Crowdsourcing Software Architecture: The Distributed Architect

Stefan Toth
The cross-functional idea

Analyst (RE)  SW Architect Designer  Developer  Build Engineer  Operations

Problem  Solution
My background on this

- Classic Architect by training
- 10+ years of experience in agile projects
- 40+ projects to gather concepts that work
1

Why think about crowd-sourcing?
Classic Software Architects

Often not on the train...
Software architects defined...

The following is defined...

Character of an architect

Know How that an architect needs

Duties of an architect
Character and Know-How

Character

- courage
- diligence
- Pragmatism
- powers of comprehension
- negotiating skills
- capacity of reaction
- patience
- diplomacy
- charisma
- accuracy
- empathy
- accountability
- accessibility
- exploratory urge
- vitality
- flexibility
- entrepreneurial mindset
- strategic view
- moderation skills
- authority
- mutability
- ability to mediate
- passion
- motivation
- creativity
- receptiveness
- Prozessverständnis
- strong leadership skills
- ambition
- sense of proportion

Know-How

- risk management
- organisational processes
- enterprise strategy
- time management
- domain
- static dynamic Code analysis
- decision methods
- evaluation methods
- programming (actual doing)
- project history
- design
- architecture documentation
- IT trends
- competitors
- Implementations techniques
- architectural concepts
- (Patterns, Practices, styles, quality attributes, technological concepts, ...)
- modelling
- stakeholder process models
- relevant technologies
- (not only superficial)
Even if you find such a person...

there are still the **Duties**...

- technically lead the project
- pass on knowledge
- develop prototypes
- mentor
- reflect on architectural work
- ensure integrity
- identify risks
- hand out tasks
- develop the architecture vision
- ensure network
- document architecture
- model
- maintain requirements
- integrate and test solutions
- communicate with stakeholders
- present architecture
- check architecture implementation
- influence the company’s strategy
- escalate important problems
- motivate developers
- implement
- develop spikes
- form teams
- implement
- make decisions
- specify quality requirements
- integrate architecture into the organization
- give context
In general...

You stand a **better chance** by finding a **group of people** who jointly live up to this role.

You are **quicker if more people can react** to requirements or technical problems.

You get a **better system** if the **developers** think **architecture** is their problem.

*Amazon*: “**Nobody take care of a problem like the owner does**”
Agile Principles

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Continuous attention to technical excellence and good design enhances agility.

The best architectures, requirements, and designs emerge from self-organizing teams.
So why crowd-sourcing?

- Realistic concepts
- Broader know-how
- Better motivation of engineers
- Faster reaction/cycle times
- Bottom-up standards (proven / often more stable)
- Better evolution of architecture over time
- Reduced / omitted overburden of ‘the architect’
2

Challenges and Risks
Software Architecture in jeopardy?

2 Challenges

- Getting a useful initial design
- Distributing the ongoing architectural activities

2 Risks

- Lacking Conceptual Integrity
- Wasteful process with low efficiency
3

Initial Design
Twin peaks model

- Vision, Goals
- Quality attributes
- Stories
- Use Cases
- Scenarios
- Acceptance criteria

- Architectural style
- Platform, Standards
- Architectural principles
- Structure
- Integration, Interfaces
- Technical aspects
- Optimizations

Requirements

Solution dependency

Architecture

Level of abstraction

abstract
detailed
neutral
specific
What do we need upfront?

**Architecture Vision**

Focus: critical aspects for the success of the project
important base for the implementation to start

Typical Contents

- **System context** (Boundary)
- **Important constraints**
- **Quality attributes** (prioritized)
- **Quality scenarios**
- **Risks**
- **Technologies** (incl. Plattform, Language)
- **Architectural style** (dominant pattern(s))
- **Architectural principles**
- **High level structure**
- **Domain model** (important domain concepts)

As an idea, not as an irreversible decision!
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
Distribute? Don’t… Tiger Team

1. **Bring together some specialists** with expertise and experience

2. **Let them formulate** an architecture **vision** that is powerful and accepted

3. **Try to integrate them into (different) software development teams to**
   - Represent the vision
   - Recognize it’s non-applicability
   - …
4

Ongoing Architectural Activity
The OODA Cycle

Act… Implement the idea (operational part)

Observe… Recognize potentials, risks or problems

Decide… Select an alternative or candidate (hard part)

Orient… Design and evaluate options (creative part)
1 - Observe

Responsibility for architecture is officially with the team (more eyes)

Architectural awareness within the iteration – Not only implementation, but also a conceptual step back

Tests for quality attributes (from architecture vision) show problems or the decline of effectiveness

Recurring evaluation workshops for feedback and communication purposes (sync)

Observe… Recognize potentials, risks or problems
Architectural Awareness within an iteration
The implementation cycle...
Direct Problem Solving

+ Focused analysis
+ Fast decisions
+ Good to find local optimum

- No synthesis
- Little strategic thought
- Little abstraction
Non-trivial Problems

- High quality requirements
- Complex Trade-Offs
- restrictive constraints
- many options (all not good/perfect)
- many stakeholders
- many teams or system parts affected

Great insights don‘t come from permanent low-level activity
The architecture cycle...
Good teams do this often...

