Concept/Topic: *Solution Strategy*

a.k.a. Big Decision Making

Context

All architectural decisions are hard to change by definition; some are even harder to change due to their impact on budgets and people. In solution strategy, such *executive decisions* are made.

Definition(s)

The term *solution strategy* comes from arc42[1] and is also used in Starke (2015): "Fundamental solution decisions and ideas". The term *executive decision* was coined by P. Kruchten in en early paper on architectural decisions[2] and elaborated upon in the decision level concept of SOA Decision Modeling (see this paper[3]). Executives usually have budgets, are involved in strategy work, and look after long-term consequences of decisions, which explains the choice of the name.

In previous weeks, we have already worked with two approaches to prioritizing architecture work, ASR criteria and NFR/QA prioritizing according to business value and technical risk. In solution strategy, the focus should be on the long lasting architectural decisions. For selection and timing advice see:

- Agile Modeling practice Architecture Envisioning[4]
- OpenUP task envisioning the architecture[5]
- P. Kruchten's website and presentations, for instance post on value of architecture[6] in context of technical debt management.

The figure "Solution Outline/Solution Strategy as first Phase in Architectural Synthesis" shows solution strategy (or solution outline, using an IBM term) in its context;design; the output of this first phase can be touched and refined in later phases (iterative and incremental design and development predates the agile movement).

Component identification (as part of Solution Strategy)

The identification of candidate components primarily, but not only for the logical viewpoint is one of the key activities in solution strategy; in exercise one, you were tasked to do this in an ad hoc fashion. We will get to know slightly more structured, systematic approaches later in the lecture. Two related heuristics exist, one starting from functional requirements, one for non-functional requirements.

For any user story or use case, ask yourself:

- Is the actor/role already represented in the context diagram? If not, is the described functionality really in scope? If yes, how many users exist and how many requests do they typically send (of what size)?
- Which domain model elements or other abstractions are required to perform the task (verb) outlined by the case/the story?

[1] http://docs.arc42.org/section-4/
[2] http://courses.ece.ubc.ca/417/public/Kruchten-2004.pdf
[3] http://ozimmer.de/download/QOSA2007_4880_0015_0032.pdf
[4] http://agilemodeling.com/essays/initialArchitectureModeling.htm
[5] http://epf.eclipse.org/wikis/openup/practice.tech.evolutionary_arch.base/tasks/envision_the_arch_FF123A81.html
[6] https://philippe.kruchten.com/2013/12/11/the-missing-value-of-software-architecture/

