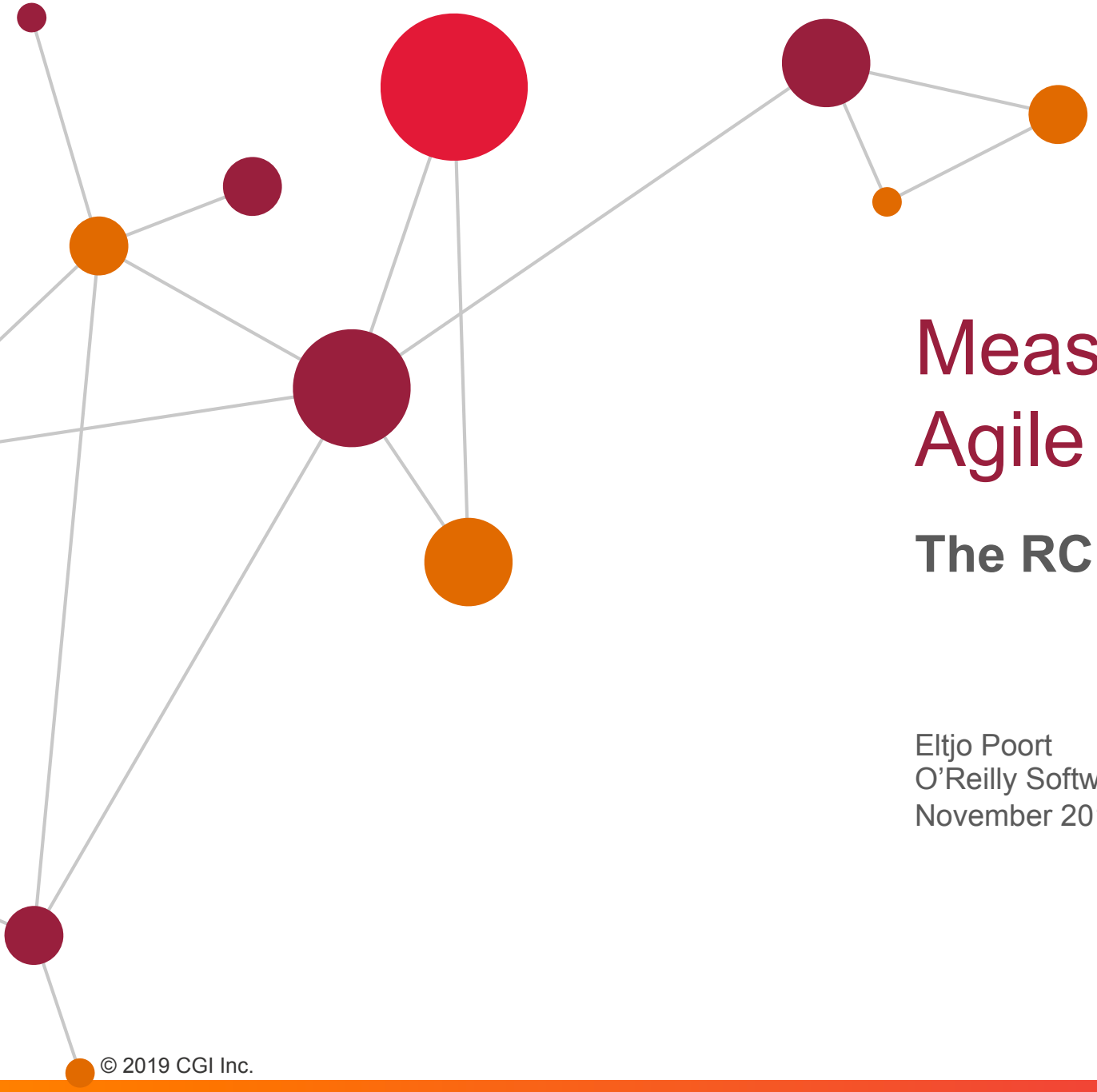


CGI

Experience the commitment®

Measure Your Agile Architecture Maturity

The RCDA Maturity Model

Eltjo Poort

O'Reilly Software Architecture Conference, Berlin

November 2019

Eltjo Poort

CGI

http://eltjopoort.nl

RCDA

Founder / owner

CGI's agile architecture approach

KLM TOMTOM ProRail NXP

PhD

VU UNIVERSITY AMSTERDAM

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

Software Engineering Institute
Carnegie Mellon University

Not another maturity model…

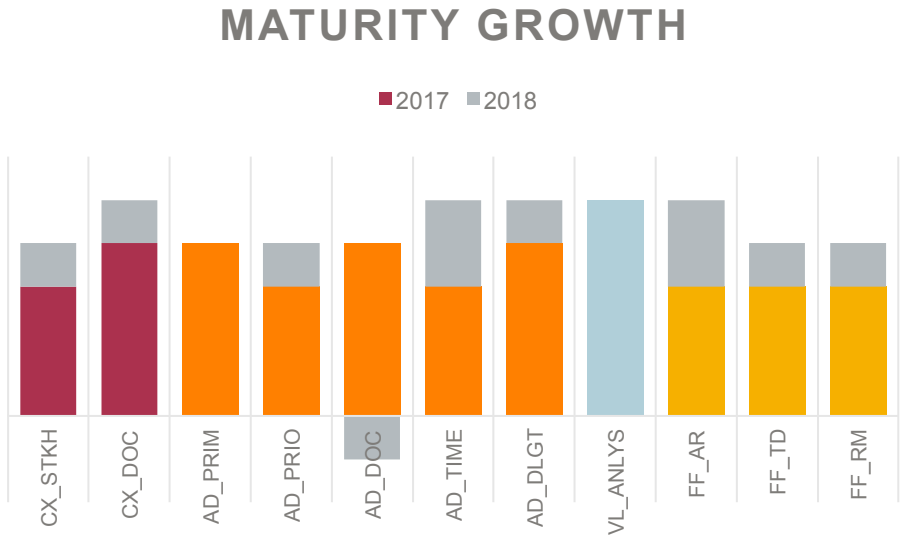
What is a maturity model?

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

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

Why measure agile architecture maturity?

- To identify areas of improvement
- To track improvement progress



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

[2] Jacobson, I., Ng, P. W., & Spence, I. (2007, July-August). Enough process - let's do practices. *Journal of Object Technology, 6*(6), 41-66

Foundation

Risk and Cost Driven Architecture

- Agile solution shaping, quick feedback loop
- Focus on economic reality
- Scalable architecting
- Better risk and cost control in delivery

RCDA publications:

- Journal of Systems and Software
- IEEE Software Magazine (Sept. 2014, Nov. 2016)
- Leading architecture conferences: SATURN, LAC, WICSA (best paper award)

