DESIGNED AND DELIVERED TODAY, ERODED TOMORROW?

TOWARDS AN OPEN AND LEAN ARCHITECTING FRAMEWORK BALANCING AGILITY AND SUSTAINABILITY

supported by ADMentor, ART and Service Cutter tools

Prof. Dr. Olaf Zimmermann (ZIO)
The Open Group Certified Distinguished IT Architect (Chief/Lead)
Institute für Software, HSR FHO
ozimmerm@hsr.ch
ZIO Past and Present

- **Research & development and professional services since 1994**
  - em. IBM Solution Architect & Research Staff Member
    - Systems & Network Management, J2EE, Enterprise Application Integration/SOA
  - em. ABB Senior Principal Scientist
    - Enterprise Architecture Management/Legacy System Modernization/Remoting

- **Industry projects and coachings**
  - Product development and IT consulting (middleware, SOA, information systems, SE tools)
  - Tutorials: UNIX/RDBMS, OOP/C++/J2EE, MDSE/MDA, Web Services/XML

- **Focus @ HSR: design of distributed/service-oriented systems**
  - Cloud computing, Web application development & integration (runtime)
  - Model-driven development, architectural decisions (build time)
  - (Co-)Editor, [Insights column](#), IEEE Software
  - PC member, e.g., ECSA, WICS, QoSA, SATURN, SummerSoC
Service-Oriented Architecture (SOA) for Core Banking

- Core banking application, shared service/service provider model
  - Layers pattern, data stored in backend, Web frontend, Web services

Process-Enabled Order Mgmt. SOA for Telecom Service Provider


“We decided for the Model-View-Controller (MVC) pattern to control Web page flow because we gained positive experience with it on many similar projects.”

“We decided for the BPEL language as workflow technology because it is standardized and supported by tools.”

“We decided for Apache Axis as our Enterprise Service Bus (ESB) asset because it performs and scales well.”
Existing metamodels and templates refactored and extended for reuse

- Before: documentation – after the fact (past tense)
- With SOAD: design guidance – forward looking (future tense)

"When designing a presentation layer, you will have to select a pattern to control the Web page flow."

"Model View Controller (MVC) is a common architectural pattern to control the Web page flow."

"We decided for the MVC alternative to resolve the web page flow issue because we gained positive experience with it on many similar projects."

Agenda

- Context and Motivation
  - Agility vs. sustainability

- Towards an Open and Lean Architecting Framework
  1. SMART NFRs and quality stories
  2. C4 architecture modeling
  3. Decision sharing with Y-statements
  4. Architecturally evident coding styles
  5. Architecture roadmapping
  6. Architectural refactoring
  7. (Micro-)services principles and patterns

- The Software Architect’s Role in the Digital Age (IEEE Software)
  - Changes
  - Challenges
Cost-effective longevity and endurance, adherence to principles:

The term *sustainability* is derived from the Latin word sustinere (sus: up; tinere: to hold) and is often used solely in the environmental sense. The scope of the SAGRA workshop does not exclude this interpretation, but the main focus of our workshop is on another usage of this term: *cost-effective longevity and endurance*. By the sustainability of software architecture we mainly mean the adherence to design principles (such that modularity, hierarchy, abstraction, separation of concerns, information hiding, etc.) along with its entire lifecycle.

It is crucial for software systems to be cost-efficiently maintained and evolved over their entire life-cycle, i.e. to be sustainable, whereas the sustainability of the system is largely determined by the sustainability of its architecture. Low quality architectures slow the progress of software evolution and reduce profits. Designing a sustainable software architecture is a non-trivial task, especially in the case of large scale and/or long-living systems. The task becomes even more complicated when we take into account collaborative and intercultural aspects of the software development, requirements traceability and big data management. Practitioners require architecture metrics that support sustainable software architectures reflecting quality attributes such as maintainability, extensibility, reliability, integrity, etc.

