Copyright (unless noted otherwise): Olaf Zimmermann, 2017. All rights reserved.

Topic: *Architectural Decisions (ADs)*

Context

Pattern-centric service design as covered in lessons 9 to 12 involves a number of Architectural Decisions (ADs) that must be made consciously and documented efficiently – like many other ADs (see the group discussion on architectural decisions in the guest lecture in week 13 for evidence). The IBM UMF work product description ART 0513 f.k.a. ARC 100 (in company-internal use at least since 1998) motivates the need for AD capturing:

"The purpose of the Architectural Decisions work product is to:

- Provide a single place to find important architectural decisions
- Make explicit the rationale and justification of architectural decisions
- Preserve design integrity in the provision of functionality and its allocation to system components
- Ensure that the architecture is extensible and can support an evolving system
- Provide a reference of documented decisions for new people who join the project
- Avoid unnecessary reconsideration of the same issues"

The IEC/IEEE/ISO 42010:2011[1] standard for architecture description specifies ADs and their rationale (dt. "Begründungen") as first class entities in architecture documentation (template).

Concept(s) and Definition(s)

Architectural Decisions (ADs)[2] are "the design decisions that are costly to change" (G. Booch, 2009). AD Records (ADRs) capture the rationale justifying a design, in addition to the design itself; they answer "why" questions.

More elaborately, "architectural decisions directly or indirectly determine the non-functional characteristics of a system. Each decision describes a concrete, architecturally significant design issue (design problem) for which several potential solutions (options) exist, and provides rationale for the selection of the chosen solution(s), e.g., by referencing desired quality attributes. Architectural decisions concern a software system as a whole, or one or more of the core components of such a system. Examples are the selection of programming languages and tools, of architectural patterns, of integration technologies, and of middleware assets." (Zimmermann (2009)).

Example(s)

A short decision capture is: "We selected the Layers pattern to make the core banking SOA future proof, e.g., to be able to add user channels in a flexible manner."

(WH)Y statements, introduced in a SATURN 2012 presentation[3] and an IEEE Software/InfoQ article Sustainable Architectural Design Decisions[4] (Zdun et al. (2013)), can link ADs to (non-)functional requirements and design context and express tradeoffs between quality attributes.

The following AD capture uses the Y template:

[1] http://www.iso-architecture.org/ieee-1471/index.html
[2] https://en.wikipedia.org/wiki/Architectural_decision
[3] https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=22132
[4] https://www.infoq.com/articles/sustainable-architectural-design-decisions

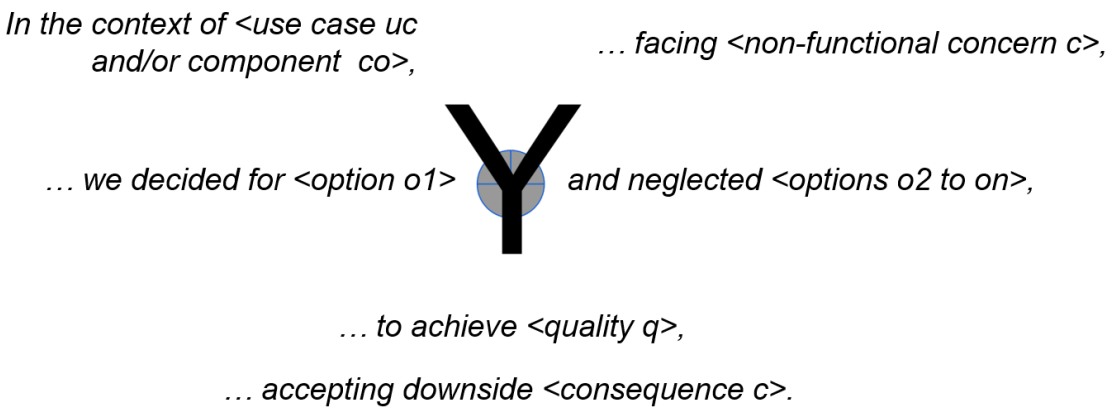

Figure 1: *Y Statement Template (Source: ABB, first presented at SATURN 2012 conference)*

```
AD 1: Apply Messaging pattern and RPC pattern

In the context of the order management scenario at ”T”,
facing the need to process customer orders synchronously, without loosing any messages,
we decided to apply the Messaging pattern and the RPC pattern
and neglected File Transfer, Shared Database, no physical distribution (local calls)
to achieve guaranteed delivery and request buffering when dealing with unreliable data sources
accepting that follow-on detailed design work has be performed
and that we need to select, install, and configure a message-oriented middleware provider.
```

The Figure “AD captured in IBM table format” shows a decision record that uses a different, even more elaborate template.