1400 Architects trained

18 Proven practices

Open Group recognized in the Certified Architect program

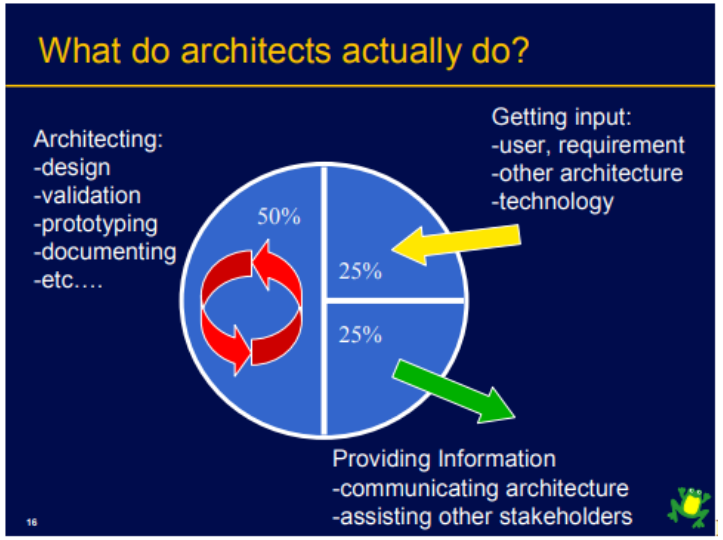
300 Pages of guidance

14 Peer-reviewed publications

Award by Software Engineering Institute

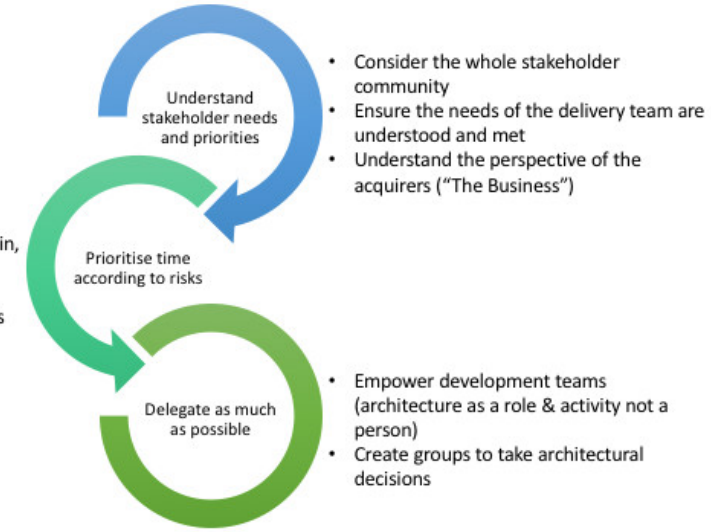
Inspiration

Philippe Kruchten
Professor (UBC),



Eoin Woods
CTO at Endava

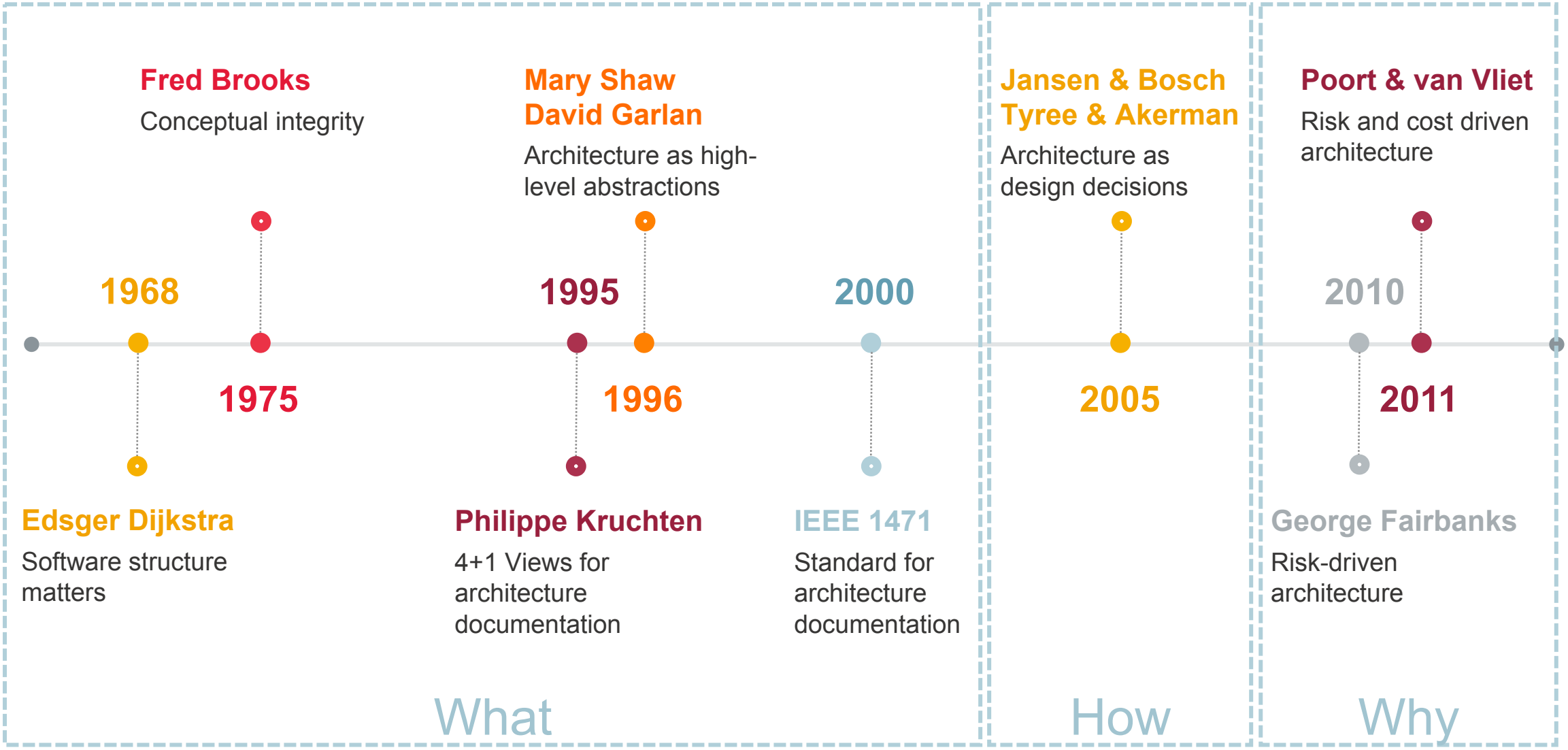
- Consider external dependencies
- Look for novel aspects of the domain, problem or solution
- Identify the high impact decisions
- Analyse your local situation for risks



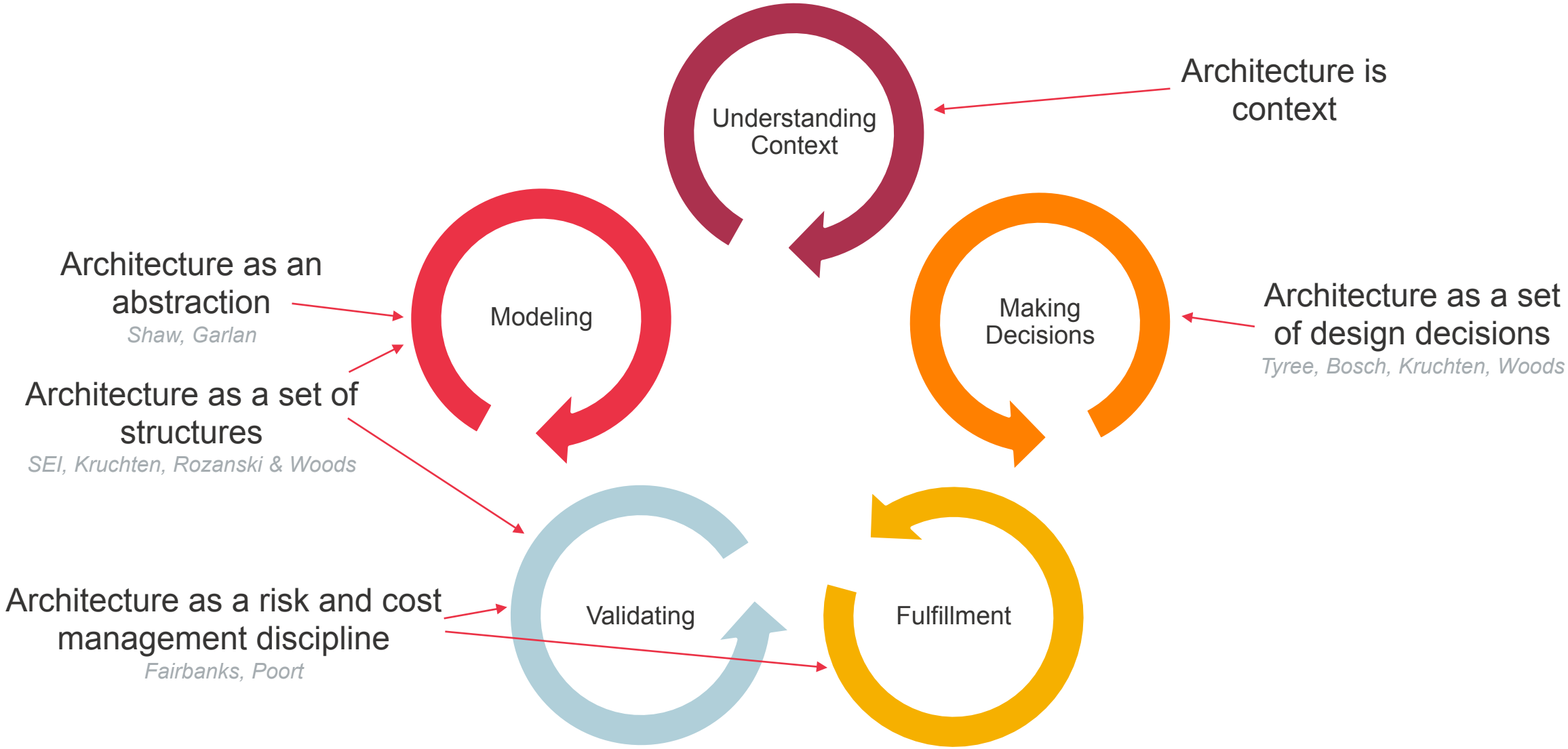
- Consider the whole stakeholder community
- Ensure the needs of the delivery team are understood and met
- Understand the perspective of the acquirers ("The Business")

- Empower development teams (architecture as a role & activity not a person)
- Create groups to take architectural decisions

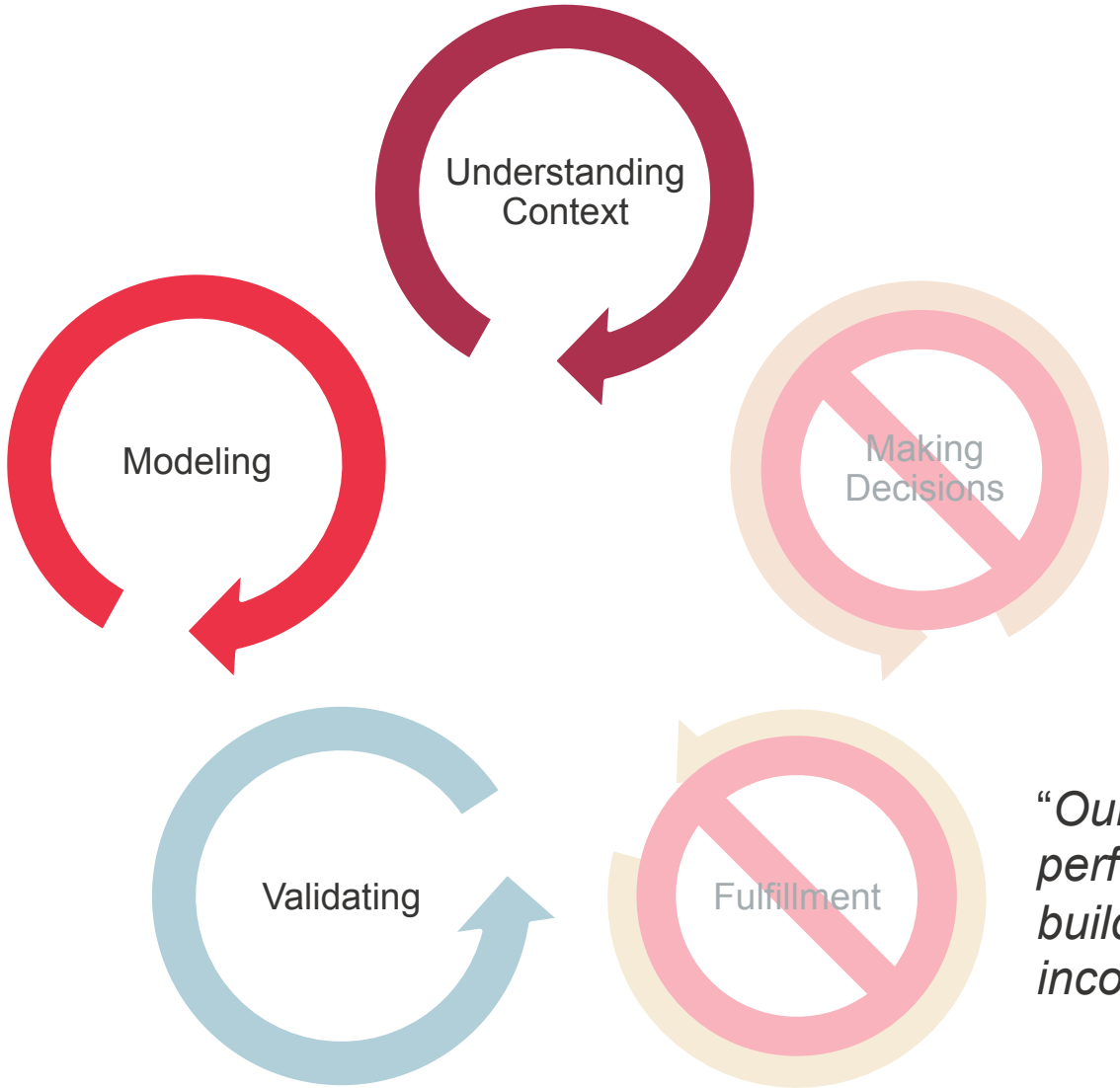
History of digital (software) architecture