Reference: [https://sagra2016.wordpress.com/](https://sagra2016.wordpress.com/)
(Plenty of) Candidate Assets by Architecting Phase

- **Arch. Synthesis (AA) of Design**
  - e.g. ADD, OOD (RUP, OpenUP)

- **Arch. Analysis (AA) of ASRs a.k.a. NFR/QA Elicitation**
  - e.g. QAS, OOA (RUP, OpenUP)

- **Arch. Evaluation (AE) a.k.a. Review**
  - e.g. ATAM, TARA

**Figure reference:** Christine Hofmeister, Philippe Kruchten, Robert L. Nord, J. Henk Obbink, Alexander Ran, Pierre America, , A general model of software architecture design derived from five industrial approaches, *Journal of Systems and Software*, 2007
Key Agile Concepts (Principles, Practices)

- Welcome change – or at least accept it
  - And 11 more agile principles (in Agile Manifesto)

- Big ideas of selected agile methods (as identified in Agile!):  
  - Reduce waste (Lean)
  - Implement then simplify (XP)
  - Freeze requirements during short iterations (Scrum)
  - Osmotic communication (Crystal)

- Agile practices (as seen by Agile Alliance):
  - One subway line per Agile “tribe”
  - Links to structured glossary entries
    - Definition, a.k.a.
    - Common pitfalls
    - Signs of use
    - Expected benefits
    - Origins

https://www.agilealliance.org/agile101/subway-map-to-agile-practices/
Agility – is it “80% Common Sense, 20% Cult”?

- “Agile! The Good, the Hype and the Ugly” (B. Meyer)
  - Controversial, many examples and anecdotes (and humorous at times); mixed reactions
  - Architecture defined as the result of all design activities

- See IFS website for pointers on agile vs. architecture
  - IEEE Software issue on agility and architecture
  - Agile architecture (P. Kruchten)
  - Original Scrum paper
  - SEI Agile Architecting Library
  - ThoughtWorks view (R. Parsons, M. Fowler)

- So what remains to be said and done?
  - Agile practices are mainstream – but software engineering is more than programming (quality attributes, patterns, 4+1, …)
  - Goal: agilize IT architecting practices – to sustain them and their output
    - Push them to target audience (rather than wait for them to be pulled)
Agenda

- **Context and Motivation**
  - Agility vs. sustainability

- **Towards an Open and Lean Architecting Framework**
  1. SMART NFRs and quality stories
  2. C4 architecture modeling
  3. Decision sharing with Y-statements
  4. Architecturally evident coding styles
  5. Architecture roadmapping
  6. Architectural refactoring
  7. (Micro-)services principles and patterns

- **The Software Architect’s Role in the Digital Age (IEEE Software)**
  - Changes
  - Challenges
**SMART Non-Functional Requirements (NFRs)**

- **SMART criteria** are frequently used in project and people management

<table>
<thead>
<tr>
<th>Letter</th>
<th>Most common</th>
<th>Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Specific[4][5][6][7][8][9][10][11][12][13][14]</td>
<td>(Strategic and specific)[15]</td>
</tr>
<tr>
<td>M</td>
<td>Measurable[4][5][6][7][8][9][10][11][12][13][14][15]</td>
<td>Motivating (Source: One Minute Manager)</td>
</tr>
</tbody>
</table>

- The SMART criteria can also be applied to NFR engineering, which is complementary to specifying full Quality Attribute Scenarios (QAS):
  - S – Which feature or part of the system should satisfy the requirement?
  - M – How can testers and other stakeholders find out whether the requirement is met (or not)? Is the requirement quantified?
  - A, R, T – Issues for Requirements Engineering and Project Management

- **Example and counter example (SMART?)**:
  - “The system should be highly usable, perform well and easy to maintain.”
  - “The ‘place order’ user story must complete in less than a second.”
User Story Template – Extended with Quality Goal

Software User

As a [role],

I would like to [goal]

so that [effect (business value, impact)]

To achieve this goal, I expect the following qualities (in descending order of priority):

• [measurable usability quality property, e.g. input steps required to complete story]

• [measurable performance quality, e.g. average and worst case story response time]

