OST
Eastern Switzerland
University of Applied Sciences

Software Architecture @ 32: Responsible Decision Making or Prompt Engineering?

CIT Schaffhausen, December 11, 2024

Dr. Olaf Zimmermann (ZIO), https://ozimmer.ch/
olaf.zimmermann@ost.ch

Abstract

The software architecture toolbox has become broader and deeper over the years. Agile architecting practices, managed cloud services, and DevOps automation are some of the established additions; architecture generation and other applications of generative AI arrived more recently. The architect's core responsibilities remain – serving as decision makers, technical risk and debt removers, and advocates of development teams.

This talk first presents selected milestones in the evolution of the field since its inception in the 1990s. It then investigates whether generative AI has a place in the modern architect's toolbox: Can and should some or all architectural decision making be handed over to probabilistic auto-completion systems without consciousness? We tackle possible answers by usage scenario, reporting on first experiments (spoiler alert: "no" for core decision making, "it depends" for other usage scenarios).

The final part of the talk presents emerging research to position ethical values such as honesty, fairness, and sustainability as special types of architecturally significant requirements, to be addressed by existing and new design tactics. We conclude with an outline of open research problems: How will a responsible approach to decision making and tradeoff analysis look like? How can values be integrated into component specifications and API contracts? How can generative AI contribute to value-driven analysis and design methods?

Structure and first Take-aways of this Talk

1. **Structure/behavior and decisions** go hand in hand in architecture design.
2. **Generative AI** is a tool not a threat – a powerful but expensive one.
3. **Ethics form** a design concern that yields architecturally significant requirements. How to address them is only partially understood.

Stability regarding the "why" and what" architecture is about, variety in the "how" (dimensions: ceremony, risk, complexity).

Creativity-rigor balance looks different in different schools of thought and throughout phases in evolution of the field.

OST
Eastern Switzerland
University of Applied Sciences

Software Architecture @ 32
Part 1: Evolution and Examples

Dr. Olaf Zimmermann
olaf.zimmermann@ost.ch

Software Architecture: Early Work (1992)

By analogy to building architecture, we propose the following model of software architecture:

Software Architecture =
{ Elements, Form, Rationale }

Foundations for the study of software architecture

Authors: Dewayne E. Perry, Alexander L. Wolf | Authors Info & Claims

ACM SIGSOFT Software Engineering Notes, Volume 17, Issue 4 • Pages 40 - 52 • https://doi.org/10.1145/141874.141884

Published: 01 October 1992 Publication History

Check for updates

1,427 12,448

PDF eReader

https://dl.acm.org/doi/10.1145/141874.141884

Phase I: All about Structure (and Interactions)

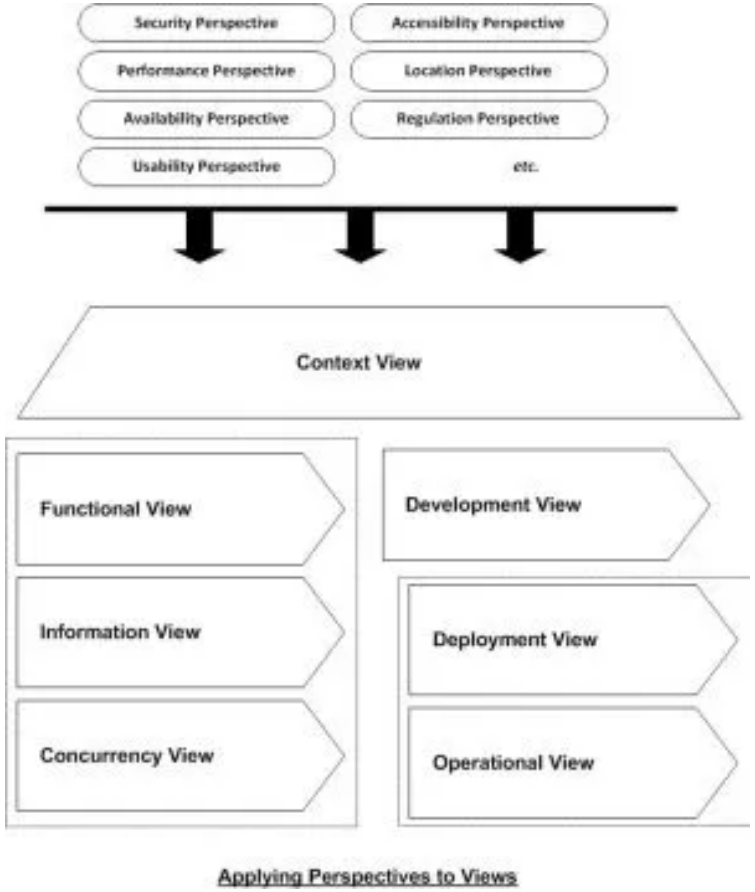
- Viewpoints and perspectives, e.g. 4+1 views (Kruchten, UP)
- Technical layers
- Functional partitioning

