

Curriculum

Certified Professional
for
Software Architecture
(CPSA)

– Foundation Level –



Version 2.0 (July 1st, 2009)

Table of contents

0 INTRODUCTION..... 4

0.1 FOUNDATION LEVEL TRAINING: WHAT ARE YOU GOING TO LEARN? 4

0.2 STRUCTURE OF THE CURRICULUM, AND RECOMMENDED SCHEDULE 4

0.3 DURATION, DIDACTICS AND FURTHER DETAILS OF ACCREDITED TRAININGS 5

0.4 REQUIREMENTS 5

0.5 STRUCTURE OF THE iSAQB LEARNING MODULE 5

0.6 SCOPE OF THE TRAINING 6

0.7 ADDITIONAL INFORMATION, TERMS, TRANSLATIONS 6

0.8 INTRODUCTION TO THE iSAQB CERTIFICATION PROGRAM 6

0.9 TERMS AND CONCEPTS 6

0.10 LEARNING OBJECTIVES..... 6

1 BASIC TERMS OF SOFTWARE ARCHITECTURE..... 7

1.1 TERMS AND CONCEPTS 7

1.2 LEARNING OBJECTIVES 7

1.3 REFERENCES..... 9

2 SPECIFICATION AND COMMUNICATION OF SOFTWARE ARCHITECTURE 10

2.1 TERMS AND CONCEPTS 10

2.2 REFERENCES..... 11

3 DEVELOPMENT OF SOFTWARE ARCHITECTURE..... 12

3.1 TERMS AND CONCEPTS 12

3.2 LEARNING OBJECTIVES..... 12

3.3 REFERENCES..... 14

4 SOFTWARE ARCHITECTURE AND QUALITY..... 15

4.1 TERMS AND CONCEPTS 15

4.2 LEARNING OBJECTIVES..... 15

4.3 REFERENCES..... 16

5 TOOLS FOR SOFTWARE ARCHITECTS 17

5.1 TERMS AND CONCEPTS 17

5.2 LEARNING OBJECTIVES..... 19

5.3 REFERENCES..... 19

<u>6</u>	<u>EXAMPLES OF SOFTWARE ARCHITECTURE.....</u>	<u>20</u>
6.1	TERMS AND CONCEPTS	20
6.2	LEARNING OBJECTIVES.....	20
6.3	REFERENCES.....	20
<u>7</u>	<u>SOURCES AND REFERENCES FOR SOFTWARE ARCHITECTURE.....</u>	<u>21</u>

0 Introduction

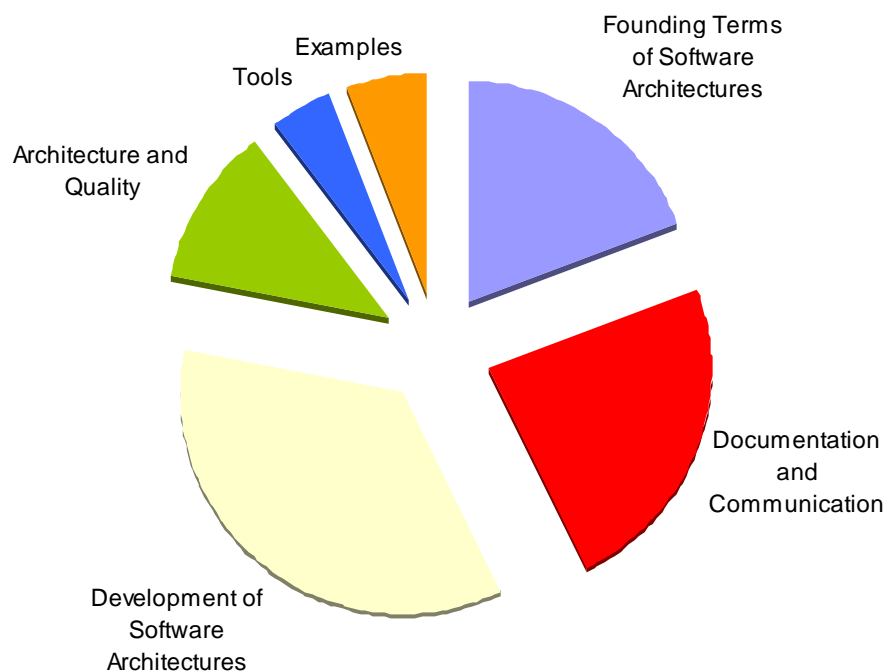
0.1 Foundation Level training: What are you going to learn?