• […]
As a [role concerned with system quality, e.g. a leadership or maintenance role],

I would like to [achieve quality goal A]  
– without changing the functional scope of the system –

so that future versions of the system can benefit from:

• [technical debt reduction effect]  
• [improved service level/system property]  
• [positive impact on other technical constraints and environment]

To achieve this goal, I am willing to invest/accept:

• [impact on other quality attributes, e.g. performance penalty for security feature]  
• [impact on project plan (cost, timeline)]  
• [impact on technical dependencies and risk]
As a Development and Operations (DevOps) engineer at a social network/media firm,

I would like to be able to add attributes to the database w/o having to migrate data – without changing the functional scope of the system –

so that future versions of the system can benefit from:
• New features of the Web software can be introduced more often.
• It is no longer needed to migrate the large amount of existing data to new schemas.
• We become independent of the provider of the current RDBMS.

To achieve this goal, I am willing to accept:
• Data access and data validation logic becomes more complex.
• Approximately five developer days and additional test effort have to be invested.
• Technical feasibility and performance have to be validated in a PoC.
Four abstractions: **Context**, **Containers**, **Components**, **Classes**

- GOTO 2014 talk: [https://www.youtube.com/watch?v=GAFZcYIO5S0](https://www.youtube.com/watch?v=GAFZcYIO5S0)
- Article: [https://www.voxxed.com/blog/2014/10/simple-sketches-for-diagramming-your-software-architecture](https://www.voxxed.com/blog/2014/10/simple-sketches-for-diagramming-your-software-architecture)
- Book: [https://leanpub.com/software-architecture-for-developers](https://leanpub.com/software-architecture-for-developers)

Critique:

- Too simple?
- Violating Single Responsibility Principle (SRP)?
- What about connectors, cross-cutting concerns? Other viewpoints?
  - From C4 to C5+2 (add connectors and cross-cutting concerns)?

Competition:

- ArchiMate, [http://www.opengroup.org/subjectareas/enterprise/archimate](http://www.opengroup.org/subjectareas/enterprise/archimate)
- Cloud Computing Patterns (CCP) icons and cloud DSLs
  - See e.g. AWS [https://aws.amazon.com/de/architecture/icons/](https://aws.amazon.com/de/architecture/icons/)
Architectural Decisions (ADs) – Answering “Why?” Questions

- AD capturing matters, e.g. ISO/IEC/IEEE 42010 has a rationale element
  - But it remains an unpopular documentation task – particularly, but not only in agile communities
  - Effort vs. gain (feeding the beast)?

- Example (from cloud application design): Session State Management
  - Shopping cart in online commerce SaaS (e.g., Amazon) has to be stored while user is logged in; three design options described in literature

  ![Diagram of session state management](image)

  "In the context of the Web shop service, facing the need to keep user session data consistent and current across shop instances, we decided for the Database Session State Pattern from the PoEAA book (and against Client Session State or Server Session State) to achieve cloud elasticity, accepting that a session database needs to be designed, implemented, and replicated."


© Olaf Zimmermann, 2016.
Y-Template for Architectural Decision Capturing

- Link to (non-)functional requirements and design context
- Tradeoffs between quality attributes

In the context of <use case uc and/or component co>, … facing <non-functional concern c>, … we decided for <option o1> and neglected <options o2 to oN>, … to achieve <quality q>, … accepting downside <consequence c>.

Approach: Refactor decision capturing templates into problem-option-driver fragments and change tone, to separate concerns and to ease reuse

“In the context of the Web shop service, facing the need to keep user session data consistent and current across shop instances, we decided for the Database Session State Pattern from the PoEAA book (and against Client Session State or Server Session State) to achieve cloud elasticity, accepting that a session database needs to be designed, implemented, and replicated.”

“When designing a stateful user conversation (for instance, a shopping basket in a Web shop), you will have to decide whether and how session state is persisted and managed.” (question: is this a requirement or stakeholder concern?)

“Your conceptual design options will be these patterns: Client Session State, Server Session State, and Database Session State.” (question: are patterns the only types of options in AD making?)

