architecture CHEAT SHEET

Conceptual development expertise in a condensed format

FIND MORE CHEAT SHEETS ONLINE:
Free pdf-Download of the architecture cheat sheet collection:
www.architektur-spicker.de

NO. 6

Agile Architecture

Software architecture in an agile context becomes more dynamic, smaller-scaled and more distributed. This cheat sheet summarises the most important aspects.

IN THIS ISSUE

What's it about? (challenges / goals)

- An agile mindset creates cross-functionality, flexibility, and the ability to work iteratively. How does this change architectural work?
- Agile projects are lean – also in regard to up-front work. How can you work on the architecture in a well-founded and focused fashion?
- Agile methodologies are short on how to do software architecture. How can you deal with architectural tasks and the role of the architect?
- Communication and ad-hoc decisions are more difficult in large scale development environments. How can agile architectural work be done in a reasonable fashion?

Fundamentals on agile ideas & architecture

The top 3 links for an agile mindset:

TOP-3 LINKS

- The principles of the agile manifesto: bit.ly/2xWsrEh
- The complex area of the Cynefin framework: goo.gl/ZNt87M
- The OODA loop for quick development processes and feedback: goo.gl/exBftJ

What do central agile ideas from these sources mean in regard to architectural work?

➔ **Agile architecture** is driven by **requirements**, the **effort spent is adequate** to the given problem, it is influenced by current insights into **collaboration** and **ways of working** and it is **interwoven** with **iterative software development**.

Central ideas on agility	Relevant architectural aspects	
Iterative approach	No big design upfront, lean architecture vision instead Architectural tasks backlog Decide at the last responsible moment	**methodological aspects** • Described in more detail in Vorgehensmuster für Softwarearchitektur Carl Hanser, 2. Auflage 2015, Stefan Toth
Fine-granular feedback	Tests quality aspects Periodically reflect quality goals Continuously identify technical debt	
Cross-functionality/ Team respnsibility	Distribute the architect's role Consensus decisions Principles guiding architectural directions	
Transparency & direct communication	Informative workplace incl. architecture wall Ad-hoc architecture workshops Communities of practice	
Responsiveness/ flexibility	Vertical architecture & domain focus Deep technical isolation Self-Service platforms and infrastructure Technical excellence and focus on maintainability	**technical aspects** • Microservices/Self-Contained Systems • Containerization • Public/Private Cloud (see also Cheat-Sheet Nr.5; currently only available in German)
Product orientation	Evolutionary achitecture Soft architecture standards and eventual integrity Achieve anti-viscosity in architecture implementation	**organisational aspects** • Long-term focus on certain topics • Soft governance • see also Cheat-Sheet p. 4

especially interesting in scaled environments (large scale projects, strong dynamics)

Conception and preliminary work

Architecture Vision

A very lean compilation of architectural drivers (What?) and ideas (How?) as counterpart to a business-oriented product vision.

What?

- **System context** (boundary)
- **Constraints**
- **Quality requirements** (prioritised)
- **Risks** (business and technical)

Foundation for architectural work – importance is independent of context!

How?

- **Basic technologies** (incl. frameworks etc.)
- **Concepts, patterns, principles**
- **Domain-specific structure** (+ domain model)
- **Solutions for coordination and communication**
- **Solutions for integration and interfaces**
- **Persistence strategies and data-model**

First solution ideas* - amount and detail depending on the context!

* The goal is to be able to make a first effort estimation for the system (macro level) – NOT the final specification!

Complexity drivers

- High quality requirements
- Tight project constraints (time, budget)
- Large development team
- High spatial distribution
- New technologies
- Little experience in the solution space
- Slim technical framework
- Many (external) dependencies
- Deviation from standard architecture
- Conflicting objectives

The top 2 differences in contrast to Big Design Up Front (BDUF):

1. **Leaner approach:** Details are decided iteratively and risk-oriented. Pending questions are OK if these can be processed in a planned manner.
2. **Candidates, not decisions:** Final decisions only in non-innovative/known areas without risk. "Candidates" are communicated.

Architecture work in iterations

Work done in **early iterations** should include useful functionality and **touch** as **many architectural approaches** as possible. It is important to touch those architectural approaches at least a little bit in order to falsify them as early and as painlessly as possible. Foundation: The "**Walking Skeleton**".

„A Walking Skeleton is a tiny implementation of the system that performs a small end-to-end function. It need not use the final architecture, but it should link together the main architectural components. The architecture and the functionality can then evolve in parallel."

Alistair Cockburn

Architecture approaches

Domain-specific structure
- [x] .....
- [ ] .....

Concepts, patterns and principles
- [x] .....
- [x] .....

Basic technologies and frameworks
- [x] .....
- [x] .....
- [x] .....

Solutions for coordination & communication
- [x] .....
- [ ] .....

Solutions for integration and interfaces
- [x] .....