The participants will learn the following in the accredited training **Certified Professional for Software Architecture – Foundation Level**:

- Terminology and relevancy of software architecture
- Tasks and accountabilities within software architecture
- The role of the software architect within a project
- State-of-the-Art measures and techniques in developing software architecture and the following skills:
 - decide about critical software architecture with other participants of the project - from Requirement Management to Project Management, testing and development
 - Document and communicate software architecture based on views, architectural patterns and technical concepts
 - Understand the most important steps in the design of software architecture and how to apply them independently for smaller and medium-sized systems

The training “Foundation Level” introduces the necessary knowledge and skills to design and document a software architecture based on sufficiently detailed, specified requirements that address the problem. It forms the foundation for the implementation and serves as a template. Participants learn the tools and based upon their knowledge decide on a design that relates to a particular problem.

0.2 Structure of the curriculum, and recommended schedule



0.3 Duration, didactics and further details of accredited trainings

The following schedule is a recommendation only. The training should not be shorter than 3 days, but might as well be longer. Provider may offer different durations, didactical approaches or types and structure of exercises and different levels of detail in regards to the training's structure.

The curriculum leaves decisions on the types of examples and exercises (in the technical and professional domain) unanswered.

0.4 Requirements

Participants should show the following knowledge and experience:

- Sufficient practical knowledge in software development, acquired through a variety of projects or systems outside of their formal education
- Familiarity and knowledge in at least one high-level programming language
- Basics of modeling
- Basics of UML (class, package, component and sequence diagram) and its mapping in source code
- Knowledge of distributed systems (development of any system that does not run on one single computer alone)

To gain a deeper understanding of some of the concepts, it might be helpful to know about:

- Object orientation
- Knowledge of at least one object oriented language
- Knowledge of the conception and implementation of distributed applications, e.g. client/server systems or web applications
- Experience with CORBA, RMI, Web Services
- Knowledge of technical documentation, especially in the documentation of source code, system design and technical concepts

0.5 Structure of the iSAQB learning modules

The modules follow the following structure:

- terms/concepts: critical key concepts of this learning module
- teaching time/practice time: defines the minimal teaching and practice time in an accredited training required for this topic
- Learning objectives: describe the content to be thought and its key terms and concepts

The next section outlines the questions that will be answered during the according part of the training. The learning objectives are differentiated in categories and sub categories.

- What the participants should be able to master... The participants should be able to apply the content independently at the end of the training. The training offers exercises relating to the content. The content is part of the exam "Foundation Level".
- What the participants should be able to understand... The content may be part of the exam "Foundation Level".
- What the participants should know...The content may support understanding (terms, concepts, methods, techniques etc.) or motivate the topic. The content is not part of the exam. It is discussed during the training but might not be taught in detail.
- References: for further reading, standards or other sources. You may find a detailed list of books and further sources in *iSAQB-FachlicheQuellen*.

0.6 Scope of the training

This curriculum does not provide a complete description of the software architecture domain. It helps to better understand the “Foundation Level” learning objectives by providing the necessary and practical content. It is defined from the point of view of the members of iSAQB and based on the current circumstances.

The following topics and concepts are not part of the “Foundation Level”:

- Concrete implementation technologies, frameworks or libraries
- Programming or programming of high-level Languages
- Fundamentals of modeling
- Comprehensive presentation of UML
- System Analysis, Requirements Engineering
- Project Management

0.7 Additional information, terms, translations

Wherever essential for the understanding of the curriculum, we included and defined technical terms in the iSAQB glossary and, if required, complemented them with translations of the original literature.

