Curriculum for

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Module
FLEX

Flexible Architecture Models - Modularization, Integration and Operation of Modern Software Systems

2024.1-RC0-EN-20241215

Table of Contents

List of Learning Goals . . . 2
Introduction: General information about the iSAQB Advanced Level . . . 4
What is taught in an Advanced Level module? . . . 4
What can Advanced Level (CPSA-A) graduates do? . . . 4
Requirements for CPSA-A certification . . . 4
Essentials . . . 5
What does the module “FLEX” convey? . . . 5
Curriculum Structure and Recommended Durations . . . 5
Duration, Teaching Method and Further Details . . . 5
Prerequisites . . . 6
Structure of the Curriculum . . . 6
Supplementary Information, Terms, Translations . . . 6
1. Why approach Flexible Systems . . . 8
1.1. Terms and Principles . . . 8
1.2. Learning Goals . . . 8
1.3. References . . . 9
2. Modularization of Systems of Systems . . . 10
2.1. Terms and Principles . . . 10
2.2. Learning Goals . . . 10
2.3. References . . . 11
3. Software Modules and the Organization . . . 12
3.1. Terms and Principles . . . 12
3.2. Learning Goals . . . 12
3.3. References . . . 13
4. Integration methods & protocols . . . 14
4.1. Terms and Principles . . . 14
4.2. Learning Goals . . . 14
4.3. References . . . 15
5. Deployment & Runtime/Platform Methods . . . 16
5.1. Terms and Principles . . . 16
5.2. Learning Goals . . . 16
5.3. References . . . 17
6. Service Operation Models . . . 18
6.1. Terms and Principles . . . 18
6.2. Learning Goals . . . 18
6.3. LG 6-1: Explain and choose different operational models and their impacts . . . 18
6.4. References . . . 20

References 21

List of Learning Goals

- LG 1-1: Classify Key Topics and Industry Buzzwords
- LG 1-2: Understand and Analyze Prerequisites for Distributed Systems
- LG 1-3: Communicate and Adapt Trade-offs of Presented Architecture Types
- LG 1-4: Define Long-term Quality Goals of Flexible Architectures
- LG 1-5: Explain and justify Typical Architectural Decisions in Flexible Architectures
- LG 2-1: Designing decomposition into components based on requirements
- LG 2-2: Describing and justifying different types of components
- LG 2-3: Evaluating and selecting modularization concepts
- LG 2-4: Assessing modularization strategies
- LG 2-5: Contrasting the costs and benefits of modularization strategies
- LG 3-1: Analyze and name the interaction between architecture types and organization
- LG 3-2: Consider the organization's communication structure when decomposing
- LG 3-3: Use context maps for stakeholder management
- LG 3-4: (OPTIONAL) Use terms like team organization and socio-technical architectures confidently
- LG 3-5: Identify macro-architecture decisions made outside your sphere of influence
- LG 4-1: Compare Integration Strategies (Using the Example of DDD Strategic Design)
- LG 4-2: Select and Justify Technical Integration Mechanisms
- LG 4-3: Explain and Select Consistency Models (CAP Theorem)
- LG 4-4: Identify and select Resilience Patterns
- LG 4-5: Understand and Consider Security Implications of Integration Methods
- LG 4-6: (Optional) Understand and Design Event-Driven Architectures (EDA)
- LG 5-1: Specify prerequisites and implications for Continuous Deployment
- LG 5-2: (OPTIONAL) Explain and select differences between IaaS, PaaS, CaaS, and FaaS
- LG 5-3: Identify and select Zero Downtime methodologies and their implications
- LG 5-4: Explain differences between Continuous Integration, Continuous Deployment, and Continuous Delivery
- LG 5-5: (OPTIONAL) Specify deployment-specific security requirements
- LG 5-6: (OPTIONAL) Explain the role of observability in the Deployment Process
- LG 5-7: (OPTIONAL) Shop options to optimize cost and resource efficiency in the deployment process
- LG 6-2: Understand and properly use observability - differences between metrics, logs, and traces
- LG 6-3: Facilitate troubleshooting in distributed systems
- LG 6-4: (OPTIONAL) Derive Service Level Objectives (SLOs) from quality goals
- LG 6-5: (OPTIONAL) Architecture can support incident management and fast MTTR
- LG 6-6: (OPTIONAL) Contribution of architecture to disaster recovery and business continuity

management

- LG 6-7: (OPTIONAL) Design and conduct chaos engineering based on quality goals

Introduction: General information about the iSAQB Advanced Level

What is taught in an Advanced Level module?