Practices and notations:

- Quality attributes, design tactics (SEI)
- UML diagrams (starting with stereotyped class diagrams), UML tools
- Patterns: POSA series, M. Fowler (PoEAA), Hohpe/Woolf (EIP), …

Reference: N. Rozanski and E. Woods, Software Systems Architecture, https://www.viewpoints-and-perspectives.info

Viewpoints and Perspectives (Example)

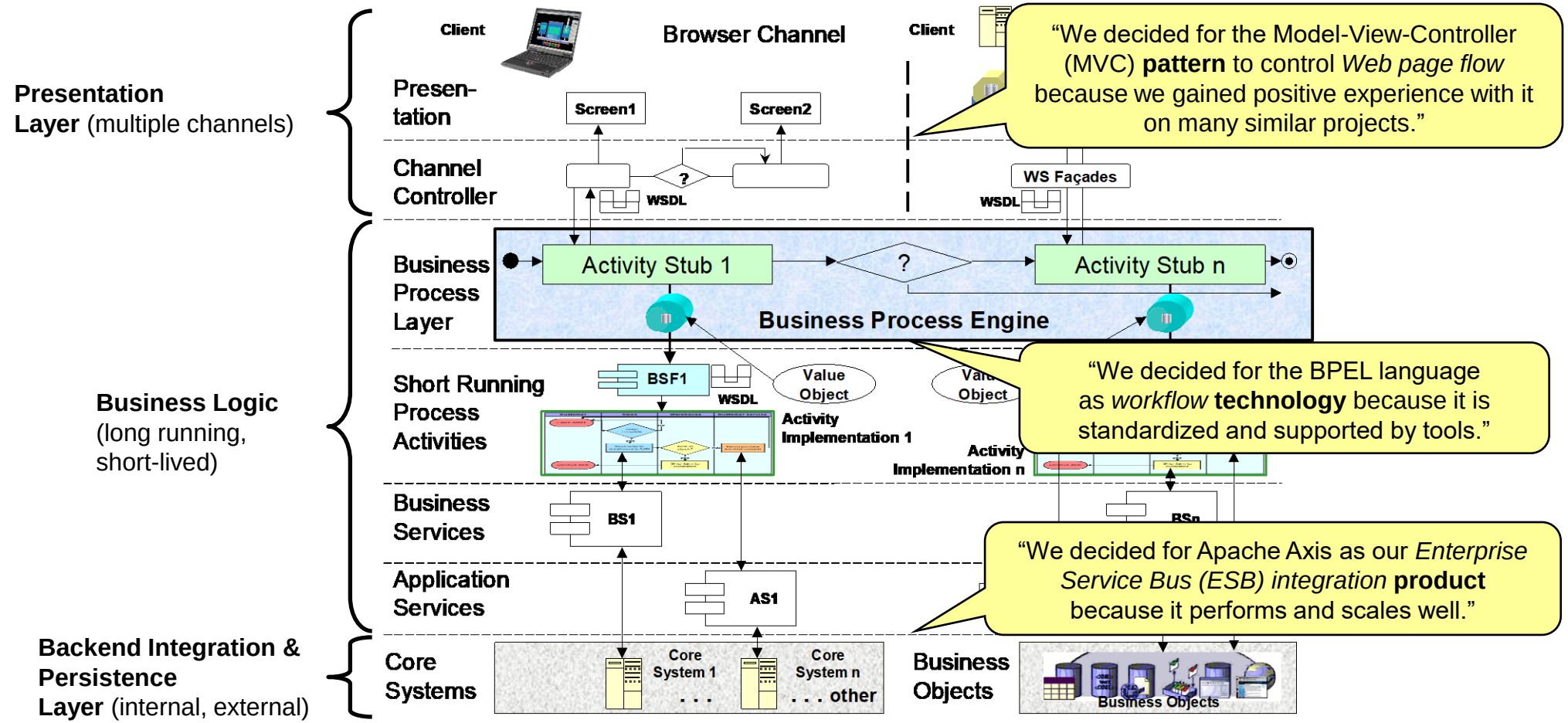


Applying Perspectives to Views

- "A *viewpoint* is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views. "
- "An *architectural perspective* is a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural views. "

Reference: Zimmermann et al, „Service-Oriented Architecture (SOA) and Business Process Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned“, OOPSLA 2005 conference companion, ACM Press, 2005

ADs in a Typical Enterprise Application (2004)



Phase II (since 2004): Decisions Emphasized

- Focus on journey in addition to destination, answer “why?” questions
  - e.g., in ISO/IEC/IEEE 42010 definition of software architecture
- Architectural Decision (AD) identification, making, enforcement
- Explicit capturing of formerly tacit knowledge (after-the-fact)

Practices and notations:

- *Architectural Decision Record (ADR)* templates to preserve design rationale
  - Context, criteria, consequences (good and bad)
- Vision of guidance models: ADs that recur in a genre/style as mentors
  - Example: SOA Decision Modeling (SOAD)
- Decision-centric architecture reviews (DCAR)

Two ADR Templates: Nygard and WH(Y)

- Cognitect Blog 2011: “ADR”
  - Context
  - Outcome
  - Status
  - Consequences
- ABB 2012: “WH(Y)”
  - Two-part context
  - Chosen and neglected options
  - Good and bad consequences
  - Metadata left out in template

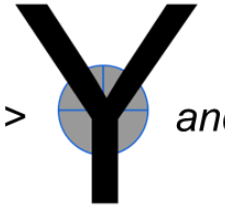
DOCUMENTING ARCHITECTURE DECISIONS

Michael Nygard - November 15, 2011 AGILITY ARCHITECTURE

In the context of <use case uc and/or component co>,

… facing <non-functional concern nfc>,

… we decided for <option o1>



and neglected <options o2 to on>,

… to achieve <positive consequence/quality q>,

… accepting that <negative consequence c>.

Reference: Sustainable Architectural Design Decisions, IEEE Software, Vol. 30, Issue 6, 2013 and SEI SATURN 2012

Sample WH(Y): Logical Decomposition

- *In the context of* the entire order management system,
- *facing* the need to organize the overall system and manage complexity,
- *we decided for* the Layer-based decomposition pattern
- *and neglected* other decomposition pattern such as pipes-and-filters or process-based decomposition (workflow)
- *to achieve* a) high flexibility regarding technology selections within the layers (changeability) and b) that teams can work on system parts in parallel
- *accepting that* there might be a performance penalty for each level of indirection and some undesired replication of implementation artifacts.

AD Management Process and Practices

- Architectural significance test: 5+2 criteria
- Definition of ready for single AD: START
- “Big” AD criteria (most responsible moment)
- Definition of done for single AD: ecADR
- Advice regarding good and bad ADR practices (justifications, …)
- (emerging) AD adoption model, catalogs for domains/genres

Slides: https://ozimmer.ch/

Concepts proposed in blog posts
(no scientific publications yet)

Phase III: Architect as Team Player, Facilitator

- Role metaphors: gardener, primus inter pares, …
- Shared responsibility in autonomous teams (virtual role)
- Architect elevator: engine room (DevOps) to penthouse/executive floor

Practices, notations, tools:

- Agile architecture, documentation as code (Markdown/Pandoc, …)
- Collaborative Modelling, Quality Attribute Workshops
- Any drawing tool for overview diagrams, component views, …
- Cloud providers providing reference architectures, methods, icon libraries

The Software Architect's Role in the Digital Age

The Software Architect's Role in the Digital Age

Gregor Hohpe, Allianz SE

Ipek Ozkaya, Carnegie Mellon Software Engineering Institute

Uwe Zdun, University of Vienna

Olaf Zimmermann, University of Applied Sciences of Eastern Switzerland, Rapperswil

TRADITIONALLY, SOFTWARE architects were entrusted with making "decisions that are costly to change."[1] Because these decisions often had to be made early in the project, architects drew on their experience and ability to abstract to get them right. Repeated project cost and timeline overruns have demonstrated, though, that trying to plan all features and decide the system structure early in a project is difficult at best. This insight, coupled with the increasing demand for delivering high-quality software more quickly, has changed how development teams approach architectural decision making.

Reversing Irreversible Decision Making

Software teams are increasingly embracing tools and practices that help them avoid, decouple, or break down big, up-front decisions. For example, agile practices have reduced the need to make irreversible decisions at a

https://ieeexplore.ieee.org/document/7725214

- Third special theme issue on Software Architecture in IEEE Software (2016)
  - First: November 1995
  - Second: April 2006
- Trend report in guest editorial:
  - Architect can be a virtual role or shared responsibility
  - Decisions made throughout iterations
  - Lightweight ADRs
  - PlantUML
  - Architecture as/in code

Article PDF: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7725214

State of the Art and the Practice in 2016

- Dimensions:
  - Context and requirements
  - Structure
  - Design decisions (rationale)
  - Realization (implementation)

TABLE A

Architectural dimensions and the evolution of the software architecture field.

