Table of Contents

Several languages have been configured, but the language-specific headings (like "Table of Contents" will be displayed in a single language only (currently set to EN))

Rechtliches

Die Abkürzung "e. V." ist Teil des offiziellen Namens des iSAQB und steht für "eingetragener Verein", der seinen Status als juristische Person nach deutschem Recht beschreibt. Der Einfachheit halber wird iSAQB e. V. im Folgenden ohne die Verwendung dieser Abkürzung als iSAQB bezeichnet.

Legal Notice

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.

This version of this document has been produced with comments (like this one) enabled. It is **NOT** intended for public distribution or publication, but primarily for internal iSAQB purposes.

Verzeichnis der Lernziele

- LZ 4-2: Qualitätsanforderungen an Softwarearchitekturen klären (R1)
- LZ 4-3: Softwarearchitekturen qualitativ analysieren und bewerten (R2-R3)
- LZ 4-4: Softwarearchitekturen quantitativ bewerten (R2)
- LZ 5-1: Bezug von Anforderungen und Randbedingungen zu Lösung erfassen (R3)
- LZ 5-2: Technische Umsetzung einer Lösung nachvollziehen (R3) == List of Learning Goals
- LG 1-1: Discuss definitions of software architecture (R1)
- LG 1-2: Understand and explain the goals and benefits of software architecture (R1)
- LG 1-3: Understand software architecture as part of the software life cycle (R2)
- LG 1-4: Understand software architects' tasks and responsibilities (R1)
- LG 1-5: Relate the role of software architects to other stakeholders (R1)
- LG 1-6: Can explain the correlation between development approaches and software architecture (R1)
- LG 1-7: Differentiate between short- and long-term goals (R1)
- LG 1-8: Distinguish explicit statements and implicit assumptions (R1)
- LG 1-9: Responsibilities of software architects within the greater architectural context (R3)
- LG 1-10: Differentiate types of IT systems (R3)
- LG 1-11: Challenges of distributed systems (R3)
- LG 2-1: Select and use approaches and heuristics for architecture development (R1,R3)
- LG 2-2: Design software architectures (R1)
- LG 2-3: Identify and consider factors influencing software architecture (R1-R3)
- LG 2-4: Design and implement cross-cutting concepts (R1)
- LG 2-5: Describe, explain and appropriately apply important solution patterns (R1, R3)
- LG 2-6: Explain and use design principles (R1-R3)
- LG 2-7: Manage dependencies between building blocks (R1)
- LG 2-8: Achieve quality requirements with appropriate approaches and techniques (R1)
- LG 2-9: Design and define interfaces (R1-R3)
- LG 3-1: Explain and consider the quality of technical documentation (R1)
- LG 3-2: Describe and communicate software architectures (R1,R3)
- LG 3-3: Explain and apply notations/models to describe software architecture (R2-R3)
- LG 3-4: Explain and use architectural views (R1)
- LG 3-5: Explain and apply context view of systems (R1)
- LG 3-6: Document and communicate cross-cutting concepts (R2)
- LG 3-7: Describe interfaces (R1)
- LG 3-8: Explain and document architectural decisions (R1-R2)
- LG 3-9: Use documentation as written communication (R2)
- LG 3-10: Know additional resources and tools for documentation (R3)

- LG 4-1: Discuss quality models and quality characteristics (R1)
- LG 4-2: Clarify quality requirements for software architectures (R1)
- LG 4-3: Qualitative analysis and assessment of software architectures (R2-R3)
- LG 4-4: Quantitative evaluation of software architectures (R2)
- LG 5-1: Know the relation between requirements, constraints, and solutions (R3)
- LG 5-2: Know the rationale of a solution's technical implementation (R3)

Introduction

Was vermittelt eine Foundation-Level-Schulung?

Lizenzierte Schulungen zum *Certified Professional for Software Architecture – Foundation Level* (CPSA-F) vermitteln grundlegende Kenntnisse und Fertigkeiten für den Entwurf einer angemessenen Softwarearchitektur für kleine und mittlere IT-Systeme. Die Teilnehmenden erweitern und vertiefen ihre bestehenden Erfahrungen und Fähigkeiten in der Softwareentwicklung um relevante Vorgehensweisen, Methoden und Prinzipien für die Entwicklung von Softwarearchitekturen. Durch das Gelernte können sie auf Grundlage angemessen detaillierter Anforderungen und Randbedingungen eine adäquate Softwarearchitektur entwerfen, kommunizieren, analysieren, bewerten und weiterentwickeln. Schulungen zum CPSA-F vermitteln Grundlagenwissen unabhängig von spezifischen Entwurfsmethoden, Vorgehensmodellen, Programmiersprachen oder Werkzeugen. Dadurch können die Teilnehmenden ihre erworbene Fertigkeiten auf ein breites Spektrum von Einsatzfällen anwenden.

Im Mittelpunkt steht der Erwerb folgender Fähigkeiten:

- mit anderen Beteiligten aus den Bereichen Anforderungsmanagement, Projektmanagement, Entwicklung und Test wesentliche Architekturentscheidungen abzustimmen
- die wesentlichen Schritte beim Entwurf von Softwarearchitekturen zu verstehen sowie für kleine und mittlere Systeme selbständig durchzuführen
- Softwarearchitekturen auf Basis von Sichten, Architekturmustern und technischen Konzepten zu dokumentieren und zu kommunizieren.

Darüber hinaus behandeln CPSA-F-Schulungen:

- den Begriff und die Bedeutung von Softwarearchitektur
- die Aufgaben und Verantwortung von Softwarearchitekten
- die Rolle von Softwarearchitekt:innen in Entwicklungsvorhaben
- State-of-the-Art-Methoden und -Praktiken zur Entwicklung von Softwarearchitekturen.

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.

Abgrenzung

Dieser Lehrplan reflektiert den aus heutiger Sicht des iSAQB e.V. notwendigen und sinnvollen Inhalt zur Erreichung der Lernziele des CPSA-F. Er stellt keine vollständige Beschreibung des Wissensgebiets „Softwarearchitektur“ dar.

Folgende Themen oder Konzepte sind **nicht Bestandteil des CPSA-F**:

- konkrete Implementierungstechnologien, -frameworks oder -bibliotheken
- Programmierung oder Programmiersprachen
- Spezifische Vorgehensmodelle
- Grundlagen oder Notationen der Modellierung (wie etwa UML)
- Systemanalyse und Requirements Engineering (siehe dazu das Ausbildungs- und Zertifizierungsprogramm des IREB e. V., https://ireb.org, International Requirements Engineering Board)
- Test (siehe dazu das Ausbildungs- und Zertifizierungsprogramm des ISTQB e. V., https://istqb.org, International Software Testing Qualification Board)
- Projekt- oder Produktmanagement
- Einführung in konkrete Werkzeuge.

Ziel des Trainings ist es, die Grundlagen für den Erwerb der für den jeweiligen Einsatzfall notwendigen weiterführenden Kenntnisse und Fertigkeiten zu vermitteln.

