

Agile Architektur

Geplante Änderungen - die Basis einer agilen Architektur



Michael Kircher

Agile Architecting



Michael Kircher

Me



SIEMENS



Sections



- ❧ Introduction
- ❧ Development Activities
- ❧ Dealing with Technical Debt
- ❧ Bonus: How to become an ‘Agile Architect’

Agile Architecting



Introduction

Agile



- ❧ Lean
- ❧ Value Stream
- ❧ Discipline
- ❧ “Embrace Change”

Architecture



- ❧ “Everything that is expensive to change”, Fowler
- ❧ Three basic elements
 - ❧ Responsibility, Dependency, Interaction
- ❧ CRC – Class-Responsibility-Collaboration

Design



- ❧ Domain analysis –
 - ❧ [Domain-Driven Design]
 - ❧ Or ‘how to find the responsibilities?’ [POSA 5]
- ❧ Optimizations for ‘Qualities’ shape the design
 - ❧ Merge & split responsibilities
 - ❧ Deployment of responsibilities
 - ❧ New dependencies – new interactions
- ❧ Aligning the design with technology

Architect



∞ Technical Leadership

1. Communication
2. Consistency
3. Coaching
4. Coding

Complexity / Simplicity



- ❧ Dependencies
- ❧ Constraints
- ❧ Coupling & Cohesion

- ❧ Terms are relative

Change



- ❧ The only constant is change
 - ❧ Requirements change
 - ❧ Technology changes
 - ❧ Business changes
 - ❧ People change
- ❧ Ability to react on Change depends on the Architecture

Mindset



- ❧ “Embrace change” – “Embrace uncertainty” [North]
- ❧ YAGNI
- ❧ Fail fast – learn fast
- ❧ Systems live longer than we expect
 - ❧ Example Millennium bug
 - ❧ Introduction of SW products well planed, but
Removal/de-installation not planed

Managing Change



- ❧ Control complexity
- ❧ Look ahead
- ❧ Isolate
- ❧ Do not prepare
- ❧ Do not prohibit
- ❧ Trade-off: Cost, time, benefit

Can you plan?



❧ ... what you **know**

❧ Vision vs. Debt

❧ Mobile devices are a fact.

❧ ... what you **guess**

❧ Cloud relevance for your product?

❧ ... what you **hope**

❧ Technology bets: Will Windows Phone 8 catch on?

Size Matters



❧ From: Small Local Team

❧ To: Dispersed Global Teams

Agile Architecting



Development Activities

Vision



- ❧ Demand a product vision ... or
 - ❧ Write one and ask for confirmation
- ❧ Align all stakeholders
- ❧ Say what it is
- ❧ Say what it is NOT
- ❧ Baseline on where you are incl. your technical debt

Requirements



- ❧ Remember: Fail fast. This is your first chance!
- ❧ Domain language
- ❧ Feature model
 - ❧ Complete
 - ❧ Hierarchical
 - ❧ Domain knowledge
- ❧ Managing Variability

Backlog



- ❧ Contains
 - ❧ Customer features
 - ❧ Refactoring & Redesigns
 - ❧ Governance
 - ❧ 'Everything that makes a team busy'


- ❧ Prioritized by Product Owner and Architect

Governance



- ❧ In general: Protect your Qualities
 - ❧ Developmental
 - ❧ Operational
- ❧ In this talk: Flexibility, Extensibility, Maintainability
- ❧ You can only plan with what you control

Governance Example



Preventive	Rules & Guidelines	<ul style="list-style-type: none"> Framework usage 	<ul style="list-style-type: none"> Consistency Complexity
	Variability Management	<ul style="list-style-type: none"> Commonality/Variability analysis Extension points 	<ul style="list-style-type: none"> Extensibility
	Reference Architecture	<ul style="list-style-type: none"> Component types Allowed dependencies 	<ul style="list-style-type: none"> Consistency Maintainability
	Concept Reviews	<ul style="list-style-type: none"> Peer review of critical changes 	<ul style="list-style-type: none"> Sustainability Performance, etc.
	Static Code Checkers	<ul style="list-style-type: none"> Coding guidelines Dependencies, API violations 	<ul style="list-style-type: none"> Stability Performance
	Preventive Tests	<ul style="list-style-type: none"> Unit tests Integration & Smoke Tests 	<ul style="list-style-type: none"> Stability
Corrective	Corrective Tests	<ul style="list-style-type: none"> Integration Tests Staged multi-client-multi-modality tests Performance Tests 	<ul style="list-style-type: none"> Stability Performance Update-ability
	Architecture Review	<ul style="list-style-type: none"> ATAM review Plan big picture course corrections 	<ul style="list-style-type: none"> Sustainability Extensibility Maintainability