“The decision criteria will include development effort and cloud affinity.” (question: what else influences the decision making?)
ADMentor Tool (AddIn to Sparx Enterprise Architect)

- ADMentor is openly available at https://github.com/IFS-HSR/ADMentor

Architectural Decision Guidance across Projects
Problem Space Modeling, Decision Backlog Management and Cloud Computing Knowledge

Olaf Zimmermann, Lukas Wegmann  
Institute for Software  
Hochschule für Technik (HSR FHO)  
Rapperswil, Switzerland  
{firstname.lastname}@hsr.ch

Heiko Koziolek, Thomas Goldschmidt  
Research Area Software  
ABB Corporate Research  
Ladenburg, Germany  
{firstname.lastname}@de.abb.com

(WHY)Y?

- My version (the Y-approach):
  - In the context of <use case/user story u>, facing <concern c>, we decided for <option o> to achieve <quality q>
  - These Y-statements yield a bullet list of open/closed (design) issues (link to project management)
  - Can go to appendix of software architecture document, notes attached to UML model elements, spreadsheet, team space, or wiki

Project website http://www ifs hsr ch/index php?id=13201&L=4
Architecturally Evident Coding Styles (George Fairbanks)

- [http://dl.acm.org/citation.cfm?id=1869627](http://dl.acm.org/citation.cfm?id=1869627)

**Examples:**
- Use layering to structure namespace, e.g., packages in Java
- Use architectural role in class names (e.g. OrderController)
- Document architectural decisions in code (ZIO)
  - e.g. with comments or [custom AD annotations](http://dl.acm.org/citation.cfm?id=1869627) (WICSA/CompArch 2016 demo)

**Critique:**
- Incomplete, e.g. what about state management (IDEAL properties)?

**Competition:**
- Model-Driven Software Development (MDSD)?
- Standard agile practices and architecture recovery?
Architecture Roadmapping (Eltjo Poort)

- Identifies future events with architectural significance and assesses their impact
- Tables plus graphical roadmap visualization (SEI SATURN 2015, IEEE Software)

<table>
<thead>
<tr>
<th>Event</th>
<th>When expected</th>
<th>Impact type</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>New version of MongoDB</td>
<td>12/2016</td>
<td>Cost</td>
<td>Has new features and QoS options</td>
</tr>
<tr>
<td>Partner project finished</td>
<td>3/2017</td>
<td>Risk</td>
<td>Inbound dependency (relies on our API)</td>
</tr>
<tr>
<td>Competitor announces new clouds service</td>
<td>6/2017</td>
<td>Business value</td>
<td>Changes workload patterns, might have impact on revenue stream</td>
</tr>
</tbody>
</table>

- Full practice described in “Architecting the Time Dimension” (Insights):  

- RCDA Method (also by Eltjo Poort, in use at CGI):  
From Traditional Layer-Tier Architectures to Cloud Services

Traditional Applications
- Discrete Applications (Two or Three Tiers)
  - Users
  - Logic
  - Data

SOA
- Basket of Services

Decision-Centric Architectural Refactoring for Cloud (ARC)

Visual Model Of NIST Working Definition Of Cloud Computing
http://www.csrc.nist.gov/groups/SNS/cloud-computing/index.html

Essential Characteristics
- Resource Pooling
- Broad Network Access
- Rapid Elasticity
- Measured Service
- On-Demand Self-Service

Delivery Models
- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

Deployment Models
- Public
- Private
- Hybrid
- Community
Refactoring are “small behavior-preserving transformations” (M. Fowler 1999)

Code refactorings, e.g. “extract method”
- Operate on Abstract Syntax Tree (AST)
- Based on compiler theory, so automation possible (e.g., in Eclipse Java/C++)

