Agile Architecture Maturity Tutorial

The RCDA Maturity Model

Eltjo Poort
SATURN, Pittsburgh
May 2019
Eltjo Poort
CGI

RCDA
Founder / owner
CGI’s agile architecture approach

PhD
Improving Solution Architecting Practices

Architecture
Practice Lead
Reviewer
Coach
Agile transformer

14 Peer-reviewed pubs
2 Best paper awards
3 Best presn awards

2016
Linda Northrop Architecture Award
What is a maturity model?

Capability maturity models focus on improving processes in an organization. They contain the essential elements of effective processes for one or more disciplines.¹

Why measure agile architecture maturity?

- To identify areas of improvement
- To track improvement progress

• The benefits of defined processes in software engineering have been disputed, but establishing good practices is useful.²

1 SEI (2003), CMMI: Guidelines for Process Integration and Product Improvement

The Five Responsibilities of the Architecture Function

1. Understanding Context
2. Modeling
3. Making Decisions
4. Validating
5. Fulfillment

Architecture as a set of design decisions
Tyree, Bosch, Kruchten, Woods

Architecture as an abstraction
Shaw, Garlan

Architecture as a set of structures
SEI, Kruchten, Rozanski & Woods

Architecture as a risk and cost management discipline
Fairbanks, Poort

Architecture is about the important stuff (whatever that is)
Fowler
The Waterfall Wasteland

“We don’t take decisions, we only advise management”

“Our design was perfect, but the builders were incompetent”
The Agile Outback

“The best architectures emerge”

“Fail early and fail often”

Understanding Context

Making Decisions

Modeling

Validating

Fulfillment
An organization’s architecture function is mature if:

- It pays balanced attention to all five responsibilities
- Activities in the five responsibility areas are coherent and related to each other

Organizations can be mature irrespective of how the architecture function is organized, or what it is called (roles, teams, boards,...).
Mapping responsibilities to behavior

Understanding Context
- Effective stakeholder communication
- Context knowledge managed

Modeling
- Visual model of context
- Visual models of solution

Making Decisions
- Decisions as primary deliverable
- Prioritized by business impact
- Justified and documented
- Well-timed decisions
- Decentral unless…

Validating
- Fulfills stakeholder needs?

Fulfillment
- Architectural runway recognized
- Architecture debt control
- Just enough anticipation
Measuring behavior

• Interviews with 5 architects and stakeholders per team
• Asking open questions
• Referencing examples of evidence and counter-evidence

Behaviors scored 1-5:
1 Never
2 Seldom
3 Ad hoc/Individuals
4 Mostly
5 Habit
Agile Architecture Maturity Radar

Understanding Context
- Effective stakeholder communication
- Context knowledge managed

Modeling
- Visual model of context
- Visual models of solution

Making Decisions
- Decisions as primary deliverable
- Prioritized by business impact
- Justified and documented
- Well-timed decisions
- Decentral unless…

Validation
- Fulfills stakeholder needs?

Fulfillment
- Architectural runway recognized
- Architecture debt control
- Just enough anticipation

1 Never  2 Seldom  3 Ad hoc/Individuals  4 Mostly  5 Habit
Understanding Context
Some observations

• Written documentation is never enough to fully understand context.
  • → Talk to stakeholders!
  • make sure you know who they are (all of them)

• Architectural trade-offs can only be made if the goals behind stakeholder needs are understood.
  • → Ask why! (7 times?)

• Vocabulary mismatch is a common source of misunderstanding
  • → Ask for narratives (stories, epics,…) (by asking why)
  • → Speak your stakeholders’ language
## Understanding Context

### Effective stakeholder communication

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX_STKH</td>
<td>Architects communicate effectively with stakeholders</td>
<td>Business stakeholders with sufficient mandate are actively involved in architecture activities</td>
<td>Business stakeholders are only involved at pre-defined approval moments. Business stakeholders are only involved in requirements gathering.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delivery and operational stakeholders are actively involved in architecture activities</td>
<td>The architecture is only based on business concerns, delivery and operational concerns are left to others or too late.</td>
</tr>
<tr>
<td></td>
<td>Architects communicate with business stakeholders in business terms (business cases, risk analyses)</td>
<td>Architects use modeling language (UML, Archimate) or design pattern language (layers, APIs) to talk to business stakeholders</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Architectural requirements and concerns are discussed with all stakeholders in terms of scenarios (epics, stories)</td>
<td>Architectural concerns are discussed in vague terms like &quot;security&quot;, &quot;performance&quot;, &quot;velocity&quot;.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Architects have access to all stakeholders when necessary</td>
<td>Architects are not allowed to talk to business stakeholders</td>
<td></td>
</tr>
</tbody>
</table>
**Understanding Context**