- The iSAQB Advanced Level offers modular training in three areas of competence with flexibly designable training paths. It takes individual inclinations and priorities into account.
- The certification is done as an assignment. The assessment and oral exam is conducted by experts appointed by the iSAQB.

What can Advanced Level (CPSA-A) graduates do?

CPSA-A graduates can:

- Independently and methodically design medium to large IT systems
- In IT systems of medium to high criticality, assume technical and content-related responsibility
- Conceptualize, design, and document actions to achieve quality requirements and support development teams in the implementation of these actions
- Control and execute architecture-relevant communication in medium to large development teams

Requirements for CPSA-A certification

- Successful training and certification as a Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- At least three years of full-time professional experience in the IT sector; collaboration on the design and development of at least two different IT systems
  - Exceptions are allowed on application (e.g., collaboration on open source projects)
- Training and further education within the scope of iSAQB Advanced Level training courses with a minimum of 70 credit points from at least three different areas of competence
- Successful completion of the CPSA-A certification exam

Essentials

What does the module “FLEX” convey?

The module presents FLEX to the participants how to build very flexible systems with modern architecture approaches. We start by looking at which quality goals actually motivate a flexible, distributed architecture at all. We also review different options for modularization and the architectural concepts behind them. The participants learn how communication structures between teams and their organisation impacts their architecture decisions and how to organize macro architecture decision making.

At the end of the module, the participants know the basics of architecture styles like microservices and self-contained systems and are able to design systems based on these concepts. They will be able to suggest a meaningful business decomposition that includes insights stemming from the concepts of Continuous Delivery and sustainable service operations.

Curriculum Structure and Recommended Durations

Content	Recommended duration (minutes)	Exercise time (minutes)
1. Why approach Flexible Systems	60	30
2. Modularization of Systems of Systems	120	60
3. Software Modules and the Organisation	90	60
4. Integration Methods & Protocols	150	60
5. Deployment & Runtime/ Platform Methods	90	90
6. Service Operation Models	90	60
Total	600 (10h)	360 (6h)

In this context, "minimum duration" refers to teaching time without exercises. The OPTIONAL content and learning objectives can be added or omitted by any trainer, but should be communicated to the participants in advance. The time breakdown between theory and exercises is only a recommendation. The design of the examples and exercises aren’t specified in this curriculum

Duration, Teaching Method and Further Details

The times stated below are recommendations. The duration of a training course on the FLEX module should be at least 3 days, but may be longer. Providers may differ in terms of duration, teaching method, type and structure of the exercises, and the detailed course structure. In particular, the curriculum provides no specifications on the nature of the examples and exercises.

Licensed training courses for the FLEX module contribute the following credit points towards admission to the final Advanced Level certification exam:

Methodical Competence:	10 Points

Technical Competence:	20 Points
Communicative Competence:	00 Points

Prerequisites

Participants **should** have the following knowledge and / or experience:

- Fundamentals of the description of architectures using various views, comprehen- sive concepts, design decisions, boundary conditions etc., as taught in the CPSA-F (Foundation Level).
- Experience with implementation and architecture in agile projects.
- Experiences from the development and architecture of classical systems with the typical challenges.

Useful for understanding some concepts are also:

- Distributed systems
  - Problems and challenges in the implementation of distributed systems
  - Typical distributed algorithms
  - Internet protocols
- Knowledge about modularisations
  - Functional modularisation
  - Technical implementations like packages or libraries
- Classical operation and deployment processes

Structure of the Curriculum

The individual sections of the curriculum are described according to the following structure:

- **Terms/principles**: Essential core terms of this topic.
- **Teaching/practice time**: Defines the minimum amount of teaching and practice time that must be spent on this topic or its practice in an accredited training course.
- **Learning goals**: Describes the content to be conveyed including its core terms and principles.