Catalog and commentary: http://refactoring.com/

Architectural refactorings
- Resolve one or more architectural smells, have an impact on quality attributes
  - Architectural smell: suspicion that architecture is no longer adequate (“good enough”) under current requirements and constraints (which may differ from original ones)
- Are carriers of reengineering knowledge (patterns?)
- Can only be partially automated
<table>
<thead>
<tr>
<th><strong>Architectural Refactoring:</strong> [Name]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Context (viewpoint, refinement level):</strong></td>
</tr>
<tr>
<td>• [...]</td>
</tr>
<tr>
<td><strong>Quality attributes and stories (forces):</strong></td>
</tr>
<tr>
<td>• [...]</td>
</tr>
<tr>
<td><strong>Architectural smell (refactoring driver):</strong></td>
</tr>
<tr>
<td>• [...]</td>
</tr>
<tr>
<td><strong>Architectural decision(s) to be revisited:</strong></td>
</tr>
<tr>
<td>• [...]</td>
</tr>
<tr>
<td><strong>Refactoring (solution sketch/evolution outline):</strong></td>
</tr>
<tr>
<td>• [...]</td>
</tr>
<tr>
<td><strong>Affected components and connectors (if modelled explicitly):</strong></td>
</tr>
<tr>
<td>• [...]</td>
</tr>
<tr>
<td><strong>Execution tasks (in agile planning tool and/or full-fledged design method):</strong></td>
</tr>
<tr>
<td>• [...]</td>
</tr>
</tbody>
</table>

*Previous Work (SOAD/ADMentor)*

*Novelty*
Architectural Refactoring Example: De-SQL

<table>
<thead>
<tr>
<th>Architectural Refactoring: De-SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Context (viewpoint, refinement level):</strong></td>
</tr>
<tr>
<td>• Logical viewpoint, data viewpoint (all levels)</td>
</tr>
<tr>
<td><strong>Quality attributes and stories (forces):</strong></td>
</tr>
<tr>
<td>• Flexibility, data integrity</td>
</tr>
<tr>
<td><strong>Architectural smell (refactoring driver):</strong></td>
</tr>
<tr>
<td>• It takes rather long to update the data model and to migrate existing data</td>
</tr>
<tr>
<td><strong>Architectural decision(s) to be revisited:</strong></td>
</tr>
<tr>
<td>• Choice of data modeling paradigm (current decision is: relational)</td>
</tr>
<tr>
<td>• Choice of metamodel and query language (current decision is: SQL)</td>
</tr>
<tr>
<td><strong>Refactoring (solution sketch/evolution outline):</strong></td>
</tr>
<tr>
<td>• Use document-oriented database such as MongoDB instead of RDBMS such as MySQL</td>
</tr>
<tr>
<td>• Redesign transaction management and database administration</td>
</tr>
<tr>
<td><strong>Affected components and connectors (if modelled explicitly):</strong></td>
</tr>
<tr>
<td>• Database</td>
</tr>
<tr>
<td>• Data access layer</td>
</tr>
<tr>
<td><strong>Execution tasks (in agile planning tool and/or full-fledged design method):</strong></td>
</tr>
<tr>
<td>• Design document layout (i.e., the pendant to the machine-readable SQL DDL)</td>
</tr>
<tr>
<td>• Define index for document access</td>
</tr>
<tr>
<td>• Write new data access layer, implement SQLish query capabilities yourself</td>
</tr>
<tr>
<td>• Decide on transaction boundaries (if any), document database administration (CRUD, backup)</td>
</tr>
</tbody>
</table>
Architectural Refactoring Tool Support (Thesis Projects)

- Web Collaboration Tool based on Angular JS, Play, MySQL
  - https://github.com/bisigc/art

Architectural smells

**Migration Problems**

It takes rather long to migrate existing data after an update to the data model (database schema).