- Define the problem
- Think (Hamock time)
- Talk to others (sharpen vision)
- experiment

abstract (externalize ideas)

learn

Analyze / criticize (your own) solutions
# 1 - Observe

**Observe…** Recognize potentials, risks or problems

- **Responsibility** for architecture is officially **with the team** (more eyes)
- **Architectural awareness** within the iteration – Not only implementation, but also a conceptual step back
- **Tests for quality attributes** (from architecture vision) show problems or the decline of effectiveness
- **Recurring evaluation workshops** for feedback and communication purposes (sync)
1. Look at the big picture

- Were there any changes?
- In which part were architecture efforts?
- Any new risks?
2. Look at the tasks and requirements

- Which done tasks have impact on others?
- Worked on any quality aspects?
- Any requirement changes that have cross-cutting impact?
3. Prioritize gathered architectural topics
Multifaceted presentations – focused but specific
Discussion

How was the topic handled?
- Relevant tactics or candidate decisions

Where/are there any problems or inconvenient constraints?
- Impeding architectural approaches / organizational hurdles, technical debt

Do you understand the solutions and ideas presented?
- Comprehensability and focus

Are all aspects of the problem tackled, is something overlooked?
- additional architectural approaches, open points, risks

Do you see negative implications of the concepts/ideas presented?
- Trade-Offs, risks

Questions for the most informed (presentation)

Questions for the other participants (Exploration)
2 - Orient

**Orient**... Design and evaluate options (creative part)

- **Problem-driven assignment** of tasks to small group or individuals (no design by committee)
- **Architectural requirements** are broken down to **quality scenarios** (often are held in the backlog)
- **Ad-hoc Design Workshops** with colleagues to find options and get context
- Install **mentors for architecture** or important technical topics (role-options we discuss later)
Quality requirements (NFRs)

- Are not easily optimizable in retrospect (expensive)
- Influence high level structure and granularity
- Are often cross-functional in nature (and therefore relevant for many project members)

→ Quality requirements are the most important requirements for architectural work
Quality scenarios
Some examples

- The algorithm for calculating an articles popularity is easily adaptable and exchangeable.

- A user interacts with the platform as the assigned server fails. The problem isn’t apparent to him, even when he tries to navigate back to previous user masks or continues a complex interaction.

- It is easy to find personnel for system maintenance or extension.
Work with architecture requirements

- principle to-be
  - only decide
  - then wide application

- acceptance criteria
  - e.g. on backside of a story-card

- quality story
  - few dependencies,
  - isolated design or evaluation

Product Backlog

Sprint Backlog
Maturity model for Quality-based Software Architecture

1. Accidental architecture, Quality attributes are not known
   - Unstructured

2. Rough Quality statements guide the architectural design
   - Quality-oriented

3. Decisions are evaluated qualitatively and regularly. (With attendance of the most important stakeholders)
   - Evaluated regularly

4. Scenarios are gathered during design of the system and guide the architectural and technical decision making.
   - Scenario-driven
3 - Decide

**Decide**... Select an alternative or candidate (hard part)

*Focus on communication* of architectural drivers, ideas, principles and candidate technologies (Architecture vision, evaluations, blogs, communities, …)

*Build consensus* around architectural decisions by possibly involving every developer (responsibility should be felt)
Consensus

A definition of consensus

"Consensus is a term which describes general agreement among members of a group or community. The cooperative spirit of consensus is different from the competitive spirit which is brought to a point in majority voting."

(Wikipedia)
Consensus focuses on opposition

Best option in his opinion
Systemic process for consensus

Goal:
Support a creative process that allows to discover and combine the advantages of different options and eventually leads to a decision with the lowest possible opposition.