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.
```

Voraussetzungen

Der iSAQB e. V. kann in Zertifizierungsprüfungen die hier genannten Voraussetzungen durch entsprechende Fragen prüfen.

Teilnehmende sollten die im Nachfolgenden genannten Kenntnisse und/oder Erfahrung mitbringen. Insbesondere bilden substanzielle praktischen Erfahrungen aus der Softwareentwicklung im Team eine wichtige Voraussetzung zum Verständnis des vermittelten Lernstoffes und für eine erfolgreiche Zertifizierung.

- mehr als 18 Monate praktische Erfahrung in arbeitsteiliger Softwareentwicklung (d.h. in Teams), erworben durch Programmierung unterschiedlicher Systeme außerhalb der Ausbildung
- Kenntnisse und praktische Erfahrung in mindestens einer höheren Programmiersprache, insbesondere:
  - Konzepte der
    - Modularisierung (Pakete, Namensräume)
    - Parameterübergabe (*Call-by-Value, Call-by-Reference*)
    - Gültigkeit (*scope*), beispielsweise von Typ- oder Variablendeklaration und -definition
  - Grundlagen von Typsystemen (statische und dynamische Typisierung, generische Datentypen)
  - Fehler- und Ausnahmebehandlung in Software
  - Mögliche Probleme von globalem Zustand und globalen Variablen
- Grundlegende Kenntnisse von:
  - Modellierung und Abstraktion
  - Algorithmen und Datenstrukturen (etwa Listen, Bäume, HashTable, Dictionary/Map)
  - UML (Klassen-, Paket-, Komponenten- und Sequenzdiagramme) und deren Bezug zum Quellcode

Hilfreich für das Verständnis einiger Konzepte sind darüber hinaus:

- Grundbegriffe bzw. Unterschiede von imperativer, deklarativer, objektorientierter und funktionaler Programmierung
- praktische Erfahrung in
  - einer höheren Programmiersprache
  - Konzeption und Implementierung verteilt ablaufender Anwendungen, wie etwa Client/Server-Systeme oder Web-Anwendungen
  - technischer Dokumentation, insbesondere in der Dokumentation von Quellcode, Systementwürfen oder technischen Konzepten

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

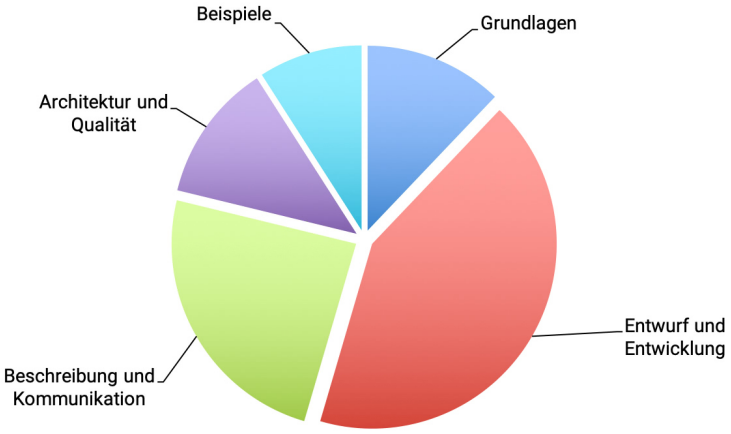
Furthermore, the following will be useful for understanding several concepts:

- Basics and differences of imperative, declarative, object-oriented and functional programming
- Practical experience in
  - 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

GS: March 2019: added several (programming and development) prerequisites

Struktur, Dauer, Didaktik

Die in den nachfolgenden Kapiteln des Lehrplans genannten Zeiten sind lediglich Empfehlungen. Die Dauer einer Schulung sollte mindestens 3 Tage betragen, kann aber durchaus länger sein. Anbieter können sich durch Dauer, Didaktik, Art und Aufbau der Übungen sowie der detaillierten Kursgliederung voneinander unterscheiden. Insbesondere die Art (fachliche und technische Domänen) der Beispiele und Übungen können die jeweiligen Schulungsanbieter individuell festlegen.

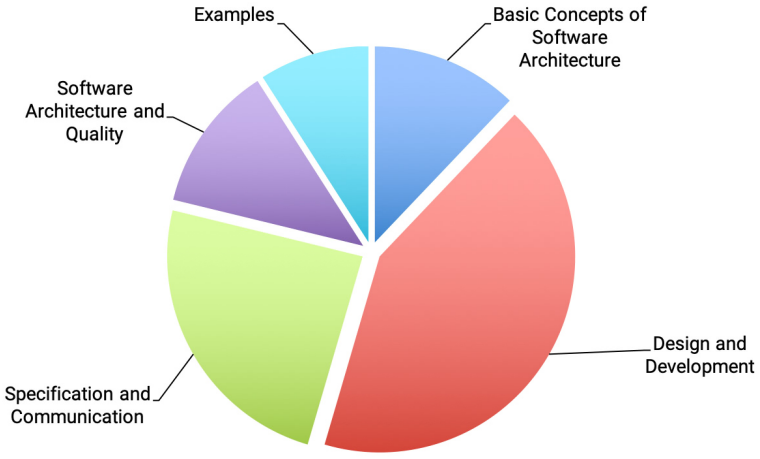


Empfohlene Zeitaufteilung

Inhalt	Empfohlene Dauer (min)
1. Grundlagen	120
2. Entwurf und Entwicklung	420
3. Beschreibung und Kommunikation	240
4. Architektur und Qualität	120
5. Beispiele	90
Summe	990

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

Content	Recommended Duration (min)
1. Basic Concepts of Software Architecture	120
2. Design and Development	420
3. Specification and Communication	240
4. Software Architecture and Quality	120
5. Examples	90
Total	990

Lernziele und Prüfungsrelevanz

Die Kapitel des Lehrplans sind anhand von priorisierten Lernzielen gegliedert. Die Prüfungsrelevanz dieser Lernziele beziehungsweise deren Unterpunkte ist beim jeweiligen Lernziel ausdrücklich gekennzeichnet (durch Angabe der Kennzeichen R1, R2 oder R3, siehe nachstehende Tabelle).

Jedes Lernziel beschreibt die zu vermittelnden Inhalte inklusive ihrer Kernbegriffe und -konzepte. Bezüglich der Prüfungsrelevanz verwendet der Lehrplan folgende Kategorien:

ID	Lernziel-Kategorie	Bedeutung	Relevanz für Prüfung
R1	Können	Diese Inhalte sollen die Teilnehmenden nach der Schulung selbständig anwenden können. Innerhalb der Schulung werden diese Inhalte durch Übungen und Diskussionen abgedeckt.	Inhalte **werden** geprüft.
R2	Verstehen	Diese Inhalte sollen die Teilnehmenden grundsätzlich verstehen. Sie werden in Schulungen i. d. R. nicht durch Übungen vertieft.	Inhalte **können** geprüft werden.
R3	Kennen	Diese Inhalte (Begriffe, Konzepte, Methoden, Praktiken oder Ähnliches) können das Verständnis unterstützen oder das Thema motivieren. Sie werden in Schulungen bei Bedarf thematisiert.	Inhalte **werden nicht** geprüft.

Bei Bedarf enthalten die Lernziele Verweise auf weiterführende Literatur, Standards oder andere Quellen. Die Abschnitte "Begriffe und Konzepte" zu Beginn jedes Kapitels zeigen Worte, die mit dem Inhalt des Kapitels in Verbindung stehen und z. T. auch in den Lernzielen verwendet werden.

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.

Aktuelle Version sowie öffentliches Repository

Den aktuellen Stand dieses Dokumentes finden Sie auf der offiziellen Download-Seite unter https://isaqb-org.github.io/.

Das Dokument wird in einem öffentlichen Repository unter https://github.com/isaqb-org/curriculum-foundation gepflegt, alle Änderungen sind dort sichtbar.

Bitte melden Sie eventuelle Probleme in unserem öffentlichen Issue Tracker bei https://github.com/isaqb-org/curriculum-foundation/issues.

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. Grundbegriffe

Dauer: 120 Min.	Übungszeit: Keine

Wesentliche Begriffe

Softwarearchitektur; Architekturdomänen; Struktur; Bausteine; Komponenten; Schnittstellen; Beziehungen; Querschnittskonzepte; Nutzen von Softwarearchitektur; Softwarearchitekt:innen und deren Verantwortlichkeiten; Rolle; Aufgaben und benötigte Fähigkeiten; Stakeholder und deren Anliegen; funktionale Anforderungen; Qualitätsanforderungen; Randbedingungen; Einflussfaktoren; Typen von IT-Systemen (eingebettete Systeme; Echtzeitsysteme; Informationssysteme etc.)

2. Basic concepts of software architecture

Duration: 120 min.	Exercises: none

Relevant terms

Software architecture; architecture domains; structure; building blocks; components; interfaces; relationships; cross-cutting-concepts; 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

LZ 1-1: Definitionen von Softwarearchitektur diskutieren (R1)

Softwarearchitekt:innen kennen mehrere Definitionen von Softwarearchitektur (u. a. ISO 42010/IEEE 1471, SEI, Booch etc.) und können deren Gemeinsamkeiten benennen:

- Komponenten/Bausteine mit Schnittstellen und Beziehungen
- Bausteine als allgemeiner Begriff, Komponenten als eine spezielle Ausprägung davon
- Strukturen, Querschnittskonzepte, Prinzipien
- Architekturentscheidungen mit systemweiten oder den gesamten Lebenszyklus betreffenden Konsequenzen

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, cross-cutting concepts, principles
- Architecture decisions and their consequences on the entire systems and its lifecycle

LZ 1-2: Nutzen und Ziele von Softwarearchitektur verstehen und erläutern (R1)

Softwarearchitekt:innen können folgenden Nutzen und wesentlichen Ziele von Softwarearchitektur begründen:

- Entwurf, Implementierung, Pflege und Betrieb von Systemen zu unterstützen
- Qualitätsanforderungen wie Zuverlässigkeit, Wartbarkeit, Änderbarkeit, Sicherheit zu erreichen
- funktionale Anforderungen zu erreichen bzw. deren Erfüllbarkeit sicherzustellen
- Verständnis für Strukturen und Konzepte des Systems zu vermitteln, bezogen auf sämtliche relevanten Stakeholder
- systematisch Komplexität zu reduzieren
- architekturrelevante Richtlinien für Implementierung und Betrieb zu spezifizieren

LG 1-2: 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 quality requirements such as reliability, maintainability, changeability, security, etc.
- achieve functional requirements or ensure that they can be met
- ensure that the the system's structures and concepts are understood by all relevant stakeholders
- systematically reduce complexity
- specify architecturally relevant guidelines for implementation and operation

GS, August 2019: after discussions and comments from Andreas Rausch and Mahbouba Gharbi, re-formulated first item (removed "in contrast to functionality").
GS/CL: sprachlich leicht umformuliert

RR: Keine klare Abgrenzung zwischen Nutzen und Zielen, deswegen vereinfacht.

LZ 1-3: Softwarearchitektur in Software-Lebenszyklus einordnen (R2)

Softwarearchitekt:innen können Ihre Aufgaben und Ergebnisse in den gesamten Lebenszyklus von IT-Systemen einordnen. Sie können:

- Konsequenzen von Änderungen bei funktionalen Anforderungen, Qualitätsanforderungen, Technologien oder der Systemumgebung im Bezug auf die Softwarearchitektur erkennen
- inhaltliche Zusammenhänge zwischen IT-Systemen und den unterstützten Geschäfts- und Betriebsprozessen aufzeigen

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

GS (April 2019, in reaction to remarks from RR): Aligned DE and EN versions, changed order of items, slightly rephrased second item. Removed references to DWH and other kinds of information systems, as it didn't fit the contents of this LG.

GS (Feb 2019): I'd prefer to replace "in die gesamte Entwicklung" by "Lifecycle" (intro sentence)

LZ 1-4: Aufgaben und Verantwortung von Softwarearchitekt:innen verstehen (R1)

Softwarearchitekt:innen tragen die Verantwortung für die Erreichung der geforderten oder notwendigen Qualität und die Erstellung des Architekturentwurfs der Lösung. Sie müssen diese Verantwortung, abhängig vom jeweiligen Prozess- oder Vorgehensmodell, mit der Gesamtverantwortung der Projektleitung

oder anderen Rollen koordinieren.

Aufgaben und Verantwortung von Softwarearchitekt:innen:

- Anforderungen und Randbedingungen klären, hinterfragen und bei Bedarf verfeinern Hierzu zählen neben den funktionalen Anforderungen (Required Features) insbesondere die geforderten Qualitätsmerkmale (*Required Constraints*)
- Strukturentscheidungen hinsichtlich Systemzerlegung und Bausteinstruktur treffen, dabei Abhängigkeiten und Schnittstellen zwischen den Bausteinen festlegen
- Querschnittskonzepte entscheiden (beispielsweise Persistenz, Kommunikation, GUI) und bei Bedarf umsetzen
- Softwarearchitektur auf Basis von Sichten, Architekturmustern sowie technischen und Querschnittskonzepten kommunizieren und dokumentieren
- Umsetzung und Implementierung der Architektur begleiten, Rückmeldungen der beteiligten Stakeholder bei Bedarf in die Architektur einarbeiten, Konsistenz von Quellcode und Softwarearchitektur prüfen und sicherstellen
- Softwarearchitektur analysieren und bewerten, insbesondere hinsichtlich Risiken bezüglich der geforderten Qualitätsmerkmale
- Die Konsequenzen von Architekturentscheidungen erkennen, aufzeigen und gegenüber anderen Stakeholdern argumentieren

Sie sollen selbständig die Notwendigkeit von Iterationen bei allen Aufgaben erkennen und Möglichkeiten für entsprechende Rückmeldung aufzeigen.

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 concepts (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.

RER, 2019-02-26: changed English translations. RER, 2019-02-26: changed first sentence so that the architect is responsible for the quality and architecture design rather than the quality and functionality. GS, Feb 2019: added "analyze" to the "evaluate" activity, both DE and EN

GS, Feb 2019: TODO: Übesetzung von "gegenüber anderen Stakeholdern argumentieren" mit "discuss with…" finde ich nicht passend.

LZ 1-5: Rolle von Softwarearchitekt:innen in Beziehung zu anderen Stakeholdern setzen (R1)

Softwarearchitekt:innen können ihre Rolle erklären. Sie sollten ihren Beitrag zur Systementwicklung in Verbindung mit anderen Stakeholdern und Organisationseinheiten kontextspezifisch ausgestalten, insbesondere zu:

- Produktmanagement, Product-Owner
- Projektleitung und -management
- Anforderungsanalytiker:innen (System-/Businessanalyse, Anforderungsmanagement, Fachbereich)
- Entwicklung
- Qualitätssicherung und Test
- IT-Betrieb (Produktion, Rechenzentren), zutreffend primär für Informationssysteme
- Hardwareentwicklung
- Unternehmensarchitektur, Architekturboard.

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 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
- enterprise architects and architecture board members

LZ 1-6: Zusammenhang zwischen Entwicklungsvorgehen und Softwarearchitektur erläutern können (R1)

- Softwarearchitekt:innen können den Einfluss von iterativem Vorgehen auf Architekturentscheidungen erläutern (hinsichtlich Risiken und Prognostizierbarkeit).
- Sie müssen aufgrund inhärenter Unsicherheit oftmals iterativ arbeiten und entscheiden. Dabei müssen sie bei anderen Stakeholdern systematisch Rückmeldung einholen.

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.

LZ 1-7: Kurz- und langfristige Ziele differenzieren (R1)

Softwarearchitekt:innen können:

- langfristige Qualitätsanforderungen sowie deren Abgrenzung gegen (kurzfristige) Projektziele erklären
- Potentielle Zielkonflikte zwischen kurz- und langfristigen Zielen erklären, um eine für alle Beteiligten tragfähige Lösung zu erarbeiten
- Qualitätsanforderungen identifizieren und präzisieren

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

- G.Starke (April 2019): changed order of items. Removed "Architecture goals", added "specify quality requirements" - fixing both issue #33 and issue 11.
- P.Ghadir (April 2019): add "handle conflicts between short-term and long-term goals", see issue #33
- GS/CL (Jan 2019): Begriffe "Architektur- und Projektziele" durch kurz- und langfristige Ziele ersetzt. Formulierung vereinfacht.
- RR: Removed redundancies, especially with LG 2-3

LZ 1-8: Explizite von impliziten Aussagen unterscheiden (R1)

Softwarearchitekt:innen:

- können Annahmen oder Voraussetzungen explizit darstellen und dadurch implizite Annahmen vermeiden

- wissen, dass implizite Annahmen potentielle Missverständnisse zwischen beteiligten Stakeholdern bewirken
- können implizit formulieren, wenn es im gegebenen Kontext angemessen ist

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

Software architects:

- should explicitly present assumptions or prerequisites, therefore avoiding implicit assumptions
- know that implicit assumptions can lead to potential misunderstandings between stakeholders
- can formulate implicitly, if it is appropriate in the given context

LZ 1-9: Zuständigkeit von Softwarearchitekt:innen in organisatorischen Kontext einordnen (R3)

Der Fokus des iSAQB CPSA-Foundation Level liegt auf Strukturen und Konzepten einzelner Softwaresysteme.

Darüber hinaus kennen Softwarearchitekt:innen weitere Architekturdomänen, beispielsweise:

- Unternehmens-IT-Architektur (*Enterprise IT Architecture*): Struktur von Anwendungslandschaften
- Geschäfts- bzw. Prozessarchitektur (*Business and Process Architecture*): Struktur von u.a. Geschäftsprozessen
- Informationsarchitektur: systemübergreifende Struktur und Nutzung von Information und Daten
- Infrastruktur- bzw. Technologiearchitektur: Struktur der technischen Infrastruktur, Hardware, Netze etc.
- Hardware- oder Prozessorarchitektur (für hardwarenahe Systeme)

Diese Architekturdomänen sind nicht inhaltlicher Fokus vom CPSA-F.

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. GS, May 31st 2019: The previous wording was difficult to understand, so I rephrased a bit. See issue #53

LZ 1-10: Typen von IT-Systemen unterscheiden (R3)

Softwarearchitekt:innen kennen unterschiedliche Typen von IT-Systemen, beispielsweise:

- Informationssysteme
- Decision-Support, Data-Warehouse oder Business-Intelligence Systeme
- Mobile Systeme
- Batchprozesse oder -systeme
- hardwarenahe Systeme; hier verstehen sie die Notwendigkeit des Hardware-/Software-Codesigns (zeitliche und inhaltliche Abhängigkeiten von Hard- und Softwareentwurf).

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)

April 2019: Roger Rhoades proposed to maybe remove this LG, see issue 12.

LZ 1-11: Herausforderungen verteilter Systeme (R3)

Softwarearchitekt:innen können:

- die Verteilung in einer gegebenen Software-Architektur identifizieren
- Konsistenzkriterien für ein gegebenes fachliches Problem analysieren
- Kausalität von Ereignissen in einem verteilten System erklären

Softwarearchitekt:innen wissen:

- dass Kommunikation in einem verteilten System fehlschlagen kann
- dass es bei verteilten Systemen Einschränkungen hinsichtlich der Konsistenz in Datenbanken gibt
- was das "Split-Brain"-Problem ist und warum es schwierig zu lösen ist
- dass es unmöglich ist, die exakte zeitliche Reihenfolge der Ereignisse in einem verteilten System zu bestimmen ==== LG 1-11: 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

September 2020: Mike Sperber proposed to add this LG, see issue 148.

References

[Bass+ 2012], [Gharbi+2020], [iSAQB References], [Starke 2020], [vanSteen+Tanenbaum]

3. Entwurf und Entwicklung von Softwarearchitekturen

Dauer: 330 Min.	Übungszeit: 90 Min.