The Five Responsibilities of the Architecture Function



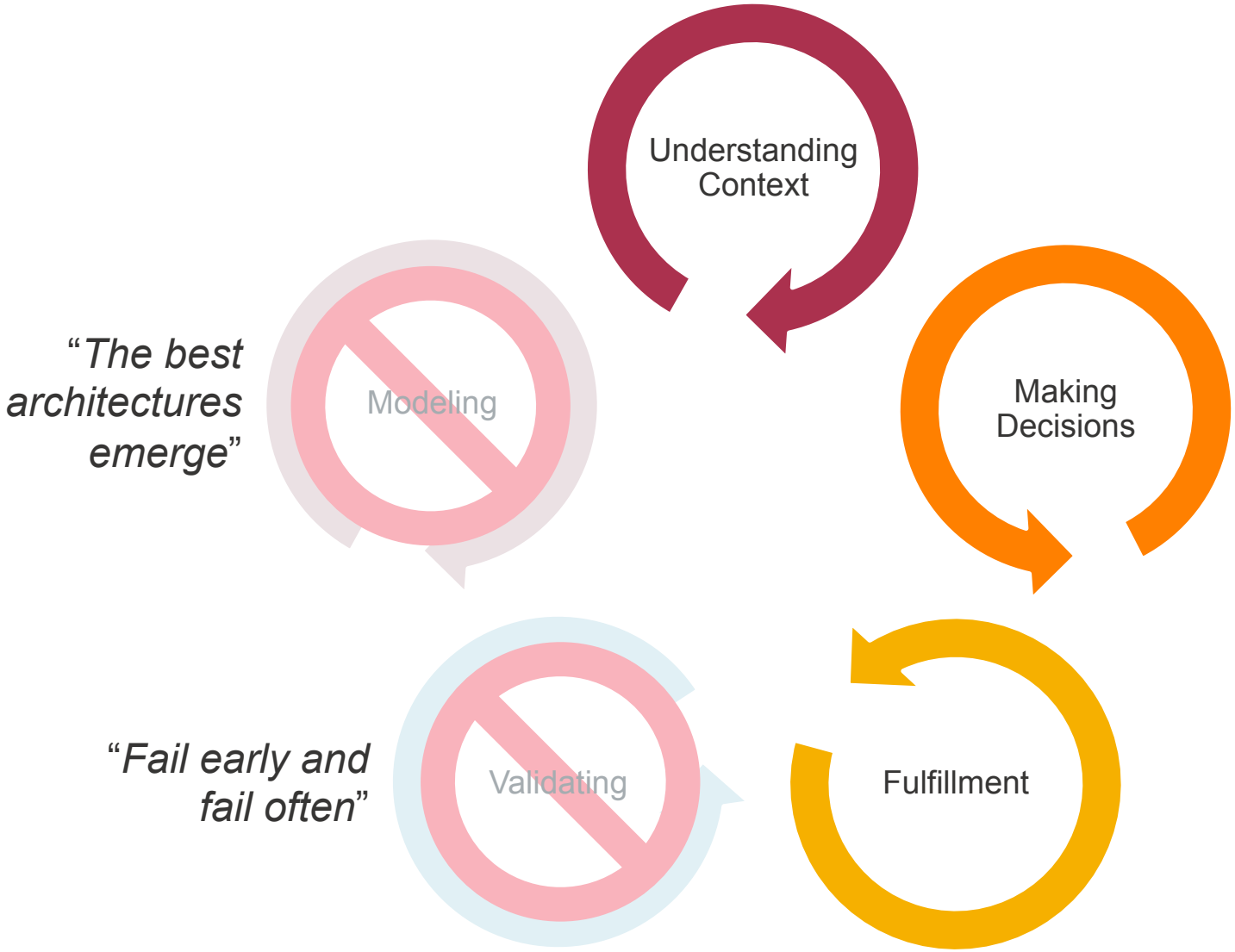
The Waterfall Wasteland



"We don't take decisions, we only advise management"

"Our design was perfect, but the builders were incompetent"

The Agile Outback



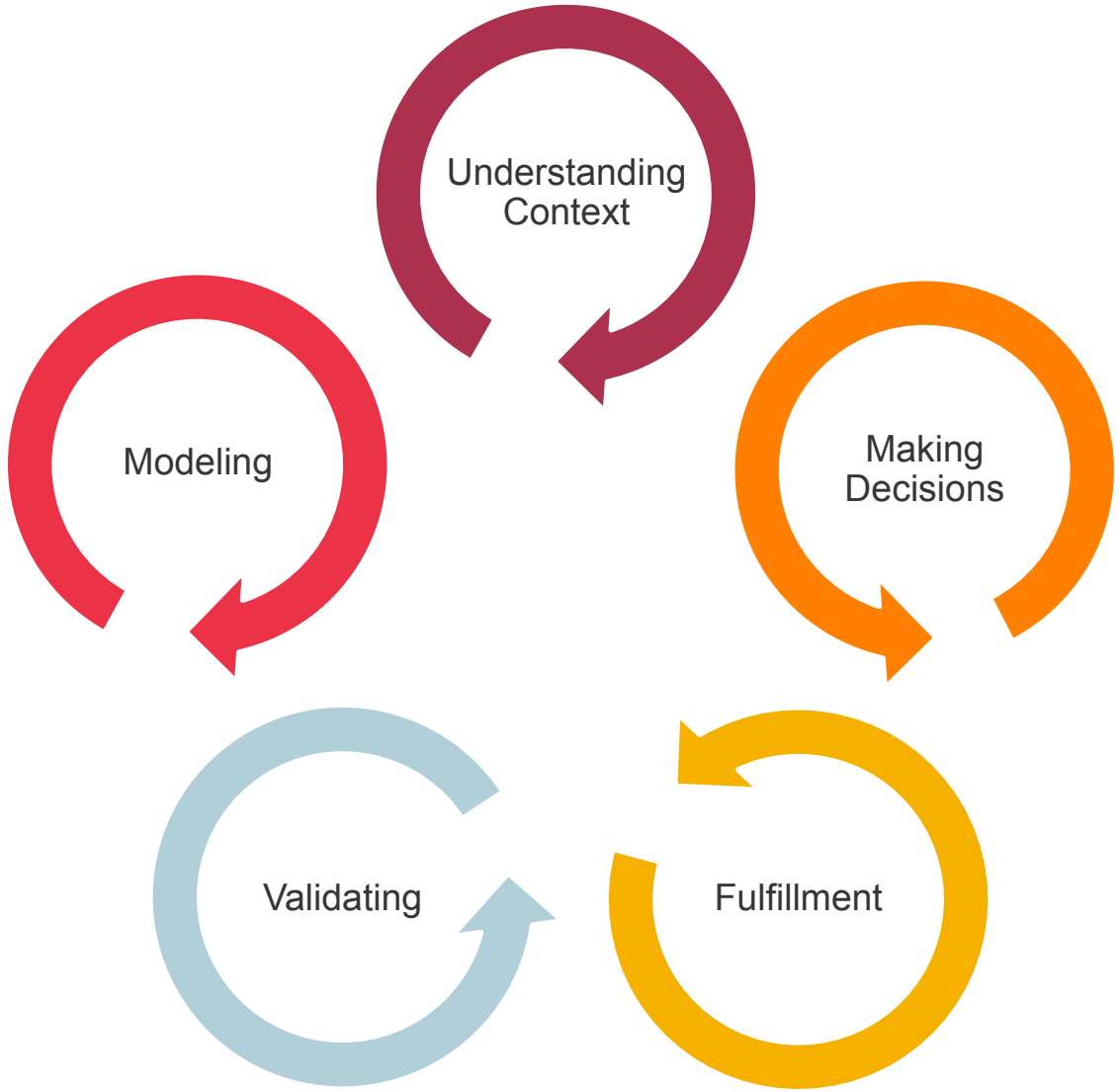
Agile Architecture Maturity

Steering clear of the Waterfall Wasteland and Agile Outback

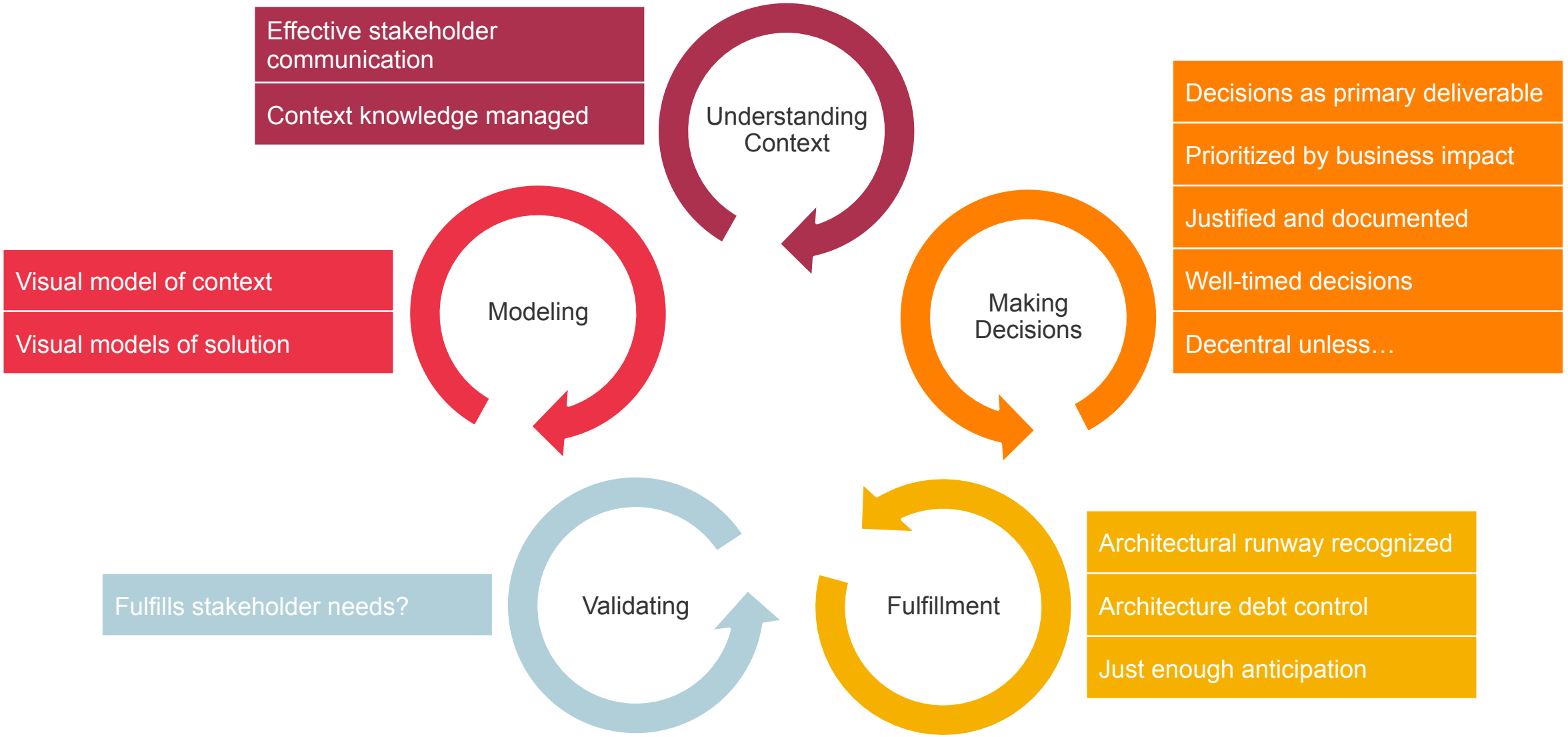
An organization's architecture function is mature if:

- 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



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

Evidence	Counterevidence
Business stakeholders with sufficient mandate are actively involved in architecture activities	Business stakeholders are only involved at pre-defined approval moments. Business stakeholders are only involved in requirements gathering.
Delivery and operational stakeholders are actively involved in architecture activities	The architecture is only based on business concerns, delivery and operational concerns are left to others or too late.
Architects communicate with business stakeholders in business terms (business cases, risk analyses)	Architects use modeling language (UML, Archimate) or design pattern language (layers, APIs) to talk to business stakeholders
Architecture documentation only describes how the architecture addresses the relevant stakeholder concerns at each point in time.	Architecture documentation contains all knowledge gathered up till now. Architecture documentation includes viewpoints addressing concerns that may become relevant later in time.
Architectural requirements and concerns are discussed with all stakeholders in terms of scenarios (epics, stories)	Architectural concerns are discussed in vague terms like "security", "performance", "velocity".
Architects have access to all stakeholders when necessary	Architects are not allowed to talk to business stakeholders
Architects actively look for (business) goals and drivers behind the problem they have been asked to address.	Architecture is based on a fixed set of documents (requirements, specs) without full awareness of the underlying business drivers.
Specific architectural drivers and the (business) goals behind them are documented and validated.	No description of the architectural context is made, or it is made but not validated with stakeholders.
Information system context is documented and	No context diagrams are made.

Agile Architecture Maturity Radar

Scope/team:	

Effective stakeholder communication	
Context knowledge managed	

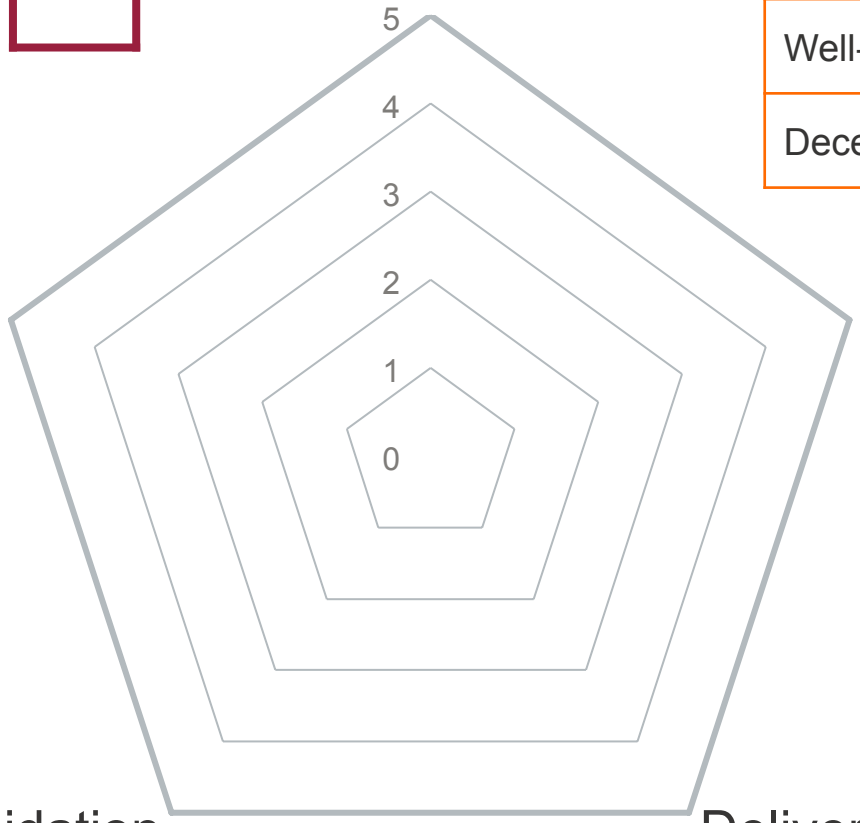
Understanding Context

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

Making Decisions

Modeling

Visual model of context	
Visual models of solution	



Architectural runway recognized	
Architecture debt control	
Just enough anticipation	

Fulfills stakeholder needs?	

Validation

Delivery

1 Never **2** Seldom **3** Ad hoc/Individuals **4** Mostly **5** Habit **1-100%** Quality multiplier

Agile Architecture Maturity Radar

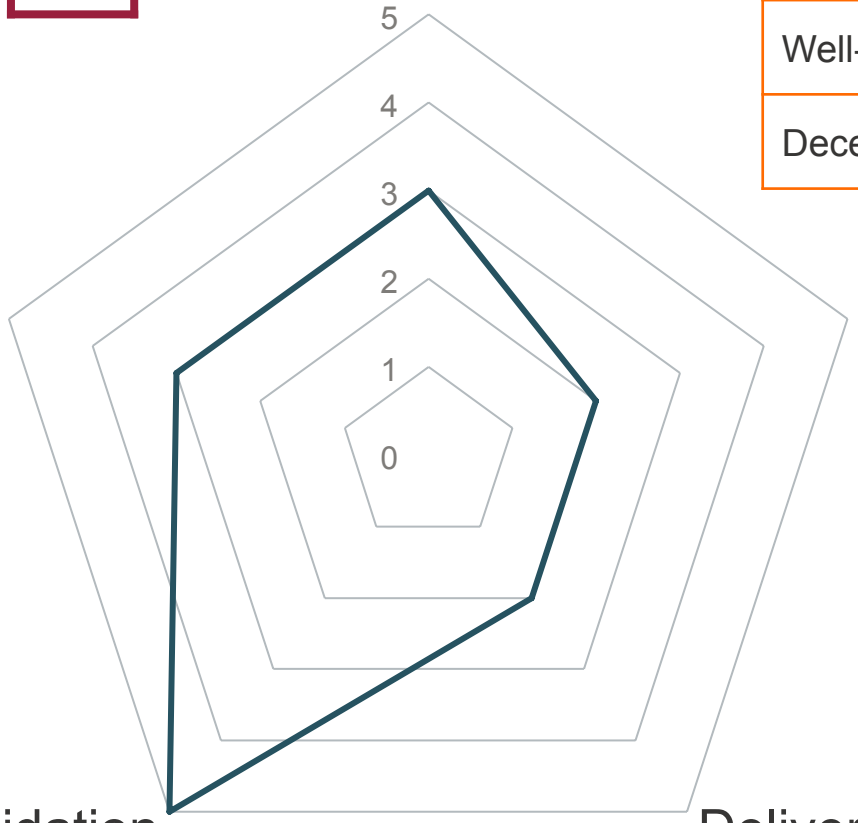
Scope/team:	WOSP Product Team

Effective stakeholder communication	4
Context knowledge managed	2

3 Understanding Context

Decisions as primary deliverable	4
Prioritized by business impact	3
Justified and documented	2
Well-timed decisions	2
Decentral unless…	4

Making Decisions 2



3 Modeling

Visual model of context	3
Visual models of solution	4

Architectural runway recognized	2
Architecture debt control	3
Just enough anticipation	2

Delivery 2

Fulfills stakeholder needs?	5

5 Validation

1 Never **2** Seldom **3** Ad hoc/Individuals **4** Mostly **5** Habit **1-100%** Quality multiplier

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

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

Understanding Context

Context knowledge managed

Behavior	Description	Evidence	Counter-evidence
CX_DOC	Knowledge about architectural context is gathered, validated and preserved	Architects actively look for (business) goals and drivers behind the problem they have been asked to address.	Architecture is based on a fixed set of documents (requirements, specs) without full awareness of the underlying business drivers.
		Specific architectural drivers and the (business) goals behind them are documented and validated.	No description of the architectural context is made, or it is made but not validated with stakeholders.

Making Architectural Decisions
Your primary deliverable

CGI

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)

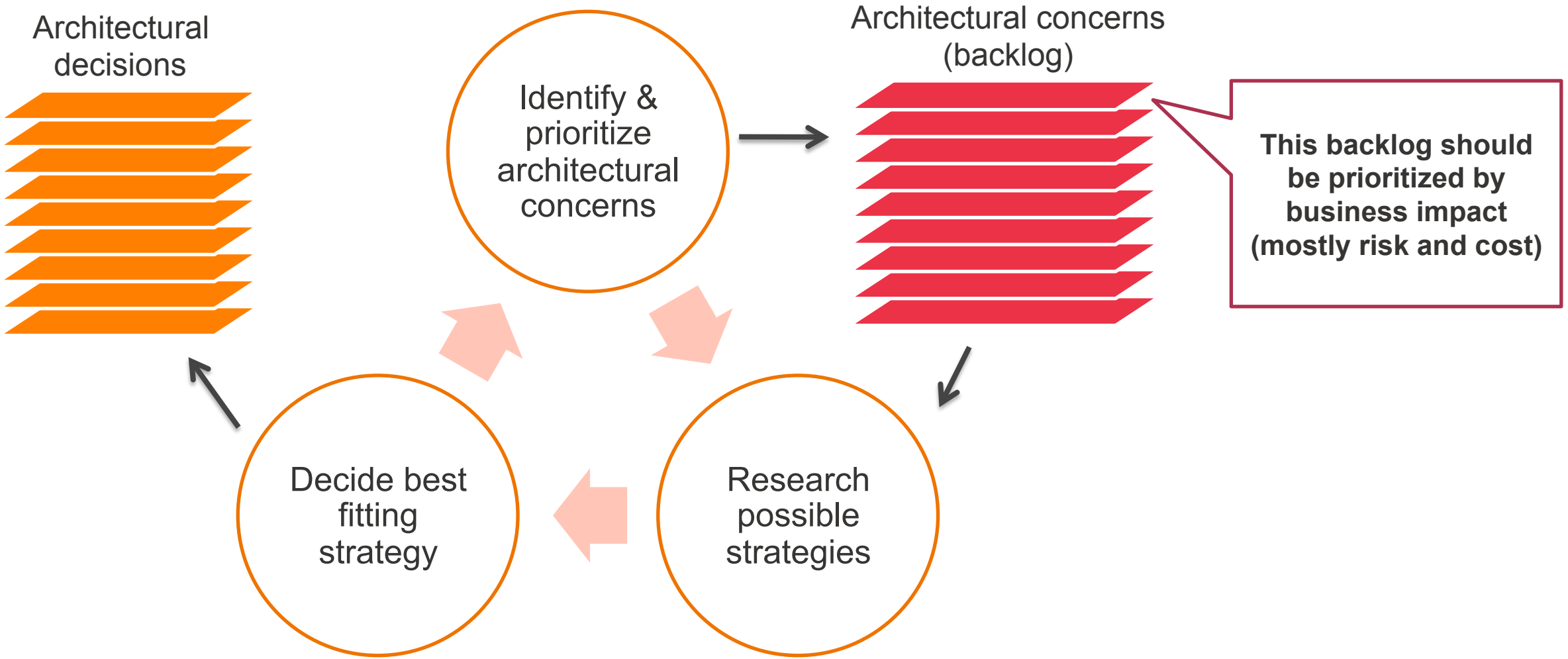
- Finer granularity of artifact
- Easier to speed up feedback cycle
- Allows individual decision timing