**Context knowledge managed**

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CX_DOC</strong></td>
<td>Knowledge about architectural context is gathered, validated and preserved</td>
<td>Architects actively look for (business) goals and drivers behind the problem they have been asked to address.</td>
<td>Architecture is based on a fixed set of documents (requirements, specs) without full awareness of the underlying business drivers.</td>
</tr>
<tr>
<td></td>
<td>Specific architectural drivers and the (business) goals behind them are documented and validated.</td>
<td>No description of the architectural context is made, or it is made but not validated with stakeholders.</td>
<td></td>
</tr>
</tbody>
</table>
Making Architectural Decisions
Your primary deliverable

Say goodbye to The Architecture Document (as your primary deliverable)
- Takes weeks or months to produce
- Forces approval of all decisions in one go

Say hello to The Architectural Decision (as your primary deliverable)
- Finer granularity of artifact
- Easier to speed up feedback cycle
- Allows individual decision timing
## Making Architectural Decisions

### Decisions as primary deliverable

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD_PRIM</td>
<td>Architectural decisions are a primary architectural deliverable</td>
<td>Architectural decisions are communicated individually, whenever they are taken (e.g. collectively (implicitly or explicitly) as part of in an architectural decision register)</td>
<td>Architectural decisions are communicated collectively (implicitly or explicitly) as part of a large architecture document.</td>
</tr>
</tbody>
</table>

Stakeholder feedback is gathered on individual architectural decisions

Stakeholders only review the collective architecture document.
Making Architectural Decisions

The architecture microcycle

1. Architectural decisions
2. Identify & prioritize architectural concerns
3. Decide best fitting strategy
4. Research possible strategies
5. Architectural concerns (backlog)

This backlog should be prioritized by business impact (mostly risk and cost)
## Making Architectural Decisions
Prioritized by economic impact

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD_PRIO</td>
<td>Architectural work is prioritized by economic impact.</td>
<td>In the architectural workload, those concerns and decisions that have the highest risk and cost impact on their collective stakeholders are considered first, and receive the most attention.</td>
<td>Architecture work is based on / prioritized by what is mentioned in a documentation template, project plan or job description. Architectural decisions are only made when stakeholders ask for them.</td>
</tr>
<tr>
<td></td>
<td>Architectural concerns and their priorities are clear to stakeholders.</td>
<td></td>
<td>Architectural concerns and their priorities are clear to stakeholders.</td>
</tr>
<tr>
<td></td>
<td>Stakeholders have no clue about the relevance of architectural concerns.</td>
<td></td>
<td>Stakeholders have no clue about the relevance of architectural concerns.</td>
</tr>
</tbody>
</table>
Making Architectural Decisions
Justified and documented

• Include rationale, decision criteria, consequences, drawbacks and alternatives not chosen
• Easily accessible to stakeholders
• Let everyone know you welcome feedback
## Making Architectural Decisions

### Justified and documented

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD_DOC</td>
<td>Architectural decisions are justified and documented</td>
<td>Architectural decisions, including their status and justification, are visible to stakeholders.</td>
<td>Stakeholders only see the effect of decisions after they have been taken in a set of views or models.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The criteria on which each decision is based, and their relation to specific architectural drivers and business goals, are visible to stakeholders.</td>
<td>Justification for decisions is not documented, or is mainly based on generic principles and not related to the specific context.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alternatives for a decision and the reason for their rejection are visible to stakeholders.</td>
<td>Only the chosen alternative is documented.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Architects take delivery and operational consequences of decisions into account and document them.</td>
<td>The target architecture is chosen, how to deliver it is left to others. Only the decision is documented, not its impact.</td>
</tr>
</tbody>
</table>
Making Architectural Decisions

Timing of architectural decisions

Certainty of correct architectural decision depends on knowledge:

- relative cost of the alternative solutions
- value and impact on the business
- delivery times

Timing architectural decision is balancing risk, cost and delivery time:

- too little information → risk of not meeting key requirements
- waiting too long → project delays, wasted resources

There's an art of knowing when.
Never try to guess.
Toast until it smokes and then twenty seconds less.

