Curriculum for Certified Professional for Software Architecture (CPSA)® Advanced Level

AGENTA – *Architecture for Agentic Software Engineering Contexts*

2026.1-rev5-EN-20260731

License and Copyright

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.

Table of Contents

Learning Goals Overview

- LG 1-1: Know a selection of the most important terms for AI disciplines and their differences
- LG 1-2: Being familiar with the fundamentals of large language models (LLMs) and their core concepts and terminology
- LG 1-3: Understand the impact of prompt and context engineering on the quality of LLM outputs
- LG 1-4: Know what AI agents are and what agentic engineering means
- LG 1-5: Know the different practical approaches (schools) to agentic engineering
- LG 1-6: Understand the relation of AI-supported and agentic engineering to classical software architecture work
- LG 2-1: Understand the impact of AI on the lifecycle of architectural decisions
- LG 2-2: Be able to refine quality goals into architectural drivers (top-down)
- LG 2-3: Be able to identify architectural / structural pain points (bottom-up)
- LG 2-4: Be able to use AI for design exploration and architectural variant generation
- LG 2-5: Be able to treat architectural decisions as testable hypotheses
- LG 2-6: Understand the limitations of AI in architectural decision support
- LG 2-7: Be able to shift architectural work from implementation to orchestration
- LG 3-1: Understand why and how the information available to an AI agent determines the quality of its output
- LG 3-2: Know how to design agent-facing representations of architectural knowledge
- LG 3-3: Be able to design delivery mechanisms that make architectural knowledge available to agents at the right time
- LG 3-4: Be able to assess and improve how effectively architectural knowledge reaches agents in a given project
- LG 4-1: Understand why AI agents require explicit architectural control
- LG 4-2: Be able to design a harness that enforces architectural decisions on AI-generated work
- LG 4-3: Be able to decide where in the development process architectural controls should be applied
- LG 4-4: Be able to introduce architectural controls for AI agents into existing codebases
- LG 4-5: Be able to design processes that improve architectural controls over time
- LG 5-1: Understand the difference between as-is, to-be, human-facing, and agent-facing architecture documentation
- LG 5-2: Understand how they maintain architectural decisions and documentation through agentic development loops
- LG 5-3: Know how to use AI and agents to extract architectural information from existing system parts
- LG 5-4: Understand how to generate architectural views from gathered architectural information
- LG 6-1: Understand, classify, and properly address the legal framework conditions when using AI tools in the organization
- LG 6-2: Know how to ensure traceability of AI-generated outputs and their context

- LG 6-3: Know how to design consistent scaling of AI governance across teams and processes
- LG 6-4: Understand the environmental costs of AI-supported software development and derive sustainable usage strategies
- LG 6-5: Know how to handle sensitive information securely, confidentially, and in compliance with data protection when using AI tools
- LG 6-6: Know how to ensure the security of agent and tool integrations (prompt injection, tool misuse, sandboxing)
- LG 7-1: Know about the importance of code ownership and the preservation of analytical skills
- LG 7-2: Be able to preserve architectural judgment when using coding agents
- LG 7-3: Know what drives the adoption of AI-supported development practices in organizations
- LG 7-4: Know about evolving roles, teams, and architectural scope for AI-supported organizations

Introduction: General information about the iSAQB Advanced Level

What is taught in an Advanced Level module?

The module can be attended independently of a CPSA-F certification.

- 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 “AGENTA” convey?

The module presents AGENTA to the participants as an advanced approach to software architecture in contexts where generative AI, LLMs, and agents are used as working tools in software development. It focuses on how software engineers and architects can use AI to support architectural work, how architectural knowledge and constraints can be made available to agents, and how architectural control, governance, and accountability can be maintained in AI-supported engineering environments. The module also addresses the extraction of architectural information from existing systems and the changing role of the architect in organizations that increasingly rely on AI-supported development.

At the end of the module, the participants know the fundamental concepts, opportunities, and limitations of AI-supported and agentic engineering in the context of software architecture. They are able to use AI to support architectural decision-making, provide architectural knowledge to agents, establish mechanisms that keep AI-generated work aligned with architectural goals, and derive architectural insights and documentation from existing development artifacts. They also know how to introduce governance, traceability, and quality controls for AI usage and are able to adapt architectural work, collaboration, and responsibilities to organizations in which AI becomes part of everyday engineering practice.

Curriculum Structure and Recommended Durations

Content	Duration (minutes)	Practice (minutes)	Total (minutes)
1. Introduction and Fundamentals	60	0	60
2. AI-Assisted Architectural Decision-Making	90	60	150
3. Providing Architectural Knowledge to Agents	150	60	210
4. Keeping Agents Aligned with Architectural Goals	90	60	150
5. Architecture Information Extraction	105	75	180
6. Governance and Quality Gates for AI Usage	60	30	90
7. The Evolving Discipline of Software Architecture	30	30	60
Total	585	315	900

Duration, Teaching Method and Further Details

The times stated below are recommendations. The duration of a training course on the AGENTA 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 AGENTA module contribute the following credit points towards admission to the final Advanced Level certification exam:

Methodical Competence:	20 Points
Technical Competence:	10 Points
Communicative Competence:	0 Points

Prerequisites

Participants **should** have the following prerequisite knowledge:

- Practical experience in software development or software architecture
- Basic understanding of software architecture concepts and typical architecture artifacts
- Basic experience with modern software development processes and collaboration in development teams
- Basic understanding of source code repositories, build pipelines, and common development tooling

Knowledge in the following areas may be **helpful** for understanding some concepts:

- Software Architecture:
  - Experience with architectural decisions and trade-off analysis
  - Knowledge of quality requirements and constraints
  - Experience with architecture documentation, views, and Architecture Decision Records (ADRs)
  - Understanding of architecture reviews and modernization of existing systems
- Software Development and Delivery:
  - Experience with continuous integration and delivery
  - Knowledge of static analysis, testing, and code review practices
  - Understanding of system decomposition, interfaces, and integration styles
- AI-supported Engineering:
  - Basic understanding of generative AI and Large Language Models
  - First practical experience with AI assistants or coding agents
  - Awareness of typical limitations of AI systems, such as hallucinations and non-deterministic behavior
- Requirements and Domain Understanding:
  - Experience with translating business or quality goals into technical decisions
  - Understanding of domain concepts, business rules, and stakeholder communication

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.

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. Introduction and Fundamentals

Duration: 60 min	Practice time: 0 min

1.1. Terms and Principles

Definition of basic concepts, Artificial intelligence, Large Language Models, Prompt Engineering, Agents

1.2. Goal of the chapter

Establish and revisit the fundamentals necessary for understanding all subsequent topics in this curriculum. Help to bring along participants with no prior AI experience so that the chapters building on it become comprehensible.