**ZO-Smell**

dds

Architectural Refactoring Tool

ARs deal with architecture documentation and the architecture's manifestation in the code and run-time artifacts. A single architectural syntax tree doesn’t exist. ARs pertain to components and connectors (modeled, sketched, or represented implicitly in code), design decision logs (in structured or unstructured text), and planning artifacts such as work items in project management tools. ARs address architectural smells, which are suspicions or indications that something in the architecture is no longer adequate under the current requirements and constraints, which might differ from the original ones. An AR, then, is a coordinated set of deliberate architectural activities that remove a particular architectural smell and improve at least one quality attribute without changing the system's scope and functionality. An AR might negatively influence other quality attributes, owing to conflicting requirements and tradeoffs.
What is SOA? (Source: OOPSLA Tutorials 2004-2008)

No single definition – “SOA is different things to different people”

- A set of services that a business wants to expose to their customers and partners, or other portions of the organization.

- An architectural style which requires a service provider, a service requestor (consumer) and a service contract (a.k.a. client/server).

- A set of architectural patterns such as enterprise service bus, service composition, and service registry, promoting principles such as modularity, layering, and loose coupling to achieve design goals such as separation of concerns, reuse, and flexibility.

- A programming and deployment model realized by standards, tools and technologies such as Web services and Service Component Architecture (SCA).

Adapted from: [IBM SSS]
The Seven ZIO Tenets for Microservices Implementations of SOA

1. *Fine-grained interfaces* to single-responsibility units that encapsulate data and processing logic are exposed remotely, typically via RESTful HTTP resources or asynchronous message queues.

2. Business-driven development practices such as *Domain-Driven Design (DDD)* are employed to identify and conceptualize services.

3. Cloud-native application design principles are followed, e.g., as summarized in *Isolated State, Distribution, Elasticity, Automated Management and Loose Coupling (IDEAL)*.

4. Multiple storage paradigms are leveraged (SQL and NoSQL) in a *polyglot persistence* strategy.

5. *Lightweight containers* are used to deploy services.

6. *Decentralized continuous delivery* is practiced during service development.

7. Lean, but holistic and largely automated approaches to configuration and fault management are employed (a.k.a. *DevOps*).

A Software Architect’s Dilemma….

Step 1: Analyze System
- Entity-relationship model
- Use cases
- System characterizations
- Aggregates (DDD)

Coupling information is extracted from these artifacts.

Step 2: Calculate Coupling
- Data fields, operations and artifacts are nodes.
- Edges are coupled data fields.
- Scoring system calculates edge weights.
- Two different graph clustering algorithms calculate candidate service cuts (=clusters).

Step 3: Visualize Service Cuts
- Priorities are used to reflect the context.
- Published Language (DDD) and use case responsibilities are shown.

Technologies:
Java, Maven, Spring (Core, Boot, Data, Security, MVC), Hibernate, Jersey, Jhipster, AngularJS, Bootstrap
https://github.com/ServiceCutter
Putting it all together…

To craft architectures and systems of longevity and endurance:

1. Specify ASRs in a SMART way, e.g., in the form of quality stories.
2. Make conscious decisions and provide rationale, e.g., in Y-statements.
3. Model context, containers, components, classes (C4) – but not more.
4. Apply an Architecturally Evident Coding Style (AECS).
5. Architect the time dimension, e.g. in the form of event tables and roadmaps.
6. Practice architectural refactoring, revisiting the team’s decisions along the way.
7. Consider the SOA style and its microservices implementation approach.

Future work:

- Document these seven framework/method elements in a lean way (template)
- Provide examples and discuss relationship to existing methods and practices
Agenda

- **Context and motivation**
  - Agility vs. sustainability

- **Towards an Open and Lean Architecting Framework**
  1. SMART NFRs and quality stories
  2. C4 architecture modeling
  3. Decision sharing with Y-statements
  4. Architecturally evident coding styles
  5. Architecture roadmapping
  6. Architectural refactoring
  7. (Micro-)services principles and patterns

- **The Software Architect’s Role in the Digital Age (IEEE Software)**
  - Changes
  - Challenges
CURRENT ISSUE: THE ROLE OF THE SOFTWARE ARCHITECT

The Software Architect’s Role in the Digital Age

Giuseppe Delisa, Allianz SE

Spek-Ostapa, Carnegie Mellon Software Engineering Institute

NoviEditor, University of Vienna

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

Architectural Refactoring for the Cloud (ARC)
Cloud Knowledge Sources
Microservices Resources and Positions
Domain-Driven Design Overview and Links
DevOps Resources and Positions

Online Resources for Software Architects

The November/December 2016 Theme Issue of IEEE Software on the Role of the Software Architect in the Digital Age is a good starting point (Guest Editor’s Introduction to Theme Issue as PDF).

