2012 Haskell Symposium, September 2012 https://doi.org/10.1145/2364506.2364520
- [Zimmermann+2022] Olaf Zimmermann, Mirko Stocker, Daniel Lübke, Uwe Zdun, Cesare Pautasso: Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges. Addison-Wesley, 2022.
- [Zörner 2021] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. 3. Auflage, Carl Hanser Verlag, 2021.