

CPSA Certified Professional for Software Architecture®

- Foundation Level Curriculum -

Version V5.0
-DE;EN, 26. Aug 2019



Table of Contents

Rechtliches	2
Legal Notice	3
Introduction	5
Was vermittelt eine Foundation-Level-Schulung?	5
What does a Foundation Level training convey?	5
Abgrenzung	7
Out of scope	7
Voraussetzungen	8
Prerequisites	8
Struktur, Dauer, Didaktik	10
Structure, duration and teaching methods	10
Lernziele und Prüfungsrelevanz	12
Learning goals and relevance for the examination	12
1. Grundbegriffe	14
Wesentliche Begriffe	14
2. Basic concepts of software architecture	15
Relevant terms	15
Learning goals	15
3. Entwurf und Entwicklung von Softwarearchitekturen	22
Wesentliche Begriffe	22
4. Design and development of software architectures	23
Relevant terms	23
Learning goals	23
References	30
5. Beschreibung und Kommunikation von Softwarearchitekturen	31
Wesentliche Begriffe	31
6. Specification and communication of software architectures	32
Relevant terms	32
Learning goals	32
References	36
7. Softwarearchitektur und Qualität	37
Wesentliche Begriffe	37
8. Software architecture and quality	38
Relevant terms	38
Learning goals	38
References	40
9. Beispiele für Softwarearchitekturen	41

10. Examples of Software architectures	42
Learning goals	42
References	43



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

© (Copyright), International Software Architecture Qualification Board e. V. (iSAQB® e. V.) 2019

Die Nutzung des Lehrplans ist nur unter den nachfolgenden Voraussetzungen erlaubt:

1. Sie möchten das Zertifikat zum CPSA Certified Professional for Software Architecture Foundation Level® erwerben. Für den Erwerb des Zertifikats ist es gestattet, die Textdokumente und/oder Lehrpläne zu nutzen, indem eine Arbeitskopie für den eigenen Rechner erstellt wird. Soll eine darüber hinaus gehende Nutzung der Dokumente und/oder Lehrpläne erfolgen, zum Beispiel zur Weiterverbreitung an Dritte, Werbung etc., bitte unter info@isaqb.org nachfragen. Es müsste dann ein eigener Lizenzvertrag geschlossen werden.
2. Sind Sie Trainer oder Trainingsprovider, ist die Nutzung der Dokumente und/oder Lehrpläne nach Erwerb einer Nutzungslizenz möglich. Hierzu bitte unter info@isaqb.org nachfragen. Lizenzverträge, die alles umfassend regeln, sind vorhanden.
3. Falls Sie unter keine der vorstehenden Kategorien fallen, aber dennoch die Dokumente und/oder Lehrpläne nutzen möchten, nehmen Sie bitte ebenfalls Kontakt unter info@isaqb.org zum iSAQB e.V. auf. Sie werden dort über die Möglichkeit des Erwerbs entsprechender Lizenzen im Rahmen der vorhandenen Lizenzverträge informiert und können die gewünschten Nutzungsgenehmigungen erhalten.

Grundsätzlich weisen wir darauf hin, dass dieser Lehrplan urheberrechtlich geschützt ist.

Alle Rechte an diesen Copyrights stehen ausschließlich dem International Software Architecture Qualification Board e. V. (iSAQB® e. V.) zu.

Legal Notice

© (Copyright), International Software Architecture Qualification Board e. V. (iSAQB® e. V.) 2019

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

1. You wish to obtain the CPSA Certified Professional for Software Architecture Foundation Level® certificate. For the purpose of obtaining the certificate, it shall be permitted to use these text documents and/or curricula by creating working copies for your own computer. If any other use of documents and/or curricula is intended, for instance for their dissemination to third parties, for advertising etc., please write to info@isaqb.org to enquire whether this is permitted. A separate license agreement would then have to be entered into.
2. If you are a trainer or training provider, it shall be possible for you to use the documents and/or curricula once you have obtained a usage license. Please address any enquiries to info@isaqb.org. License agreements with comprehensive provisions for all aspects exist.
3. If you fall neither into category 1 nor category 2, but would like to use these documents and/or curricula nonetheless, please also contact the iSAQB e. V. by writing to info@isaqb.org. You will then be informed about the possibility of acquiring relevant licenses through existing license agreements, allowing you to obtain your desired usage authorizations.