This section therefore also outlines the skills to be acquired in corresponding training courses.

The structure of the chapters follows a clear progression:

- First, understand the big picture
- Then, know the prerequisites
- Grasp the organizational aspects
- Prepare and evaluate specific architectural decisions
- Keep long-term goals in mind
- And finally: Be able to argue decisions and explain them soundly

Supplementary Information, Terms, Translations

To the extent necessary for understanding the curriculum, we have added definitions of technical terms to the iSAQB glossary and complemented them by references to (translated) literature.

1. Why approach Flexible Systems

Duration: 60 min	Practice time: 30 min

1.1. Terms and Principles

Availability, resilience, time-to-market, flexibility, predictability, reproducibility, internet/web scale, distributed systems, parallelizing feature development, evolution of architecture (build for replacement), heterogenity, automation.

1.2. Learning Goals

Chapter Structure: A Clear Learning Progression

- First, understand the big picture
- Then, grasp the prerequisites
- Master the organizational aspects
- Prepare and evaluate concrete architectural decisions
- Keep long-term goals in focus
- Finally, justify decisions with solid reasoning

LG 1-1: Classify Key Topics and Industry Buzzwords

- Explain the differences between architectural styles like microservices, self-contained systems, moduliths, and monoliths
- Understand the role of DevOps and Continuous Delivery in flexible architectures
- Recognize the relationship between cloud computing and flexible architectures
- Evaluate current architectural trends based on specific quality goals

LG 1-2: Understand and Analyze Prerequisites for Distributed Systems

- Identify key technical requirements for distributed systems
- Recognize how architecture influences rapid feature delivery
- Understand how team dependencies affect development velocity
- Identify essential team skills required for distributed systems
- Understand the role of uniform development environments in error reduction and reproducibility
- Recognize the relationship between CI, CD, and architecture
- Understand why automation, repeatability, and resilience are crucial for distributed systems
- Evaluate benefits of different isolation types (devtime, runtime, deployment, team)

LG 1-3: Communicate and Adapt Trade-offs of Presented Architecture Types

- Weigh advantages and disadvantages of different architectural styles
- Know how to adapt architectural decisions to business context
- Understand and use remote communication pros and cons as decision criteria for or against distribution

- Know different isolation types to find the right balance between isolation and complexity
- Understand when combining different architectural styles makes sense
- Realistically assess costs of distribution and service communication

LG 1-4: Define Long-term Quality Goals of Flexible Architectures

- Know core quality characteristics of flexible architectures
- Understand why flexibility and rapid feedback are strategic goals
- Explain why the balance between short-term and long-term optimization isn't always 50:50
- Develop methods to measure quality goals
- Break down long-term goals into incremental improvements and show relationships
- Understand the role of reproducibility and predictability
- Recognize automation capability as a quality goal

LG 1-5: Explain and justify Typical Architectural Decisions in Flexible Architectures

- Justify architectural decisions through business goals and quality scenarios
- Explain the necessity of specific architectural patterns
- Make architectural decision costs transparent
- Communicate the value of architectural flexibility
- Justify decisions for or against remote communication
- Justify the choice of isolation boundaries

1.3. References

[Wolff 2016], [Humble, et al. 2014], [Lewis, Fowler, et al. 2013], [Takada 2013]

2. Modularization of Systems of Systems

Duration: 120 min	Practice time: 60 min

2.1. Terms and Principles

- Motivation for decompositioning into smaller systems
- Different kinds of modularisation, coupling
- System limits as a way of isolation
- Hierarchical structure
- Application, Self-Contained System, Microservice
- Domain-Driven Design Concepts and "Strategic Design", Bounded Contexts

2.2. Learning Goals

LG 2-1: Designing decomposition into components based on requirements

- Decomposition approaches: Creating a system decomposition into components while considering functional and technical requirements.
- Strategic Design: Using Domain-Driven Design (DDD) to define module boundaries based on business domains (e.g., Bounded Contexts).
- Hierarchical structures: Considering the hierarchy of modules during decomposition, such as subdomains and context mapping.
- Naming with clear semantics: Components require a name and a description that leave no ambiguity about their purpose and function.