1.3. Learning Goals

LG 1-1: Know a selection of the most important terms for AI disciplines and their differences

Participants can distinguish the following important disciplines in the context of agentic engineering and know which tasks they are used for:

- AI-assisted, augmented, agentic coding
- AI orchestration
- Agentic workflow design
- Agentic engineering vs vibe coding
- Harness, context, prompt, and human-in-the-loop engineering

LG 1-2: Being familiar with the fundamentals of large language models (LLMs) and their core concepts and terminology

- Participants can explain the terms AI, ML, GenAI, and LLMs and classify how they relate to each other.
- Participants know the terms data, model, training, and inference.
- Participants know that data is the central foundation of AI (garbage in, garbage out).
- Participants understand LLMs as probabilistic tools whose outputs are samples from a distribution and can explain the implications for architecture work.
- Participants know basic concepts of LLMs such as tokens, prompts, context, and reasoning.
- Participants know the most important characteristics of LLMs and can make well-founded decisions about using specific providers and models.
- Participants know the capabilities (e.g., language, knowledge, structured output) and limitations of LLMs (e.g., hallucinations, bias, knowledge cutoff, context window).
- Participants can weigh whether tasks should be done by humans, classic software tools, or LLM agents.

LG 1-3: Understand the impact of prompt and context engineering on the quality of LLM outputs

- Participants understand the concept of “context engineering” and know why it matters.
- Participants know a selection of prompt-engineering techniques to optimize the quality of outputs, e.g., role playing, consistency, or repeating.

LG 1-4: Know what AI agents are and what agentic engineering means

- Participants know the difference between assistants and agents.
- Participants have a basic understanding of the agentic loop (ReAct pattern): prompting, reasoning, tool usage, state management (memory).
- Participants are familiar with advanced techniques such as planning, sub-agents, hand-offs, and human-in-the-loop.
- Participants have a basic understanding of the structure and functioning of modern coding agents.
- Participants know the strengths and broad applicability of the agent-based approach in software development and architecture work.

LG 1-5: Know the different practical approaches (schools) to agentic engineering

- Participants understand autonomy as a spectrum with multiple levels and can place tools/workflows appropriately. They understand how the levels differ across typical activities (from idea to feedback).
- Participants can name concrete current practices in agentic engineering, such as Spec-Driven Development, Agent Swarms, Conversational Programming, and related workflows, and classify them within the autonomy spectrum.

LG 1-6: Understand the relation of AI-supported and agentic engineering to classical software architecture work

- Participants know which activities are typically referred to as "architecture work" and understand that AI can affect these activities.
- Participants can explain how AI assistants and agents can support core architecture activities, such as quality goal definition, decision documentation, and architecture reviews. They can identify which activities can be supported, accelerated, or partially automated by AI, and where human architectural judgment remains necessary.
- Participants can map architecture work to agentic engineering. Instead of merely augmenting traditional activities, they can anchor architecture work in agentic processes and tools — for example, by embedding architecture principles through context engineering, supporting architecture decisions with guardrails, evaluating alternatives by implementing them in agentic loops, and continuously validating architectural assumptions through automated feedback.

1.4. References

[Bass et al. 2021], [Beck 2025], [Boeckeler 2025c], [Boeckeler 2025d], [Brown et al. 2020], [Feng et al. 2025] [Karpathy 2025b], [Park et. al. 2023], [Raschka 2026], [Starke 2024], [Willison 2026a], [Jahic et al. 2024], [Ouyang 2025]

2. AI-Assisted Architectural Decision-Making

Duration: 90 min	Practice time: 60 min

2.1. Terms and Principles

Architectural decision lifecycle, evidence-based architecture, architectural drivers, quality scenarios, fracture planes, set-based design, trade-off analysis, hypothesis-driven development, lightweight experiments, LLM model limitations, LLM model switching, human-in-the-loop

2.2. Learning Goals

LG 2-1: Understand the impact of AI on the lifecycle of architectural decisions

Participants understand the phases of an architectural decision lifecycle, from problem identification to eventual deprecation. They understand that this lifecycle is cyclical rather than linear and can explain how agentic engineering affects this lifecycle (effects, risks and opportunities). In detail, participants:

- understand the phases of the decision lifecycle: problem identification, option exploration, (in)validation of alternatives, decision making, decision establishment, deprecation, and iteration
- understand the effect of agentic engineering: It can accelerate exploration and (prototype) development, and - paired with continuous validation - can increase decision cycle frequency
- understand that it is important to establish feedback loops from development, testing, and operations into ongoing architectural reasoning
- can explain how these effects make it possible to establish a continuous, evidence based architecture process, and why this is a good thing.

LG 2-2: Be able to refine quality goals into architectural drivers (top-down)

Participants are able to decompose quality goals into concrete architectural drivers and use them to reason about the output of agentic development. They understand why it is important to handle these drivers and the resulting architectural decisions separately and distinctively. In detail, participants:

- can decompose quality goals into concrete, assessable criteria, like quality scenarios
- can identify conflicts or contradictions between quality goals, constraints, and architectural intentions
- know how to use these drivers to support goal-oriented design and development of software solutions (see chapter 3)
- understand why it is important to have different concepts in the agentic workflow to handle architectural drivers & goals on one side and architectural decisions & concepts on the other side.

LG 2-3: Be able to identify architectural / structural pain points (bottom-up)

Participants are able to identify architectural drivers and structural pain points by using a mixture of established tools and AI-assisted techniques.

- can explain bottom-up needs for architectural work, especially in agentic engineering contexts: structural weaknesses, unintended drift, or emerging complexity in evolving systems
- know established tools to identify architectural drivers: technical debt assessment, static structural analysis, monitoring-based feedback and dynamic analysis, metrics and test coverage analysis, architecture reviews

- can use AI-assisted techniques to support bottom-up driver identification: AI-driven architecture reviews and structural assessments, consistency checks and quality assessments

LG 2-4: Be able to use AI for design exploration and architectural variant generation

Participants are able to use AI to support the exploration of architectural options. They understand why architectural drivers, constraints, existing decisions, and architectural principles are important for the generation of architectural variants.

- Generating and refining architectural options for a given problem situation
- Evaluating candidate options against drivers, existing decisions, constraints, and architectural principles
- Understanding the limits of AI-generated suggestions and the omnipresent nature of trade-offs in architectural decisions

LG 2-5: Be able to treat architectural decisions as testable hypotheses

Participants can formulate architectural decisions as hypotheses with explicit assumptions, expected impact on the system's quality characteristics, risks, and validation criteria. They can design lightweight experiments and use the results to support, revise, or reject architectural options.