Aspect	The state of the art: At the field's inception (1990s)[3]	After a decade (mid 2000s)[1]	Today
Context and requirements	Not an explicit part of the early definition[3]	Quality attributes (QAs) and constraints	QAs plus explicit representation of context;[4] more emphasis on business speed and value, cost and risk, architectural principles, and technical-debt management for strategic architecting
Structure	Elements • Processing • Data • Connectors Form • Properties (of elements) • Relationships (between elements)	4+1 views, components and connectors in UML and architecture description languages, informal box-and-line diagrams created by following processes and guidance in architecture design methods, and general architectural patterns; and first domain-specific architectural tactics and patterns (for example, for enterprise application architectures)[5]	More notations, such as domain-specific languages (for example, context maps in domain-driven design); more emphasis on data (for example, information viewpoints) and on architecting runtime relationships (for example, in cloud deployments); design by composition through frameworks; and many more domain-specific architectural tactics and patterns
Design decisions (reasoning behind chosen structures)	Rationale	Architectural decisions recognized as a key architectural concept in many articles and books, but no detailed coverage in most methods and tools[6]	Architecture knowledge management and decision making as a major research field and early adoption in practice (for example, inclusion in ISO/IEC/IEEE 42010:2011)
Realization	Not an explicit part of the early definition	Architecture design often embedded into end-to-end software engineering methods, International Federation for Information Processing (IFIP) subarea "realization," and model-driven software engineering and code generation attempts	Agile practices, continuous delivery, and DevOps; increased emphasis on the time dimension; better enactment and enforcement of architectural decisions (for example, architecturally evident coding styles); and continuous feedback cycles[7]

Subset of topics covered here: https://medium.com/olzzio

More Recent Toolbox Entries (or Comebacks?)

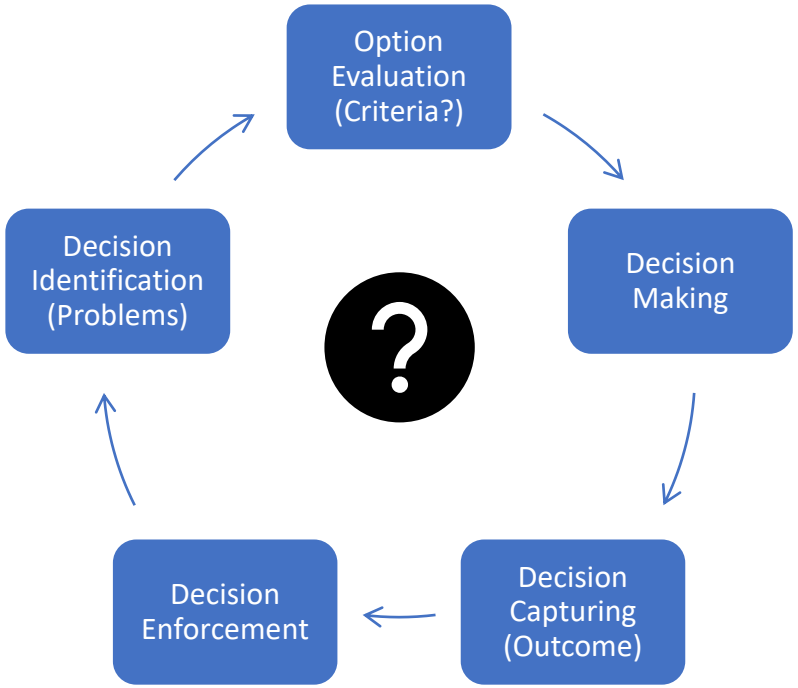
- Cloud-Native Application Architectures and Microservices
- API design and management patterns
  - Polyglot protocol: RESTful HTTP, gRPC, GraphQL; many messaging system
- DevOps including CI/CD and observability
- Collaborative modeling as input for analysis and synthesis/design:
  - Domain Storytelling, Event Storming
  - Example Mapping, Story Mapping, Story Splitting
- C4 Model: Context to Container, Component, Code (zoom in)
- Domain-Driven Design (DDD): one way of doing OOAD

OST
Eastern Switzerland
University of Applied Sciences

Software Architecture (SWA) @ 32
Part 2: Generative AI in SWA – Experiments and Position(s)

Dr. Olaf Zimmermann
olaf.zimmermann@ost.ch

Gen AI Use Cases (in Software Architecture)



- Jim Highsmith on LinkedIn:
  - Different context: advice about agile software development
  - “Byron is my AI writing companion (ChatGPT with Canvas). I use Byron for research, editing, and idea generation.”
- Possible adoption to software architecture:
  - Find design options and criteria
  - Check, trim, and improve ADRs
  - Learn what others are doing

Other Architects' Views

- AI as Software Architect assistant by Avraham Poupko, GSAS 2024
  - LLMs compared to humans (from 16:33 to 20:00 in the presentation)
  - Comments interesting too

“QOC” for Gen AI in SWA

- **Questions:**
  - When to use what? And how? Many recurring decisions, so architects needed ☺
- **Options:**
  - No use, playground mode, productions use (core tasks, edge tasks)
  - Local computations (training, prompting) vs. cloud service
  - Large vs. small LLM
  - Prompting patterns (see Tech Radar by TW, see ICSA paper)
- **Criteria:**
  - Criticality of task
  - Complexity and context-sensitivity of task
  - Popularity of task
  - Existing knowledge in the public domain, in proprietary/custom LLMs
  - AI hype vs. long-term perspective

Decision Made!