0.8 Introduction to the iSAQB certification program

Duration: 15 min.	Training time: none
-------------------	---------------------

This part will not be covered in the exam.

0.9 Terms and concepts

iSAQB, certification, foundation and advanced level, exam

0.10 Learning objectives

The participants get to know the iSAQB certification program, the related exams and their procedures.

0.10.1 What the participants should know...

- iSAQB as an association
- the scope of the “Foundation Level” in contrast to other level
- Constraints and approach of the iSAQB certification program
- Other certification programs (e.g. TOGAF)

1 Basic concepts of software architecture

Duration: 135 min.	Training time: 45 min.
--------------------	------------------------

1.1 Terms and Concepts

Software architecture; structure; modules/components; interfaces; relationships; comprehensive concepts/aspects; objectives of the architecture; software architects and their accountabilities; tasks and required skills; non-functional and functional requirements of systems; constraints; factors, types of IT systems (integrated systems, real time systems, information systems etc.)

1.2 Learning objectives

1.2.1 What the participants should be able to master...

1.2.1.1 Definition of software architecture

- Comparison of several definitions of software architecture (et al. SEI, Booch etc.) and their similarities:
 - Components and modules with interfaces and relationships
 - Structures and comprehensive concepts
 - Comprehensive decision regarding the design and their system-wide consequences
 - Design as small scale architecture
- Our definition of software architecture and components and its relation to the code
 - How do we define software architecture in this training?
 - How do we define component, interface, and relationship between components in this training?
 - How do these definitions (software architecture/components/interfaces/relationships) map code?
- Explain the correspondence of software architecture with architectural drawings and construction plans
 - Software architecture as the description of a solution
 - Software architecture as support of the production process, especially the implementation

1.2.1.2 Benefits and Targets of software architecture

Name and explain the targets of software architecture and architects

Software architecture focuses on quality features. These are in contrast to a mere functionality. It focuses on issues like durability, maintainability, changeability instead.

- As a rule architecture targets the long-term, whereas projects target the short-term

1.2.1.3 Relation of the software architecture to the overall development of IT systems

Know and explain the role of software architecture in relation to other stakeholders, highlighting the relation to:

- Requirement Analysis (System Analysis, Requirements Management, department asking for the solution)
- Implementation
- Project Management
- Quality assurance

- IT Operations (production, datacenter) [refers primarily to information systems]
- Development of hardware

1.2.1.4 Tasks of software architecture

Software architects are responsible for achieving the required targets or necessary quality of the solution. Essential is the coordination between this responsibility and the overall project responsibility.

- Name and characterize tasks of software architects
 - Clarify, question and refine, if required, requirements and constraints. These include functional requirements (required features) especially quality features (required constraints). Show consequences of these constraints for the architecture (Hofmeister+2000 Global Analysis/Hofmeister+2005).
 - Determine the structure in regards to the decomposition of the system and of the modules. Include a definition of dependencies and interfaces between the modules.
 - Determine and, if necessary, implement comprehensive technical concepts (e.g. persistence communication, GUI etc.).
 - Communicate and document software architecture.
 - Monitor implementation of the architecture; integrate feedback of the stakeholders, if necessary. Control and ensure consistency of the source code and software architecture.
 - Assess software architecture, especially in regards to risks of required quality features.
- Name and characterize relationships between tasks' results: Which tasks provide or influence which deliverable?
- Explain the connection between requirements and solutions
- detect and prioritize constraints and influencing factors for the architectural design
- Identify and specify architectural targets on the basis of the current requirements
- Detect and demonstrate consequences of architectural decisions and discuss them with other stakeholders
- Perceive the necessity of iterations with all tasks independently and demonstrate possibilities for an appropriate feedback

1.2.1.5 Decision about architecture and design

- Demonstrate the significance of architectural targets, constraints and factors for the design of software architecture
 - Explanation of requirements (required features as well as required constraints)
 - Explanation of constraints /influencing factors
 - Explanation of the distinction between (long-term) architectural targets and (short-term) project objectives
- Decisions about structures and technical concepts, in particular possible interdependencies of these decisions
- Necessity to explain explicit and operational architectural targets: they are the only means to assess the success of architecture, architectural design
- Explain the influence of repetitive procedures on architectural descriptions (in regards to risks and predictabilities)