- Framing architectural decisions as hypotheses with assumptions, expected outcomes, and measurable criteria for validation or invalidation.
- Replacing potentially existing meeting-based approaches for architectural design with experimental loops
- Designing lightweight experiments such as spikes, simulations, prototypes, and side-by-side variant comparisons
- Identifying architectural risks and using targeted experiments to address them
- Using experimental outcomes to support, weaken, revise, or revisit architectural decisions and understand that these agentic experiments can be cheaper than highly developed "specs"

LG 2-6: Understand the limitations of AI in architectural decision support

Participants understand the limitations of AI in architectural reasoning and the support of decision-making. They know why AI-generated results require critical review and understand the need to retain human responsibility for consequential architectural decisions.

- Known risks include model-inherent aspects that need attention: hallucinations, bias, sensitivity to prompt structure and instruction ordering, differences in context effectiveness, potentially sponsored technologies, …
- Participants know the value and risks of switching models during a decision process
- Participants are able to define review checkpoints and know how to maintain human responsibility over consequential architectural decisions by enforcing these checkpoints and using additional methods (e.g., setting boundaries for agentic decisions, enforcing guardrails)

LG 2-7: Be able to shift architectural work from implementation to orchestration

- The course participants can distinguish whether they are directing agents through a fixed plan or composing the agent workflow itself, and can select the appropriate style for a given context.

- The course participants can reason about the effects of orchestration on throughput (especially regarding fewer hand-offs causing less friction), traceability, and the development process as a whole.
- The course participants know about orchestration variants and can identify meaningful checkpoints at which human judgment is required.
- The course participants can identify failure modes specific to orchestration (e.g. runaway agents, infinite loops, lost human override points, cascading errors across orchestrated steps) and can design containment mechanisms that limit their impact.
- The course participants can specify targets, constraints, and acceptance criteria rather than prescribing the exact path, and can explain why this style of guidance is more effective when working with AI agents.
- The course participants understand that orchestration design has direct cost and sustainability implications (token consumption, agentic loop length, tool-call frequency) and can take these into account when shaping orchestration patterns. See also LG 7-4.

2.3. References

[Bass et al. 2021], [Cui et al. 2026], [Dhar et al. 2024], [Diaz-Pace et al. 2024], [Dieste et al. 2024], [Silva et al. 2025], [Toche et. al. 2020], [Toth 2025], [Wohlin et. al. 2012], [Yao et al. 2023]

3. Providing Architectural Knowledge to Agents

Duration: 150 min	Practice time: 60 min

3.1. Terms and Principles

Context engineering, Architectural knowledge delivery to AI agents, Agent-readable architecture artifacts

3.2. Learning Goals

LG 3-1: Understand why and how the information available to an AI agent determines the quality of its output

- Participants can explain the relationship between the information an agent receives and the quality and architectural consistency of its output.
- Participants understand why providing architectural knowledge to agents is a design problem that requires engineering discipline, not just writing instructions.
- Participants can explain the inherent limitation: providing knowledge increases the probability of good results but cannot guarantee them.

LG 3-2: Know how to design agent-facing representations of architectural knowledge

- Participants can identify which architectural decisions, constraints, quality goals, principles, domain concepts, integration rules, and system boundaries are relevant for AI agents working in a given codebase, and can assess which are missing.
- Participants can evaluate different forms of representing architectural knowledge for agent consumption and assess their trade-offs in terms of precision, maintainability, and effectiveness.
- Participants can select suitable representation forms, such as ADRs, architecture rules, context files, knowledge bases, retrieval indexes, diagrams-as-code, or structured constraints.

LG 3-3: Be able to design delivery mechanisms that make architectural knowledge available to agents at the right time

- Participants can assess different strategies for delivering architectural knowledge to agents (e.g. Retrieval Augmented Generation, Skills) and can select appropriate strategies for different situations.
- Participants can reason about who or what should decide when specific knowledge is loaded, and can explain the trade-offs involved.
- Participants understand the architectural implications of standardized integration protocols (e.g. Model Context Protocol) for agent-tool communication and can identify where such mechanisms add value in an organization.

LG 3-4: Be able to assess and improve how effectively architectural knowledge reaches agents in a given project

- Participants can evaluate whether the architectural knowledge currently available to agents in a project is sufficient, well-structured, and effectively delivered.
- Participants understand that more knowledge is not always better. Irrelevant or contradictory information degrades agent performance. They can design strategies that balance comprehensiveness with signal quality.
- Participants can design an iterative process for improving architectural knowledge delivery based on

observed agent behavior.

- Participants are aware that shared documents are not shared understanding and that establishing such involves other disciplines and methodologies (e.g. Domain-driven Design, Collaborative Modeling)

3.3. References

[Vasilopoulos 2026], [Mei et al. 2025], [Anthropic 2025a], [Anthropic 2024], [Boeckeler 2025a], [Boeckeler 2026a], [Boeckeler and Thiagarajan 2025], [Fang et al. 2025], [Hargis et al. 2004], [Karpathy 2025a], [Konrad et al. 2026], [LangChain 2025], [Marco 2026], [Qishuo et. al. 2025], [Rothman 2025], [Willison 2025b]

4. Keeping Agents Aligned with Architectural Goals

Duration: 90 min	Practice time: 60 min

4.1. Terms and Principles

Harness engineering, Architectural constraints for AI agents, Automated architecture compliance, Agent control systems

4.2. Learning Goals

LG 4-1: Understand why AI agents require explicit architectural control

- Participants can explain why AI agents, without explicit constraints, tend to produce work that violates architectural decisions, accumulates technical debt, and introduces inconsistencies.
- Participants can describe the factors that determine an AI agent's behavior (e.g. context and tools) and identify where architects can intervene most effectively.
- Participants understand the relationship between providing architectural knowledge to agents and constraining agent behavior, and can explain why knowledge alone (e.g. in agent's context) is insufficient.
- Participants understand that explicit architectural controls can hinder evolvability and are therefore a temporary, "sinking" investment — maintained only until models and agents can reliably enforce the controls themselves, and reduced as their capabilities grow.

LG 4-2: Be able to design a harness that enforces architectural decisions on AI-generated work

- Participants can design preventive and detective controls that work together to keep AI agents within architectural boundaries and aligned with architectural decisions, quality goals, and system constraints.
- Participants can assess the trade-offs between different types of enforcement mechanisms, in particular the trade-off between precision and coverage, and between speed and depth of analysis.
- Participants can select appropriate enforcement mechanisms for different architectural concerns (e.g., structural rules, quality attributes) and can explain why some are harder to enforce than others — atomic structural rules allow fast, deterministic checks, whereas holistic quality attributes like maintainability or usability yield only holistic feedback that is hard to obtain automatically (cf. atomic vs. holistic fitness functions).
- Participants know about examples for enforcement mechanisms, such as architecture tests, static analysis, linters, dependency rules, policy checks, review gates, sandboxing, and manual approval workflows.