We like to point out that this curriculum is protected by copyright.

The International Software Architecture Qualification Board e. V. (iSAQB® e. V.) has exclusive entitlement to these copyrights. 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.

Unresolved directive in foundation-curriculum.adoc - include::/home/travis/build/isaqb-org/curriculum-foundation/build/learning-objectives.adoc[tags=FEEDBACK;DE;EN;REMARK]

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 Softwarearchitekten 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., <http://ireb.org>, International Requirements Engineering Board)
- Test (siehe dazu das Ausbildungs- und Zertifizierungsprogramm des ISTQB e. V., <http://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., <http://ireb.org>, International Requirements Engineering Board)
- Software testing (please refer to the education and certification program by ISTQB e.V., <http://istqb.org>, International Software Testing Qualification Board)
- Project or product management
- Introduction to specific software tools.

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 objektorientierten Programmiersprache (etwa Java oder C#);
 - 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:

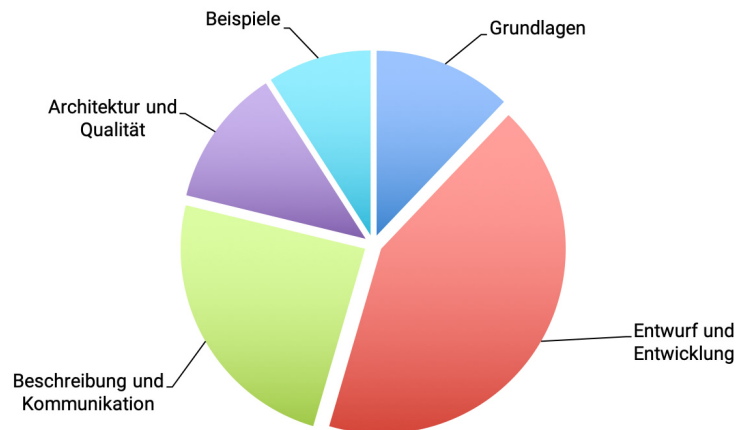
- Basics and differences of imperative, declarative, object-oriented and functional programming
- Practical experience in
 - an object-oriented programming language (i.e. Java or C#)
 - designing and implementing distributed applications, such as client-server systems or web applications
 - technical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



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.

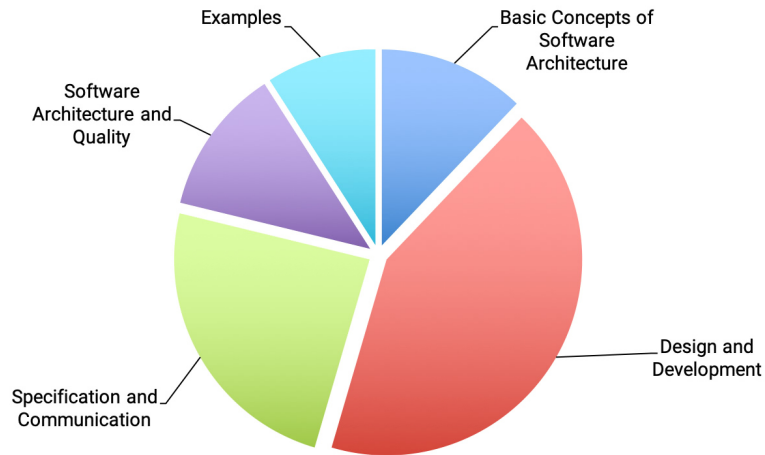


Empfohlene Zeitaufteilung

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 Teilnehmer 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 Teilnehmer 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.

1. Grundbegriffe

Dauer: 120 Min.	Übungszeit: Keine
-----------------	-------------------

Wesentliche Begriffe

[Softwarearchitektur](#); Architekturdomänen; [Struktur](#); [Bausteine](#); [Komponenten](#); [Schnittstellen](#); [Beziehungen](#); Querschnittskonzepte; Nutzen von Softwarearchitektur; Softwarearchitekten 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 concerns; software architects and their responsibilities; tasks and required skills; stakeholders and their concerns; functional and quality requirements of systems; constraints; influencing factors; types of IT systems (embedded systems; real-time systems; information systems etc.)