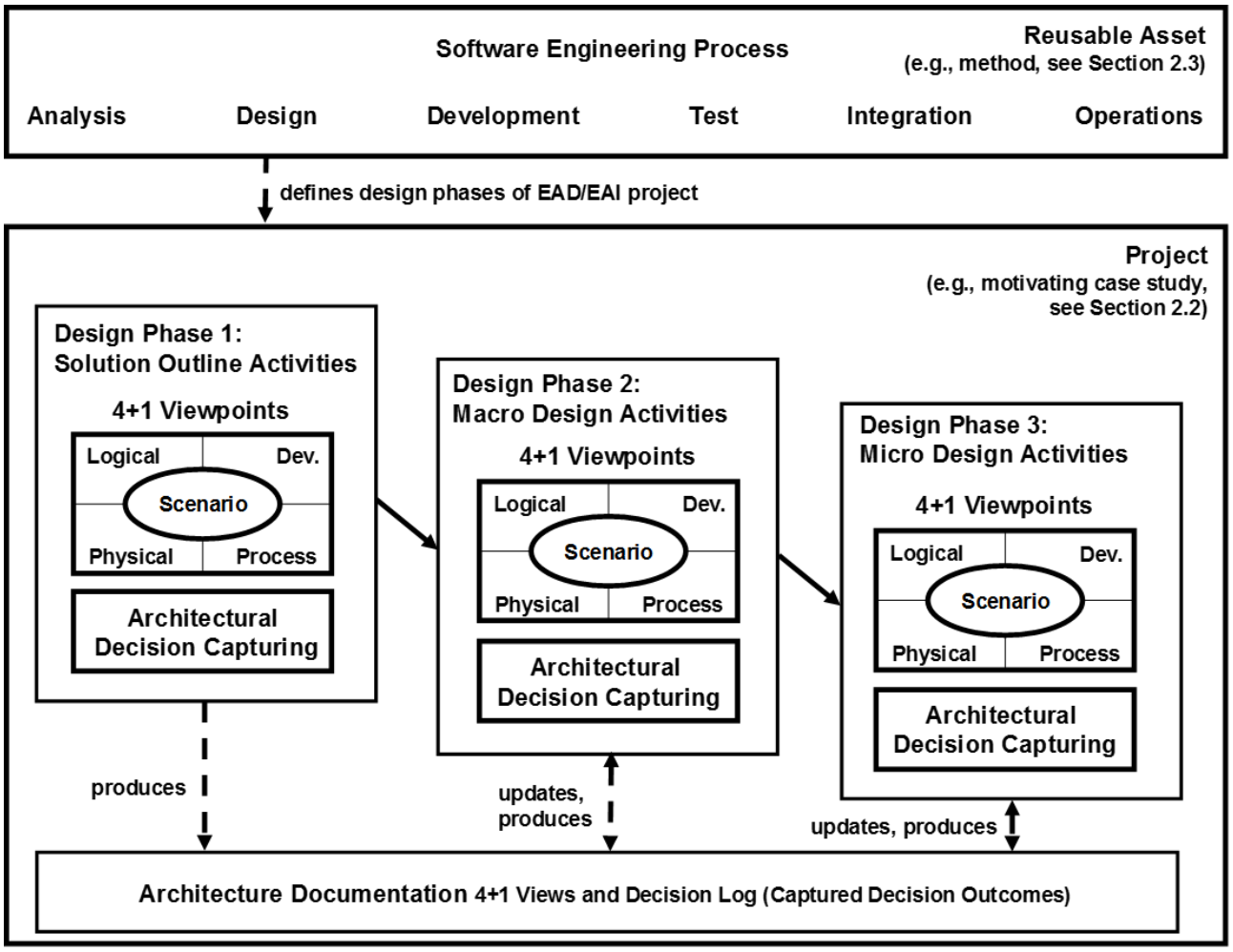

Figure 1: *Solution Outline/Solution Strategy as first Phase in Architectural Synthesis*

- What is the impact of the required functionality on presentation, processing, storage (or other logical layers)?
- Is the feature something the system can do by itself (with the help of a component already present or to be implemented), or does it need help from downstream backend systems (that already appear in the context diagram of have to be added to it?
- Which patterns are eligible, which components do the pattern sketches suggest?
- Are these systems represented in the system context? If not, is the requirement really in scope? If yes, which protocols are used and which service level agreements apply (for performance and availability, for instance)? How do typical and unusual invocations look like (amount and structure of data exchanged, etc.)?
- How can the successful execution be measured (incl. "so that" part of user story or postcondition of use case)?

For any Quality Attribute Scenario (QAS) or quality story, ask yourself:

- Which already identified logical components are affected (the "S" in SMART)? Can they deliver all the quantified behavior (the 'M' in SMART)?
- Are additional components needed? Which layer/tier should they go to?
- What are candidate implementation technologies for them?

In the group work in the Web whiteboard and the discussion during the lecture, the following sources *of candidate* components were mentioned:

- Interviews with customer
- External interfaces to other systems
- Older projects, existing systems
- Identity keywords (nouns) appearing in requirements analysis artifacts, e.g. the system should display customers of a rental customer and rental are possible classes
- Business process models and domain models
- Pattern sketches

Recurring issues (on all layers/tiers) are identified in an article "ADs as reusable assets" in the Jan./Feb. 2011 issue of IEEE Software[7] (more on this in other fact sheets and later in the lecture).

Notation and Templates

arc42 recommends a table format to record the results of solution strategy (in its Section 4[8]; the C4 model[9] suggests to draw a container diagram to give an architecture overview that covers both logical and physical aspects and represents by technology decisions.