1.2.2 What the participants should understand...

- Software architecture is a description of the solution. It is not the solution itself.
- Software architecture supports the achievement of non-functional quality features
- Software architecture is responsible for achieving the required and necessary quality of the solution

- Software architecture is proactive
- Software architecture should present assumptions and prerequisites explicitly and avoid implicit assumptions
- Due to an inherent uncertainty, software architects often have to work and decide iteratively. To do so, a systematic feedback by stakeholders is essential.
- Quality requirements (required constraints), like changeability, efficiency and security etc., have a greater influence on the software architecture than functional requirements.
- Understand different types of software architecture of IT systems:
 - Relationship to systems or overall architecture for integrated and hardware related systems
 - Understand particularities of software/hardware code design (dependencies to hardware and software design in relation to time and content)
 - Relationship with business and operational processes of the information systems
 - Relationship to business or operational processes for decision support systems (Data Warehouse, Management Information)

1.2.3 What the participants should know...

- Additional levels of architecture, e.g. enterprise IT architecture, business architecture, infrastructure architecture (in accordance with Dern 2006). Several different level of architecture exist within the IT of information systems:
 - Infrastructure architecture: structure of the technical infrastructure, hardware, network etc.
 - Hardware architecture (for hardware-related systems)
 - Software architecture: structure of the individual software architecture. Software architecture, according to iSAQB, focuses on this kind of structure.
 - Enterprise IT architecture: structure of application landscapes. iSAQB does not focus on this topic.
 - Business process architecture, business architecture: structure of business processes (not a focus of iSAQB).
- Taxonomy of quality features according to ISO 9126 (excluding their exact definition).

1.3 References

[Bass+2003]

[Hofmeister+2000]

[Posch 2004]

[Reussner+2006]

[Siedersleben 2004]

[Starke 2008]

[Vogel+2005]

2 Specification and communication of software architecture

Duration: 150 min.	Training time: 90 min.
--------------------	------------------------

2.1 Terms and concepts

Views, structures, (technical) concepts, documentation, communication, description meta structures to specify and communication, modules, view of modules,

Kommunikation, specification Meta-Strukturen zur Beschreibung und Kommunikation, Building Blocks, Building Block View, Lifecycle View, Runtime View, View of distribution, links and nodes, channel, artifacts distribution, mapping of modules and artifacts distribution, description of interfaces

This part of the curriculum is strongly tied to the following chapter

Learning objectives

2.1.1 What the participants should be able to master...

- Specify and communicate software architecture relevant to the needs of specific stakeholder groups
- Define essential architectural views and their significance
- Document different architectural views
 - Module and component view (system build made of software modules)
 - Runtime View (dynamic view, collaboration of the software modules over the life cycle, model in regards to the state)
 - Distribution/deployment view (map the software modules with the hardware or the execution environment)
- Explain comprehensive technical concepts or architectural aspects
 - Name some typical concepts (e.g. persistence, process control, GUI, distribution/integration)
- Specification and communication of interfaces
 - define interfaces, especially external interfaces
 - Resource-oriented approach
 - Service-oriented approach
- Explain the fundamentals and quality features of the technical documentation:
 - Quality features of the technical documentation: comprehensibility, efficiency, accuracy

2.1.2 What the participants should know...

- The specification (or documentation) is a form of communication in writing
- descriptors for software architecture also support the design and the build
- useful UML diagrams for the notation of architectural views:
 - class, package and component diagrams
 - distribution diagrams
 - sequence diagrams
- benefits of template-based documentation
 - blackbox and whitebox templates