Persistence strategies and data-model
- [x] .....
- [x] .....

Platform- and infrastructure elements
- [x] .....

Quality goals should be used as the basis for planning and discussing further product development with the product owner. A high-level checklist to asses **iterative architectural work maturity**:

Level 1	Level 2	Level 3	Level 4	Level 5
Coarse quality statements are known to the team and are considered (e.g. in reviews)	Explicit events asses the achievement of quality goals	Quality scenarios support communication with stakeholders and the product owner	Quality scenarios become acceptance criteria or are kept as dedicated stories in the product backlog	All architecture activities are motivated by quality goals (incl. technical debt)

(Non-functional) tests and CI/CD practices provide feedback regarding achieved objectives: coarse → finegrained

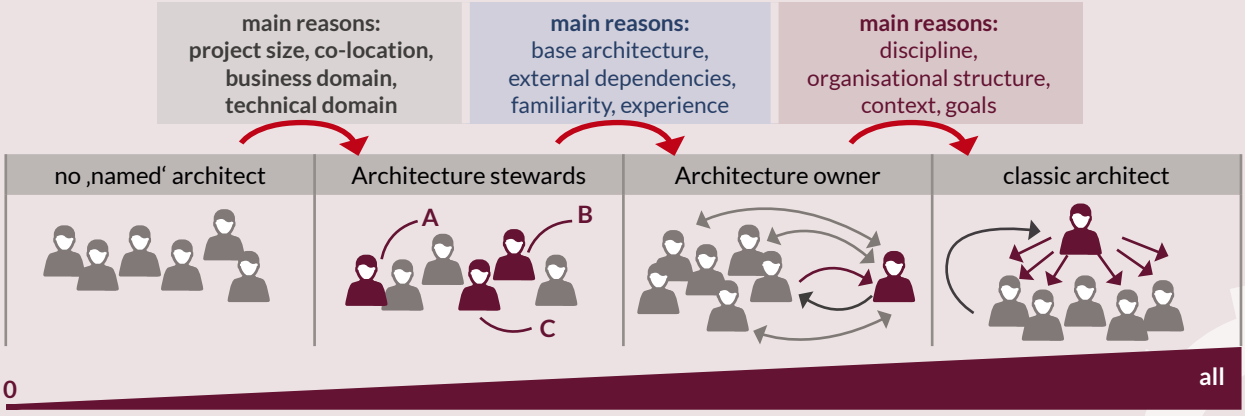
→ Quality scenarios: See also Cheat-Sheet Nr. 4 – Architecture-Reviews

The architect's role

Apart from the classical software architect, there are several options to collaboratively implement the architect's role.
The "right" choice is to be determined according to the environment and problem – "Factors influencing the architect's role".

Factors influencing the architect's role

- **project size:** many teams
- **co-location:** distributed
- **businessdomain:** complex, new
- **technical domain:** hard, challenging, new
- **architecture base:** green field
- **external dependencies:** high
- **familiarity:** first project in this setup
- **experience:** many inexperienced developers
- **discipline:** little responsibility of individuals
- **org. structure:** hierarchical, top-down
- **context:** regulated or heavily standardized
- **goals:** conflicting architecture goals in conflict (also with project goals)



Staying more on the left side of the graphics allows to act more dynamically without neglecting architecture work.
Tactics and practices to achieve this:

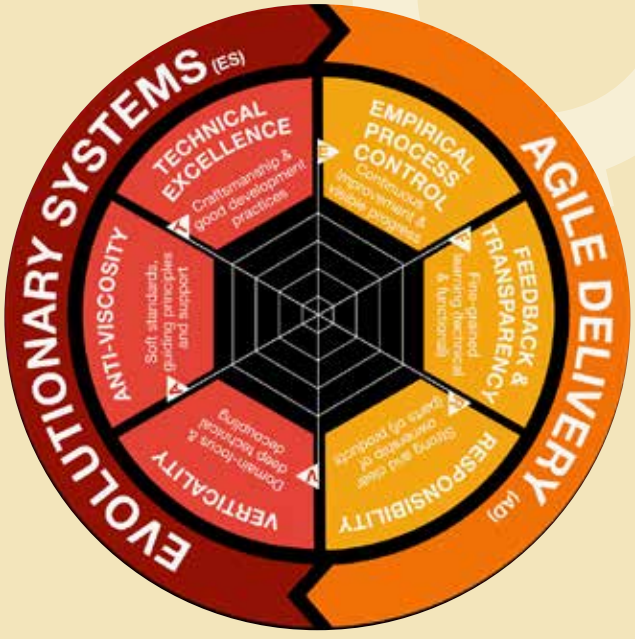
- **Informative workplace:** Visible architectural artefacts (e.g. architecture wall) as basis for communication
- **Group decision:** Consent-based decision-making process to increase sense of accountability
- **Repeated reflection:** Explicit events check the achievement of quality and synchronize software developers.
- **Architectural communities:** Knowledge is shared in dedicated architecture events.
- **Architectural work in the backlog:** Architectural work is made transparent and can be shared by using quality scenarios within the backlog.
- **Architectural principles:** Communicate important insights and mindset – increase the integrity of architectural work.
- **Qualitative, automated tests:** Feedback about achievement of architectural goals creates real accountability.