Guiding principle: Fail fast!

Always running system, Continuous Integration, Gated Check-in, Staged testing, Continuous System Test

Technology



- ❧ One of the hardest things to change
- ❧ Isolation recommended
 - ❧ Check out 'Quasar' [sd&m, J. Siedersleben]

Coupling & Cohesion



- ❧ Decouple only where you expect change
- ❧ Inversion of control
- ❧ Dependency injection
- ❧ Protocols, Standards, etc.

Patterns



- ❧ Structure
 - ❧ Initial Context
 - ❧ Problem with Forces
 - ❧ Solution with Consequences
 - ❧ Resulting Context
- ❧ Careful usage: Resulting context should be positive!
- ❧ Pattern-Oriented-Software Architecture [POSA]

Test



- ❧ No Test = No Change
- ❧ Interplay of test layers
 - ❧ Unit Tests incl. Mock
 - ❧ Smoke Tests
 - ❧ Automated System Tests

Quality



- ❧ What you do not measure, you do not know.
- ❧ Quality is relative
 - ❧ Define your quality model
 - ❧ Base is the product quality tree
 - ❧ Align the rules with the qualities
- ❧ Derive your Code Checker Rules and their Criticality

Performance



- ❧ No Test, No Trending = No Protection
- ❧ “Worry about it later” is typically the wrong strategy
- ❧ Instead define it up front: Quality Model/Tree
- ❧ Continuously trend it

Documentation



- ❧ “Design decisions as first class citizen” [Jan Bosch]
- ❧ Minimum:
 - ❧ Assumptions / Constraints
 - ❧ Alternatives
 - ❧ Rationale
 - ❧ Decision
- ❧ Architects use also written words to lead
[Writing with Style]

Examples



- ❧ Example: Sync via Dropbox vs. iCloud
 - ❧ Staying in control

- ❧ Example: System Integration
 - ❧ Decoupling release lifecycles

Agile Architecting



Technical Debt

Technical Debt



- ❧ Identification
 - ❧ Explicit business drivers
 - ❧ Quality tree incl. scenarios
 - ❧ Assessment [ATAM]
- ❧ Mapping risks to business drivers & scenarios

Arguing for Big Change



- ❧ Impact & Consequences
- ❧ Big Picture Vision
- ❧ Business Case
 - ❧ Difficult but sometimes necessary

Introducing Big Change



- ❧ Plan
- ❧ Communicate broadly esp. upwards
- ❧ Step-wise
 - ❧ Limit risk
 - ❧ Gain confidence
 - ❧ Celebrate successes

Examples



- ❧ Example: Re-implementation
 - ❧ Workflow module

- ❧ Example: Changing paradigms
 - ❧ Data loading strategy

Agile Architecting



Bonus:
How to become an 'Agile
Architect'

How to become an Agile Architect



- ❧ Know how
 - ❧ Easy to gather
- ❧ Experience
 - ❧ Time will teach
- ❧ Capabilities
 - ❧ Hard to change

Identify the need for each individual architect.

Know How



- ❧ Agile / Lean
- ❧ Test Driven Development
- ❧ Architecture
- ❧ Technologies

Experiences



- ❧ Architecture Reviews
- ❧ Code Reading
- ❧ Patterns
- ❧ Great frameworks [Cocoa]
- ❧ Communities

Capabilities



- ❧ Communication
- ❧ Feedback
- ❧ Leadership (Styles)
- ❧ Coaching (Peer-to-Peer)

Agile Architecting



michael@nature-software.com