Wesentliche Begriffe

Entwurf; Vorgehen beim Entwurf; Entwurfsentscheidung; Sichten; Schnittstellen; technische Konzepte und Querschnittskonzepte; Architekturmuster; Entwurfsmuster; Mustersprachen; Entwurfsprinzipien; Abhängigkeit; Kopplung; Kohäsion; fachliche und technische Architekturen; Top-down- und Bottom-Up-Vorgehen; modellbasierter Entwurf; iterativer/inkrementeller Entwurf; Domain-Driven Design

4. Design and development of software architectures

Duration: 330 min.	Excercises: 90 min.

Relevant terms

Design; design approach; design decision; views; interfaces; technical and cross-cutting concepts; 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

LZ 2-1: Vorgehen und Heuristiken zur Architekturentwicklung auswählen und anwenden können (R1,R3)

Softwarearchitekt:innen können grundlegende Vorgehensweisen der Architekturentwicklung benennen, erklären und anwenden, beispielsweise:

- Top-down- und Bottom-Up-Vorgehen beim Entwurf (R1)
- Sichtenbasierte Architekturentwicklung (R1)
- iterativer und inkrementeller Entwurf (R1)
  - Notwendigkeit von Iterationen, insbesondere bei unter Unsicherheit getroffenen Entscheidungen (R1)
  - Notwendigkeit von Rückmeldungen zu Entwurfsentscheidungen (R1)
- Domain-Driven Design, siehe [Evans 2004] (R3)
- Evolutionäre Architektur, siehe [Ford 2017] (R3)
- Globale Analyse, siehe [Hofmeister et. al 1999] (R3)
- Modellgetriebene Architektur (R3)

LG 2-1: 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 (R1)
- view-based architecture development (R1)
- iterative and incremental design (R1)
  - necessity of iterations, especially when decision-making is affected by uncertainties (R1)
  - necessity of feedback on design decisions (R1)
- domain-driven design, see [Evans 2004] (R3)
- evolutionary architecture, see [Ford 2017] (R3)
- global analysis, see [Hofmeister et. al 1999] (R3)
- model-driven architecture (R3)

- GS (April 2019) incorporated suggestion by Roger Rhoades from issue 13 to lower priority of MDA and DDD
- GS (Feb. 2019): re-ordered approaches, re-phrased *iterative and incremental*

LZ 2-2: Softwarearchitekturen entwerfen (R1)

Softwarearchitekt:innen können:

- Softwarearchitekturen auf Basis bekannter funktionaler und Qualitätsanforderungen für nicht sicherheits- oder unternehmenskritische Softwaresysteme entwerfen und angemessen kommunizieren und dokumentieren
- Strukturentscheidungen hinsichtlich Systemzerlegung und Bausteinstruktur treffen, dabei Abhängigkeiten zwischen Bausteinen festlegen
- gegenseitige Abhängigkeiten und Abwägungen bezüglich Entwurfsentscheidungen erkennen und begründen
- Begriffe *Blackbox* und *Whitebox* erklären und zielgerichtet anwenden
- schrittweise Verfeinerung und Spezifikation von Bausteinen durchführen
- Architektursichten entwerfen, insbesondere Baustein-, Laufzeit- und Verteilungssicht
- die aus diesen Entscheidungen resultierenden Konsequenzen auf den Quellcode erklären
- fachliche und technische Bestandteile in Architekturen trennen und diese Trennung begründen
- Risiken von Entwurfsentscheidungen identifizieren.

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.

- RR: (Oct 2020): added hyphen when using "building block" as an adjective
- GS (May 2019): fixed typo in "Effizienz" (DE)
- GS (April 2019): fixed issue 14 by adjusting translation.
- GS (Feb 2019): re-ordered bullets in first paragraph, minor rephrasing
- GS/CL (Jan 2019): diverse detaillierte DDD-Konstrukte entfernt
- RR: Removed text relating to decisions. This is covered in LG 3-8

LZ 2-3: Einflussfaktoren auf Softwarearchitektur erheben und berücksichtigen können (R1-R3)

Softwarearchitekt:innen können Einflussfaktoren (Randbedingungen) als Einschränkungen der Entwurfsfreiheit erarbeiten und berücksichtigen. Sie verstehen, dass ihre Entscheidungen weitere Anforderungen und Einschränkungen für das zu entwerfende System, seine Architektur oder den Entwicklungsprozess mit sich bringen können.

Sie erkennen und berücksichtigen den Einfluss von:

- produktbezogenen Faktoren wie (R1)
  - funktionale Anforderungen
  - Qualitätsanforderungen und Qualitätsziele
  - zusätzliche Faktoren wie Produktkosten, beabsichtigtes Lizenzmodell oder Geschäftsmodell des Systems
- technischen Faktoren wie (R1-R3)
  - extern beauftragte technische Entscheidungen und Konzepte (R1)
  - bestehende oder geplante Hardware- und Software-Infrastruktur (R1)
  - technologische Beschränkungen für Datenstrukturen und Schnittstellen (R2)
  - Referenzarchitekturen, Bibliotheken, Komponenten und Frameworks (R1)
  - Programmiersprachen (R3)
- organisatorischen Faktoren wie
  - Organisationsstruktur des Entwicklungsteams und des Kunden (R1)
  - Unternehmens- und Teamkultur (R3)
  - Partnerschaften und Kooperationen (R2)
  - Normen, Richtlinien und Prozessmodelle (z.B. Genehmigungs- und Freigabeprozesse) (R2)
  - Verfügbarkeit von Ressourcen wie Budget, Zeit und Personal (R1)
  - Verfügbarkeit, Qualifikation und Engagement von Mitarbeitenden (R1)
- regulatorischen Faktoren wie (R2)
  - lokale und internationale rechtliche Einschränkungen
  - Vertrags- und Haftungsfragen
  - Datenschutzgesetze und Gesetze zum Schutz der Privatsphäre
  - Fragen der Einhaltung oder Verpflichtungen zur Beweislast

- Trends wie (R3)
  - Markttrends
  - Technologietrends (z.B. Blockchain, Microservices)
  - Methodik-Trends (z.B. agil)
  - (potenzielle) Auswirkungen weiterer Stakeholderinteressen und vorgegebener oder extern festgelegter Designentscheidungen (R3) ==== LG 2-3: Identify and consider factors influencing software architecture (R1-R3)

Software architects are able to gather and consider constraints and influencing factors that restrict their decisions. They understand that their decisions may imply further requirements and constraints on the system being designed, its architecture, or the development process. (R1-R2)

They should recognize and account for the impact of:

- product-related factors such as (R1)
  - functional requirements
  - quality requirements and quality goals
  - additional factors such as product cost, intended licensing model, or business model of the system
- technological factors such as (R1-R3)
  - externally mandated technical decisions and concepts (R1)
  - 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 (R3)
- organizational factors such as
  - organizational structure of the development team and of the customer (R1)
  - 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 factors 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. blockchain, microservices)
- methodology trends (e.g. agile)
- (potential) impact of further stakeholder concerns and mandated design decisions (R3)

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

GS (May 31st 2019): fixed punctuation, added R2 to last sentence in DE version RR: Stakeholder sind neu.

LZ 2-4: Querschnittskonzepte entwerfen und umsetzen (R1)

Softwarearchitekt:innen können:

- die Bedeutung von Querschnittskonzepten erklären
- Querschnittskonzepte entscheiden und entwerfen, beispielsweise Persistenz, Kommunikation, GUI, Fehlerbehandlung, Nebenläufigkeit
- mögliche wechselseitige Abhängigkeiten dieser Entscheidungen erkennen und beurteilen.

Softwarearchitekt:innen wissen, dass solche Querschnittskonzepte systemübergreifend wiederverwendbar sein können.

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

Software architects are able to:

- explain the significance of such cross-cutting concepts
- decide on and design cross-cutting concepts, 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.

LZ 2-5: Wichtige Lösungsmuster beschreiben, erklären und angemessen anwenden (R1, R3)

Softwarearchitekt:innen kennen verschiedene Architekturmuster (siehe unten) und können sie gegebenenfalls anwenden.

Sie wissen (R3):

- dass Muster ein Weg sind, bestimmte Qualitäten für gegebene Probleme und Anforderungen innerhalb gegebener Kontexte zu erreichen
- dass es verschiedene Kategorien von Mustern gibt
- zusätzliche Quellen für Muster, die sich auf ihre spezifische technische oder Anwendungsdomäne beziehen

Softwarearchitekt:innen können die folgenden Muster erklären und Beispiele dafür liefern (R1):

- Schicht (Layers):
  - Abstraktionsschichten (Abstraction layers) verbergen Details, Beispiel: ISO/OSI-Netzwerkschichten oder "Hardware-Abstraktionsschicht". Siehe https://en.wikipedia.org/wiki/Hardware_abstraction.