- Templates for the design/documentation of notes and channels
- Templates for the documentation of interfaces
- Documentation basics
 - Style and choice of words should be chosen with respect to the skills and the goals of the reader
 - the reader alone can assess how comprehensible a documentation is written
 - Essential quality features of a technical documentation are:
 - accuracy
 - comprehensibility
 - efficiency (reading and writing)
 - document important architectural decisions
 - explain architectural decision with explanations and derivations
 - due to a variety of stakeholders the specification of software architecture has particular requirements
 - Different audiences: management, developer, QS and other software architects
 - Different authors: software architects, developer und maybe more

2.1.3 What the participants should know ...

- Fundamentals of at least two (more is better) conceptual frameworks to specify the software architecture:
 - TOGAF
 - FMC
 - RM/ODP
 - IEEE-1471
 - arc42
- Ideas and examples of check lists for the design, documentation and assessment of software architecture
- Possible tools to design and maintain architectural documentation

2.2 References

[Clements+2003]

[Hargis+2004]

[Starke 2008]

[Starke+2009]

3 Development of software architecture

duration: 270 min.	Training time: 90 min.
--------------------	------------------------

3.1 Terms and concepts

Design, design procedure, design decision, views, technical concept of architectural patterns, principles of design, professional and technical architecture, model-based design, interactive/incremental design, domain driven design, top-down und bottom-up procedure

3.2 Learning objectives

This topic introduces procedures and tools used in the design and development of software architecture.

Participants will practice to develop one (or more) architectural designs using the results of chapter 3.

3.2.1 What the participants should be able to master

- name, specify and explain the approach and heuristics to develop architecture:
 - Model-based and view-based development of architecture
 - Model-based design and domain driven design
 - Iterative and incremental design
 - Necessity of iterations, especially when the decision took place in a situation of uncertainty
 - Feedback about design decisions on the basis of source code and from a qualitative point of view
 - top-down and bottom-up procedure in design
 - constraints and influencing factors as limitations in the design of architecture (Global Analysis)
- design architecture based on functional and non-functional requirements for security or essential business software systems and develop the appropriate documentation
- explain the terms “blackbox” and “whitebox” and their goal-oriented application
- refine and specify modules sequentially (use a hierarchy)
- design individual architectural views, especially distribution, module and life cycle view, and describe its consequences for the according source code
- define, design and translate architecture and its consequences into code
- justify and apply separation of technical and professional modules in the architecture
- design and justify professional structures
 - identify, justify and document professional modules (units, services)
 - explain and apply the abstraction of the transition from professional parts to technical parts
- architecture’s influence on quality requirements
- influence of technical decisions and concepts on the architecture, especially the structures of modules
- Name, explain and apply basic concepts of architectural design

- Name and explain modules of software architecture and their features:
 - Desired features (encapsulation, information hiding, limited access principle) of modules
 - blackbox and whitebox modules
- Types of composition of modules (interleaving, usage/delegation, inheritance)
- UML notation for different modules and their compositions
 - Packages: semantically weak type of module aggregation
 - Components: semantically strong type of aggregation due to clearly defined interfaces
- Specify, explain and apply important architectural patterns accordingly
 - Layer
 - Pipes and filter
 - Model-View-Controller
- Explain and apply design principles
 - limited access principle
 - coupling and cohesion
 - Separation of Concern
 - Open-/Closed-Principle
 - Reversion of dependencies between interfaces
 - Dependency Injection to externalize dependencies
 - Relationships between dependencies in the model and in the source code of programming languages
- Understand and use dependencies and coupling of modules systematically
 - Identify consequences of coupling
 - Types of coupling (structural, in regards to schedule, data type or hardware etc.)
 - Identify possibilities to break up and reduce coupling
- Design and describe interfaces

3.2.2 What the participants should understand ...

- Details about coupling and their consequences for programming languages
 - Implementation of relationships in (object-oriented) programming languages
 - Constructors
 - Factory patterns
 - Dependency Injection
- Use of architectural pattern to meet demands
 - According to POSA (layer, pipes and filter, blackboard, microkernel)
 - According to Fowler/PoEAA
 - According to Hohpe (messaging, async pattern)
 - Other patterns
- Deploy language independent design pattern
 - Adapter, Wrapper, Gateway
 - Facade
 - Registry
 - Broker