LG 4-3: Be able to decide where in the development process architectural controls should be applied

- Participants can design architectural controls at appropriate points in the development and delivery process, and can justify why different controls belong at different points.
- Participants understand that AI-assisted development can cause architectural drift at a pace that exceeds traditional review cycles, and can design mechanisms for continuous verification.

LG 4-4: Be able to introduce architectural controls for AI agents into existing codebases

- Participants can assess how amenable an existing codebase and technology stack are to architectural control of AI agents, and can identify the properties that make control easier or harder.
- Participants can design a strategy for incrementally introducing architectural controls to systems that were not built with AI-assisted development in mind, and can prioritize which controls to establish first.
- Participants can reason about the trade-off between the effort of establishing controls and the risk of operating AI agents without them.

LG 4-5: Be able to design processes that improve architectural controls over time

- Participants can design feedback processes where failures in AI-generated work inform improvements to the control system itself.
- Participants can reason about the extent to which the improvement of controls can itself be automated, and where human architectural judgment remains essential.

4.3. References

[Anthropic 2025b], [Ashby 1956], [Boeckeler 2026b], [Ford et al. 2023], [GitClear 2025], [Lopopolo 2026], [Morris 2026], [OReilly 2026a]

5. Architecture Information Extraction

Duration: 105 min	Practice time: 75 min

5.1. Terms and Principles

Architecture discovery, architecture recovery, technology detection, pattern recognition, business rule extraction, architecture views, documentation templates, living documentation, Architecture Decision Records (ADRs), knowledge graphs

5.2. Learning Goals

LG 5-1: Understand the difference between as-is, to-be, human-facing, and agent-facing architecture documentation

Participants can distinguish key types of architecture documentation and understand the different role AI plays in creating and maintaining each. They understand that documents are a means to shared understanding, not an end in themselves — and that this principle applies especially in agentic contexts.

- Distinguish **as-is** documentation (current system state — benefits most from AI-tool support) from **to-be** documentation (vision, target architecture - AI plays a fundamentally different and smaller role)
- Understand that vision documents require active human processes: talking, discussing, evaluating, deciding. The quality of the resulting document depends on the quality of those human interactions — AI can support but not replace them
- Know that for human-facing documentation, audience-oriented format (who reads it, for what purpose, in what context) is at least as important as quantity
- Know that agent-facing documentation (e.g. as context for coding agents or AI-assisted workflows) has more options than traditional formats or templates, and that concise, machine-readable representations often outperform traditional documents

LG 5-2: Understand how they maintain architectural decisions and documentation through agentic development loops

Participants understand that agentic development loops continuously evolve the system architecture — often implicitly. They know how to accompany these loops with appropriate decision tracking and documentation practices that ensure traceability and human accountability without becoming a bottleneck.

- Understand that long agentic iteration loops can drift the architecture away from original decisions, and know why this must be deliberately tracked and surfaced
- Know how AI can help detect when existing decisions are being implicitly changed during agentic development, and flag these for human review
- Use ADRs as a core tool for capturing decisions arising from agentic development loops: context, alternatives, rationale, and consequences — including decisions that emerge incrementally rather than in a single moment
- Know how AI can support ADR drafting by collecting and synthesizing information from diverse agentic context sources: context logs, guardrail outputs, chat histories, emails, and meeting minutes. Understand that the resulting draft must go through a human-in-the-loop review — for validation, correction, and completion

- Understand the limitations of AI-generated ADR content: organizational commitment, stakeholder buy-in, implicit knowledge, and the social contract that makes a decision binding cannot be generated or substituted by AI — these require human processes
- Understand how to maintain traceability between implementation artifacts and the decisions that shaped them, even when those decisions were made across many short agentic iterations
- Know where to place human review checkpoints in agentic loops to ensure accountability for consequential architectural decisions is never fully delegated to the agent

LG 5-3: Know how to use AI and agents to extract architectural information from existing system parts

Participants know how AI tools and agents can extract relevant architectural information from existing system artifacts. They understand what kinds of information can be recovered, which techniques are available, and how to validate and combine findings — including the strengths and limitations of each approach.

- Summarize and analyze available system artifacts: source code, deployment descriptors, configuration, tests, and interfaces
- Interpret structural information from code, repositories, interfaces, and dependency graphs; combine findings into usable representations such as knowledge graphs or structure models, and keep them continuously up-to-date
- Combine AI-generated insights with traditional static and dynamic analysis to improve accuracy and coverage
- Detect technologies and frameworks using code property graphs (e.g. Joern, see [Joern]) or AI coding agents; recognize recurring architectural and design patterns in a domain-informed way; relate findings to architectural structures and modernization concerns
- Extract candidate business rules from code, tests, and configuration; relate them to documented domain concepts and business processes; prepare them for validation with domain experts and stakeholders
- Identify gaps, inconsistencies, and uncertainties in recovered information, and validate findings using static/dynamic analysis tools, code inspection, and stakeholder communication

LG 5-4: Understand how to generate architectural views from gathered architectural information

Participants understand how AI can support the creation and ongoing maintenance of architectural views and documentation. They know how to combine AI-generated insights with static and dynamic analysis to derive accurate, up-to-date architectural representations.

- Generate architecture-related documentation and views from code, repositories, interfaces, and other development artifacts, including knowledge representations of these artifacts
- Reconstruct and maintain relevant system views in alignment with system evolution; understand the value of text-based diagram formats (e.g. Mermaid, Structurizr) for version-controllable, living documentation
- Validate generated views and documentation against developer knowledge and actual system behavior
- Understand the value of current system views for analysis, communication, and modernization planning

5.3. References

[Clements et al. 2003], [Equal Experts 2025], [Joern], [Johansson et al. 2024], [Marco 2026], [Newman 2019], [Nygard 2011], [Brown 2026], [Zörner 2021]

6. Governance and Quality Gates for AI Usage

Duration: 60 min	Practice time: 30 min

6.1. Terms and Principles

AI Governance, Compliance, EU Act, OWASP Top 10 LLM, MITRE ATLAS, Human-In-The-Loop, Prompt Injection, Sovereignty, Sustainability of LLMs, LLM Footprint

6.2. Learning Goals

LG 6-1: Understand, classify, and properly address the legal framework conditions when using AI tools in the organization

- Participants can explain copyright issues such as copyleft licenses and assess which obligations may arise when using, adapting, and especially distributing software/artifacts in an AI context.
- Participants are aware of the resulting legal risks and understand that an AI-augmented architectural process needs to establish controls and processes that shield the generated code and artifacts from copyright-related liabilities.
- Participants can, when using AI tools, check from a data protection perspective whether and which personal data are processed, classify roles and responsibilities (controller/processor), and derive when contractual requirements must be clarified.
- Participants can apply the EU AI Act in its basic features (roles, risk logic, phased applicability) to typical AI tool scenarios and explain the timeline for application with key dates.
- Participants can decide for concrete initiatives when they must involve which internal and external stakeholders (e.g., data protection officer, information security) and can document the results in a way that is audit-ready and decision-ready.