Learning goals

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

Softwarearchitekten 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, crosscutting concepts, principles
- Architecture decisions and their consequences on the entire systems and its lifecycle

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

Wesentliche Ziele von Softwarearchitektur sind:

- Qualitätsanforderungen wie Zuverlässigkeit, Wartbarkeit, Änderbarkeit, Sicherheit sowie funktionale Anforderungen zu erreichen
- Erstellung und Wartung von Software, insbesondere die Implementierung zu unterstützen
- Verständnis für Strukturen und Konzepte des Systems zu vermitteln, bezogen auf sämtliche relevanten Stakeholder.
- Architekturelevante Richtlinien zur Implementierung zu spezifizieren

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

Essential goals of software architecture are:

- the achievement of quality requirements such as reliability, maintainability, changeability, security etc. and functional requirements
- supporting the creation and maintenance of software, particularly its implementation
- convey the understanding of structures and concepts of the system, related to all relevant stakeholders.
- specify architecture-relevant guidelines for the implementation



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)

Softwarearchitekten 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 lifecycle 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 Softwarearchitekten verstehen (R1)

Softwarearchitekten 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 Softwarearchitekten:

- 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 concerns (for instance persistence, communication, GUI etc.)
- Communicate and document software architecture based on views, architectural patterns, cross-cutting and technical concepts
- Accompany the realization and implementation of the architecture; integrate feedback from relevant stakeholders into the architecture if necessary; review and ensure the consistency of source code and software architecture
- Analyze and evaluate software architecture, especially with respect to risks involving the quality requirements
- Identify, highlight, and justify the consequences of architectural decisions to other stakeholders

They should independently recognize the necessity of iterations in all tasks and point out possibilities for appropriate and relevant feedback.



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: Übersetzung von "gegenüber anderen Stakeholdern argumentieren" mit "discuss with..." finde ich nicht passend.

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

Softwarearchitekten können ihre Rolle erklären und ihren Beitrag zur Systementwicklung in Verbindung mit anderen Stakeholdern kontextspezifisch ausgestalten, insbesondere zu:

- Anforderungsanalyse (Systemanalyse, Anforderungsmanagement, Fachbereich)
- Implementierung
- Projektleitung und -management
- Produktmanagement, Product-Owner
- Qualitätssicherung
- 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, in particular to:

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

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

- Softwarearchitekten 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)

Softwarearchitekten 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)

Softwarearchitekten:

- können Annahmen oder Voraussetzungen explizit darstellen und vermeiden implizite Annahmen
- wissen, dass implizite Annahmen potentielle Missverständnisse zwischen beteiligten Stakeholdern bewirken.

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

Software architects:

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

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

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

Darüber hinaus kennen Softwarearchitekten 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)

Softwarearchitekten 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](#).

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; Stereotypen; 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.

Exercises: 90 min.

Relevant terms

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

Learning goals

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

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

- Top-down- und Bottom-Up-Vorgehen beim Entwurf
- Sichtenbasierte Architekturentwicklung
- iterativer und inkrementeller Entwurf
 - Notwendigkeit von Iterationen, insbesondere bei unter Unsicherheit getroffenen Entscheidungen
 - Notwendigkeit von Rückmeldungen zu Entwurfsentscheidungen
- Domain-Driven Design (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 the basic approaches of architecture development, for example:

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



- 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)

Softwarearchitekten 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.



- 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-R2)

Softwarearchitekten können Einflussfaktoren (Randbedingungen) als Einschränkungen der Entwurfsvfreiheit erarbeiten und berücksichtigen.

Sie erkennen und berücksichtigen den:

- Einfluss von Qualitätsanforderungen
- Einfluss technischer Entscheidungen und Konzepte
- (möglichen) Einfluss organisatorischer und juristischer Faktoren (R2)
- Einfluss verschiedener Stakeholder (R2)

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

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

They should recognize and consider the:

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



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)

Softwarearchitekten können:

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

Softwarearchitekten wissen, dass solche Querschnittskonzepte systemübergreifend wiederverwendbar sein können.

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

Software architects are able to:

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

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



GS/CL: leicht umformuliert. Modified EN translation of "Konzept" to "concept" instead of "concern".