3.2.3 What the participants should know...

- Further design pattern – that is not necessarily part of an accredited training
 - Factory
- Essential sources of architectural patterns
 - POSA series
 - conferences about patterns
- example of further languages for pattern
 - Organizational patterns
 - Reengineering patterns
 - Security patterns
 - Client/Server patterns
 - Patterns for distributed systems
 - more patterns depending on the focus of the training

3.3 References

[Martin+2003]

[Fowler 2003]

POSA-Serie, insbesondere [POSA-1] und [POSA-4]

[Demeyer+2002]

4 Software architecture and quality

duration: 60 min.	Training time: 60 min.
-------------------	------------------------

4.1 Terms and concepts

Quality, features of quality, DIN/ISO 9126, ATAM, scenarios, quality tree, compromises (when implementing quality features), quality assessment of architecture

4.2 Learning objectives

This topic focuses on getting to know measures to achieve certain quality features in software systems and on the methodological procedures for quality assessment of software architecture.

4.2.1 What the participants should be able to master...

- Define quality (following DIN/ISO 9126) and explain quality features
- Explain models of quality (e.g. DIN/ISO 9126)
- Explain relationships and interactions between quality features
- Explain and apply tactics, Best Practices and technical possibilities to achieve important quality targets of software systems (different for integrated systems and information systems)
 - Efficiency/performance
 - Maintainability, changeability, expandability and flexibility
- Perform a quality assessment of the software architecture according to ATAM
 - Explain and perform procedures of quality assessment
 - Explain scenarios and perform quality trees
 - Perform the analysis of software architecture in regards to scenarios and identification of risks independently
- Assessment of the implementation of the software architecture, i.e. answer the question: "Has the architecture been matched to code?"
 - How can you assess the architecture through code (and operationalize and integrate this into the process, e.g. as part of the check in procedure during the build etc.)
 - Metrics and other tools to assess architecture

4.2.2 What the participants should understand...

- Explain and apply tactics, Best Practices and technical possibilities to achieve additional quality targets of software systems:
 - security
 - comprehensibility, traceability
 - robustness, tolerance of errors
 - scalability
- use prototypes or technical trials to control the quality of the architecture
- explain the metrics for the assessment of artifacts

- to assess and estimate the architecture additional media and information might be taken into account, e.g.:
 - Source code
 - Known errors in the source code, especially error cluster

4.2.3 What the participants should know...

Further metrics, especially lines of code, cyclomatic complexity, incoming and outgoing dependencies, stability, distance and abstraction.

4.3 References

[Bass+2003]

[Clements+2002]

[Martin 2003]

[Starke 2008]

5 Tools for software architects

Duration: 45 min.	Training time: none
-------------------	---------------------

5.1 Terms and concepts

The following categorizes the essential tools for architects:

5.1.1 Modeling tools

Supports the architect's modeling tasks:

- Design and communication of software architecture
- Guideline for detailed design and implementation and maybe even to generate code

Criteria:

- Support multiple users
- Validation/formal check of models
- Support code generation (check also: tools for generation)
- Support reverse engineering
- Support the generation of documentation
- Support the Configuration Management (versioning, comparison, merge of models)

5.1.2 Tools for the statistical analysis

Supports the architect with the following tasks:

- Asses existing software systems in terms of quality features (e.g. complexity: can be the basis for refactoring)
- Analyses if the implementation is according to the guidelines of the architecture (e.g. follow rules of permitted dependencies)

Criteria:

- Automation
- Prepare and visualize results
- Support criteria for the analysis and metrics

5.1.3 Dynamic analysis tools

Support the architect with the following tasks:

- Systematic investigation of the performance and load features of the existing software systems

5.1.4 Generation tools

Support the architect with the following tasks:

- Provide a framework for the implementation
- Ensure consistency between architectural model and implementation

Criteria:

- configurability of the code's generation
- readability of the generated code
- certification for applications critical to the security
- repeatability of generation phases
- interface between generated and manually implemented code