Scaling agile methods & evolutionary architectures

The biggest agile challenges for software development with 5 teams and more are **maintaining responsiveness** and **well-distributed accountability** without installing bottlenecks. Organisational/methodical aspects of agility have to be combined with the right technical/architectural concepts in order to master these challenges:

„The technical architecture is hugely important for the way we are organized. The organizational structure must play in harmony with the technical architecture. Many companies can't use our way of working because their architecture won't allow it."

Henrik Kniberg (about Spotify)



The **ADES** framework (**A**gile **D**elivery and **E**volutionary **S**ystems) interweaves technical and organisational aspects in order to effectively generate agility in a larger product and company context: www.ADES-Framework.org

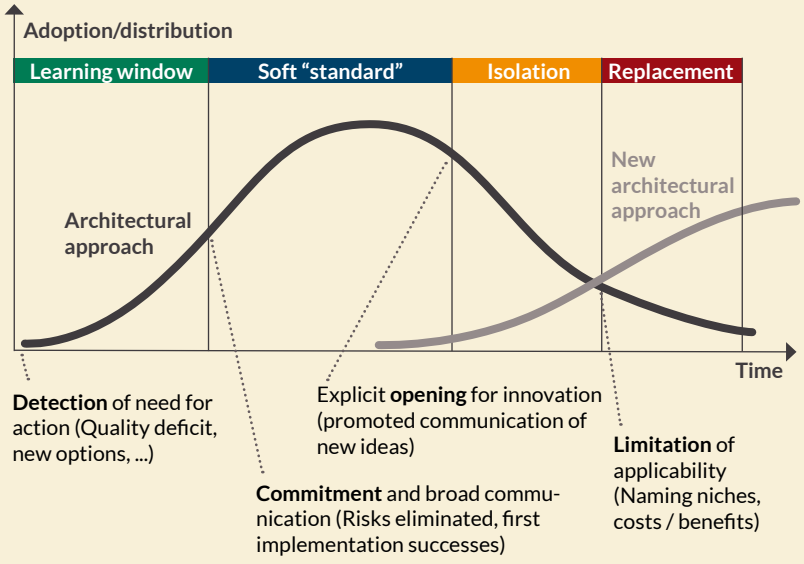
Evolutionary architectures

The left half of the ADES-Framework.

Evolutionary systems/architecture	Counter-model: project-oriented development
Product focus: A long running product delivers the solution to a problem	**Project focus:** Temporary system development, followed by maintenance, ...
Stable Teams, that are linked to the problem/product	**Changing teams** for development and maintenance.
Technically **adaptable architecture base** (not completely new developments)	**Fixed architecture base** for a project / maintenance cycle.
Deep technical **decoupling** between (sub-) domains for small-scale change.	**Technical standardization** and harmonization (not a must but common)
Eventual Integrity – Integrity of the solution is given when good ideas prevail	**Standards First** – Specifications are fixed after analysis, afterwards no deviation / learning
Consistent quality over time.	**Fluctuating quality** over project / maintenance cycle
Constant investment into the problem.	Investment via projects and **time-bound budgets**

➔ Evolutionary approaches are predominantly suitable for new topics, endeavors with high innovation or market pressure: the home of agile approaches.

Evolutionary observation of an architectural question over time:



Important concepts of agile architecture assigned to the phases:

- **Last responsible moment:** Decisions are made as late as reasonably possible in order to expand the learning window and avoid costly errors across a broader context.
- **Verticalisation/Technical isolation:** Lower technical standardisation makes it possible to test new approaches in real environments with small isolated business impact.
- **Anti-Viscosity:** The currently best solution is simplified in its application so that developers do not deviate due to laziness. Examination of achieving goals rather than hard governance.
- **Communities of Practice:** Sharing among developers and dealing with trends is especially important if architectural approaches are beyond their zenith.
- **Eventual Integrity:** Deviation and innovation are always permitted. Ideas distributed via communities that assert themselves against viscosity ultimately lead to integrity.

Further information

- Practices for Scaling Lean and Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum, Craig Larman, Addison Wesley 2010
- ADES Framework: www.ADES-Framework.org
- The top 3 links for an agile mindset (see Page 1)
- Spotify Culture: https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1

We look forward to your feedback: spicker@embarc.de

https://architektur-spicker.de

embarc Software Consulting GmbH
https://www.embarc.de
info@embarc.de

SIGS DATACOM FACHINFORMATIONEN FÜR IT-PROFESSIONALS
https://www.sigs-datacom.de
info@sigs-datacom.de