Making Architectural Decisions

Decisions as primary deliverable

Behavior	Description	Practice	Counter-Practice
AD_PRIM	Architectural decisions are a primary architectural deliverable	Architectural decisions are communicated individually, whenever they are taken (e.g. in an architectural decision register)	Architectural decisions are communicated collectively (implicitly or explicitly) as part of a large architecture document.
		Stakeholder feedback is gathered on individual architectural decisions	Stakeholders only review the collective architecture document.

Making Architectural Decisions
The architecture microcycle

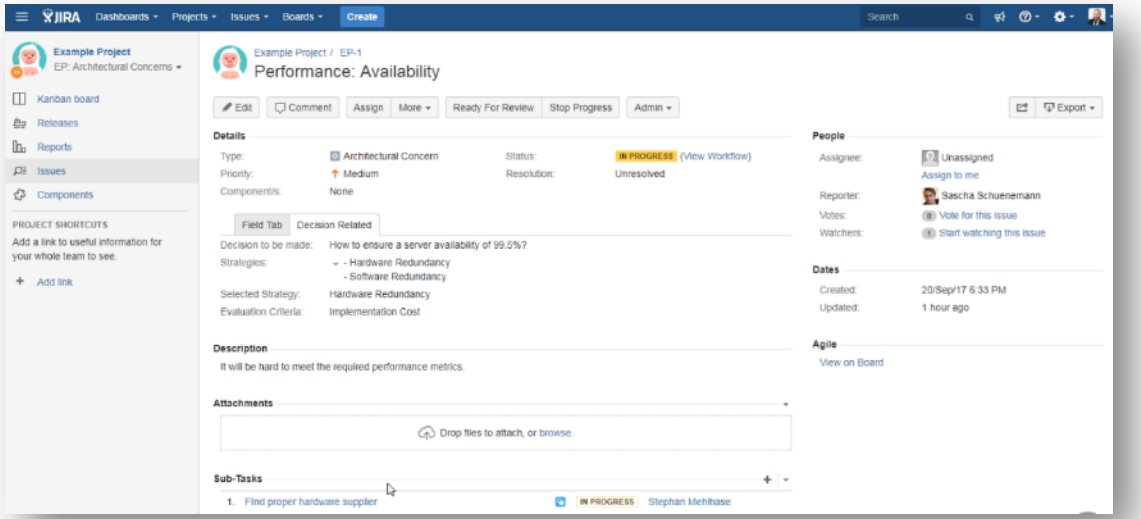


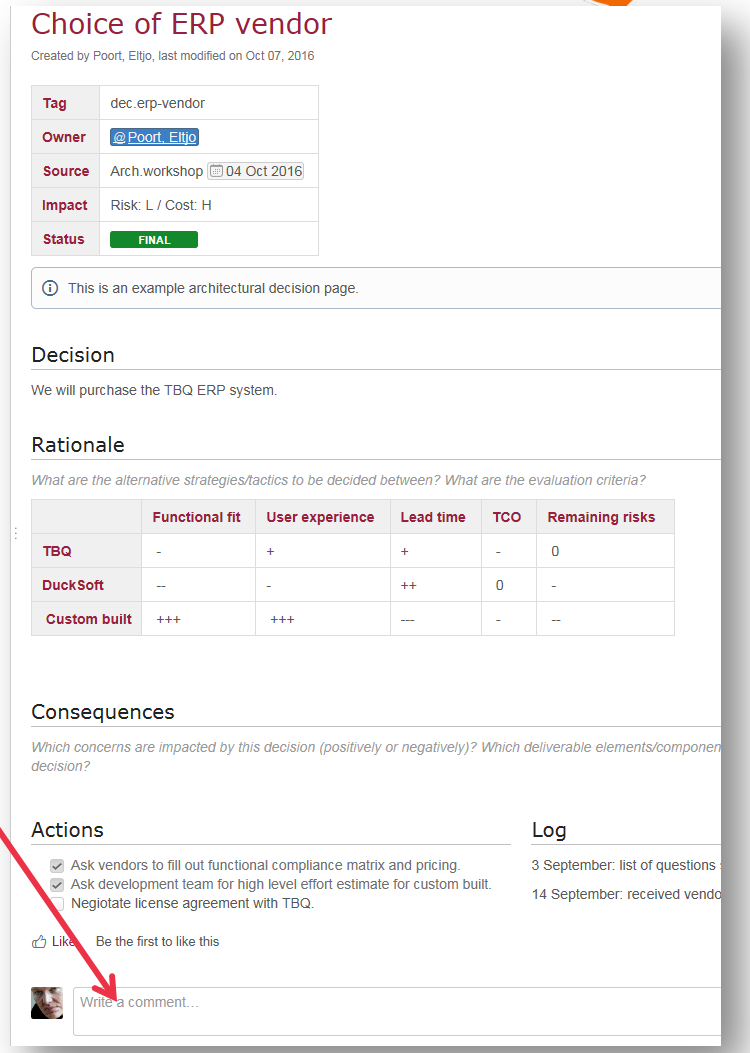
Making Architectural Decisions
Prioritized by economic impact

Behavior	Description	Practice	Counter-practice
AD_PRIO	Architectural work is prioritized by economic impact.	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.	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.
		Architectural concerns and their priorities are clear to stakeholders.	Stakeholders have no clue about the relevance of architectural concerns.

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

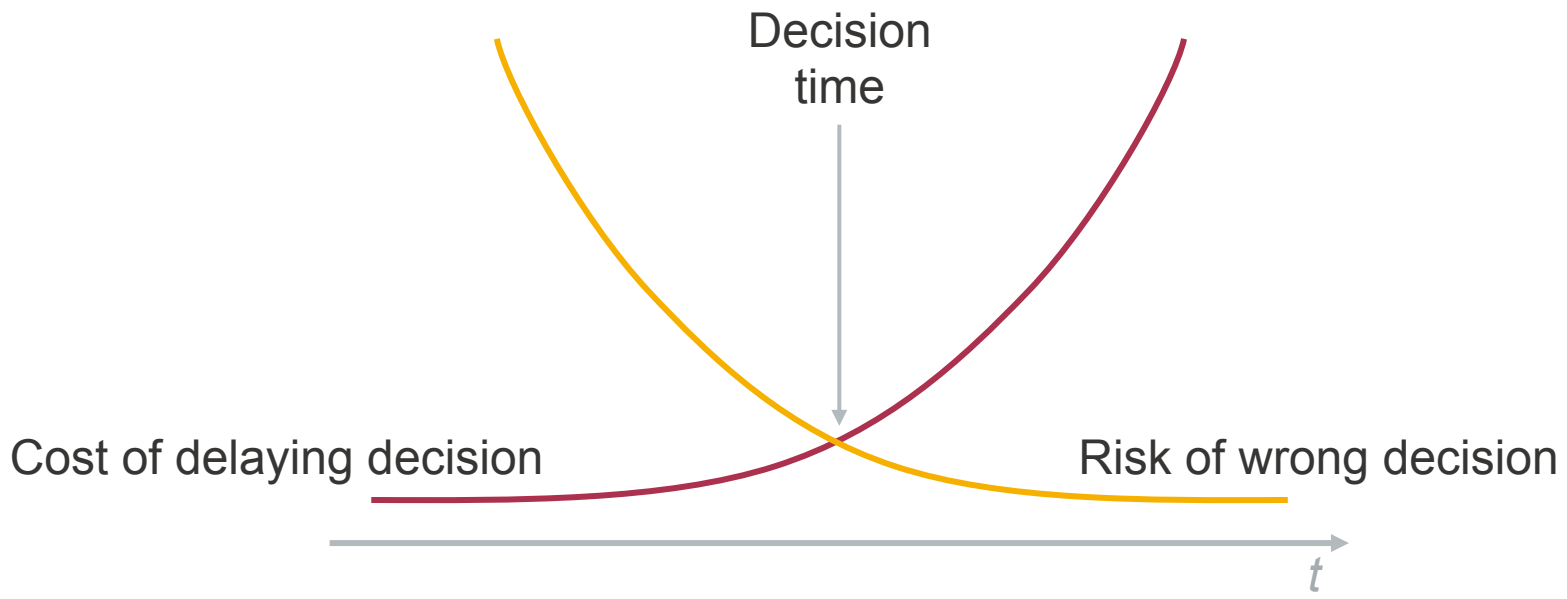




Making Architectural Decisions
Justified and documented

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

Architectural Decision Timing



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

- Pat Hein

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

Making Architectural Decisions
Well-timed decisions

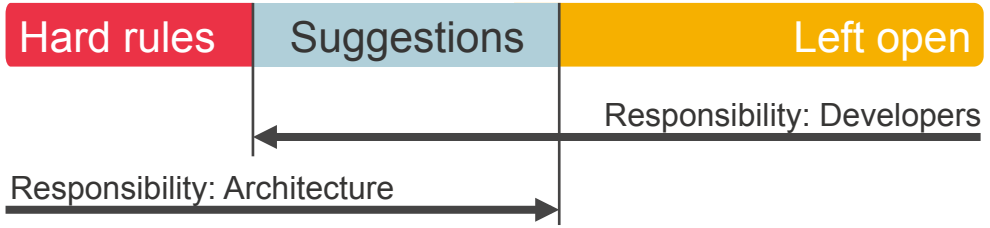
Behavior	Description	Evidence	Counter-evidence
AD_TIME	Architectural decisions are taken at the optimal time	The timing of each architectural decision is a conscious process.	Timing of architectural decisions is dictated by project milestones or (agile) principles
		A decision is postponed until more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

Making Architectural Decisions
Decentral unless…

- "Emerge" does not imply "without thinking", "for free" or "magically".
- It does stimulate re-thinking ownership of architectural decisions.