5.1.5 Requirement engineering tools

Support the architect with the following tasks:

- Analyze the requirements of the software system that needs to be designed

Criteria:

- Support the traceability between requirements and architecture

5.1.6 Documentation tools

Support the architect with the following tasks:

- Architecture's communication

Criteria:

- Automated generation of up-to-date documentation from architectural models; it is essential to choose a level of detail and scope that matches the target group's needs
- Support multiple users
- Supports the documentation's Configuration Management (versioning, traceability of modifications)
-

5.1.7 Tools for build/build systems

Support the architect with the following tasks:

- Automation of the compiler, packing and test procedure
- Monitor the compliance with structural guidelines
- Control of a continual set-up including quality assurance

Criteria:

- Performance
- Adaptability
- Homogeneity
- Support proprietary tools

5.1.8 Configuration Management

Support the architect with the following tasks:

- Allocate and select configuration elements to a configuration
- inventory
- reconstruction of a configuration

Criteria:

- Performance

- Scalability (configurations due to enormous size and due to versions)
- Methodical integration with other tools (Version Management, Issue Tracker, requirements Management Tools)

5.2 Learning objectives

5.2.1 What the participants should be able to master ...

Name and explain the most important categories of tools and their strength and weaknesses when working on software architecture.

5.2.2 What the participants should understand ...

The working environment and tools of software architecture depend on the according constraints and influencing factors.

5.2.3 What the participants should know ...

Some typical tools, that the participants experienced personally, include:

- Modeling tools (UML tools)
- Generation tools (MDA or MDSD tools)
- Documentation tools (Wikis, repository-based and file-based document management)
- Tools for Build and Configuration Management

5.3 References

None

6 Examples of software architecture

Duration: 60 min.	Training time: none
-------------------	---------------------

This topic is not part of the examination.

6.1 Terms and concepts

Each accredited training needs to introduce and discuss at least one (better two) examples of software architecture.

The participants' interests, as well as the training itself, decide which type and of which scope the presented examples should be. iSAQB does not require any specific examples.

6.2 Learning objectives

Introduction and discussion of at least one (better two) example architectures.

6.2.1 What the participants should be able to master ...

n/a

6.2.2 What the participants should understand ...

n/a

6.2.3 What the participants should know ...

Some realistic architecture examples

6.3 References

None; The training providers are responsible for the choice and description of the examples.

7 Sources and references for software architecture

The curriculum references in part or entirely to some of these enlisted references and sources.

B

[Bachmann 2000]

Bachmann, F., L. Bass, et al.: Software Architecture Documentation in Practice. Software Engineering Institute, CMU/SEI-2000-SR-004.

[Bass+2003]

Bass, L., Clements, P. und Kazman, R. (2003): Software Architecture in Practice. Addison-Wesley, Reading, Mass

[Bosch 2000]

Bosch, J.: Design & Use of Software Architectures. Addison-Wesley, 2000

[Buschmann+96]

Buschmann, F., R. Meunier, H. Rohnert,, P. Sommerlad,, M. Stal: A System of Patterns. Pattern – Oriented Software. Architecture. Wiley 1996.

[Buschmann+2007]

Frank Buschmann, Kevlin Henney, Doug Schmidt: Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing (v. 4). Wiley, 2007.

C

[Clements+2002]

Clements, P., R. Kazman, M. Klein: Evaluating Software Architectures – Methods and Case Studies. Addison Wesley, 2002.

[Clements+2003]

Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers et al: Documenting Software Architectures – Views and Beyond. Addison Wesley, 2003.

D

[Demeyer+2002]

Serge Demeyer, Stephane Ducasse, Oscar Nierstrasz. Object Oriented Reengineering Patterns. Morgan Kaufman, 2002.

[Dern 2006]

Dern, Gernot: Management von IT-Architekturen. Informationssysteme im Fokus von Architekturplanung und -entwicklung. Vieweg, Edition CIO, 2. Auflage, 2006.

E

[Evans 2004]

Evans, E.: Domain Driven Design. Addison-Wesley, 2004. Website: www.domaindrivendesign.org.