LG 6-2: Know how to ensure traceability of AI-generated outputs and their context

- Participants can explain what is meant by traceability of AI outputs and justify why this is important for quality assurance, incident analysis, governance, and auditability.
- Participants can evaluate and weigh concrete measures to ensure traceability in a system context, in particular **structured logging**, **consistent referencing** of context artifacts, and **quoting & citation** as a mechanism for verifiable provenance of statements.
- Participants can explain why versioning and observability are necessary for AI applications, and can name key components such as prompt/template versioning, model/configuration states, traceable changes, as well as logging, tracing, or monitoring.

LG 6-3: Know how to design consistent scaling of AI governance across teams and processes

- Participants can classify and compare token-based billing models versus subscription/seat-based models and select a suitable governance and budgeting model for different team and use-case contexts.
- Participants can consider dependency risks (vendor lock-in) and data sovereignty when selecting LLM providers and understand the importance of flexible options to switch providers.
- Participants can explain why guardrails are important for agent and tool usage, and can describe how they can help reduce risks such as compliance violations, data leakage, or misuse.
- Participants can implement an allowlist/whitelist strategy for tool integrations, describe appropriate

authorization and approval processes, and justify the trade-offs between flexibility and control for different organizational areas.

LG 6-4: Understand the environmental costs of AI-supported software development and derive sustainable usage strategies

- Participants can explain and distinguish the key environmental impacts of AI-supported development across the lifecycle and classify typical misconceptions about them.
- Participants can identify and assess the most important drivers of energy and emissions impact in AI-assisted development (e.g., model size, multi-stage/agentic workflows) and create awareness of them within the organization.

LG 6-5: Know how to handle sensitive information securely, confidentially, and in compliance with data protection when using AI tools

- Participants can identify and classify sensitive information and decide what must not be shared with AI tools.
- Participants can assess data flows and data retention of AI tool usage and identify key risks.
- Participants can define team-appropriate rules for selecting and using AI tools in a secure and data-protection-compliant way.

LG 6-6: Know how to ensure the security of agent and tool integrations (prompt injection, tool misuse, sandboxing)

- Participants are familiar with the OWASP LLM Top 10 and know which risks are relevant to agentic engineering (among others, prompt injection and tool misuse). They have also heard of the MITRE ATLAS knowledge base.
- Participants can recognize prompt-injection attacks (direct and indirect) in agentic systems, explain the associated "confused deputy" risk, and derive requirements for robust prompt/context boundaries (e.g. clear separation of data and instructions, no trust in externally sourced content).
- Participants are familiar with suitable measures to reduce these risks (e.g. least privilege, scoped credentials, explicit approvals / human-in-the-loop, monitoring).
- Participants can select suitable sandboxing strategies for tool execution and know the advantages and disadvantages of container-based versus VM-based solutions.

6.3. References

[AI Act 2024], [Gumbley and Ryan 2025], [MITRE ATLAS 2026], [OReilly 2025b], [OWASP 2026], [Shamsujjoha et al. 2025], [Vries 2023], [Willison Various]

7. The Evolving Discipline of Software Architecture

Duration: 30 min	Practice time: 30 min

7.1. Terms and Principles

Hypothesis-driven AI adoption, Orchestration of AI-supported development, Skill preservation and deliberate practice, Accountability in AI-supported engineering, Blast-radius and risk-based reasoning, Shared terminology and domain glossary, Deterministically verifiable architectural rules (architecture tests, linters, fitness functions), Set-based design, Sociotechnical collaboration patterns (Human + Human + AI), Architecture levels (software, solution, enterprise)

7.2. Learning Goals

LG 7-1: Know about the importance of code ownership and the preservation of analytical skills

- Participants understand that the developer stays responsible for the result: "you push it, you own it".
- Participants understand the risk of skill atrophy through overreliance on AI and can design personal and team practices (e.g. deliberate AI-free practice, human-first / AI-second reasoning practices, traditional and AI-supported katas, using AI as a learning instrument for explanation, summarization, and questioning) to preserve core engineering skills.
- Participants understand the remaining importance of critical thinking (keep asking questions and researching until an understanding of a problem is gained as well as scrutinizing and cross-checking ‘AI-generated results’ against their own knowledge and observations)

LG 7-2: Be able to preserve architectural judgment when using coding agents

- Participants understand that agent decisions are shaped by available context and by both biases in training data (e.g. weighting toward particular vendors, frameworks, or stacks) and biases in existing artifacts of the own organization, and can identify situations in which such biases distort architectural choices.
- Participants can distinguish AI-supported reasoning from delegated decision-making.
- Participants can identify which architecture activities must remain human-owned and know that this might change in the future.
- Participants know that shared terminology and a consistent domain vocabulary (e.g. a domain glossary, ubiquitous language) act as anchors that make agent decisions more predictable and aligned with the system’s design (see [Gu et al. 2025]).
- Participants know about indicators for whether teams are improving architectural capability, not only producing faster outputs, and that the goal is to maximize outcome and impact for the user rather than output (avoiding what’s increasingly called 'AI slop')

LG 7-3: Know what drives the adoption of AI-supported development practices in organizations

- Participants can apply a hypothesis-driven approach to AI adoption (identifying a problem, formulating hypotheses, running experiments, measuring outcomes) and explain why this is preferable to broad, untargeted rollouts.
- Participants understand AI adoption as a continuous, multi-stage process (e.g. access, adoption, proficiency, ways of working, organizational change) rather than a one-time initiative, and can position an organization within such a model.

- Participants know about enabling measures such as trainings, hackathons, and dedicated time for experimentation, and can argue why protected time for learning is a precondition for sustainable adoption.
- Participants know that explicit ownership of AI adoption within a team or organization is crucial for its success.
- Participants understand the importance of a learning culture that tolerates mistakes and creates psychological safety for asking questions, and can identify cultural anti-patterns that block adoption.
- Participants can balance ease of access (recommending sensible default tooling) with openness to alternatives, and can justify this trade-off.
- Participants can support the shift in engineering identity from writing code toward shipping outcomes, and can explain its implications for individuals and teams.
- Participants know meaningful indicators for the success of AI adoption (e.g. DORA metrics, lead time, change failure rate, developer experience signals) and can use them to validate the hypotheses driving adoption.