- Pat Hein
Making Architectural Decisions
Well-timed decisions

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD_TIME</td>
<td>Architectural decisions are taken at the optimal time</td>
<td>The timing of each architectural decision is a conscious process.</td>
<td>Timing of architectural decisions is dictated by project milestones or (agile) principles</td>
</tr>
<tr>
<td></td>
<td>A decision is postponed until more knowledge is available if the risk of a</td>
<td></td>
<td>A decision is made now because the deadline for the architecture document is getting close.</td>
</tr>
<tr>
<td></td>
<td>wrong decision outweighs the cost of delay.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2019 CGI Inc.
Making Architectural Decisions
Decentral unless…

• “Emergence” does not imply “without thinking”, “for free” or “magically”.
• It does stimulate re-thinking ownership of architectural decisions.

Classic architecture: broad standardization

<table>
<thead>
<tr>
<th>Hard rules</th>
<th>Left open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibility: Architecture</td>
<td>Responsibility: Developers</td>
</tr>
</tbody>
</table>

Evolutionary architecture: eventual integrity

<table>
<thead>
<tr>
<th>Hard rules</th>
<th>Suggestions</th>
<th>Left open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibility: Developers</td>
<td>Responsibility: Architecture</td>
<td></td>
</tr>
</tbody>
</table>
Making Architectural Decisions
Decentral unless…

Crowd-sourced

No ‘named’ architect

Architecture marshalls

Architecture owner

Classic architect

Centralized

Project size
Different locations
Domain expertise

Architecture base
External dependencies
Familiarity, experience

Discipline
Organizational context

Source: Stefan Toth
## Making Architectural Decisions

**Decentral unless…**

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AD_DLGT</strong></td>
<td>Architectural decisions are delegated to the optimal level of decentralization</td>
<td>An architectural decision is taken at the central level (only) if decentral interests can be conflicting or cause complexity that outweighs local benefits</td>
<td>All topics mentioned in the reference architecture template are made at the central level.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An architectural decision is taken at the decentral level (only) if the benefits of local optimization outweigh the risk and costs of diversity and non-standardization</td>
<td>All decisions are made at the decentral level because we are a self-organizing team applying agile principles.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Responsibility for architectural decisions is shared between architects owning the wider context and architects owning specific solution (elements).</td>
<td>Architects demand a clear delineation of their own responsibility.</td>
</tr>
</tbody>
</table>
Architecture Modeling
Visual model of context

Context Diagram: Solution in its operational environment

• “What’s in scope and what is not?” → Solution Boundary
• “What external systems/actors?” → Interface Overview

## Architecture Modeling
### Visual model of context

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_CX</td>
<td>There is a visual model of the solution context</td>
<td>Information system context is documented and validated (context diagram), showing solution boundary and external dependencies.</td>
<td>No context diagrams are made.</td>
</tr>
</tbody>
</table>
Architecture Modeling

Visual models of solution

All architecture documentation methods use views

• ISO 42010, TOGAF, Archimate, 4 + 1, ‘Views and Beyond’, C4

• Viewpoints address concerns per stakeholder (group)
• Concerns evolve over time, use only relevant views at each time

https://c4model.com
Simon Brown

https://en.wikipedia.org/wiki/4+1_architectural_view_model
Philippe Kruchten
### Architecture Modeling

#### Visual models of solution

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_SL</td>
<td>Appropriate visual models of the solution are created</td>
<td>Visual models are created that show how the architecture addresses the relevant stakeholder concerns at each point in time.</td>
<td>Architecture documentation contains all knowledge gathered up till now. Architecture documentation includes viewpoints addressing concerns that may become relevant later in time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Visual models are used to validate that key stakeholder needs are fulfilled by the design.</td>
<td>Diagrams are only used to clarify written documentation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Visual models are used as the basis for creating and delivering the solution.</td>
<td>Models are used to get approval and then abandoned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Visual models are made at the appropriate level of abstraction to allow reasoning about the architecture.</td>
<td>Diagrams are created at prescribed levels of abstraction only.</td>
</tr>
</tbody>
</table>
**Architecture Validation**

**Fulfills stakeholder needs?**

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>VL_ANLYS</td>
<td>The architect/team checks whether the architecture (still) fulfills the stakeholder needs.</td>
<td>Architecture models and checklists are used to validate the architecture against stakeholder needs.</td>
<td>We'll first build the architecture and then test it.</td>
</tr>
</tbody>
</table>

Architectural requirements, such as non-functional requirements, are made part of the "definition of done" and are tested. Only functionality is tested. There are performance/security tests because they are mandatory.
Architecture Fulfillment
Architectural runway recognized