Classic architecture: broad standardization

Evolutionary architecture: eventual integrity

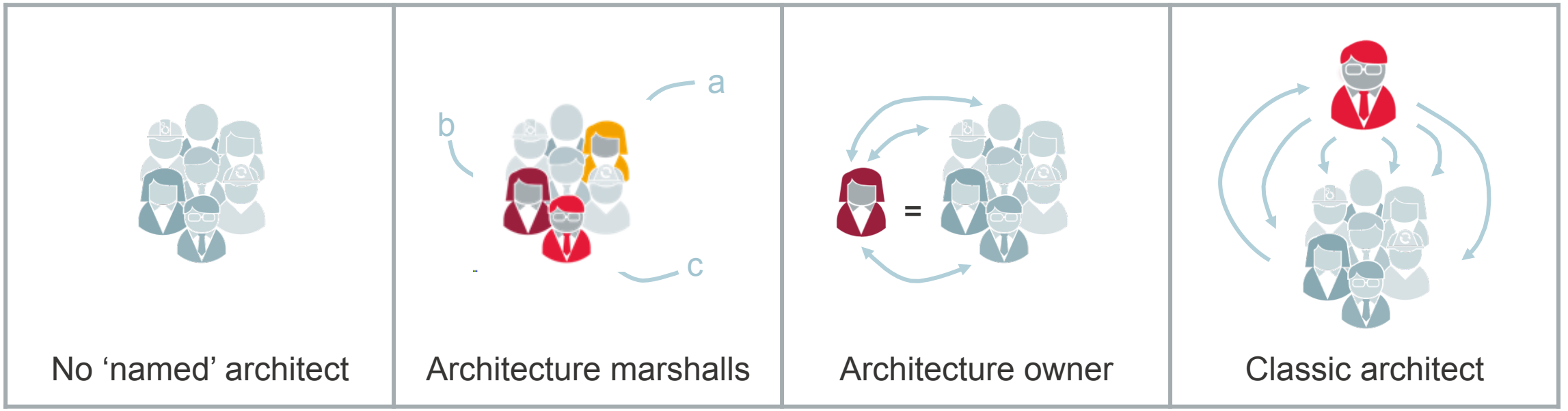


Source: Stephan Toth

Making Architectural Decisions

Decentral unless…

Crowd-sourced — Centralized



Project size
Different locations
Domain expertise

Architecture base
External dependencies
Familiarity, experience

Discipline
Organizational context

Source: Stephan Toth

Making Architectural Decisions

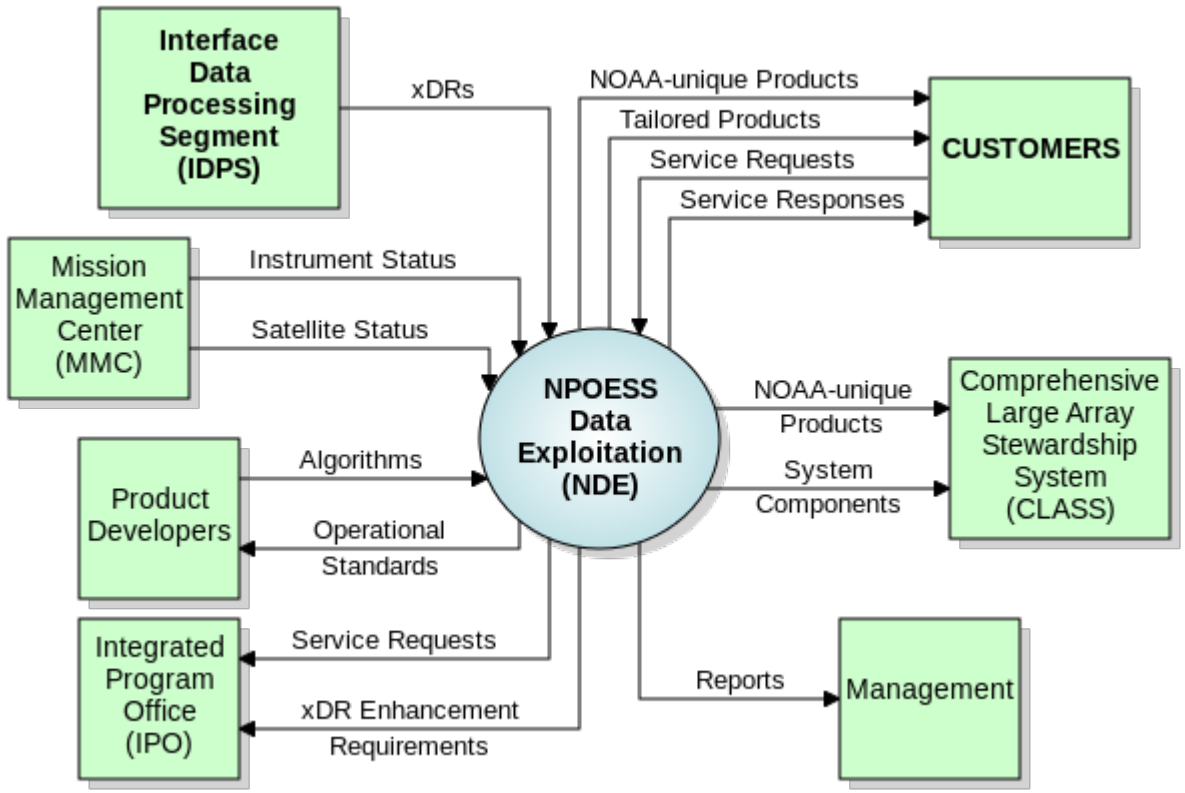
Decentral unless…

Behavior	Description	Evidence	Counter-evidence
AD_DLGT	Architectural decisions are delegated to the optimal level of decentralization	An architectural decision is taken at the central level (only) if decentral interests can be conflicting or cause complexity that outweighs local benefits	All topics mentioned in the reference architecture template are made at the central level.
		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	All decisions are made at the decentral level because we are a self-organizing team applying agile principles.
		Responsibility for architectural decisions is shared between architects owning the wider context and architects owning specific solution (elements).	Architects demand a clear delineation of their own responsibility.

Architecture Modeling
Visual model of context

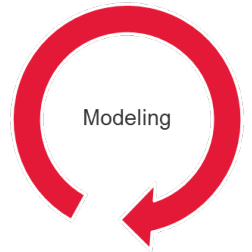
Context Diagram: Solution in its operational environment

- "What's in scope and what is not?" → Solution Boundary
- "What external systems/actors?" → Interface Overview



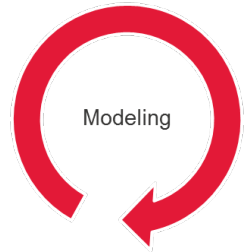
https://en.wikipedia.org/wiki/System_context_diagram

Architecture Modeling
Visual model of context



Behavior	Description	Evidence	Counter-evidence
MD_CX	There is a visual model of the solution context	Information system context is documented and validated (context diagram), showing solution boundary and external dependencies.	No context diagrams are made.

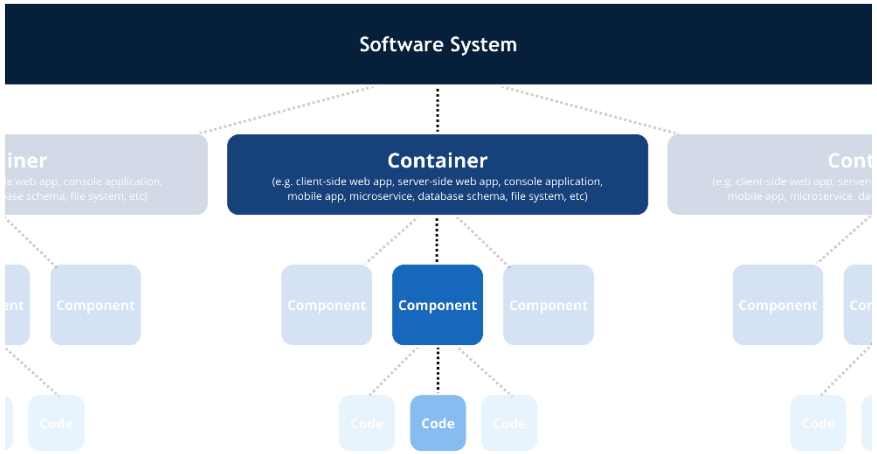
Architecture Modeling
Visual models of solution



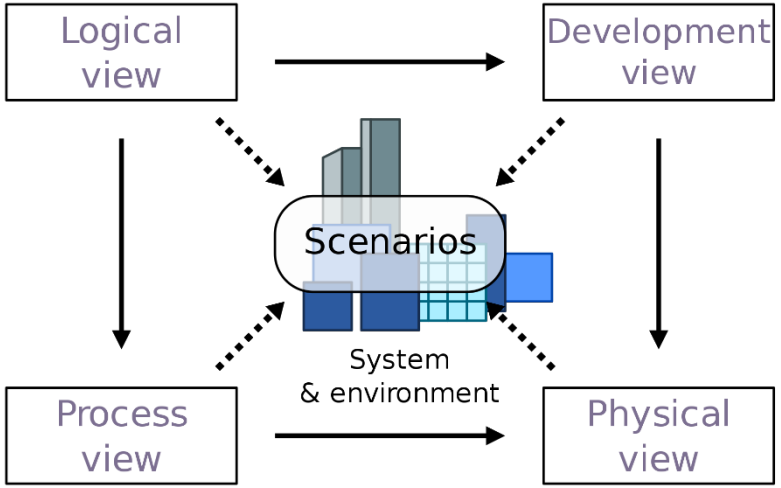
All architecture documentation methods use views

- 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

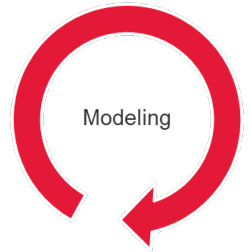


https://c4model.com
Simon Brown



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

Architecture Modeling
Visual models of solution



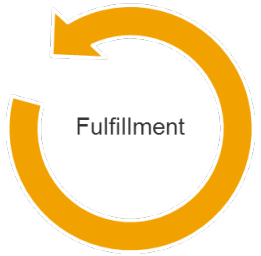
CGI

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

Architecture Validation
Fulfills stakeholder needs?

Behavior	Description	Evidence	Counter-evidence
VL_ANLYS	The architect/team checks whether the architecture (still) fulfills the stakeholder needs.	Architecture models and checklists are used to validate the architecture against stakeholder needs.	We'll first build the architecture and then test it.
		Architectural requirements, such as non-functional requirements, are made part of the "definition of done" and are tested.	Only functionality is tested. There are peformance/security tests because they are mandatory.