LG 7-4: Know about evolving roles, teams, and architectural scope for AI-supported organizations

- Participants know new collaboration patterns that combine humans and AI (e.g. Human + Human + AI pairing) and can identify safeguards that ensure people continue to talk to each other rather than only to the agent and that insights encountered within an AI session might be shared with the human developer team. Correspondingly the participants know about agent topologies (e.g. Solo, Sub-Agent, Agent Teams)
- Participants know that collaboration patterns must match team context and that different team roles can perform different AI-supported activities (e.g. that product roles might trigger agent workflows, that developers might draft features).
- Participants can shift their own focus from line-level coding toward a broader architectural, generalist and product-centric thinking and they can anticipate the implications of this shift for hiring, skill development, and team size and composition.
- Participants understand that established agile practices will evolve under AI-supported development (e.g. pull request size, faster feedback loops, changed cadence).
- Participants can reason about the evolving role of the architect across architecture levels (software, solution, enterprise) and can identify where AI support has the greatest leverage at each level.

7.3. References

[Anthropic 2026], [Boeckeler 2025b], [Fowler 2025a], [Fowler 2025b], [Gu et al. 2025], [Hohpe 2020], [Holliday et. al. 2025], [Kim 2025], [Kim and Spear 2023], [Morris 2026]

References

This section contains references that are cited in the curriculum.

A

- [Vasilopoulos 2026] Vasilopoulos, A. (2026): "Codified Context: Infrastructure for AI Agents in a Complex Codebase." arXiv:2602.20478. https://arxiv.org/abs/2602.20478
- [Mei et al. 2025] Mei, L. et al. (2025): "A Survey of Context Engineering for Large Language Models." arXiv:2507.13334. https://arxiv.org/abs/2507.13334
- [AI Act 2024] EU AI Act — Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence. Official Journal of the EU, 12 July 2024. https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng
- [Anthropic 2026] Anthropic (2026): "2026 Agentic Coding Trends Report: How Coding Agents Are Reshaping Software Development." resources.anthropic.com. https://resources.anthropic.com/2026-agentic-coding-trends-report
- [Anthropic 2025a] Anthropic (2025): "Effective context engineering for AI agents." anthropic.com/engineering, 29 September 2025. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents
- [Anthropic 2025b] Anthropic (2025): "Effective harnesses for long-running agents." anthropic.com/engineering, 26 November 2025. https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents
- [Anthropic 2024] Anthropic (2024): "Introducing the Model Context Protocol." anthropic.com/news, 25 November 2024. https://www.anthropic.com/news/model-context-protocol
  Specification and documentation: modelcontextprotocol.io. https://modelcontextprotocol.io
- [Ashby 1956] Ashby, W. R. (1956): An Introduction to Cybernetics. Chapman & Hall, London. (Law of Requisite Variety) https://archive.org/details/introductiontocy00ashb

B

- [Bass et al. 2021] Bass, L., Clements, P. & Kazman, R. (2021): Software Architecture in Practice. 4th Edition. Addison-Wesley Professional. ISBN 978-0-13-688609-9. https://www.informit.com/store/software-architecture-in-practice-9780136886099
- [Beck 2025] Beck, K. (2025): "Augmented Coding: Beyond the Vibes." Tidy First? (newsletter.kentbeck.com), 25 June 2025. https://newsletter.kentbeck.com/p/augmented-coding-beyond-the-vibes
- [Boeckeler 2025a] Böckeler, B. (2025): "Anchoring AI to a reference application." martinfowler.com, 25 September 2025. https://martinfowler.com/articles/exploring-gen-ai/anchoring-to-reference.html
- [Boeckeler 2025b] Böckeler, B. (2025): "The role of developer skills in agentic coding." martinfowler.com, 25 March 2025. https://martinfowler.com/articles/exploring-gen-ai/13-role-of-developer-skills.html
- [Boeckeler 2025c] Böckeler, B. (2025): "Understanding Spec-Driven-Development: Kiro, spec-kit, and Tessl." martinfowler.com, 15 October 2025. https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html
- [Boeckeler 2025d] Böckeler, B. (2025): "To vibe or not to vibe." martinfowler.com, 23 September 2025. https://martinfowler.com/articles/exploring-gen-ai/to-vibe-or-not-vibe.html

- [Boeckeler 2026a] Böckeler, B. (2026): "Context Engineering for Coding Agents." martinfowler.com, 5 February 2026. https://martinfowler.com/articles/exploring-gen-ai/context-engineering-coding-agents.html
- [Boeckeler 2026b] Böckeler, B. (2026): "Harness engineering for coding agent users." martinfowler.com, 2 April 2026. https://martinfowler.com/articles/harness-engineering.html
- [Boeckeler and Thiagarajan 2025] Böckeler, B. & Thiagarajan, C. (2025): "Context engineering: Tackling legacy systems with generative AI." ThoughtWorks Technology Podcast, 21 August 2025. https://www.thoughtworks.com/insights/podcasts/technology-podcasts/context-engineering-tackling-legacy-systems-generative-ai
- [Brown et al. 2020] Brown, T. B. et al. (2020): "Language Models are Few-Shot Learners." arXiv:2005.14165. https://arxiv.org/abs/2005.14165
- [Brown 2026] Brown, S. (2026): The C4 Model: Visualizing Software Architecture. O'Reilly Media. ISBN 979-8-3416-6011-3. https://www.oreilly.com/library/view/the-c4-model/9798341660113/

C

- [Clements et al. 2003] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J. et al. (2003): Documenting Software Architectures – Views and Beyond. Addison-Wesley Professional. ISBN 978-0-201-70372-6. https://www.informit.com/store/documenting-software-architectures-views-and-beyond-9780201703726
- [Cui et al. 2026] Kevin Zheyuan Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, Tobias Salz (2026) The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers. Management Science 0(0). https://doi.org/10.1287/mnsc.2025.00535

D

- [Dhar et al. 2024] Dhar, R., Vaidhyanathan, K., Varma, V.: Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical Study. In: 2024 IEEE 21st International Conference on Software Architecture (ICSA), pp. 79–89 (2024). https://doi.org/ 10.1109/ICSA59870.2024.00016 https://www.researchgate.net/publication/378693248_Can_LLMs_Generate_Architectural_Design_Decisions_-An_Exploratory_Empirical_study
- [Diaz-Pace et al. 2024] Díaz-Pace, J.A., Tommasel, A., Capilla, R.: Helping Novice Architects to Make Quality Design Decisions Using an LLM-Based Assistant. In: Software Architecture: 18th European Conference, ECSA 2024, Luxembourg City, Luxembourg, September 3–6, 2024, Proceedings, pp. 324–332. Springer-Verlag, Luxembourg City, Luxembourg (2024). https://doi.org/10.1007/978-3-031-70797-1_21
- [Dieste et al. 2024] Dieste, O., Grimán, A., Juristo, N.: Developing search strategies for detecting relevant experiments. Empirical Software Engineering 14, 513–539 (2009)