  - Eine andere Interpretation sind Schichten zur (physischen) Trennung von Funktionalität oder Verantwortung, siehe https://en.wikipedia.org/wiki/Multitier_architecture.
- Pipes-and-Filter: Repräsentativ für Datenflussmuster, die die schrittweise Verarbeitung in eine Reihe von Verarbeitungsaktivitäten ("Filter") und zugehörige Transport/Puffer ("Pipes") separieren.
- Microservices teilen Anwendungen in separate ausführbare Diensteauf, die über Netzwerk (remote) kommunizieren.
- Dependency Injection als eine mögliche Lösung für das Dependency-Inversion-Prinzip

Softwarearchitekt:innen können einige der folgendene Muster erklären, ihre Relevanz für konkrete Systeme erläutern und Beispiele dafür liefern (R3):

- Blackboard: Behandlung von Problemen, die nicht durch deterministische Algorithmen lösbar sind, sondern vielfältiges Wissen erfordern.
- Broker: verantwortlich für die Koordination der Kommunikation zwischen Anbieter(n) und Verbraucher(n), angewandt in verteilten Systemen. Verantwortlich für die Weiterleitung von Anfragen und/oder die Übermittlung von Ergebnissen, Fehlern und Ausnahmen.
- Kombinator (Synonym: Closure of Operations), für Domänenobjekte vom Typ T, suchen Sie nach Operationen sowohl mit Input- als auch Output-Typ T. Siehe [Yorgey 2012]
- CQRS (Command-Query-Responsibility-Segregation): Trennung von Lese- und Schreibvorgängen in Informationssystemen. Erfordert Einblicke in konkrete Datenbank-/Persistenztechnologie, um die unterschiedlichen Eigenschaften und Anforderungen von "Lese-" und "Schreib"-Operationen zu verstehen.
- Event-Sourcing: Behandlung von Datenoperationen durch eine Abfolge von Ereignissen (Events), von denen jedes in einem Append-only Speicher aufgezeichnet wird.
- Interpreter: repräsentieren Domänenobjekt oder DSL als Syntax, bieten eine Funktion, die eine semantische Interpretation des Domänenobjekts getrennt vom Domänenobjekt selbst implementiert.
- Integrations- oder Messaging-Patterns (z.B. aus Hohpe+2004])
- Die MVC-, MVVM-, MV-Update-, PAC-Musterfamilie, die die externe Repräsentation (Ansicht) von Daten von Operationen Diensten und deren Koordination trennt .
- Schnittstellenmuster wie Adapter, Fassade, Proxy. Solche Muster helfen bei der Integration von Subsystemen und/oder bei der Vereinfachung von Abhängigkeiten. Architekt:innen sollten wissen, dass diese Muster unabhängig von (Objekt-)Technologie verwendet werden können.
  - Adapter: Entkopplung von Konsument und Provider - wenn die Schnittstelle des Providers nicht genau mit der des Konsumenten übereinstimmt.
  - Fassade: vereinfacht die Verwendung eines Providers für den/die Consumer durch vereinfachten Zugriff.
  - Proxy: Ein Vermittler/Stellvertreter zwischen Consumer und Provider, der beispielsweise die zeitliche Entkopplung, das Caching von Ergebnissen oder die Zugriffkontrolle auf den Provider ermöglicht.
- Observer (Beobachter): ein Produzent von Werten benachrichtigt eine zentrale Vermittlungsstelle, bei

der sich Interessenten (Consumer) registrieren können, um über Änderungen benachrichtigt zu werden.

- Plug-In: erweitert das Verhalten einer Komponente.
- Ports&Adapters (syn. Onion-Architecture, Hexagonale Architektur): konzentrieren die Domänenlogik im Zentrum des Systems, und besitzen lediglich an den Rändern Verbindungen zur Außenwelt (Datenbank, UI). Abhängigkeiten von aussen nach innen (Outside-In), niemals von innen nach aussen (Inside-Out).
- Remote Procedure Call: eine Funktion oder einen Algorithmus in einem anderen Adressraum ausführen lassen.
- SOA: Service-orientierte Architektur. Ein Ansatz zur Bereitstellung abstrakter Dienste statt konkreter Implementierungen für die Benutzer des Systems, um die Wiederverwendung von Diensten über Abteilungen und zwischen Unternehmen zu fördern.
- Template und Strategy: spezifische Algorithmen durch Kapselung flexibel machen.
- Visitor (Besucher): Traversierung von Datenstrukturen von spezifischer Verarbeitung trennen.

Softwarearchitekt:innen kennen wesentliche Quellen für Architekturmuster, beispielsweise die POSA-Literatur (z.B. [Buschmann+ 1996]) und PoEAA ([Fowler 2003]) (für Informationssysteme) (R3)

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

Software architects know:

- various architectural patterns and can apply them when appropriate
- that patterns are a way to achieve certain qualities for given problems and requirements within given contexts
- that various categories of patterns exist (R3)
- additional sources for patterns related to their specific technical or application domain (R3)

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

- Layers:
  - Abstraction layers hide details, example: ISO/OSI network layers, or "hardware abstraction layer". See https://en.wikipedia.org/wiki/Hardware_abstraction
  - Another interpretation are Layers to (physically) separate functionality or responsibility, see https://en.wikipedia.org/wiki/Multitier_architecture
- Pipes-and-Filter: Representative for data flow patterns, breaking down stepwise processing into a series of processing-activities ("Filter") and associated data transport/buffering capabilities ("Pipes").
- Microservices split application into separate executable that communicate via network
- Dependency Injection as a possible solution for the Dependency-Inversion-Principle

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

- Blackboard: handle problems that cannot be solved by deterministic algorithms but require diverse knowledge

- Broker: responsible for coordinating communication between provider(s) and consumer(s), applied in distributed systems. Responsible for forwarding requests and/or transmitting results and exceptions
- Combinator (synonym: closure of operations), for domain object of type T, look for operations with both input and output type T. See [Yorgey 2012]
- CQRS (Command-Query-Responsibility-Segregation): Separates read- from write concerns in information systems. Requires some context on database-/persistence technology to understand the different properties and requirements of "read" versus "write" operations
- Event-Sourcing: handle operations on data by a sequence of events, each of which is recorded in an append-only store
- Interpreter: represent domain object or DSL as syntax, provide function implementing a semantic interpretation of domain object separately from domain object itself
- Integration and messaging patterns (e.g. from Hohpe+2004])
- The MVC, MVVM, MV-Update, PAC family of patterns, separating external representation (view) from data, services and their coordination
- Interfacing-patterns like Adapter, Facade, Proxy. Such patterns help in integration of subsystems and/or simplification of dependencies. Architects should know that these patterns can be used independent of (object) technology
    - Adapter: decouple consumer and provider - where the interface of the provider does not exactly match that of the consumer. The Adapter decouples one party from interface-changes in the other
    - Facade: simplifies usage of a provider for consumer(s) by providing simplified access
    - Proxy: An intermediate between consumer and provider, enabling temporal decoupling, caching of results, controlling access to the provider etc.
- Observer: a producer of values over time notifies a central switchboards where consumers can register to be notified of them
- Plug-In: extend the behaviour of a component
- Ports&Adapters (syn. Onion-Architecture, Hexagonal-Architecture): concentrate domain logic in the center of the system, have connections to the outside world (database, UI) at the edges, dependencies only outside-in, never inside-out
- Remote Procedure Call: make a function or algorithm execute in a different address space
- SOA: Service-Oriented Architecture. An approach to provide abstract services rather than concrete implementations to users of the system to promote reuse of services across departments and between companies
- Template and Strategy: make specific algorithms flexible by encapsulating them
- Visitor: separate data-structure traversal from specific processing

Software architects know essential sources for architectural patterns, such as POSA (e.g. [Buschmann+ 1996]) and PoEAA ([Fowler 2003]) (for information systems) (R3).

GS (August 2019): Added reference for PoEAA, replaced POSA by Buschmann-references GS (Feb 2019): re-ordered to match priorities.

GS/CL (Jan 2019): significantly shortenend, removed "styles" → followed RRs' suggestion below.