ADR-style rationale for my position:

Context: *See this and previous slides for (N)FRs; productivity gains always desired*
Status: *decided*
Decision: *ADOPT (for ideation); TRIAL (for documentation); HOLD for decision making (architecture generation); ASSESS (for other usage scenarios)*
Consequences:

(+) stay in control, acknowledge context
(+) be able to explain, defend, revise designs
(o) be able to/have to keep on experimenting
(-) risk of funding issues (academia, industry)
(-) be perceived as skeptic among pioneers/early adopters

Other Gen AI Positions (Software Engineering)

- Pro (with technical explanations, use cases and risks identified: ”Generative AI for Software Practitioners”
  - IEEE Software column, https://ieeexplore.ieee.org/document/10176168
- Neutral (clarifications, call to action, …): “Being a Responsible Developer in the Age of AI Hype”
  - QCON presentation, transcribed
- Contra (with historic perspective on Software Engineering): “Is AI a Silver Bullet?
  - Blog post, https://ian-cooper.writeas.com/is-ai-a-silver-bullet
- Recent SI-SE/JUG event “Will AI Replace Software Engineering?”
- Many more opinions…

OST
Eastern Switzerland
University of Applied Sciences

Software Architecture @ 32
Part 3: Ethics as a Design Concern

Dr. Olaf Zimmermann
olaf.zimmermann@ost.ch

Values… there are way more than $ and UX

Reference: https://ieeexplore.ieee.org/ielx7/52/8693065/08693077.pdf

> There will always be adversarial threats, either internal or external, that will breach data and abuse systems and resources. However, embracing ethics as an explicit, nonnegotiable software design concern will be a start toward conscious progress.

- Codes of Conduct (ACM, IEEE, …)
  - Related work: ProactiveCARE, SoDIS
- Method engineering at OST (open source):
  - VDAD process, ESE practices

https://github.com/socadk/ethical-software-engineering/

NFR: Non-Functional Requirement

IEEE Std. 7000

Standards

7000-2021 - IEEE Standard Model Process for Addressing Ethical Concerns during System Design