Subject Area	Process and service layer design	**Topic**	Integration
Name	Integration Style	**AD ID**	2
Decision Made	We decided for RPC and the Messaging pattern (Enterprise Integration Patterns)		
Issue or Problem	How should process activities and underlying services communicate?		
Assumptions	Process model and NFRs/QA requirements are valid and stable		
Motivation	If logical layers are physically distributed, they must be integrated.		
Alternatives	File transfer, shared database, no physical distribution (local calls)		
Justification	This is an inherently synchronous scenario: VSP users as well as internal Telco staff expect immediate responses to their requests. Messaging system will ensure guaranteed delivery and buffer requests to unreliable data sources.		
Implications	Need to select, install, and configure a message-oriented middleware provider.		
Derived Requirements	Many finer grained patterns are now eligible and have to be decided upon: message construction, channel design, message routing, message transformation, system management (see Enterprise Integration Patterns book).		
Related Decisions	Next, we have to decide on one or more integration technologies implementing the selected two integration styles. Many alternatives exist, e.g., Java Message Service (JMS) providers.		

Figure 2: *AD captured in IBM table format*

Application of the Concept in Products and Projects

Usage of Concept/Topic

The importance of decisions has been recognized since the beginnings of the software architecture field in the 1990s (see an IEEE Software article for a brief history as of 2016[5]), but the concept remained somewhat neglected in methods and tools; this has changed since 2004. AD capturing is now mainstream: ADs are captured in various notations and formats in industry. Filled out template instances can be captured e.g. in Markdown files, office documents, or wiki pages (and even spreadsheets are used).

Architectural Decision Records (ADRs) can also be embedded into code, e.g., via comments or custom annotations, or formatted with markdown (and versioned jointly). See for instance the ADR project[6], a community effort on GitHub that provides an e-adr Annotations library for Java as well as support for Markdown ADRs (MADRs)[7].

See IFS website on Architectural Knowledge Management (AKM)[8] for links to additional templates.

Tips and Tricks

Good and bad AD justifications are listed in the two figures "Examples of AD justifications".

Decision driver type	Valid justification	Counter example
Wants and needs of external stakeholders	Alternative A best meets user expectations and functional requirements as documented in user stories, use cases, and business process model.	End users want it, but no evidence for a pressing business need. Technical project team never challenged the need for this feature. Technical design is prescribed in the requirements documents.
Architecturally significant requirements	Nonfunctional requirement XYZ has higher weight than any other requirement and must be addressed; only alternative A meets it.	Do not have any strong requirements that would favor one of the design options, but alternative B is the market trend. Using it will reflect well on the team.
Conflicting decision drivers and alternatives	Performed a trade-off analysis, and alternative A scored best. Prototype showed that it's good enough to solve the given design problem and has acceptable negative consequences.	Only had time to review two design options and did not conduct any hands-on experiments. Alternative B does not seem to perform well, according to information online. Let's try alternative A.

Figure 3: *Examples of AD justifications (1/2)*

Practical challenges include that retrospective decision capturing takes time and does not yield sufficient *immediate* benefits. Furthermore, the relation to other architectural concepts and viewpoints (quality attributes, patterns) is not understood fully yet and not supported well in methods and tools.

Related Topics and Concepts

Methods, agile architecting, architectural evaluation.

[5] http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7725214
[6] https://github.com/adr
[7] https://github.com/adr/madr
[8] https://www.ifs.hsr.ch/index.php?id=13191&L=4

Decision driver type	Valid justification	Counter example
Reuse of an earlier design	Facing the same or very similar NFRs as successfully completed project XYZ. Alternative A worked well there. A reusable asset of high quality is available to the team.	We've always done it like that. Everybody seems to go this way these days; there's a lot of momentum for this technology.
Prefer do-it-yourself over commercial off-the-shelf (build over buy)	Two cornerstones of our IT strategy are to differentiate ourselves in selected application areas, and remain master of our destiny by avoiding vendor lock-in. None of the evaluated software both meets our functional requirements and fits into our application landscape. We analyzed customization and maintenance efforts and concluded that related cost will be in the same range as custom development.	Price of software package seems high, though we did not investigate total cost of ownership (TCO) in detail. Prefer to build our own middleware so we can use our existing application development resources.
Anticipation of future needs	Change case XYZ describes a feature we don't need in the first release but is in plan for next release. Predict that concurrent requests will be x per second shortly after global rollout of the solution, planned for Q1/2009.	Have to be ready for any future change in technology standards and in data models. All quality attributes matter, and quality attribute XYZ is always the most important for any software-intensive system.