RR: I've seen people differentiate between architecture styles and patterns. I don't think the difference is important enough to differentiate. I do make a difference between architecture patterns and design patterns, but it's not always clear if a pattern is an architecture pattern, a design pattern, or both.

I suggest using "architecture-level design patterns" or simply "architecture and design patterns". – or simply "architecture pattern" / "Architekturmuster"

LZ 2-6: Entwurfsprinzipien erläutern und anwenden (R1-R3)

Softwarearchitekt:innen sind in der Lage zu erklären, was Entwurfsprinzipien sind. Sie können deren grundlegende Ziele und deren Anwendung im Hinblick auf Softwarearchitektur skizzieren. (R2)

Mit einer Prüfungsrelevanz, die jeweils von dem unten aufgeführten konkreten Prinzip abhängt, sind Softwarearchitekt:innen in der Lage:

- die unten aufgeführten Gestaltungsprinzipien zu erläutern und mit Beispielen zu illustrieren
- zu erklären, wie diese Prinzipien angewendet werden sollen
- darzulegen, wie Qualitätsanforderungen die Anwendung dieser Prinzipien beeinflussen
- die Auswirkungen der Entwurfsprinzipien auf die Implementierung zu erläutern
- Quellcode- und Architekturentwürfe zu analysieren, um zu beurteilen, ob diese Entwurfsprinzipien angewendet wurden oder angewendet werden sollten

Abstraktion (R1)

- im Sinne eines Vorgehens zur Erarbeitung zweckmäßiger Generalisierungen
- als ein Enwurfskonstrukt, bei dem die Bausteine von Abstraktionen und nicht von Implementierungen abhängen
- Schnittstellen als Abstraktionen

Modularisierung (R1)

- Geheimnisprinzip (Information Hiding) und Kapselung (R1)
- Trennung von Verantwortlichkeiten (Separation of Concerns - SoC) (R1)
- Lose, aber funktionell ausreichende Kopplung (R1) von Bausteinen, siehe LG 2-7
- Hohe Kohäsion (R1)
- SOLID-Prinzipien (R1-R3), soweit sie auf architektonischer Ebene von Relevanz sind:
  - S: Single-Responsibility-Prinzip (R1) und seine Beziehung zu SoC
  - O: Offen/geschlossen-Prinzip (R1)

- L: Liskovsches Substitutionsprinzip (R3) als eine Möglichkeit, Konsistenz und konzeptionelle Integrität beim objektorientierten Design zu erreichen
- I: Interface-Segregation-Prinzip (R2) und seine Beziehung zu Lernziel 2-9 "Schnittstellen entwerfen und festlegen"
- D: Dependency-Inversion-Prinzip (R1) - Umkehrung von Abhängigkeiten (R1) durch Schnittstellen oder ähnlichen Abstraktionen

Konzeptionelle Integrität (R2)

- bedeutet Einheitlichkeit (Homogenität, Konsistenz) von Lösungen für ähnliche Probleme zu erreichen (R2)
- als ein Mittel um Prinzips der geringsten Überraschung zu erreichen (principle of least surprise) (R3)

Einfachheit (R1)

- als Mittel zur Verringerung von Komplexität (R1)
- als Motiv der Prinzipien KISS (R1) und YAGNI (R2)

Erwarte Fehler (R1-R2)

- als Mittel für den Entwurf robuster und widerstandsfähiger Systeme (R1)
- als eine Verallgemeinerung des Robustheitsgrundsatzes (Postel's law) (R2)

LG 2-6: Explain and use design principles (R1-R3)

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

With relevance for the examination depending on the specific principle listed below, 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 the quality 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 (R1)

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

Modularization (R1-R3)

- information hiding and encapsulation (R1)
- separation of concerns - SoC (R1)
- loose, but functionally sufficient, coupling (R1) of building blocks, see LG 2-7
- high cohesion (R1)
- SOLID principles (R1-R3), which have, to a certain extent, relevance at the architectural level
  - S: Single responsibility principle (R1) and its relation to SoC
  - O: Open/closed principle (R1)
  - L: Liskov substitution principle (R3) as a way to achieve consistency and conceptual integrity in OO design
  - I: Interface segregation principle (R2), including its relation to LG 2-9
  - D: Dependency inversion principle (R1) by means of interfaces or similar abstractions

Conceptual integrity (R2)

- meaning uniformity (homogeneity, consistency) of solutions for similar problems (R2)
- as a means to achieve the principle of least surprise (R3)

Simplicity (R1-R2)

- as a means to reduce complexity (R1)
- as the driving factor behind KISS (R1) and YAGNI (R2)

Expect Errors (R1-R2)

- as a means to design for robust and resilient systems (R1)
- as a generalization of the robustness principle aka Postel's law (R2)

- GS, May 2019: moved *dependency injection* to LG-02-07
- GS, Feb. 2019: fixed wrong indentation for modularization (SRP and OCP are principles on their own). Added links to leaning goals. Adjusted intro sentence of EN version.
- CL/GS, Jan 2019: Minor rephrasing (DE version)

LZ 2-7: Abhängigkeiten von Bausteinen managen (R1)

Softwarearchitekt:innen verstehen Abhängigkeiten und Kopplung zwischen Bausteinen und können diese gezielt einsetzen. Sie:

- kennen und verstehen unterschiedliche Arten der Kopplung von Bausteinen (beispielsweise Kopplung über Benutzung/Delegation, Nachrichten/Ereignisse, Komposition, Erzeugung, Vererbung, zeitliche Kopplung, Kopplung über Daten, Datentypen oder Hardware)
- verstehen, wie Abhängigkeiten die Kopplung vergrößern
- können solche Arten der Kopplung gezielt einsetzen und die Konsequenzen solcher Abhängigkeiten einschätzen

- kennen Möglichkeiten zur Auflösung bzw. Reduktion von Kopplung und können diese anwenden, beispielsweise:
  - Muster (siehe LZ 2-5)
  - Grundlegende Entwurfsprinzipien (siehe LZ 2-6)
  - Externalisierung von Abhängigkeiten, d.h. konkrete Abhängigkeiten erst zur Installations- oder Laufzeit festlegen, etwa durch Anwendung von *Dependency Injection*.

LG 2-7: 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 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 LG 2-5)
  - Fundamental design principles (see LG 2-6)
  - Externalization of dependencies, i.e. defining concrete dependencies at installation- or runtime, for example by using *Dependency Injection*.

GS, Feb 2019: re-formated DE version, adjusted EN translation to modifications of DE version

LZ 2-8: Qualitätsanforderungen mit passenden Ansätzen und Techniken erreichen (R1)

Softwarearchitekt:innen kennen und berücksichtigen den starken Einfluss von Qualitätsanforderungen in Architektur- und Entwurfsentscheidungen, beispielsweise für:

- Effizienz, Laufzeitperformance
- Verfügbarkeit
- Wartbarkeit, Modifizierbarkeit, Erweiterbarkeit, Adaptierbarkeit

Sie können:

- Lösungsmöglichkeiten, *Architectural Tactics*, angemessene Praktiken sowie technische Möglichkeiten zur Erreichung wichtiger Qualitätsanforderungen von Softwaresystemen (unterschiedlich für eingebettete Systeme bzw. Informationssysteme) erklären und anwenden
- mögliche Wechselwirkungen zwischen solchen Lösungsmöglichkeiten sowie die entsprechenden Risiken identifizieren und kommunizieren.

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, runtime performance
- availability
- maintainability, modifiability, extensibility, adaptability

They can:

- explain and apply solution options, *Architectural 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

RER: Juni 2019: Redundanzen zwischen LZ 2-2 und 2-8 eliminiert GS: Feb 2019: Ich bin der Meinung, dass dieses LZ ins Kapitel 2 (statt bisher 5) des Lehrplanes gehört, und habe es zum neuen LZ 2-8 gemacht

LZ 2-9: Schnittstellen entwerfen und festlegen (R1-R3)

Softwarearchitekt:innen kennen die hohe Bedeutung von Schnittstellen. Sie können Schnittstellen zwischen Architekturbausteinen sowie externe Schnittstellen zwischen dem System und Elementen außerhalb des Systems entwerfen bzw. festlegen.

Sie kennen:

- wünschenswerte Eigenschaften von Schnittstellen und können diese beim Entwurf einsetzen:
  - einfach zu erlernen, einfach zu benutzen, einfach zu erweitern
  - schwer zu missbrauchen
  - funktional vollständig aus Sicht der Nutzer:innen oder nutzender Bausteine.
- die Notwendigkeit unterschiedlicher Behandlung interner und externer Schnittstellen
- unterschiedliche Implementierungsansätze von Schnittstellen (R3):
  - ressourcenorientierter Ansatz (REST, REpresentational State Transfer)
  - serviceorientierter Ansatz (wie bei WS-*/SOAP-basierten Webservices).

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.

GS (June 2019): re-ordered sub-goals to have R1 before R3.

GS (Feb 2019): Added the differentiation between external and internal interfaces.