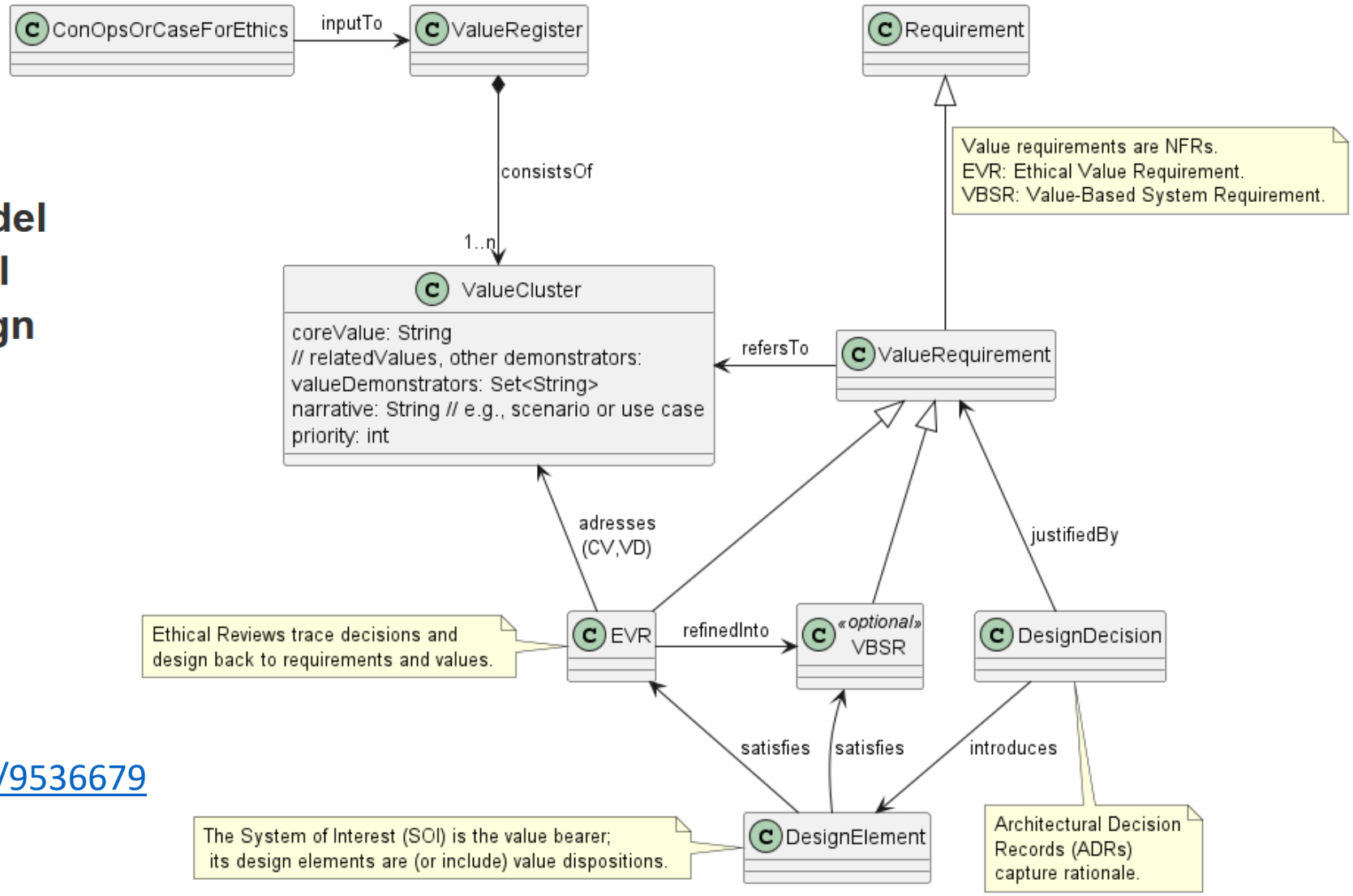
Publisher: IEEE Cite This

Status: Active - Approved

Available through the **IEEE GET Program**™

Free Access

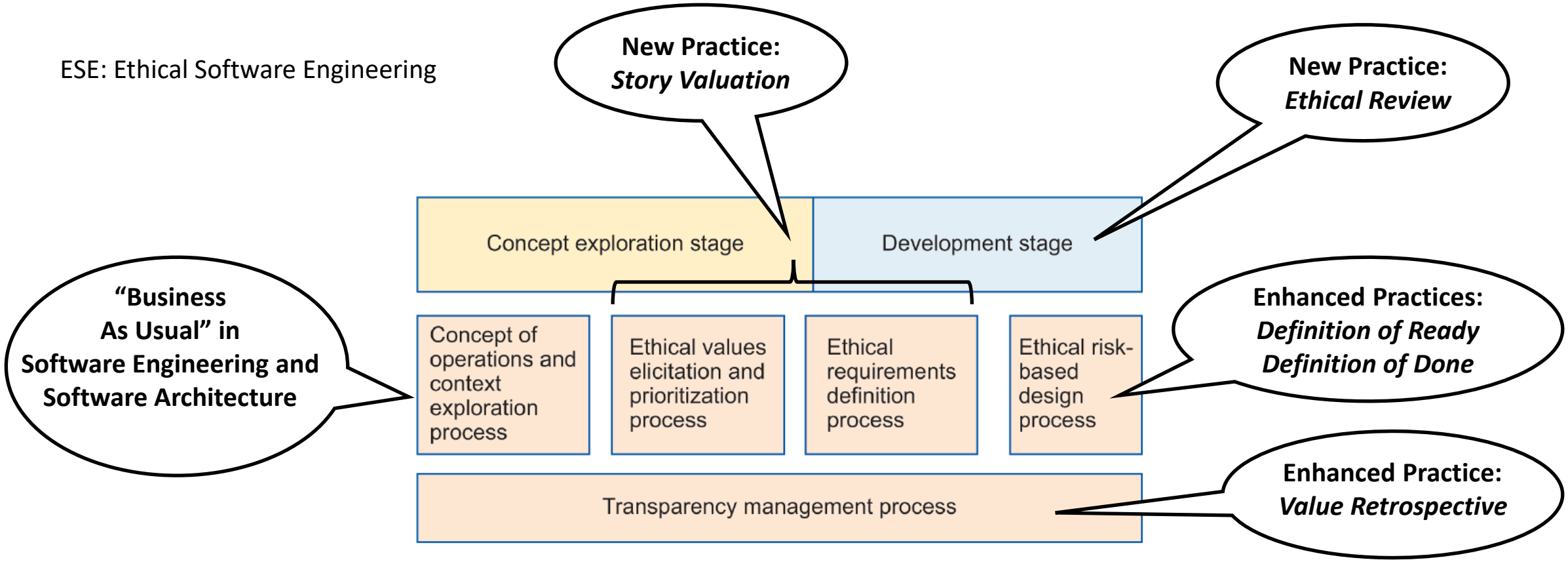
https://ieeexplore.ieee.org/document/9536679