Visible

Positive Value
New features Added functionality

Invisible

Architecture Runway

Negative Value
Defects
Technical Debt

Source: Philippe Kruchten

What’s in your backlog?
(or Work Breakdown Structure / Project Portfolio / Change Requests)
Architecture Fulfillment

Architectural runway recognized
## Architecture Fulfillment

### Architectural runway recognized

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_AR</td>
<td>Work on architectural runway is recognized and planned.</td>
<td>The product backlog or product breakdown structure contains stories or activities to realize architectural elements and technical debt reductions (&quot;enablers&quot;) whose business value is indirect (i.e. they derive their business value from other improvements that depend on them).</td>
<td>The product backlog only contains stories that add direct business value. The work breakdown structure only contains work items to realize functionality.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ownership for architectural decision making is clearly allocated (to an individual or team).</td>
<td>Nobody takes ownership of the architecture.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Improvements considered for the next release/sprint are fed by a variety of sources, among which is the architect(ure team).</td>
<td>Only new functionality or features are considered for the next release/sprint.</td>
</tr>
</tbody>
</table>
Make Architectural Debt visible as **business risk**

- Put on risk register
- Find business owner(s) who feel the pain of the risk (and can do something about it)
- Make the business case

<table>
<thead>
<tr>
<th>Item</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Benefits</strong></td>
<td></td>
</tr>
<tr>
<td>Reduced recurrent maintenance cost</td>
<td>M/yr</td>
</tr>
<tr>
<td>Reduced risk exposure</td>
<td>R/yr</td>
</tr>
<tr>
<td>Total benefits per year</td>
<td>M+R</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td></td>
</tr>
<tr>
<td>Principal: effort of migration/refactoring/…</td>
<td>P</td>
</tr>
<tr>
<td>Opportunity cost (delayed features)</td>
<td>F</td>
</tr>
<tr>
<td>Total cost</td>
<td>P+F</td>
</tr>
<tr>
<td><strong>TOTAL RETURN ON INVESTMENT (1 YEAR)</strong></td>
<td>(M+R) – (P+F)</td>
</tr>
</tbody>
</table>
## Architecture Fulfillment
### Architecture Debt Control

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_TD</td>
<td>Architectural debt is identified and controlled.</td>
<td>Technology upgrades and work to fix architectural shortcuts are visible on the product backlog or project schedule.</td>
<td>The product backlog only contains stories that add direct business value. The work breakdown structure only contains work items to realize functionality.</td>
</tr>
</tbody>
</table>

Decisions to fix architectural debt are based on economic trade-offs taking into account interest and cost of delay. Architectural debt is only fixed when there is time left after implementing all the necessary functionality.
Architecture Fulfillment
Just enough anticipation

Just Enough Anticipation achieved by:

- Dependency Analysis
- Technical Debt Control
- Economic Reasoning

Source: Nanette Brown, Rod Nord, Ipek Ozkaya
Architecture Fulfillment
Strategy 1: Value-first roadmapping

- In line with Agile philosophy
- May increase TCO (more refactoring)
- Too “greedy” algorithm may run project into wall (complete rebuild)
- Good in volatile environments
Architecture Fulfillment
Strategy 2: Architecture-first roadmapping

- In line with plan-driven philosophy
- Late delivery of value → risk of cancellation
- Risk of building wrong architecture (if context changes)
- Good for complex solutions
# Architecture Fulfillment
## Just enough anticipation

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Description</th>
<th>Evidence</th>
<th>Counterevidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_RM</td>
<td>Architecturally significant features, events and stories are anticipated.</td>
<td>Architects/teams work with stakeholders to identify future events that impact the risk, cost and value of solutions/systems/landscapes</td>
<td>Urgent events that have significant economic impact take architects/teams or stakeholders by surprise.</td>
</tr>
</tbody>
</table>

- Dependency/critical path analysis is used to start work on the architecture runway in time to mitigate the risk of disrupting events.
- Architecture runway improvements are identified late.
- Teams have to break promises or do significant extra work to prevent disasters.
- The timing of realization of enablers is a conscious decision, based on economic trade-offs.
- Enablers are only implemented when they become really urgent.
Our commitment to you
We approach every engagement with one objective in mind—to help clients succeed.
References


Contact us to continue the conversation.

Eltjo Poort  
Distinguished Solution Architect  
etjpoort@cgi.com  
linkedin.com/in/eltjopoort  
@eltjopoort

Joost van der Burg  
Vice-President Aviation  
joost.van.der.burg@cgi.com  
linkedin.com/in/joostvanderburg