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

Softwarearchitekten kennen diverse Architekturmuster und können diese bedarfsgerecht einsetzen:

- Schichten (*Layer*) (R1)
- Pipes und Filter (R1)
- Client/Server (R1)
- Adapter und Facade (R1)
- Proxy (R1)
- Plugin (R1)
- Blackboard (R2)
- Model-View-Controller und Varianten (R2)
- Broker (R2)
- Remote Procedure Call (R2)
- Messaging (R2), z.B. mit Events und Commands

Softwarearchitekten 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 architectural patterns (R1-R3)

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

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

Software architects know essential sources for architectural patterns, such as POSA (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)

Softwarearchitekten können die folgenden Entwurfsprinzipien erläutern und anwenden:

- Abstraktion
- Modularisierung, unter Berücksichtigung von:
 - Geheimnisprinzip (*Information Hiding*) und Kapselung
 - Trennung von Verantwortlichkeiten (*Separation of Concerns*)
 - Hoher Kohäsion
- Single-Responsibility-Prinzip
- Offen-Geschlossen-Prinzip (*Open-/Closed-Principle*)
- möglichst lose, aber funktional ausreichende Kopplung der Bausteine (siehe auch [LZ 2-7](#))
- Umkehrung von Abhängigkeiten durch Schnittstellen oder ähnliche Abstraktionen (*Dependency Inversion*)
- konzeptioneller Integrität zur Erreichung der Gleichförmigkeit (Homogenität, Konsistenz) von Lösungen für ähnliche Probleme (R2)

Softwarearchitekten verstehen die Einflüsse der Entwurfsprinzipien auf Quellcode und können diese gezielt einsetzen.

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

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

- Abstraction
- Modularization, with regard to:
 - information hiding and encapsulation
 - separation of concerns
 - high cohesion
- Single-Responsibility

- Open/closed principle
- Loose, but functionally sufficient coupling of building blocks (cf. [LG 2-7](#))
- Dependency inversion by means of interfaces or similar abstractions
- Conceptual integrity to achieve uniformity (homogeneity, consistency) of solutions for similar problems (R2)

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



- 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 planen (R1)

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

- kennen unterschiedliche Arten der Abhängigkeiten von Bausteinen (beispielsweise strukturelle Kopplung über Benutzung/Delegation, Schachtelung, Besitz, Erzeugung, Vererbung, zeitliche Kopplung, Kopplung über Datentypen oder über Hardware)
- 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 [Lernziel 2-5](#))
 - Grundlegende Entwurfsprinzipien (siehe [Lernziel 2-6](#))
 - Externalisierung von Abhängigkeiten, d.h. konkrete Abhängigkeiten erst zur Installations- oder Laufzeit festlegen, etwa durch Anwendung von *Dependency Injection*.

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

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

- know different types of dependencies of building blocks (e.g. structural coupling via use/delegation, nesting, ownership, generation, inheritance, temporal coupling, coupling via data types or via hardware)
- can use such types of coupling in a targeted manner and can assess the consequences of such dependencies
- know and can apply possibilities to reduce or eliminate coupling, for example:
 - Patterns (see [learning goal 2-5](#))
 - Basic design principles (see [learning goal 2-6](#))

- Externalization of dependencies, i.e. defining concrete dependencies at installation- or runtime, for example by using *Dependency Injection*.



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)

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

- Effizienz / Performance
- Verfügbarkeit
- Wartbarkeit, Modifizierbarkeit, Erweiterbarkeit, Adaptierbarkeit

Sie können:

- Lösungsmöglichkeiten, *Design 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 / Performance
- Availability
- Maintainability, modifiability, extensibility, adaptability

They can:

- explain and apply solution options, *Design Tactics*, suitable practices as well as technical possibilities to achieve important quality requirements of software systems (different for embedded systems or information systems)
- identify and communicate possible trade-offs between such solutions and their associated risks.



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)

Softwarearchitekten 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 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+2017], [Gamma+94], [Martin 2003], [Buschmann+1996] and [Buschmann+2007], [Starke 2017], [Lilienthal 2017]

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)

Softwarearchitekten 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)

Softwarearchitekten:

- können Architekturen stakeholdergerecht dokumentieren und kommunizieren und dadurch unterschiedliche Zielgruppen adressieren, z. B. Management, Entwicklungsteams, QS, andere Softwarearchitekten sowie möglicherweise zusätzliche Stakeholder
- sind in der Lage, die Beiträge unterschiedlicher Autorengruppen stilistisch und inhaltlich zu konsolidieren und harmonisieren

- kennen den Nutzen von Template-basierter Dokumentation.