https://github.com/socadk/ethical-software-engineering/tree/main/practices

Agile Practices for IEEE Std. 7000: ESE

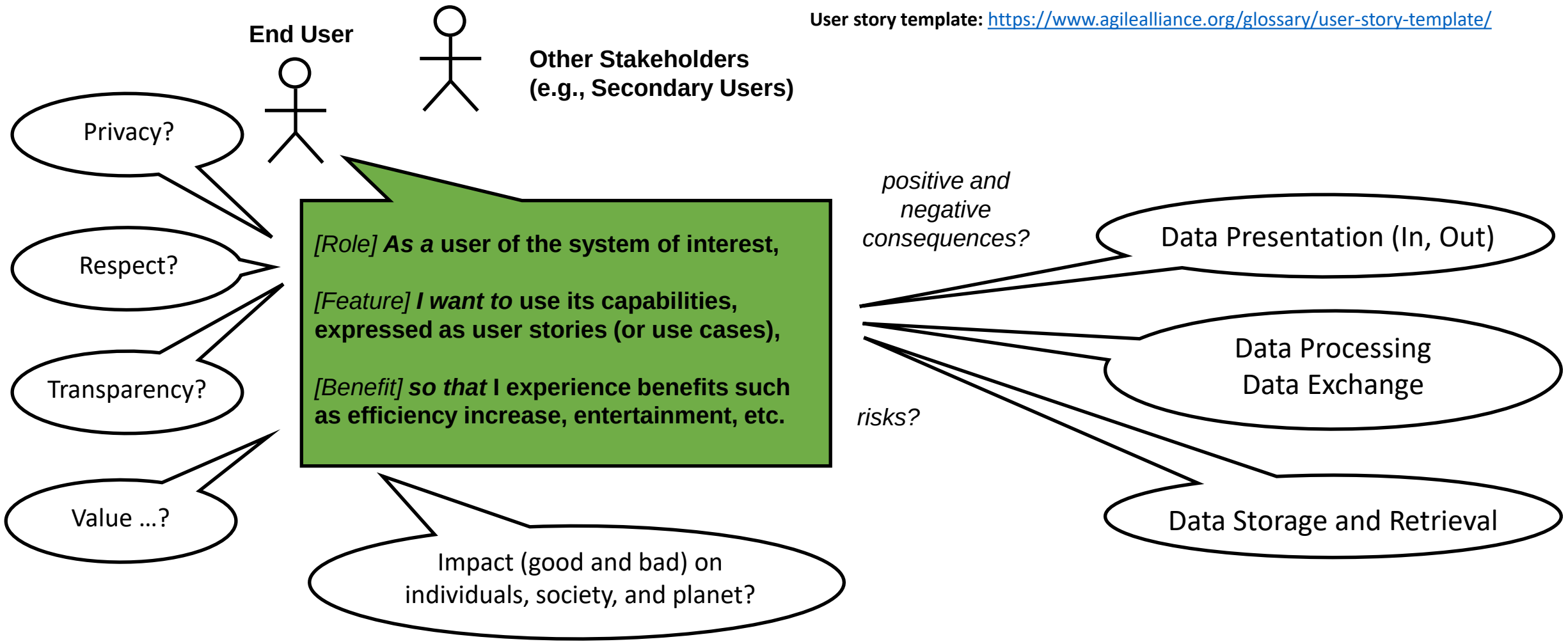
ESE: Ethical Software Engineering



Reference: "Relationship of processes and stages in IEEE Std 7000" (Figure 1 in "IEEE Standard Model Process for Addressing Ethical Concerns during System Design")

https://github.com/socadk/ethical-software-engineering/blob/main/practices/ESE-StoryValuation.md

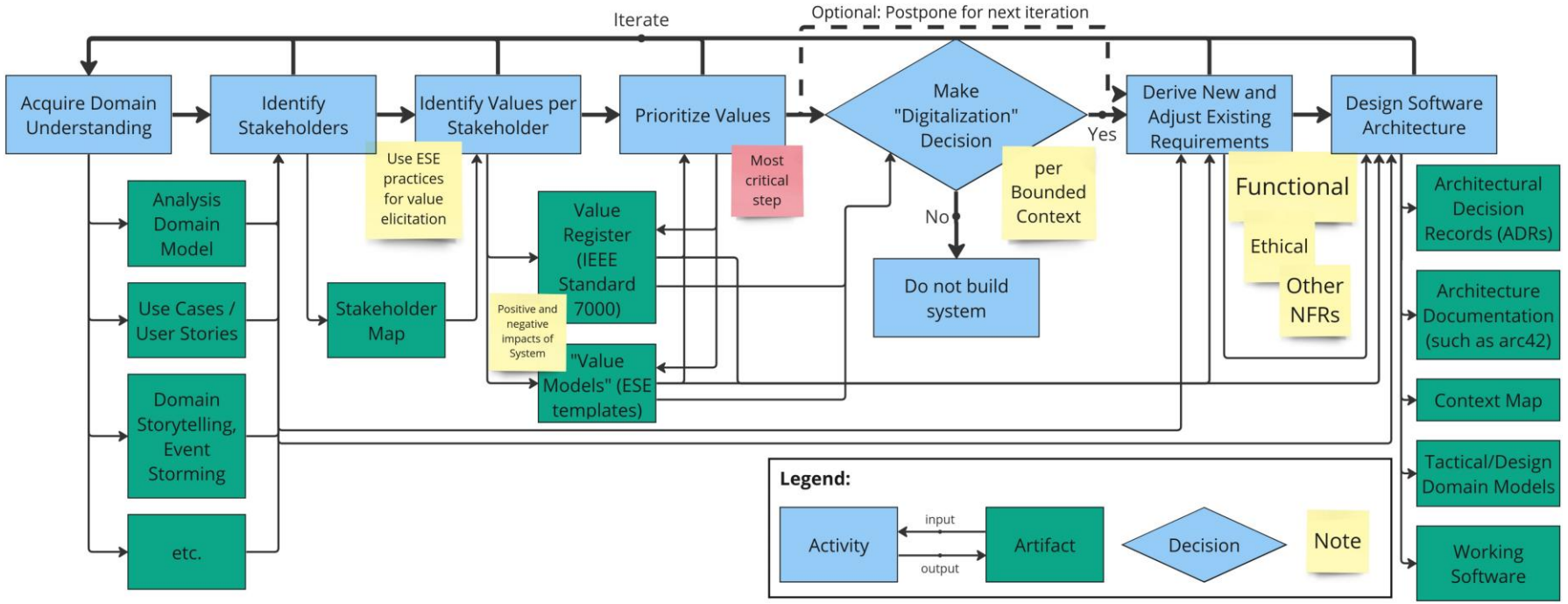
User Story as Source of Value Requirements



