Agile Architektur
Geplante Änderungen - die Basis einer agilen Architektur

Michael Kircher
Agile Architecting

Michael Kircher
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
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

<table>
<thead>
<tr>
<th>Preventive</th>
<th>Corrective</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rules &amp; Guidelines</strong></td>
<td><strong>Reference Architecture</strong></td>
</tr>
<tr>
<td>- Framework usage</td>
<td>- Component types</td>
</tr>
<tr>
<td>- Consistency</td>
<td>- Allowed dependencies</td>
</tr>
<tr>
<td>- Complexity</td>
<td>- Sustainability</td>
</tr>
<tr>
<td><strong>Variability Management</strong></td>
<td><strong>Concept Reviews</strong></td>
</tr>
<tr>
<td>- Commonality/ Variability analysis</td>
<td>- Peer review of critical changes</td>
</tr>
<tr>
<td>- Extension points</td>
<td>- Stability</td>
</tr>
<tr>
<td>- Extensibility</td>
<td>- Performance</td>
</tr>
<tr>
<td><strong>Static Code Checkers</strong></td>
<td><strong>Preventive Tests</strong></td>
</tr>
<tr>
<td>- Coding guidelines</td>
<td>- Integration Tests</td>
</tr>
<tr>
<td>- Dependencies, API violations</td>
<td>- Stability</td>
</tr>
<tr>
<td>- Performance</td>
<td>- Update-ability</td>
</tr>
<tr>
<td><strong>Concept Reviews</strong></td>
<td><strong>Corrective Tests</strong></td>
</tr>
<tr>
<td>- Peer review of critical changes</td>
<td>- Integration Tests</td>
</tr>
<tr>
<td>- Sustainability</td>
<td>- Stability</td>
</tr>
<tr>
<td>- Performance, etc.</td>
<td>- Performance</td>
</tr>
<tr>
<td><strong>Preventive Tests</strong></td>
<td><strong>Architecture Review</strong></td>
</tr>
<tr>
<td>- Unit tests</td>
<td>- ATAM review</td>
</tr>
<tr>
<td>- Integration &amp; Smoke Tests</td>
<td>- Plan big picture course corrections</td>
</tr>
<tr>
<td><strong>Corrective Tests</strong></td>
<td><strong>Guiding principle: Fail fast!</strong></td>
</tr>
</tbody>
</table>

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

**Guiding principle: Fail fast!**
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
“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