LG 2-2: Describing and justifying different types of components

- Types of modules: Defining and identifying characteristics of different components, such as microservices, self-contained systems, and deployment monoliths.
- Lifecycle of components: Explaining how a component is created, integrated, tested, deployed, and scaled.
- Levels of isolation: Differentiating modular isolation in code, runtime environments, and network interfaces.

LG 2-3: Evaluating and selecting modularization concepts

- Technical modularization: Assessing concepts such as files, libraries, processes, microservices, or self-contained systems.
- Integration and coupling: Analyzing levels of coupling (source code, compile-time, network protocols) and their implications.
- Considering quality goals: Selecting modularization concepts based on requirements such as "development parallelization" or "independent module deployment."

LG 2-4: Assessing modularization strategies

- Strategies and consequences: Evaluating the impact of different modularization approaches, e.g., monolith vs. distributed modules.

- Technology dependency: How modularization technologies (e.g., containers or runtime environments) influence strategies.
- Effort-benefit trade-offs: Weighing organizational and technical efforts against expected benefits.

LG 2-5: Contrasting the costs and benefits of modularization strategies

- Integration vs. decentralization: Identifying organizational and technical costs for integrating modular systems.
- Expected benefits: Assessing advantages such as independent deployment, parallelization of work, replaceability of technologies, and improved system comprehensibility due to clearly named modules and integration points.
- Module boundaries and complexity: Analyzing how module boundary choices affect complexity, coordination effort for changes, and development and maintenance costs.

.

2.3. References

[Eric Evans 2003], [Lewis, Fowler, et al. 2013], [Wolff 2018], [Sam Newman 2021], [Parnas 1972]

3. Software Modules and the Organization

Duration: 90 min	Practice time: 60 min

3.1. Terms and Principles

Conway's Law, Communication structures, Context Mapping, strategic design, stakeholder management, Team Topologies, socio-technical Architectures, teamautonomy, Team and system boundaries, Macro- vs. Micro architecture decisons.

3.2. Learning Goals

LG 3-1: Analyze and name the interaction between architecture types and organization

- Explain how relationships between teams or organizational units influence the architecture of systems, impacting coordination efforts and time-to-market
- Recognize how Conway's Law affects the development speed and adaptability of systems
- Analyze and explain the relationship between team boundaries and system boundaries
- (OPTIONAL) Explain the balance between technical and organizational structure
- Explain the differences between build-time and runtime dependencies for software development processes
- Understand the parallelizability of software development tasks as an architectural goal
- (OPTIONAL) Describe how organizational changes affect software and vice versa, enabling informed decisions for organizational and software architecture

LG 3-2: Consider the organization's communication structure when decomposing

- Conway's Law: Understand the significance and impact of the organization's communication structure on the choice and design of module boundaries.
- Autonomy and Collaboration: Ensure development autonomy through modular cuts along business boundaries.

LG 3-3: Use context maps for stakeholder management

- Use context maps from Domain-Driven Design (DDD) to represent the current state (AS-IS) and a desired future state (TO-BE) of a system landscape.
- (OPTIONAL) Understand and apply context maps as a communication and planning tool depending on the target audience (developers vs. management).

LG 3-4: (OPTIONAL) Use terms like team organization and socio-technical architectures confidently

- Classify terms such as team organization, socio-technical architectures, and similar concepts.
- Explain their meaning in the context of modern software development.
- Develop an understanding of the link between technical and social aspects in architecture design.

LG 3-5: Identify macro-architecture decisions made outside your sphere of influence

- Understand the difference between micro and macro architecture to identify and proactively address potential constraints and coordination efforts.