VDAD Project

Value-Driven Analysis and Design (VDAD)

- 6 usage scenarios (so far)
- Two new practices: stakeholder mapping, value impact mapping



OST
Eastern Switzerland
University of Applied Sciences

Software Architecture @ 32
Conclusions and Outlook

Dr. Olaf Zimmermann

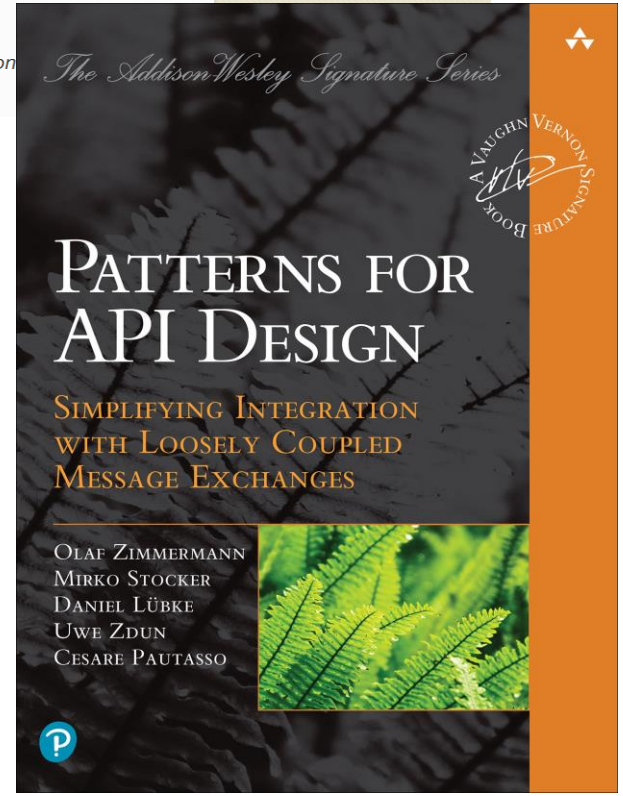
olaf.zimmermann@ost.ch

Towards Responsible Software Architecting and Engineering (Research Roadmap?)

- How does a responsible approach to tradeoff analysis and decision-making look like? How to balance creativity and rigor in it?
- How can the guiding role of reusable architectural decision models be leveraged for and/or supported by Retrieval Augmented Generation (RAG) during AD preparation (criteria and option identification/evaluation)?
- How do stakeholder values fit into existing viewpoints and perspectives models (for instance, the one in "Software Systems Architecture"?)
- How can ethical values be expressed during AD execution and enforcement, for instance in component specifications and API contracts?
- How can generative AI contribute to value-driven analysis and design methods and other initiatives (e.g., architecturally evident coding styles)?

Books, Open Repos

- Design Practice Reference: e-book on LeanPub
  - Y-Statements, SMART NFRs, Architecture Modeling (C4 plus), DDD, … (content also available on GitHub/GitPages)
- "Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges" (website)
  - Arch. Significant Requirements (ASRs) in API design
  - Many decisions (about APIs), Y-statements
  - 44 patterns, focus on message content



Thank You & Keep in Touch

Discussion time!

- I hope you find a few ideas to take away from this presentation
- I will be happy to answer questions and discuss arch. decisions of all kinds, as well as other topics – after the talk? Later on?

- mailto:olaf.zimmermann@ost.ch
- https://medium.com/olzzio
- https://www.linkedin.com/in/ozimmer/