Idea:
Measure opposition on a scale of 0-10

Result:
Option with the lowest combined opposition is chosen
**Tips**

- **Use Dummy Solution:** Everything stays the same

- **Define a Veto-Value:** everything above 7 counts as Veto

- **Simplification:** “Thumb-Voting”

  - Cool!
  - OK...
  - Veto!
Alternative: Fist of Five

Quantification of rejection and approval

Rejection

Never!
This can’t work…

Opposed
Modifications necessary…

Reservations
Let’s discuss
Some issues

Consensus

Yeah, OK
I won’t fight
this…

Good Idea
I’ll support
and promote

Love it!
Full and active
support
4 - Act

**Act**… Implement the idea (operational part)

… actual implementation is easily distributed
… more manpower, no bottlenecks, no NIH-Syndrome, …
5

The problem of scale
Scaling OODA

**Act**… Implement the idea (operational part)

**Observe**… Recognize potentials, risks or problems

**Decide**… Select an alternative or candidate (hard part)

**Orient**… Design and evaluate options (creative part)
Architecture Owner

Character
- Teamspirit
- Courage
- Capacity of reaction
- Strategic view
- Ability to abstract
- Diplomacy
- Powers of comprehension
- Ability to mediate

Duties
- Bring broader context
- Escalate bigger problems
- Raise important architectural questions
- Broaden Know-How
- Coordinate and organize
- ...

Software development
- Architectural concepts
- Design
- Programming
- Business drivers
- Domain
- Technologies

Know-How

Developers

Duties
- Decide on architecture
- Formulate principles
- Identify risks
- Implement proof-of-concepts
- Evaluate architecture
- Integrate and test solution
- Check architecture adherence
- ...

Crowdsourcing Software Architecture
© 2017 embarc Software Consulting GmbH
Architecture Owner

- Keeping an overview
- Supporting by raising questions or co-design
- Proposing principles to speed up design and make it more effective
- Organizing necessary community activities
- Supporting communication on architecture
Architecture marshalls

- Separate **Responsibilities** for Architectural Concerns to different project members (AO per topic)
- For example by **layers**: UI-, Service-, Backend-, Integration-architects
- Or by **qualitative Aspects**: Security-, Performance-, Scalability-, Maintainability-Marshalls
Factors for choosing a role-concept

- **Factors**
  - **project size:** many teams
  - **co-location:** not co-located
  - **business domain:** complex, new
  - **technical domain:** hard, challenging, new
  - **architecture base:** green field
  - **external dependencies:** high
  - **familiarity:**
    - experience:
    - discipline:
    - org. structure:
    - context:
    - goals:
  - **first project in this setup**
    - many inexperienced developers
    - little responsibility of individuals
    - hierarchical, top-down
    - regulated or heavily standardized
    - architecture goals in conflict (also to project goals)

- **Decision making stays with teams**

- **no ‘named’ architect**
  - **main reasons:**
    - project size, co-location,
    - business domain,
    - technical domain

- **Architecture marshalls**
  - **main reasons:**
    - architecture base,
    - external dependencies,
    - familiarity, experience

- **Architecture owner**
  - **main reasons:**
    - discipline,
    - organizational structure,
    - context, goals

- **classic architect**
Software Architecture in jeopardy?

2 Challenges

- Getting a useful initial design
- Distributing the ongoing architectural activities

2 Risks

- Lacking Conceptual Integrity
- Wasteful process with low efficiency
6 Conceptual Integrity
**Modes for conceptual integrity**

### Standards first
- Classic approach
- You do **architectural work** incl. prototypes or spikes until you are “sure enough”. Then you **prescribe** the concept or idea
- **Integrity** is there **from the start** (the prescription).
- **Hard governance**

### Eventual integrity
- Evolutionary approach
- You do **architectural work** to make development more effective. **Gathering** good practices, abstracting ideas.
- **Integrity** is there **when good ideas** and concepts **prevail**.
- **Lower Viscosity** for concepts with broader use or benefits
Low Viscosity

An alternative for rules and prescriptions that
- scales pretty well
- works in complex environments

“When faced with a change, engineers usually find more than one way to make the change. Some of the ways preserve the design, others do not (i.e. they are hacks.) When the design preserving methods are harder to employ than the hacks, then the viscosity of the design is high. It is easy to do the wrong thing, but hard to do the right thing.”

(Robert C. Martin)
The tradeoff between speed and standardization

Classic architecture: broad standardization

- Hard rules
  - Responsibility: Architecture

- Left open
  - Responsibility: Developers

Evolutionary architecture: eventual integrity

- Hard rules
  - Responsibility: Architecture

- Suggestions

- Left open
  - Responsibility: Developers
7

The waste within crowdsourcing architectures
Is Crowd-Sourcing wasteful?

- In theory distributing a interdependent topic is more expensive

- In practice some aspects shift that:
  - Delivery speed as an important goal
  - New technology and innovation as an important factor
  - Complex systems idea where planning is less effective
Projects have all four of these problem domains!

Is your business domain understood or new/shifting?

Is your system/architecture blueprint-based or innovative?
Thank You.

Questions are welcome!

stefan.toth@embarc.de

@st_toth

Download SLIDES:
http://www.embarc.de/blog/