Architecture Fulfillment
Architectural runway recognized



	Visible	Invisible
Positive Value	**New features Added functionality**	**Architecture Runway**
Negative Value	**Defects**	**Technical Debt**

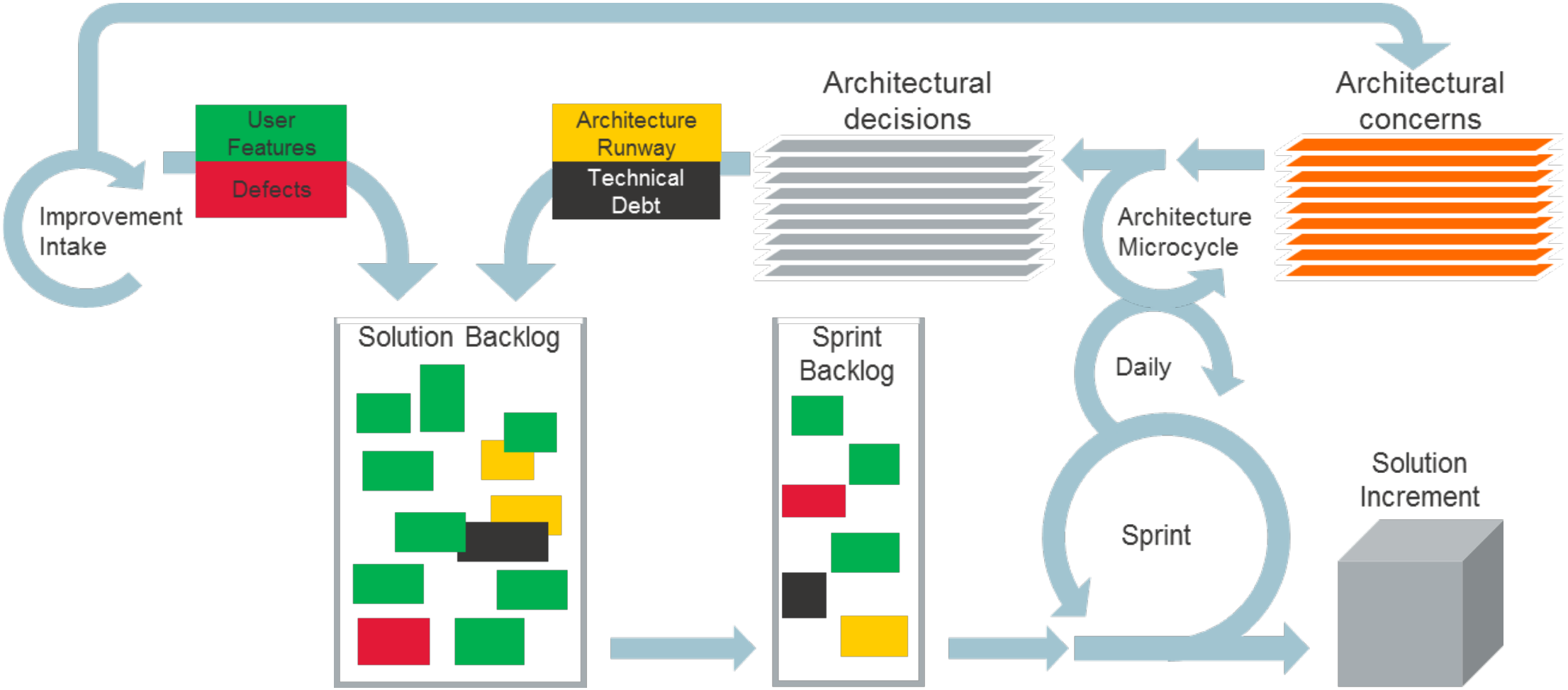
Source: Philippe Kruchten

What's in your backlog?

(or Work Breakdown Structure / Project Portfolio / Change Requests)

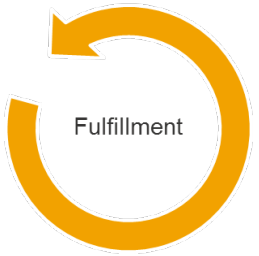
Architecture Fulfillment

Architectural runway recognized



Architecture Fulfillment

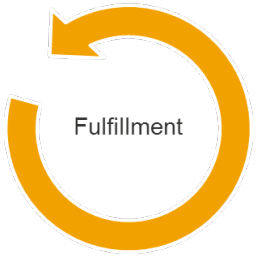
Architectural runway recognized



Behavior	Description	Evidence	Counter-evidence
FF_AR	Work on architectural runway is recognized and planned.	The product backlog or product breakdown structure contains stories or activities to realize architectural elements and technical debt reductions ("enablers") whose business value is indirect (i.e. they derive their business value from other improvements that depend on them).	The product backlog only contains stories that add direct business value. The work breakdown structure only contains work items to realize functionality.
		Ownership for architectural decision making is clearly allocated (to an individual or team).	Nobody takes ownership of the architecture.
		Improvements considered for the next release/sprint are fed by a variety of sources, among which is the architect(ure team).	Only new functionality or features are considered for the next release/sprint.

Architecture Fulfillment

Architecture Debt Control – Business Case

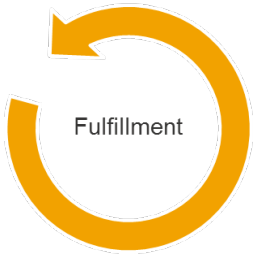


Make Architectural Debt visible as business risk

- 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

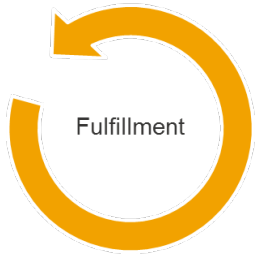
Item		Total
Benefits		
Reduced recurrent maintenance cost	M/yr	
Reduced risk exposure	R/yr	
Total benefits per year	M+R	M+R
Cost		
Principal: effort of migration/refactoring/…	P	
Opportunity cost (delayed features)	F	
Total cost	P+F	P+F
TOTAL RETURN ON INVESTMENT (1 YEAR)		(M+R) – (P+F)

Architecture Fulfillment
Architecture Debt Control



Behavior	Description	Evidence	Counter-evidence
FF_TD	Architectural debt is identified and controlled.	Technology upgrades and work to fix architectural shortcuts are visible on the product backlog or project schedule.	The product backlog only contains stories that add direct business value. The work breakdown structure only contains work items to realize functionality.
		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

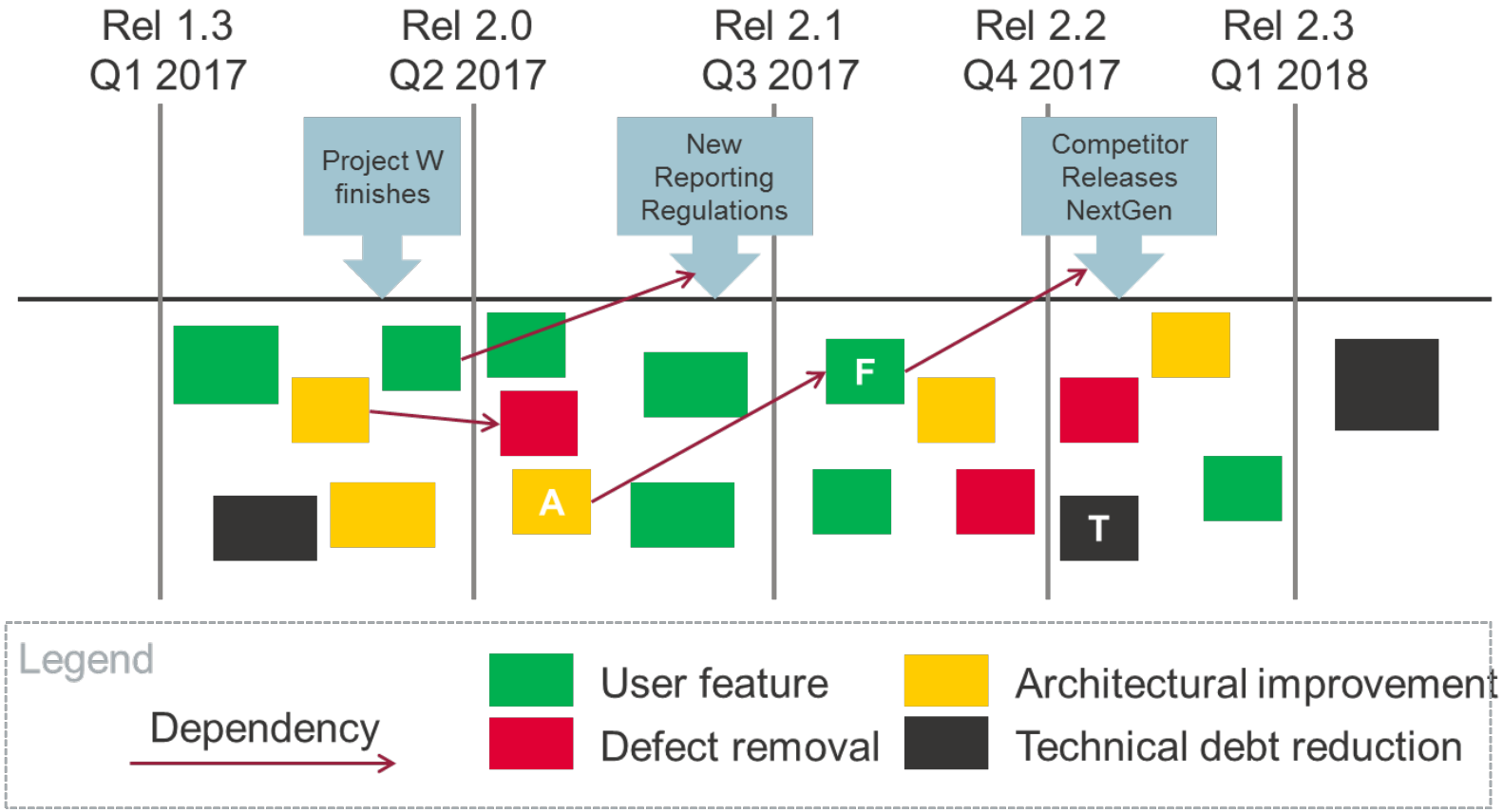


CGI

Just Enough Anticipation achieved by:

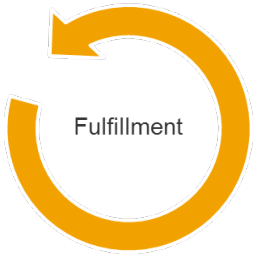
- Dependency Analysis
- Technical Debt Control
- Economic Reasoning