Figure 4: *Examples of AD justifications (2/2)*

Background Information: *AD Capturing Practice(s)*

Context

Each software system has an architecture, which is defined by the set of architectural decisions[9] made. Hopefully, these decisions are made consciously and continuously, and consensus is reached about them. This decision making progress should be recorded to avoid knowledge vaporization.[10]

Purpose (When to Use and When not to Use)

Let's express the goal of AD capturing in a user and quality story from/for a fictitious architect: "As a technical team lead (architect, lead developer, product manager), I would like to keep track of the results of all architecture design activities in a compact and searchable form so that newcomers can get a quick overview of the key decisions made and the entire team (and interested external stakeholders) have a single point of reference when making follow-on decisions. From a quality point view, I look for accountability (of decision makers), consistency (between decisions, between decisions and implementation artifacts), and continuity (or currentness of the design artifacts)."

There are no reasons not to use such practice. Two variants exist, *continuous architectural decision capturing* and *stage-based architectural decision capturing*. A stage refers to a phase or iteration, for instance a sprint in Scrum and other agile methods.

[9] https://en.wikipedia.org/wiki/Architectural_decision

[10] Not all organization can or want to follow a "products over projects" philosophy; team members come and go, e.g., external consultants. Some of these tend to apply the "I know better, why haven't you done this and that?" tactic when joining and forget to document their legacy when leaving. Keeping a decision log current is a good safety belt in such (and similar) situations.

Instructions (Synopsis, Definition)

Keep a log of architectural decisions in a single place that is accessible to the entire team. This can, for instance, be a team-wide source modeling or code repository, a project-wide wiki, or a shared office document (Word processor, spreadsheet). As the first decision to be captured, select a *verbosity level* for the decision log:

- At a minimum, document the decision outcome ("we chose") and provide basic rationale (for instance, in the form of a "because" half sentence).
- On a medium level of verbosity, use one of the lean *Architectural Decision Record (ADR)* templates such as MADR[11], Y-Statements[12] or Nygard's ADRs[13].
- If needed or desired, for instance due to external stakeholder pressure to reach certain compliance or maturity levels, apply one of the full-fledged decision modeling templates form the literature.

See this website[14] and this paper[15] for an overview of selected existing templates (both lean and full-fledged).

Examples

Example of a minimal decision capture:

```
ADR-001: We decided for MySQL as our relational database in the backend
because this database has been approved strategically by our EAM group,
has good tool support and sufficient performance under the workload
that we expect to be caused by the user stories to be implemented in the next 10 sprints.
```

See slide 18 in a SAGRA 2016 keynote[16] for a medium verbose ADR.

See a SATURN 2010 presentation for a high fidelity example (slide 14)[17][18]

Benefits vs. Effort (Expected Benefits, Skill Levels)

There are few short-term incentives for decision capturing, which has been reported to pay off in the medium to long time. Many architects report that out of all views on software architecture they provide in their software architecture documents, the AD part/decision log has been read and commented on and appreciated the most. No special skills are required; anybody who can make a decision (or contribute to a collaborative decision making effort) should be able to capture the outcome and the supporting arguments.

Comparing the two definitions of software architecture (components and connectors vs. set of design decisions), one can view the latter as the software engineering/architecting process equivalent of event sourcing[20], with all its pros and cons.

See this SATURN 2013 BoF session report[21] for more information.