E

- [Equal Experts 2025] Louw, T., Hmeid, R. & Brabban, P. (2025): "Accelerating Architectural Decision Records (ADRs) with Generative AI." Equal Experts blog, 27 October 2025. https://www.equalexperts.com/blog/our-thinking/accelerating-architectural-decision-records-adrs-with-generative-ai/

F

- [Fang et al. 2025] Fang, H. et al.: A Holistic Approach to Design Understanding Through Concept

Explanation. IEEE Transactions on Software Engineering 51(2), 449–465 (2025). https://doi.org/10.1109/TSE.2024.3522973

- [Feng et al. 2025] Kevin Feng, David W. McDonald, and Amy X. Zhang, Levels of Autonomy for AI Agents, 25-15 Knight First Amend. Inst. (July 28, 2025), https://knightcolumbia.org/content/levels-of-autonomy-for-ai-agents-1 [https://perma.cc/ETV7-M4Q9].
- [Ford et al. 2023] Ford, N., Parsons, R., Kua, P. & Sadalage, P. (2023): Building Evolutionary Architectures: Automated Software Governance. 2nd Edition. O'Reilly Media. https://www.oreilly.com/library/view/building-evolutionary-architectures/9781492097532/
- [Fowler 2025a] Fowler, M. (2025): "LLMs bring new nature of abstraction." martinfowler.com, 24 June 2025. https://martinfowler.com/articles/2025-nature-abstraction.html
- [Fowler 2025b] Fowler, M. (2025): "Some thoughts on LLMs and Software Development." martinfowler.com, 28 August 2025. https://martinfowler.com/articles/202508-ai-thoughts.html

G

- [GitClear 2025] GitClear (2025): "AI Copilot Code Quality: 2025 Data Suggests 4x Growth in Code Clones." gitclear.com. https://www.gitclear.com/ai_assistant_code_quality_2025_research
- [Gu et al. 2025] Gu, X., Chen, M., Lin, Y., Hu, Y., Zhang, H., Wan, C., Wei, Z., Xu, Y. & Wang, J. (2025): "On the Effectiveness of Large Language Models in Domain-Specific Code Generation." ACM Transactions on Software Engineering and Methodology (TOSEM). DOI: 10.1145/3697012. https://doi.org/10.1145/3697012 (preprint: https://arxiv.org/abs/2312.01639)
- [Gumbley and Ryan 2025] Gumbley, J. & Ryan, L. (2025): "Coding Assistants Threaten the Software Supply Chain." martinfowler.com, 13 May 2025. https://martinfowler.com/articles/exploring-gen-ai/software-supply-chain-attack-surface.html

H

- [Hargis et al. 2004] Hargis, G., Carey, M., Hernandez, A. K., Hughes, P., Longo, D., Rouiller, S. & Wilde, E. (2004): Developing Quality Technical Information: A Handbook for Writers and Editors. 2nd Edition. IBM Press / Prentice Hall. ISBN 978-0-13-147749-0. https://www.informit.com/store/developing-quality-technical-information-a-handbook-9780131477490
- [Hohpe 2020] Hohpe, G. (2020): The Software Architect Elevator: Redefining the Architect's Role in the Digital Enterprise. O'Reilly Media. https://www.oreilly.com/library/view/the-software-architect/9781492077534/

I

- [Holliday et. al. 2025] Holliday, D., Gonçalves, J. C. & Yadav, M. K. (2025): "Where Architects Sit in the Era of AI." infoq.com, 19 December 2025. https://www.infoq.com/articles/architects-ai-era/

J

- [Jahic et al. 2024] Jahić, J., Sami, A.: State of Practice: LLMs in Software Engineering and Software Architecture. In: 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pp. 311–318 (2024). https://doi.org/10.1109/ICSAC63560.2024.00059
- [Joern] Joern — The Bug Hunter's Workbench: an open-source platform for static code analysis based on code property graphs (CPGs), supporting multiple languages. https://joern.io Documentation: https://docs.joern.io — Source: https://github.com/joernio/joern

- [Johansson et al. 2024] Johansson, N., Caporuscio, M., Olsson, T.: Mapping Source Code to Software Architecture by Leveraging Large Language Models. In: Software Architecture. ECSA 2024 Tracks and Workshops: Luxembourg City, Luxembourg, September 3–6, 2024, Proceedings, pp. 133–149. Springer-Verlag, Luxembourg City, Luxembourg (2024). https://doi.org/10.1007/978-3-031-71246-3_13

K

- [Karpathy 2025a] Karpathy, A. (2025): Post on X, 25 June 2025. https://x.com/karpathy/status/1937902205765607626
  Quoted in: Willison, S., "Context engineering," simonwillison.net, 27 June 2025 (see [Willison 2025b])
- [Karpathy 2025b] Karpathy, A. (2025): "2025 LLM Year in Review." karpathy.bearblog.dev, 19 December 2025. https://karpathy.bearblog.dev/year-in-review-2025/
- [Kim 2025] Kim, G. (2025): "Potential GenAI Impact On DORA Metrics: Five Dimensions Of Value For Developers—Especially Creating Option Value!" itrevolution.com, 9 January 2025. https://itrevolution.com/articles/genai-metrics-of-value-for-developers-option-value-dora/
- [Kim and Spear 2023] Kim, G. & Spear, S. J. (2023): Wiring the Winning Organization: Liberating Our Collective Greatness through Slowification, Simplification, and Amplification. IT Revolution Press. https://itrevolution.com/product/wiring-the-winning-organization/
- [Konrad et al. 2026] Konrad, Phongsakon & Adam, Tim & Terrenzi, Riccardo & Ayvaz, Serkan. (2026). Architecture Without Architects: How AI Coding Agents Shape Software Architecture. 10.48550/arXiv.2604.04990. https://www.researchgate.net/publication/403604864_Architecture_Without_Architects_How_AI_Coding_Agents_Shape_Software_Architecture

L

- [LangChain 2025] LangChain (2025): "Context Engineering for Agents." langchain.com, 2 July 2025. https://www.langchain.com/blog/context-engineering-for-agents (four strategies: write, select, compress, isolate)
- [Lopopolo 2026] Lopopolo, R. (2026): "Harness engineering: leveraging Codex in an agent-first world." openai.com, 11 February 2026. https://openai.com/index/harness-engineering/
- [Marco 2026] Marco, E. (2026): Agentic Coding with Claude Code: The Everyday Developer's Guide to Agentic Coding with Claude Code. Packt, 27 March 2026. https://www.packtpub.com/en-us/product/agentic-coding-with-claude-code-9781806022595