Websites by thought leaders that we frequently consult (among many others) are:
1. Martin Fowler’s bliki
2. Gregor Hohpe’s Ramblings
3. Philippe Kruchten’s Weblog
4. Eoin Wood’s website and blog at Arteche
5. Michael Stal’s software architecture blog
6. The Software Architecture Handbook website by Grady Booch
7. Personal page of Gernot Starke (mostly in German) - arc42, aim42, IT architect profession
8. Technical Reports and other publications in the Digital Library of the Software Engineering Institute (SEI)
9. The Open Group website - IT Architect Certification, TOGAF, ArchMate, XML
10. Object Management Group (OMG) - UML, SPEM, MDA, CORBA, ADM, KDM
11. IEEE Software, as well as SWEBOK and the very readable standard for architecture descriptions, ISO/IEC/IEEE 42010

The following conferences have a practitioner focus on all things software architecture are (most of the presentations are available online and can be accessed from the conference websites):
1. SEI SATURN, e.g. SATURN 2011
2. Industry Day at CompArch/WICSA 2011
3. ECSA 2014 also had an Industry Day
4. OOP (most talks in German, presentations not available online by default)
5. SPLASH and OOPSLA (e.g. practitioners reports program at OOPSLA 2008)

If you are new to the field, you can get started by reviewing the arc42 site (in German) or look for architectural guidance and practices in OpenUP. If you have a little more time to study, many excellent books on the topic are available to you, including (but of course not limited to):
Practical Challenges and Research Problems

In the context of “agility meets sustainability”:

- Visualization that meets designer’s needs, pragmatic modeling
  - Reverse trend in recent years: less formal, more text (due to SE tools?)
- Architectural handbook still missing, pattern languages standing in
  - Consistency (across pattern languages)? Collaboration (on content)?
    - SE-Repo project at HSR FHO (ongoing)
- Meaningful software architecture metrics (beyond static code analysis)?
  - See ICSE SAM 2015 workshop paper for my suggestions (from practice)

- More changes and challenges:
  - See Uwe Zdun’s presentation “The Future of Architecting and Architecture Decision Making in the Context of the Software Engineering Lifecycle”
  - To be given as part of ECSA Anniversary Day (Friday)
Thank you! References and Contact Information

Olaf Zimmermann
HSR FHO
Software Architecture, Architectural Decisions, Architectural Refactoring, SOA, Cloud
Bestätigte E-Mail-Adresse bei hsr.ch - Startseite

Title | Zitiert von | Jahr
--- | --- | ---
Re-engineering data-centric information systems for the Cloud—A method and architectural patterns promoting multi-tenancy | A Furda, C Fidge, A Barros, O Zimmermann | 2017
Welcome Message from the Chairs of WICSA, QoSA and CBSE | H Muccini, KE Harper, R Heinrich, J Bosch, N Plouzeau, O Zimmermann, ... | 2016
Microservices Tenets: Agile Approach to Service Development and Deployment | O Zimmermann | 2016
A Decade of Enterprise Integration Patterns: A Conversation with the Authors | O Zimmermann, C Pautasso, O Holpe, B Woolf | 3 2016
Guidelines and Metrics for Configurable and Sustainable Architectural Knowledge Modelling | C Canullo, R Capilla, O Zimmermann, U Zdun | 1 2015
Architectural decision guidance across projects—problem space modeling, decision backlog management and cloud computing knowledge | O Zimmermann, I Wamman, H Knoblach, T Goldschmidt | 6 2015

- ozimmerm@hsr.ch
- LinkedIn
- XING
- Personal IFS Website
- Research Gate
- Bibtex file (at DBLP)

(screen caption clickable)