- Understand the distinction between macro and micro architecture to identify potential limitations and opportunities in deployment strategies.
- Analyze macro-architecture decisions such as communication protocols, operational standards, and platform requirements, and evaluate their impact on deployment and runtime methods.
- Recognize the importance of central platform standards for the efficiency of communication and deployments/software deliveries.

3.3. References

[Skelton, Pais 2019], [Eric Evans 2003], [Vossen, Haselmann, Hoeren 2012], [Conway 1968], [Baxter, Sommerville 2011]

4. Integration methods & protocols

Duration: 150 min	Practice time: 60 min

4.1. Terms and Principles

4.2. Learning Goals

LG 4-1: Compare Integration Strategies (Using the Example of DDD Strategic Design)

- Understand and explain Strategic Design from Domain-Driven Design (DDD) and its relationship patterns (e.g., Anti-Corruption Layer, Shared Kernel, Customer/Supplier).
- Represent and describe the module boundaries of a system using "Bounded Context" and perform AS-IS/TO-BE comparisons.
- Justify which patterns from Strategic Design can or cannot be used for an integration/interface based on quality goals.

LG 4-2: Select and Justify Technical Integration Mechanisms

- Evaluate the advantages and disadvantages of technical integration mechanisms such as frontend integration, middle-tier integration (REST, RPC, Messaging), or database integration based on specific quality goals of the system and the required experience/skills of the development teams.
- (Optional) Compare frontend integration (e.g., links, client-side includes, microfrontends) and middleware for legacy systems and their impacts.
- (Optional) Practical application in brownfield scenarios: integration of legacy data models through transformation and mapping.

LG 4-3: Explain and Select Consistency Models (CAP Theorem)

- Explain the CAP theorem: trade-offs between consistency, availability, and partition tolerance.
- Understand the necessity of partition tolerance.
- Compare ACID and BASE models: guarantees, compromises, and trade-offs, as well as their resulting applicability in distributed systems.
- Explain the difference between "consistency" in ACID and BASE using practical examples.
- (Optional) Provide practical examples of ACID and BASE transactions in distributed systems.
- Justify CP vs. AP system design based on quality goals.
- (Optional) Practical example: choosing a suitable consistency model based on system requirements such as latency and scalability.
- (Optional) Know product examples from different categories (e.g., NoSQL, configuration tools, service discovery).

LG 4-4: Identify and select Resilience Patterns

- (Optional) Messaging Patterns: Request/Reply, Publish/Subscribe, and their resilience properties.
- Resilience strategies for decentralized data storage and redundant architecture.
- Understand the differences between high availability and resilience and provide practical examples.

- (Optional) Use mechanisms like Circuit Breaker, Bulkhead, and Graceful Degradation to ensure availability in scenarios where system components have different availability requirements.
- Understand the impact of resilience patterns on well-known operations metrics like MTTR and MTBF and their interdependency with fast delivery of bugfixes/releases.

LG 4-5: Understand and Consider Security Implications of Integration Methods

- (Optional) Compare authentication and authorization mechanisms (e.g., OAuth, Kerberos).
- Analyze the impact of synchronous (e.g., RPC) vs. asynchronous integration (e.g., Messaging) on security and data integrity.
- Implement secure interfaces and macroarchitecture for distributed systems.

LG 4-6: (Optional) Understand and Design Event-Driven Architectures (EDA)

- Introduction to Event-Driven Architectures (EDA): publishing domain events to decouple systems.
- Design an EDA with messaging systems like RabbitMQ or Kafka.
- Address challenges and solutions for backpressure and decentralized data storage.

4.3. References

[Eric Evans 2003], [Parecki et al. 2006], [Hohpe, Woolf 2003], [Tanenbaum, van Steen 2006], [Lamport 1998], [Brewer 2000], [Takada 2013], [Nygard 2018], [Hanmer 2007], [Hamilton 2007]

5. Deployment & Runtime/Platform Methods

Duration: 90 min	Practice time: 90 min

5.1. Terms and Principles

Continuous Deployment, DevOps, Platform Engineering, Infrastructure as Code, Configuration Management, cost efficiency, zero downtime deployments.