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

Software architects are able to:

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



GS: Layout, sprachlich vereinfacht, *Schablone* entfernt

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

Softwarearchitekten kennen mindestens folgende UML-Diagramme zur Notation von Architektursichten:

- Klassen-, Paket-, Komponenten- und Kompositionsstrukturdiagramme
- Verteilungsdiagramme
- Sequenz- und Aktivitätsdiagramme
- Zustandsdiagramme

Softwarearchitekten kennen Alternativen zu UML-Diagrammen. Insbesondere für die Laufzeitsicht eignen sich z. B. Flussdiagramme, nummerierte Listen, BPMN.

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

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

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

Software architects know alternative notations to UML diagrams, especially for the runtime view, e.g., flow charts, numbered lists, BPMN.

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

Softwarearchitekten 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)

Softwarearchitekten können:

- Kontext von Systemen z.B. in Form von Kontextdiagramm 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 can:

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



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

LZ 3-6: Querschnittskonzepte dokumentieren und kommunizieren (R1)

Softwarearchitekten können typische Querschnittskonzepte (auch genannt *Prinzipien* oder *Aspekte*) adäquat dokumentieren und kommunizieren, z. B. Persistenz, Ablaufsteuerung, UI, Verteilung/Integration, Protokollierung.

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

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



- GS+RR: Redundanz zu Kapitel 2 entfernt, Protokollierung aufgenommen.
- JR: Begriff an Kapitel 2 angepasst: "Querschnittskonzepte" anstelle von "querschnittliche Konzepte".

LZ 3-7: Schnittstellen beschreiben (R1)

Softwarearchitekten 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 (R2)

Softwarearchitekten können:

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

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

Software architects are able to:

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



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)

Softwarearchitekten 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)

Softwarearchitekten kennen:

- Grundlagen mehrerer publizierter Frameworks zur Beschreibung von Softwarearchitekturen, beispielsweise:
 - ISO/IEEE-42010 (vormals 1471)
 - arc42

- C4
- RM/ODP
- 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)
 - arc42
 - C4
 - RM/ODP
 - FMC
- ideas and examples of checklists for the creation, documentation, and testing of software architectures
- possible tools for creating and maintaining architectural documentation



GS: (Feb 2019): please can we remove the "checklists"?

GS: C4 aufgenommen (von Simon Brown)

GS/RR: TOGAF entfernt

References

[\[Bass+2012\]](#), [\[Clements+2010\]](#), [\[Gharbi+2017\]](#), [\[Starke 2017\]](#), [\[Zörner 2015\]](#)

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)

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



GS: Beispiele klarer formuliert

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

Softwarearchitekten 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)

Softwarearchitekten:

- 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)

Softwarearchitekten 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

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

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)

Softwarearchitekten 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. Software architects have used at least one example to identify and understand the relationship between requirements and constraints, and the solution decisions.

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

Softwarearchitekten 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

- [iSAQB Working Groups] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Freely available <https://leanpub.com/isaqbreferences>.
- [Bass+2012] Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. 3rd Edition, Addison Wesley 2012.
- [Clements+2002] Paul Clements, Rick Kazman, Mark Klein: Evaluating Software Architectures. Methods and Case Studies. Addison Wesley, 2002.
- [Clements+2010] Paul Clements, Felix Bachmann, Len Bass, David Garlan, David, James Ivers, Reed Little, Paulo Merson and Robert Nord. *Documenting Software Architectures: Views and Beyond*, 2nd edition, Addison Wesley, 2010
- [Gamma+94] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
- [Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java. Springer-Vieweg Verlag, 2. Auflage 2014.
- [Fowler 2003] Martin Fowler: Patterns of Enterprise Application Architecture. (PoEAA) Addison-Wesley, 2003.
- [Gharbi+2017] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 3. Auflage, dpunkt Verlag, Heidelberg 2017.
- [Martin 2003] Robert Martin: Agile Software Development. Principles, Patterns, and Practices. Prentice Hall, 2003.
- [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.
- [Starke 2017] Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden (in German). 8. Auflage, Carl Hanser Verlag 2017. Website: <https://esabuch.de>
- [Lilienthal 2017] Carola Lilienthal: Langlebige Softwarearchitekturen. 2. Auflage, dpunkt Verlag 2018.
- [Zörner 2015] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. 2. Auflage, Carl Hanser Verlag 2015.