F

[Fowler 2003]

Fowler, M. (2003): Patterns of enterprise application architecture. Addison-Wesley.

G

[Gamma 1995]

Gamma, E., R. Helm, R. Johnson, J. Vlissides: Design Patterns. Addison-Wesley, 1995.

[Gordon 2006]

Gordon, I.: Essential Software Architecture. Springer 2006.

H

[Hargis+2004]

Hargis, Gretchen et. al Developing Quality Technical Information. A Handbook for Writers and Editors. Prentice Hall, 2004.

[Hatley2000]

Hatley, D., P. Hruschka, I. Phirbai: Process for System Architecture and Requirements Engineering. Dorset House, 2000.

[Hofmeister+2000]

Hofmeister, C., Nord, R. und Soni, D. (2000): Applied Software Architecture. Addison-Wesley.

[Hofmeister+2005]

Hofmeister, Christine, R. Nord, D. Soni. "Global Analysis: Moving from Software requirements Specification to Structural Views of the Software Architecture," IEE Proceedings Software, Vol. 152, Issue 4, pp. 187-197, August 2005.

[Hohpe+2003]

Hohpe, G., B. Woolf: Enterprise Integration Patterns: Design-ing, Build-ing, and Deploying Messaging Solutions. Addison Wesley, 2003.

K

[Keller 2006]

Keller, Wolfgang: IT-Unternehmensarchitektur: Von der Geschäftsstrategie zur optimalen IT-Unterstützung. dpunkt, 2006.

[Kruchten 1995]

Kruchten, P.: Architectural Blueprints – The 4-1 View Model of Architecture. IEEE Software November 1995; 12(6), p. 42-50.

M

[Martin 2003]

Martin, R. C. (2003): Agile Software Development, Principles, Patterns, and Practices. Pearson Education, Upper Saddle River.

O

[Open Group 09]

The Open Group: TOGAF, Version 9. Online: www.opengroup.org/architecture/togaf9-doc/arch/

P

[Parnas 1972]

Parnas, D. L.: On the Criteria to be Used in Decomposing Systems into Modules. Communications of the ACM 1972; 15: p. 1053-1058.

[POSA-1]

siehe [Buschmann+96]

[POSA-4]

siehe [Buschmann+2007]

[Posch 2004]

Posch, T., K. Birken, M. Gerdorn: Basiswissen Softwarearchitektur: Verstehen, entwerfen, bewerten und dokumentieren. dpunkt, 2004.

[Putman 2000]

Putman, J.: Architecting with RM-ODP. Prentice Hall, 2000

R

[Rechtin+2000]

Rechtin, E., M. Maier: The Art of Systems Architecture. CRC Press, 2000

[Reussner+2006]

Reussner, R. und Hasselbring, W. (eds.) (2006): Handbuch der Software-Architektur. dpunkt.verlag, Heidelberg.

S

[Schmidt2000]

Schmidt, D., M. Stal, H. Rohnert, F. Buschmann: Pattern-Oriented Software-Architecture – Patterns for Concurrent and Networked Objects. Vol. 2. Wiley, 2000.

[Shaw 1996]

Shaw, M., D. Garlan: Software Architecture: Perspectives on an Emerging Discipline. Upper Saddle River, NJ: Prentice Hall, 1996.

[Siedersleben 2004]

Siedersleben, J. (2004): Moderne Softwarearchitektur: Umsichtig planen, robust bauen mit Quasar. dpunkt.verlag, Heidelberg

[Starke 2008]

Starke, G. (2008): Effektive Software-Architekturen - Ein praktischer Leitfaden. 3. aktualisierte und erweiterte Auflage 2008, Carl Hanser Verlag, München.

[Starke+2009]

Starke, Gernot und Peter Hruschka: Software-Architektur kompakt. Spektrum Akademischer Verlag, 2009.

V

[Vogel+2005]

Vogel, O., u.a.: Software-Architektur: Grundlagen – Konzepte – Praxis. Spektrum, 2005