5.2. Learning Goals

LG 5-1: Specify prerequisites and implications for Continuous Deployment

- Utilize quality goals to explain CI/CD pipeline requirements, including automation levels, testing, and infrastructure integration
- Understand DevOps' role in Continuous Deployment and recognize organizational implications
- Evaluate differences between traditional and modern deployment approaches, such as Immutable Infrastructure or Infrastructure as Code
- Assess risks and benefits of Continuous Deployment, including zero downtime, in various project contexts

LG 5-2: (OPTIONAL) Explain and select differences between IaaS, PaaS, CaaS, and FaaS

- Describe technical characteristics and application areas of different platform approaches (IaaS, PaaS, CaaS, FaaS)
- Establish criteria for selecting appropriate platform approaches based on project requirements like scalability, flexibility, and costs
- Analyze platform choice impacts on deployment strategy and make decisions regarding containerization (e.g., Docker, Kubernetes) or serverless approaches

LG 5-3: Identify and select Zero Downtime methodologies and their implications

- Explain various zero-downtime strategies (Blue-Green Deployment, Canary Releases, Rolling Updates) and analyze their pros and cons
- Explain the role of Immutable Infrastructure and automation for zero downtime
- Evaluate challenges and architectural requirements for zero-downtime deployments

LG 5-4: Explain differences between Continuous Integration, Continuous Deployment, and Continuous Delivery

- Explain the significance of Continuous Integration (CI) as a foundation for Continuous Deployment and Delivery
- Analyze differences in objectives and automation levels between the three concepts
- Describe how architectural decisions influence CI/CD pipeline implementation

LG 5-5: (OPTIONAL) Specify deployment-specific security requirements

- Explain requirements for protecting secrets (e.g., API keys, certificates) and security policies in the deployment process

- Integrate concepts like access controls, encryption, and compliance requirements into deployment strategies
- Implement and evaluate tools for managing security-relevant information

LG 5-6: (OPTIONAL) Explain the role of observability in the Deployment Process

- Explain monitoring system requirements and their importance for deployment success
- Explain the role of centralized logging and metrics systems (e.g., Elastic Stack, Prometheus, Grafana)
- Develop strategies to identify and address potential problems in the deployment process early

LG 5-7: (OPTIONAL) Shop options to optimize cost and resource efficiency in the deployment process

- Evaluate methods for cost optimization in deployment and runtime (e.g., Reserved Instances, Spot Instances, Serverless)
- Analyze how cloud and virtualization strategy choices impact operational costs and resource utilization
- Develop deployment methods that maximize scalability and efficiency
- Explain negative impact of cost optimization to flexibility and speed in the deployment process

5.3. References

[Vossen, Haselmann, Hoeren 2012], [Wolff, Müller, Löwenstein 2013], [Humble, et al. 2010], [Wolff 2016]

6. Service Operation Models

Duration: 90 min	Practice time: 60 min

6.1. Terms and Principles

Observability, Monitoring, Operation Models, MTBF vs. MTTR, Logging, Tracing, Metrics, Alerting, Service Level Objectives, Chaos Engineering.

6.2. Learning Goals

6.3. LG 6-1: Explain and choose different operational models and their impacts

- Operations Team vs. You Build It, You Run It:
  - Compare centralized operational models (dedicated operations team) with decentralized approaches like DevOps and "You Build It, You Run It."
  - Impact of operational models on responsibilities, communication paths, and problem-solving times.
  - The role of DevOps practices in integrating operations and development.
  - Decide on centralized or decentralized application operations based on the context - such as organization and skillset.
- MTBF (Mean Time Between Failures) vs. MTTR (Mean Time to Repair):
  - Importance of both metrics and their impact on operational models and culture.
  - (OPTIONAL) Strategies for optimizing MTBF and MTTR in the context of the chosen architecture and their effects on continuous deployment.
  - Relationship between operational models and quality criteria like changeability, reliability, and fault tolerance.