[11] https://github.com/adr/madr
[12] http://www.sei.cmu.edu/library/abstracts/presentations/zimmermann-saturn2012.cfm
[13] http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions
[14] https://www.ifs.hsr.ch/index.php?id=13191&L=4
[15] http://www.ifs.hsr.ch/fileadmin/user_upload/customers/ifs.hsr.ch/Home/projekte/ADMentor-WICSA2015submissionv11nc.pdf
[16] https://sagra2016.files.wordpress.com/2016/10/zio-towardsopenleanarchitectureframework-sagranov2016v10p.pdf
[17] http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=22124%3E
[18] This presentation was awarded the IEEE Software New Directions for Architecting Practices Award; see here[19] for a subsequently published article on decision knowledge reuse.
[20] https://martinfowler.com/eaaDev/EventSourcing.html
[21] http://www.sei.cmu.edu/library/assets/presentations/zimmermann-saturn2013.pdf

Application Hints and Pitfalls to Avoid (Common Pitfalls)

See this article for tips and tricks: https://www.infoq.com/articles/sustainable-architectural-design-decisions

Some additional rules of thumb:

- Stick to one template throughout a project; it is less relevant which one you pick – as long as you pick one.
- Focus on the essence of the decision, but make sure the individual ADRs and the entire decision log can serve their purpose (see above): quick orientation, long time reference.
- Don't spend more time on decision capturing than on decision making (and preparing this activity).
- Assign an identifier and create minimal meta information such as decision owner/makers, status and time stamp.
- See slides 51 and 52 in this presentation[22] for examples of good and bad justifications; for instance, "will look good on my CV" does not qualify as a sound decision rationale for a technology selection decision (although it is understandable why to come up with such argument).

Origins and Signs of Use

AD capturing has been a key element in iterative and incremental software engineering methods such as (R)UP and IBM UMF (and its predecessors) since the late 1990s. For instance, OpenUP[23] has the notion of an *Architecture Notebook* that calls for decision capturing with a justification as a mandatory element even in minimal project documentation. The term *rationale* also appears in one of the first articles that defined software architecture (by Perry/Woolf). Book authors has always seen decisions to take center stage in architecture design process; see, for instance, works by SEI authors including Attribute-Driven Design (ADD) 3.0, but also patterns books and supporting material such as this rambling[24] on the Enterprise Integration Patterns website. George Fairbanks suggests a single-sentence decision outcome overview as part of his *Architecture Haikus*, which have been used by Michael Keeling (see here[25]).

Around 2014, a sub-community in software architecture research jumped on the topic at conferences such as WICSA and workshops such as SHARK (Zimmermann et al. (2008)). Decision rationale also is a mandatory element in the ISO/IEC/IEEE 42010:2011, Systems and software engineering – Architecture description for architectural descriptions[26]. The lean and agile communities talk about deferring decisions until the last (but not the least) responsible moment; for instance, M. Nygard suggested the ADR format[27] in November 2011. Many organizations and team design their own templates, some of which are based on those in the literature; others are custom made, applying common sense and practices from other fields.

The architects on the project described in Brandner et al. (2004) embedded ADRs in their Software Architecture Document (SAD) directly (in shaded boxes).

More Information

- Via IFS website Architectural Knowledge Hubs[28]

[22] http://resources.sei.cmu.edu/asset_files/Presentation/2012_017_001_31349.pdf
[23] http://epf.eclipse.org/wikis/openup/
[24] http://www.enterpriseintegrationpatterns.com/ramblings/86_isthisarchitecture.html
[25] https://www.neverletdown.net/2015/03/architecture-haiku.html
[26] http://www.iso-architecture.org/ieee-1471/
[27] http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions
[28] https://www.ifs.hsr.ch/index.php?id=13193&L=4

References

Brandner, Michael, Michael Craes, Frank Oellermann, and Olaf Zimmermann. 2004. "Web Services-Oriented Architecture in Production in the Finance Industry." *Informatik-Spektrum* 27 (2):136–45. https://doi.org/10.1007/s00287-004-0380-2.

Zdun, Uwe, Rafael Capilla, Huy Tran, and Olaf Zimmermann. 2013. "Sustainable Architectural Design Decisions." *IEEE Software* 30 (6):46–53. https://doi.org/10.1109/MS.2013.97.

Zimmermann, Olaf. 2009. "An Architectural Decision Modeling Framework for Service-Oriented Architecture Design." PhD thesis, University of Stuttgart, Germany. http://elib.uni-stuttgart.de/opus/volltexte/2010/5228/.

Zimmermann, Olaf, Uwe Zdun, Thomas Gschwind, and Frank Leymann. 2008. "Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method." In *Proc. of Wicsa*, 157–66.