M

- [MITRE ATLAS 2026] MITRE ATLAS™ (2026): Adversarial Threat Landscape for Artificial-Intelligence Systems — a knowledge base of adversarial AI tactics and techniques. https://atlas.mitre.org/
- [Morris 2026] Morris, K. (2026): "Humans and Agents in Software Engineering Loops." martinfowler.com, 4 March 2026. https://martinfowler.com/articles/exploring-gen-ai/humans-and-agents.html

N

- [Newman 2019] Newman, S. (2019): Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media. ISBN 978-1-492-04783-4. https://www.oreilly.com/library/view/monolith-to-microservices/9781492047834/

- [Nygard 2011] Nygard, M. (2011): "Documenting Architecture Decisions." cognitect.com blog, 15 November 2011. https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions

O

- [Ouyang 2025] Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. 2025. An Empirical Study of the Non-Determinism of ChatGPT in Code Generation. ACM Trans. Softw. Eng. Methodol. 34, 2, Article 42 (February 2025), 28 pages. https://doi.org/10.1145/3697010
- [OReilly 2026a] Ford, N. & Richards, M. (2026): "Architecture as Code to Teach Humans and Agents About Architecture." O'Reilly Radar, 9 April 2026. https://www.oreilly.com/radar/architecture-as-code-to-teach-humans-and-agents-about-architecture/
- [OReilly 2025b] Ford, N. & Richards, M. (2025): "How Agentic AI Empowers Architecture Governance." O'Reilly Radar, 19 November 2025. https://www.oreilly.com/radar/how-agentic-ai-empowers-architecture-governance/
- [OWASP 2026] OWASP (2026): "OWASP Top 10 for LLM Applications" . https://genai.owasp.org/llm-top-10/

P

- [Park et. al. 2023] Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P. & Bernstein, M. S. (2023): "Generative Agents: Interactive Simulacra of Human Behavior." UIST '23: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, Article No. 2, pp. 1-22. DOI: 10.1145/3586183.3606763. https://doi.org/10.1145/3586183.3606763 (preprint: https://arxiv.org/abs/2304.03442)

Q

- [Qishuo et. al. 2025] Qishuo, Hua & Ye, Lyumanshan & Fu, Dayuan & Xiao, Yang & Cai, Xiaojie & Wu, Yunze & Lin, Jifan & Wang, Junfei & Liu, Pengfei. (2025). Context Engineering 2.0: The Context of Context Engineering. 10.48550/arXiv.2510.26493. https://www.researchgate.net/publication/397088090_Context_Engineering_20_The_Context_of_Context_Engineering

R

- [Raschka 2026] Raschka, S. (2026): "Components of A Coding Agent." Ahead of AI (Substack), 4 April 2026. https://magazine.sebastianraschka.com/p/components-of-a-coding-agent
- [Rothman 2025] Rothman, D. (2025): Context Engineering for Multi-Agent Systems: Move beyond prompting to build a Context Engine, a transparent architecture of context and reasoning. Packt Publishing, 18 November 2025. ISBN 978-1-80669-005-3. https://www.packtpub.com/en-us/product/context-engineering-for-multi-agent-systems-9781806690053

S

- [Shamsujjoha et al. 2025] Shamsujjoha, M. et al.: Swiss Cheese Model for AI Safety: A Taxonomy and Reference Architecture for Multi-Layered Guardrails of Foundation Model Based Agents. In: 22nd IEEE International Conference on Software Architecture (ICSA 2025). Institute of Electrical and Electronics Engineers (IEEE) (2025). https : //arxiv.org/abs/2408.02205
- [Starke 2024] Starke, G. (2002): Effektive Softwarearchitekturen – Ein praktischer Leitfaden. 10. Auflage 2024, Carl Hanser Verlag, München. ISBN 978-3-446-47672-1. https://www.hanser-fachbuch.de/Effektive-Softwarearchitekturen/978-3-446-47672-1

- [Silva et al. 2025] Silva, K., Melegati, J., Silveira, F., Wang, X., Ferreira, M. & Guerra, E. (2025): "ArchHypo: Managing Software Architecture Uncertainty Using Hypotheses Engineering." IEEE Transactions on Software Engineering, 51(2), pp. 430-448. DOI: 10.1109/TSE.2024.3520477. https://doi.org/10.1109/TSE.2024.3520477

T

- [Toche et. al. 2020] Toche, B., Pellerin, R. & Fortin, C. (2020): "Set-based design: a review and new directions." Design Science, vol. 6, e18. Cambridge University Press. DOI: 10.1017/dsj.2020.16. https://doi.org/10.1017/dsj.2020.16
- [Toth 2025] Toth, S. (2025): Vorgehensmuster für Softwarearchitektur - Kombinierbare Praktiken in Zeiten von Agile und Lean. 4. Auflage 2025, Carl Hanser Verlag, München. ISBN 978-3-446-47993-7. https://www.hanser-fachbuch.de/Vorgehensmuster-fuer-Softwarearchitektur/978-3-446-47993-7

V

- [Vries 2023] de Vries, A. (2023): "The growing energy footprint of artificial intelligence." Joule, Volume 7, Issue 10, pp. 2191-2194. DOI: 10.1016/j.joule.2023.09.004. https://doi.org/10.1016/j.joule.2023.09.004

W

- [Willison 2026a] Willison, S. (2026): "Agentic Engineering Patterns." simonwillison.net. https://simonwillison.net/2026/Feb/23/agentic-engineering-patterns/
- [Willison 2025b] Willison, S. (2025): "Context engineering." simonwillison.net, 27 June 2025. https://simonwillison.net/2025/Jun/27/context-engineering/
- [Willison Various] Willison, S.: Writings on prompt injection and the lethal trifecta. Tag archive (150+ posts, 2022–present): https://simonwillison.net/tags/prompt-injection/
  Key post: Willison, S. (2025): "The lethal trifecta for AI agents: private data, untrusted content, and external communication." simonwillison.net, 16 June 2025. https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/
- [Wohlin et. al. 2012] Wohlin, C. et al.: Experimentation in Software Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

Y

- [Yao et al. 2023] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y. & Narasimhan, K. (2023): "Tree of Thoughts: Deliberate Problem Solving with Large Language Models." NeurIPS '23: Proceedings of the 37th International Conference on Neural Information Processing Systems, Article No.: 517, Pages 11809-11822. https://arxiv.org/abs/2305.10601

Z

- [Zörner 2021] Zörner, S. (2021): Software-Architekturen dokumentieren und kommunizieren. Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten. 3. Auflage, Carl Hanser Verlag, München. ISBN 978-3-446-46928-0. https://www.hanser-fachbuch.de/fachbuch/artikel/9783446469280