GS/CL (Jan 2019): Nachsatz zu Dokumentation entfernt, der gehört in Kapitel 3

References

[Bass+ 2012], [Fowler 2003], [Gharbi+2020], [Gamma+94], [Martin 2003], [Buschmann+ 1996], [Buschmann+ 2007], [Starke 2020], [Lilienthal 2018]

5. Beschreibung und Kommunikation von Softwarearchitekturen

Dauer: 180 Min.	Übungszeit: 60 Min.

Wesentliche Begriffe

(Architektur-)Sichten; Strukturen; (technische) Konzepte; Dokumentation; Kommunikation; Beschreibung; zielgruppen- oder stakeholdergerecht; Meta-Strukturen und Templates zur Beschreibung und Kommunikation; Kontextabgrenzung; Bausteine; Bausteinsicht; Laufzeitsicht; Verteilungssicht; Knoten; Kanal; Verteilungsartefakte; Mapping von Bausteinen auf Verteilungsartefakte; Beschreibung von Schnittstellen und Entwurfsentscheidungen; UML; Werkzeuge zur Dokumentation

6. Specification and communication of software architectures

Duration: 180 min.	Exercises: 60 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

GS (Feb. 2019): Why do we have ";" semicolons in EN and comma in DE?

Learning goals

LZ 3-1: Qualitätsmerkmale technischer Dokumentation erläutern und berücksichtigen (R1)

Softwarearchitekt:innen kennen die wesentlichen Qualitätsmerkmale technischer Dokumentation und können diese bei der Dokumentation von Systemen berücksichtigen bzw. erfüllen:

- Verständlichkeit, Korrektheit, Effizienz, Angemessenheit, Wartbarkeit
- Orientierung von Form, Inhalt und Detailgrad an Zielgruppe der Dokumentation

Sie wissen, dass Verständlichkeit technischer Dokumentation nur von deren Zielgruppen beurteilt werden kann.

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.

GS: *Leser* durch *Zielgruppe* ersetzt

LZ 3-2: Softwarearchitekturen beschreiben und kommunizieren (R1)

Softwarearchitekt:innen:

- können Architekturen stakeholdergerecht dokumentieren und kommunizieren und dadurch unterschiedliche Zielgruppen adressieren, z. B. Management, Entwicklungsteams, QS, andere Softwarearchitekt:innen sowie möglicherweise zusätzliche Stakeholder
- können die Beiträge unterschiedlicher Autorengruppen stilistisch und inhaltlich konsolidieren und harmonisieren

- kennen den Nutzen von Template-basierter Dokumentation.

Softwarearchitekt:innen wissen, dass verschiedene Eigenschaften der Dokumentation von Spezifika des Systems, seinen Anforderungen, Risiken, dem Entwicklungsvorgehen, der Organisation oder anderen Faktoren abhängen.

Diese Faktoren beeinflussen beispielsweise:

- ob der schriftlichen oder mündlichen Kommunikation Vorrang eingeräumt werden sollte
- Umfang und Detaillierungsgrad der in jeder Entwicklungsphase benötigten Dokumentation
- das Dokumentationsformat
- die Zugänglichkeit der Dokumentation
- Formalitäten der Dokumentation (z.B. Diagramme, die einem Metamodell entsprechen, oder einfache Zeichnungen)
- formale Überprüfungen und Freigabeprozesse für die Dokumentation

Softwarearchitekt:innen sind sich dieser Faktoren bewusst und können die Dokumentationsmerkmale je nach Situation anpassen.

LG 3-2: Describe and communicate software architectures (R1,R3)

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

Software architects know that various properties of documentation depend on specifics of the system, its requirements, risks, development process, organization or other factors.

These factors impact:

- whether written or verbal communication should be prioritized
- the amount and level of detail of documentation needed at each stage of development
- the documentation format
- the accessibility to the documentation
- formality of documentation (e.g. diagrams compliant to a meta model or simple drawings)
- formal reviews and sign-off processes for documentation

Software architects are aware of these factors and can adjust the documentation characteristics according to the situation.

GS: Layout, sprachlich vereinfacht, *Schablone* entfernt

LZ 3-3: Notations-/Modellierungsmittel für Beschreibung von Softwarearchitektur erläutern und anwenden (R2-R3)

Softwarearchitekt:innen kennen mindestens folgende UML-Diagramme (siehe [UML]) zur Notation von Architektursichten:

- Klassen-, Paket-, Komponenten- (jeweils R2) und Kompositionsstrukturdiagramme (R3)
- Verteilungsdiagramme (R2)
- Sequenz- und Aktivitätsdiagramme (R2)
- Zustandsdiagramme (R3)

Softwarearchitekt:innen kennen Alternativen zu UML, beispielsweise (R3)

- ArchiMate, siehe [Archimate]
- für Laufzeitsichten beispielsweise Flussdiagramme, nummerierte Listen oder Business-Process-Modelling-Notation (BPMN).

LG 3-3: Explain and apply notations/models to describe software architecture (R2-R3)

Software architects know at least the following UML (see [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, see [Archimate]
- for runtime views for example flow charts, numbered lists or business-process-modeling-notation (BPMN).

LZ 3-4: Architektursichten erläutern und anwenden (R1)

Softwarearchitekt:innen können folgende Architektursichten anwenden:

- Kontextsicht (auch genannt Kontextabgrenzung)
- Baustein- oder Komponentensicht (Aufbau des Systems aus Softwarebausteinen)
- Laufzeitsicht (dynamische Sicht, Zusammenwirken der Softwarebausteine zur Laufzeit, Zustandsmodelle)
- Verteilungs-/Deploymentsicht (Hardware und technische Infrastruktur sowie Abbildung von Softwarebausteinen auf diese Infrastruktur)

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)

LZ 3-5: Kontextabgrenzung von Systemen erläutern und anwenden (R1)

Softwarearchitekt:innen können:

- Kontext von Systemen z.B. in Form von Kontextdiagrammen mit Erläuterungen darstellen
- externe Schnittstellen von Systemen in der Kontextabgrenzung darstellen
- fachlichen und technischen Kontext differenzieren.

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

Software architects are able to:

- 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 business and technical context.

GS: Schnittstellen + Kontextdiagramm zugefügt - LZ war sonst so "leer"

LZ 3-6: Querschnittskonzepte dokumentieren und kommunizieren (R2)

Softwarearchitekt:innen können typische Querschnittskonzepte (synonym *Prinzipien*, *Aspekte*) adäquat dokumentieren und kommunizieren, z. B. Persistenz, Ablaufsteuerung, UI, Verteilung/Integration, Protokollierung.

LG 3-6: Document and communicate cross-cutting concepts (R2)

Software architects are able to adequately document and communicate typical cross-cutting concepts (synonym: *principles*, *aspects*), e. g., persistence, workflow management, UI, deployment/integration, logging.

LZ 3-7: Schnittstellen beschreiben (R1)

Softwarearchitekt:innen können sowohl interne als auch externe Schnittstellen beschreiben und spezifizieren.

LG 3-7: Describe interfaces (R1)

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

GS/RR: Redundanz zu Kapitel 2 behoben.

LZ 3-8: Architekturentscheidungen erläutern und dokumentieren (R1-R2)

Softwarearchitekt:innen können:

- Architekturentscheidungen systematisch herbeiführen, begründen, kommunizieren und dokumentieren
- gegenseitige Abhängigkeiten solcher Entscheidungen erkennen, kommunizieren und dokumentieren

Softwarearchitekt:innen kennen Architecture-Decision-Records (ADR, siehe [Nygard 2011]) und können diese zur Dokumentation von Entscheidungen einsetzen (R2).

LG 3-8: Explain and document architectural decisions (R1-R2)

Software architects are able to:

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

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

GS (Juni 2019): "herleiten" durch "herbeiführen" ersetzt (engl: derive → take)

GS (Feb 2019): wording: += kommunizieren, begründen

LZ 3-9: Dokumentation als schriftliche Kommunikation nutzen (R2)

Softwarearchitekt:innen nutzen Dokumentation zur Unterstützung bei Entwurf, Implementierung und Weiterentwicklung (auch genannt *Wartung* oder *Evolution*) von Systemen.

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.

GS: Redundanz zu LZ 3-2 entfernt, *Weiterentwicklung* und Synonyme aufgenommen.

LZ 3-10: Weitere Hilfsmittel und Werkzeuge zur Dokumentation kennen (R3)

Softwarearchitekt:innen kennen:

- Grundlagen mehrerer publizierter Frameworks zur Beschreibung von Softwarearchitekturen, beispielsweise:
  - ISO/IEEE-42010 (vormals 1471), siehe [ISO 42010]
  - arc42, siehe [arc42]
  - C4, siehe [Brown]
  - FMC, siehe [FMC]
- Ideen und Beispiele von Checklisten für die Erstellung, Dokumentation und Prüfung von Softwarearchitekturen

- mögliche Werkzeuge zur Erstellung und Pflege von Architekturdokumentation