LG 6-2: Understand and properly use observability - differences between metrics, logs, and traces

- Definition and importance of observability:
  - Distinction from monitoring and explanation of the key aspects of observability.
  - The role of logs, metrics, and traces in troubleshooting and system monitoring and understanding.
  - Participants should understand the differences between logging, monitoring, and metrics collection, and the associated differences in tools used for these tasks.
- Use of logs, metrics, and traces:
  - Examples of typical use cases.
  - Tools and technologies for observability (e.g., Elastic Stack, Prometheus, Jaeger).
  - Logs are events, metrics are states at a specific point in time.
  - Monitoring can include both business-related and technical data.
  - The right selection of data is crucial for a reliable understanding of the runtime behavior of the system.

    - Participants should understand which information to obtain from traces, which from log data, and which to derive (preferably) through code instrumentation with metric probes.
- Architectural support for observability:
    - Architectural decisions that facilitate the integration and use of observability tools.
    - Supporting operations as part of architectural concepts.
    - Centralized log management and alternatives, as well as their impact on architecture (performance overhead, memory consumption, latency, locking vs. log loss).
    - Structure of typical metric architectures (collection, sampling, persistence, querying, visualization).
    - Differentiation between business, application, and system metrics.
    - Importance of key system and application metrics independent of specific tools.

LG 6-3: Facilitate troubleshooting in distributed systems

- Challenges in distributed systems:
    - Typical issues like network latencies, partial failures, and inconsistent states.
    - Complexity of troubleshooting due to asynchronous communication and decentralized data storage.
- Tools and strategies for troubleshooting:
    - Use of distributed tracing (e.g., OpenTelemetry, Jaeger) to visualize request flows.
    - Logging and monitoring strategies for distributed systems.
    - Importance of centralized data platforms (e.g., log aggregation) for troubleshooting.
    - Participants should be able to make architectural decisions that best support the use of appropriate tools while considering efficient use of system resources.
    - Depending on the specific project scenario, they can focus on logging, monitoring, and the necessary data.
- Architectural patterns to support troubleshooting:
    - Patterns like Circuit Breaker, Retry mechanisms, and Bulkhead for fault isolation.
    - Architectural decisions for transparent error reporting and propagation.

LG 6-4: (OPTIONAL) Derive Service Level Objectives (SLOs) from quality goals

- Definition and importance of SLOs:
    - Difference between SLOs, SLAs (Service Level Agreements), and SLIs (Service Level Indicators).
    - Deriving SLOs from quality scenarios, non-functional requirements, and business requirements.
- Architectural support for SLOs:
    - Influence of architectural decisions on meeting SLOs (e.g., scalability, redundancy).
    - Examples of typical SLOs such as availability, response time, and error rate.
- Tools for monitoring and measuring SLOs:
    - Automated monitoring and reporting mechanisms for SLO tracking.

LG 6-5: (OPTIONAL) Architecture can support incident management and fast MTTR

- Architectural decisions to reduce MTTR:
  - Use of self-healing mechanisms and automated recovery.
  - Support through observability-focused architectural decisions.
- Incident management and resilience:
  - Architectural patterns like Circuit Breaker, fallback strategies, and rate limiting to minimize failures.
  - Influence of deployments and rollbacks on response times during incidents.

LG 6-6: (OPTIONAL) Contribution of architecture to disaster recovery and business continuity management

- Disaster recovery strategies:
  - Use of backup and recovery mechanisms (snapshots, database replication).
  - Requirements for architectures to meet recovery goals like RTO (Recovery Time Objective) and RPO (Recovery Point Objective).
- Business Continuity Management (BCM) for architects:
  - Architectural support for high availability and failover capabilities.
  - Technologies such as multi-region deployments, failover strategies, and geo-redundancy.
- Testing disaster recovery strategies:
  - Implementation and validation of recovery plans through simulations and drills.

LG 6-7: (OPTIONAL) Design and conduct chaos engineering based on quality goals

- Basics of chaos engineering: objectives and principles (e.g., "testing in production" to increase resilience).
- Designing chaos experiments:
  - Deriving experiments from quality goals like availability and fault tolerance.
  - Identifying weaknesses and validating them through targeted disruptions.
- Architectural support for chaos engineering:
  - Use of resilience-enhancing patterns like Circuit Breaker and fallback mechanisms.
  - Integration of chaos testing tools (e.g., Gremlin, Chaos Monkey) into the CI/CD pipeline.