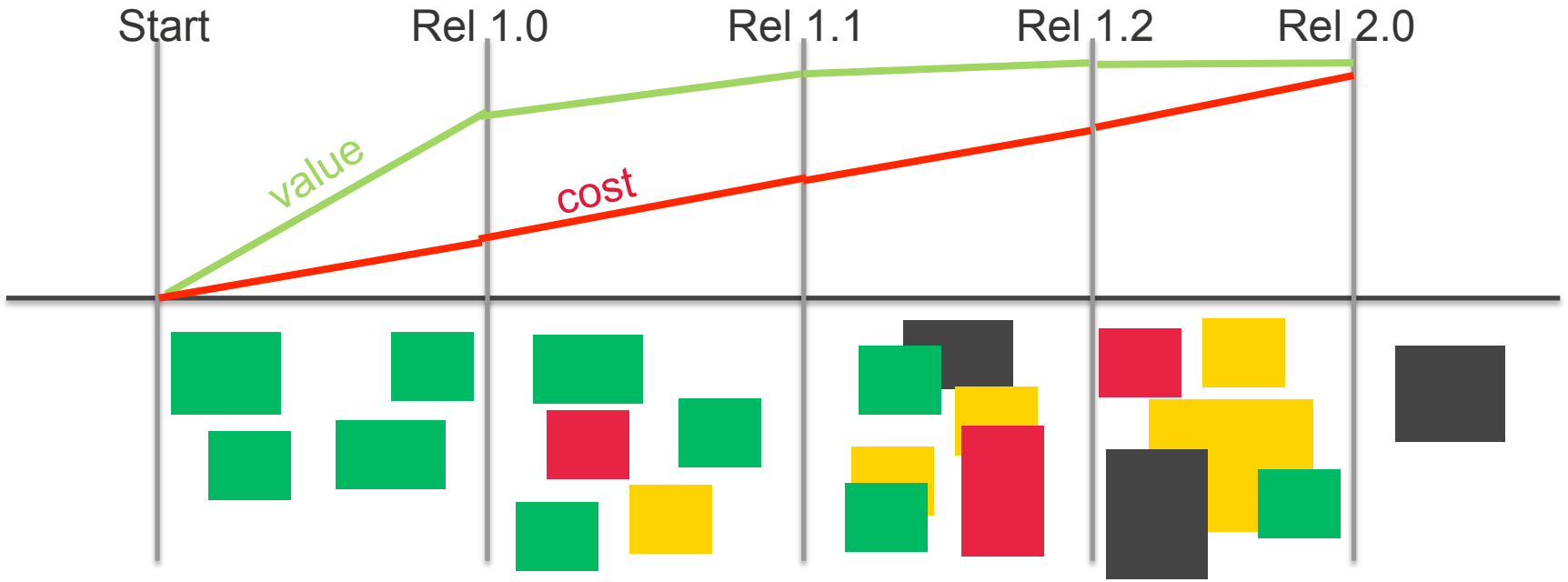
Source: Nanette Brown, Rod Nord, Ipek Ozkaya



Architecture Fulfillment

Strategy 1: Value-first roadmapping

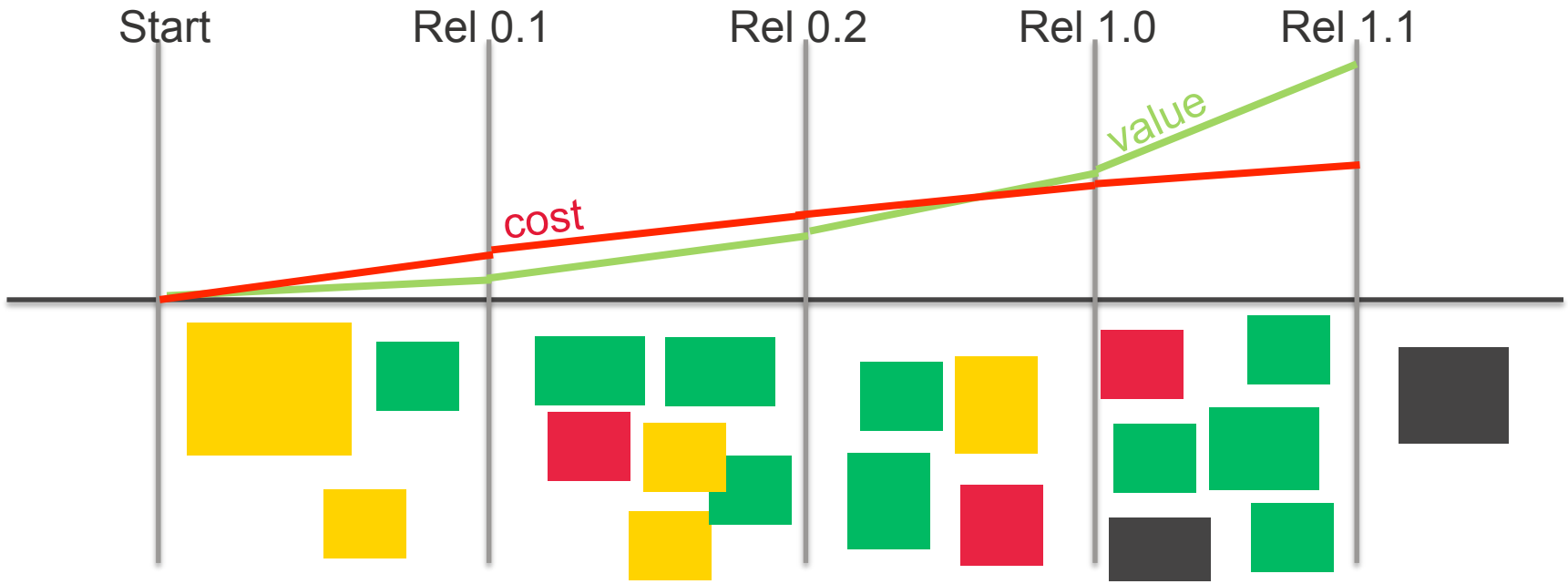




- In line with Agile philosophy
- May increase TCO (more refactoring)
- Too “greedy” algorithm may run project into wall (complete rebuild)
- Good in volatile environments

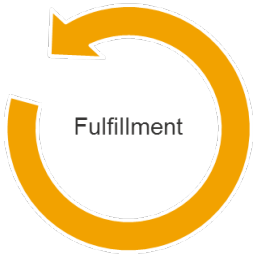
Architecture Fulfillment

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



CGI

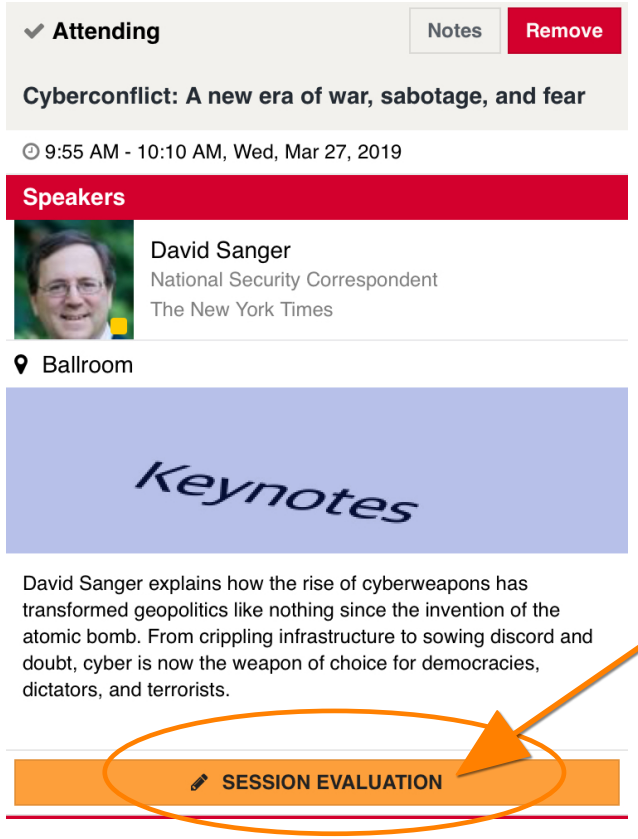
Behavior	Description	Evidence	Counter-evidence
FF_RM	Architecturally significant features, events and stories are anticipated.	Architects/teams work with stakeholders to identify future events that impact the risk, cost and value of of solutions/ systems/landscapes	Urgent events that have significant economic impact take architects/teams or stakeholders by surprise.
		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.

Maturity assessment questions

- Where is your team on the Waterfall Wasteland ←→ Agile Outback map?
- What are the most urgent improvement points to increase your maturity?

Rate this session

Session page on conference website



O'Reilly Events App

Our commitment to you

We approach every engagement with one objective in mind—to help clients succeed.

References

Abrahamsson, P., Babar, M. A., & Kruchten, P. (2010, March/April). Agility and Architecture: Can They Coexist? *IEEE Software*.

Bass, L., Clements, P., & Kazman, R. (2012). *Software Architecture in Practice (3rd Edition)*. Addison-Wesley Professional.

Brown, N., Nord, R. L., & Ozkaya, I. (2010, November/December). Enabling Agility Through Architecture. *CrossTalk*.

Fairbanks, G. (2010). *Just Enough Software Architecture: A Risk- Driven Approach.* Marshall & Brainerd.

Fowler, M. (2003, July/August). Who Needs an Architect? *IEEE Software*, pp. 2-4.

Jansen, A., & Bosch, J. (2005). Software Architecture as a Set of Architectural Design Decisions. *Working IEEE/IFIP Conference on Software Architecture.*

Kruchten, P. (2008). What do software architects really do? *Journal of Systems and Software,* 2413-2416.

Poort, E. R., & van Vliet, H. (2012). RCDA: Architecting as a Risk- and Cost Management Discipline. *Journal of Systems and Software*, 1995-2013.

Poort, E. R. (2014, Sept/Oct). Driving Agile Architecting with Cost and Risk. *IEEE Software.*

Poort, E. R. (2016, Nov/Dec). Just Enough Anticipation: Architect your Time Dimension. *IEEE Software.*

Woods, E, & Bashroush, R. (2019). How Software Architects Focus Their Attention. *Journal of Systems and Software,* submitted.

Tyree, J., & Akerman, A. (2005, 22(2)). Architecture decisions: Demystifying architecture. *IEEE Software*, pp. 19-27.

Contact us to continue the conversation.

Eltjo Poort
Vice-President Architecture
eltjo.poort@cgi.com

linkedin.com/in/eltjopoort
@eltjopoort