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), see [ISO 42010]
  - arc42, see [arc42]
  - C4, see [Brown]
  - FMC, see [FMC]
- ideas and examples of checklists for the creation, documentation, and testing of software architectures
- possible tools for creating and maintaining architectural documentation

References

[arc42], [Archimate], [Bass+ 2012], [Brown], [Clements+ 2010], [FMC], [Gharbi+2020], [Nygard 2011], [Starke 2020], [UML], [Zörner 2015]

7. Softwarearchitektur und Qualität

Dauer: 60 Min.	Übungszeit: 60 Min.

Wesentliche Begriffe

Qualität; Qualitätsmerkmale; DIN/ISO 25010; Qualitätsszenarien; Qualitätsbaum; Kompromisse/Wechselwirkungen von Qualitätseigenschaften; qualitative Architekturbewertung; Metriken und quantitative Bewertung

8. 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

LZ 4-1: Qualitätsmodelle und Qualitätsmerkmale diskutieren (R1)

Softwarearchitekt:innen können:

- den Begriff der Qualität (angelehnt an DIN/ISO 25010, vormals 9126) und der Qualitätsmerkmale erklären
- generische Qualitätsmodelle (wie etwa DIN/ISO 25010) erklären
- Zusammenhänge und Wechselwirkungen von Qualitätsmerkmalen erläutern, beispielsweise:
  - Konfigurierbarkeit versus Zuverlässigkeit
  - Speicherbedarf versus Leistungseffizienz
  - Sicherheit versus Benutzbarkeit
  - Laufzeitflexibilität versus Wartbarkeit.

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
- generic quality models (such as DIN/ISO 25010)
- correlations and trade-offs of quality characteristics, for example:
  - configurability versus reliability
  - memory requirements versus performance efficiency
  - security versus usability
  - runtime flexibility versus maintainability.

GS: Beispiele klarer formuliert

LZ 4-2: Qualitätsanforderungen an Softwarearchitekturen klären (R1)

Softwarearchitekt:innen können:

- spezifische Qualitätsanforderungen an die zu entwickelnde Software und deren Architekturen klären und konkret formulieren, beispielsweise in Form von Szenarien und Qualitätsbäumen

- Szenarien und Qualitätsbäume erklären und anwenden.

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.

RR: (March 2019) We don't (shouldn't) define (formulieren) quality requirements. This is the customer's job. We shouldn't define how fast, secure, etc. a system should be. See issue 15

LZ 4-3: Softwarearchitekturen qualitativ analysieren und bewerten (R2-R3)

Softwarearchitekt:innen:

- kennen methodische Vorgehensweisen zur qualitativen Analyse und Bewertung von Softwarearchitekturen (R2), beispielsweise nach ATAM (R3)
- können kleinere Systeme qualitativ analysieren und bewerten (R2)
- wissen, dass zur qualitativen Analyse und Bewertung von Architekturen folgende Informationsquellen helfen können (R2):
  - Qualitätsanforderungen, beispielsweise in Form von Qualitätsbäumen und -szenarien
  - Architekturdokumentation
  - Architektur- und Entwurfsmodelle
  - Quellcode
  - Metriken
  - Sonstige Dokumentationen des Systems, etwa Anforderungs-, Betriebs- oder Testdokumentation.

LG 4-3: Qualitative analysis and assessment 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.

- GS (June 2019): make relevance of sub-goals of LG-4-3 explicit, added other kinds of documentation
- GS: += *analysieren* im Titel, ATAM auf R3
- FLWG: ATAM nur noch exemplarisch
- RR: Vorgehen ist mir weniger wichtig als die Ziele und Ergebnisse von ATAM

LZ 4-4: Softwarearchitekturen quantitativ bewerten (R2)

Softwarearchitekt:innen kennen Ansätze zur quantitativen Analyse und Bewertung (Messung) von Software.

Sie wissen, dass:

- quantitative Bewertung helfen kann, kritische Teile innerhalb von Systemen zu identifizieren
- zur Bewertung von Architekturen weitere Informationen hilfreich sein können, etwa:
  - Anforderungs- und Architekturdokumentation
  - Quellcode und diesbezügliche Metriken wie Lines-of-Code, (zyklomatische) Komplexität, ein- und ausgehende Abhängigkeiten
  - bekannte Fehler in Quellcode, insbesondere Fehlercluster
  - Testfälle und Testergebnisse.

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.

GS: see issue 16

References

[Bass+ 2012], [Clements+ 2002], [Gharbi+2020], [Martin 2003], [Starke 2020]

9. Beispiele für Softwarearchitekturen

Dauer: 90 Min.	Übungszeit: Keine

Dieser Abschnitt ist nicht prüfungsrelevant.

10. Examples of software architectures

Duration: 90 min.	Exercises: none

This section is not relevant for for the exam.

Learning goals

LZ 5-1: Bezug von Anforderungen und Randbedingungen zu Lösung erfassen (R3)

Softwarearchitekt:innen haben an mindestens einem Beispiel den Bezug von Anforderungen und Randbedingungen zu Lösungsentscheidungen erkannt und nachvollzogen.

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.

LZ 5-2: Technische Umsetzung einer Lösung nachvollziehen (R3)

Softwarearchitekt:innen können anhand mindestens eines Beispiels die technische Umsetzung (Implementierung, technische Konzepte, eingesetzte Produkte, Lösungsstrategien) einer Lösung nachvollziehen.

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

- [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+ 2012] Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. 3rd Edition, Addison Wesley 2012.
- [Brown] Simon Brown: Brown, Simon: The C4 model for visualising software architecture. https://c4model.com https://www.infoq.com/articles/C4-architecture-model.
- [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
- [Evans 2004] Eric Evans: *Domain-Driven Design: Tackling Complexity in the Heart of Software,* Addison-Wesley, 2004.
- [FMC] Siegfried Wendt: Fundamental Modeling Concepts, online: http://www.fmc-modeling.org/
- [Ford 2017] Neil Ford, Rebecca Parsons, Patrick Kua: Building Evolutionary Architectures: Support Constant Change. OReilly 2017
- [Fowler 2003] Martin Fowler: Patterns of Enterprise Application Architecture. (PoEAA) Addison-Wesley, 2003.
- [Gharbi+2020] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 4. Auflage, dpunkt Verlag, Heidelberg 2020.
- [Geirhosk 2015] Matthias Geirhos. Entwurfsmuster: Das umfassende Handbuch (in German). Rheinwerk Computing Verlag. 2015 ISBN: 9783836227629
- [Gamma+94] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
- [Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java. Springer-Vieweg Verlag, 2. Auflage 2014.
- [Hofmeister et. al 1999] Christine Hofmeister, Robert Nord, Dilip Soni: *Applied Software Architecture*, Addison-Wesley, 1999
- [ISO 42010] ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description, online: http://www.iso-architecture.org/ieee-1471/
- [iSAQB References] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Freely available https://leanpub.com/isaqbreferences.
- [Keeling 2017] Michael Keeling. Design It!: From Programmer to Software Architect. Pragmatic Programmer. ISBN 978-1680502091.

- [Lilienthal 2018] Carola Lilienthal: Langlebige Softwarearchitekuren. 2. Auflage, dpunkt Verlag 2018.
- [Lilienthal 2019] Carola Lilienthal: Sustainable Software Architecture: Analyze and Reduce Technical Debt. dpunkt Verlag 2019.
- [Martin 2003] Robert Martin: Agile Software Development. Principles, Patterns, and Practices. Prentice Hall, 2003.
- [Martin 2017] Robert Martin. Clean Architecture: A craftsman's guide to software structure and design. MITP,
- [Miller et. al] Heather Miller, Nat Dempkowski, James Larisch, Christopher Meiklejohn: Distributed Programming (to appear, but content-complete) https://github.com/heathermiller/dist-prog-book.
- [Newman 2015] Sam Newman. Building Microservices: Designing Fine-Grained Systems. O'Reilly. 2015. ISBN 9781491950357.
- [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.
- [Starke 2020] Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden (in German). 9. Auflage, Carl Hanser Verlag 2020. Website: https://esabuch.de
- [Eilebrecht+2019] Karl Eilebrecht, Gernot Starke: Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung (in German). 5th Edition Springer Verlag 2019.
- [UML] The UML reading room, collection of UML resources https://www.omg.org/technology/readingroom/UML.htm. See also https://www.uml-diagrams.org/.
- [vanSteen+Tanenbaum] Andrew Tanenbaum, Maarten van Steen: Distributed Systems, Principles and Paradigms. https://www.distributed-systems.net/.
- [Yorgey 2012] Brent A. Yorgey, Proceedings of the 2012 Haskell Symposium, September 2012 https://doi.org/10.1145/2364506.2364520
- [Zörner 2015] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. 2. Auflage, Carl Hanser Verlag 2015.