As an alternative to the table format, an X-shaped four-tuple can be used (core features, high priority qualities, logical architecture cornerstones, physical architecure essentials). The example in the sample solution to exercise 3 is: "To meet the project vision of an open source, multi-cloud continuous delivery platform that helps release software changes with high velocity and confidence, which requires high availability and competitive performance, Spinnaker employs a workflow-centric thin client architecture that can be deployed to load-balanced clusters running on public infrastructure-as-a-service offerings from Amazon Web Services, Microsoft Azure, and other cloud providers".

[7] http://delivery.qmags.com/d/?pub=ISW&upid=15761SP&fl=others%2fISW%2fISW_20110101_Jan_2011.pdf
[8] http://docs.arc42.org/section-4/
[9] https://c4model.com/

Example(s)

Examples of activities in solution strategy and executive decisions include:

- Buy vs. build with options such as Do-it-Yourself (DIY) vs. Open Source Software (OSS) participation vs. procurement of Commercially-off-the-Shelf (COTS) software (many TLAs are introduced here).[10]
- Client-server cuts and architectural style(s); inclusion or exclusion of external interfaces to be consumed
- Choice of programming language (even in agile age, with autonomous teams and polyglot programing)
- Choice of database vendor (and other complex middleware)
- Choice of method or method mix (practice collection)

See lecture (case study "T" and exercise 3) for more, as well as SOAD articles and book chapter[11].

Application of the Concept in Products and Projects

Usage of Concept/Topic

All project due some solution strategy, either during project planning and preparation (presales) or in Inception (UP). In the agile community that warns of too much "Big Design Upfront (BDUF)", we find terms such as *Architectural Spike*, *Sprint 0* or *Pregame*. Deferring decisions until the last (but not the least) responsible moment is a lean principle that is discussed controversially for instance here[12] and here[13].

Tips and Tricks

- Talk to as many people as possible, and ask a lot of open questions. Some of these people might be stakeholders of the system and architecture under construction.
- Think about long term consequences of your decisions - do they maintain your client/your team's freedom to operate (to use a term from intellectual property management)?
- Apply a "worst first" strategy to decision prioritizing; if this is too challenging for instance because team or requirements are not stable enough yet, try a sandwich approach: simple-difficult-simple to get some experience with decision making and capturing before tackling the really hard ones.
- Be aware of company politics and cognitive bias. Decisions are always somewhat subjective and suboptimal.
- arc42 has 6 tips tagged with Solution Strategy[14].

More Information

A definite public list of early must-decides is still missing; in its absence, try these resources for guidance:

- S. Toths's book "Vorgehensmuster für Software-Architekturen" has a pattern "Gerade genug Architektur vorweg" (Section 4.3, in German).
- M. Fowler's "Patterns of Enterprise Application Architecture", especially the first pages in the narrative part (Fowler (2002))

[10] TLA stands for Three Letter Acronym. See lecture week 4 for coverage of this decision.
[11] http://soadecisions.org/soad.htm
[12] http://www.codeops.tech/blog/architecture/when-to-make-architecture-decisions/
[13] http://wirfs-brock.com/blog/2011/01/18/agile-architecture-myths-2-architecture-decisions-should-be-made-at-the-last-responsible-moment/
[14] http://docs.arc42.org/keywords/#solution-strategy

- The eight steps in Chapter 4 "Layered Application Guidelines" in Microsoft's Application Architecture guide, 2nd Edition[15] and the detailed coverage of each layer in subsequent chapters (note that the book is *not* specific to Microsoft technologies and products).
- The SOAD project proposed to start with recurring meta issues and reusable architectural decision models a.k.a. guidance models (if available).
- arc42 template[16] and related FAQs[17].

Related Topics and Concepts

- ASRs
- SMART NFRs
- Client-Server Cuts
- Component Modeling
- Lightweight architectural decision capturing

References

Fowler, Martin. 2002. *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Starke, Gernot. 2015. *Effektive Software-Architekturen*. 7. Auflage. Hanser-Verlag.

[15] https://msdn.microsoft.com/en-us/library/ee658109.aspx
[16] http://docs.arc42.org/section-4/
[17] http://faq.arc42.org/category_c/#c-sec-4