6.4. References

[Wolff 2016], [Nygard 2018]

References

This section contains references that are cited in the curriculum.

B

- [Baxter, Sommerville 2011] Gordon Baxter, Ian Sommerville: Sociotechnical Systems Design: Evolving Theory and Practice, 2011, https://academic.oup.com/iwc/article/23/1/4/693091
- [Brewer 2000] Eric Brewer, Towards Robust Distributed Systems, PODC Keynote, July-19-2000

C

- [Conway 1968] Melvin E. Conway, How Do Committees Invent?, Datamation, 1968, https://en.wikipedia.org/wiki/Conway%27s_law

E

- [Eric Evans 2003] Eric Evans: Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison- Wesley Professional, 2003

H

- [Hamilton 2007] James Hamilton, On Designing and Deploying Internet-Scale Services, 21st LISA Conference 2007
- [Hanmer 2007] Robert S. Hanmer, Patterns for Fault Tolerant Software, Wiley, 2007
- [Hohpe, Woolf 2003] Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley, 2003, ISBN 978-0-32120-068-6
- [Humble, et al. 2010] Jez Humble, David Farley: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010, ISBN 978-0-32160-191-9
- [Humble, et al. 2014] Jez Humble, Barry O'Reilly, Joanne Molesky: Lean Enterprise: Adopting Continuous Delivery, DevOps, and Lean Startup at Scale, O'Reilly 2014, ISBN 978-1-44936-842-5

L

- [Lewis, Fowler, et al. 2013] James Lewis, Martin Fowler, et al.: Microservices - http://martinfowler.com/articles/microservices.html, 2013
- [Lamport 1998] Leslie Lamport, The Part-Time Parliament, ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169

N

- [Sam Newman 2021] Sam Newman: Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2nd ed. edition, 2021, ISBN 978-1-49203-402-5
- [Nygard 2018] Michael T. Nygard, Release It!, 2. Auflage, Pragmatic Bookshelf, 2018

O

- [Parecki et al. 2006] Aaron Parecki et al., Explaining OAuth 2.0 http://oauth.net/

P

- [Parnas 1972] David Parnas, On the Criteria To Be Used in Decomposing Systems into Modules. Communications of the ACM 15(12):1053—1058, 1972.

T

- [Takada 2013] Mikito Takada, Distributed Systems for Fun and Profit, http://book.mixu.net/distsys/ (Guter Einstieg und Überblick)
- [Tanenbaum, van Steen 2006] Andrew Tanenbaum, Marten van Steen, Distributed Systems – Principles and Paradigms, Prentice Hall, 2nd Edition, 2006
- [Skelton, Pais 2019] Mathew Skelton, Manuel Pais - TEAM TOPOLOGIES: ORGANIZING BUSINESS AND TECHNOLOGY TEAMS FOR FAST FLOW, IT Revolution, 2019, ISBN 978-1-942788-81-2

V

- [Vossen, Haselmann, Hoeren 2012] Gottfried Vossen, Till Haselmann, Thomas Hoeren: Cloud-Computing für Unternehmen: Technische, wirtschaftliche, rechtliche und organisatorische Aspekte, dpunkt, 2012, ISBN 978-3-89864-808-0

W

- [Wolff 2016] Eberhard Wolff: Continuous Delivery: Der pragmatische Einstieg, 2. Auflage, dpunkt, 2016, ISBN 978-3-86490-371-7
- [Wolff 2018] Eberhard Wolff: Microservices - Grundlagen flexibler Software Architekturen, 2. Auflage, dpunkt, 2018, ISBN 978-3-86490-555-1
- [Wolff, Müller, Löwenstein 2013] Eberhard Wolff, Stephan Müller, Bernhard Löwenstein: PaaS - Die wichtigsten Java Clouds auf einen Blick, entwickler.press, 2013