



Agile Architecture Roadmapping

Eltjo Poort
SATURN 2016

© CGI Group Inc.

CGI

Experience the commitment®

Eltjo Poort

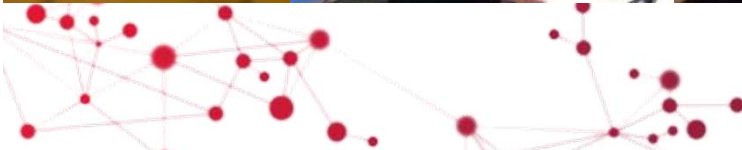
<http://eltjopoort.nl>

CGI Architecture Community of Practice lead

- Reviewing Bids & Projects
- Standardizing & Improving Architecture Practice

Researcher

- With Universities (VU, Twente, Utrecht, Eindhoven)
- Member of IFIP WG 2.10 Software Architecture

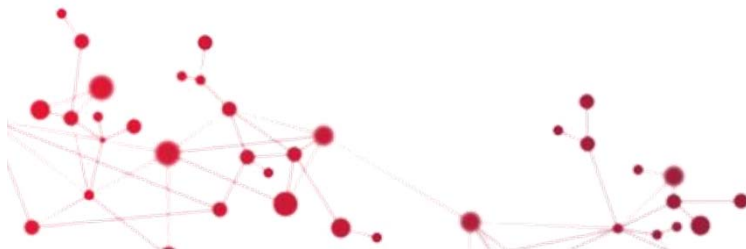


CGI

Do we still need Architecture?

Result	Improvement by applying Solution Architecture	
Budget predictability	2-3 x better	Std dev 32 → 13
Budget overrun	7 x less	22% → 3%
Time overrun	6 x less	48% → 8%
Troubled projects	3 x less	38% → 13%
Customer satisfaction	1-2 points better	10 point scale
Results delivered	+10%	

Survey among 49 software development projects between €50,000 and €2,500,000. Reported by Raymond Slot, PhD Thesis, 2010.



Principles of Agile Architecting

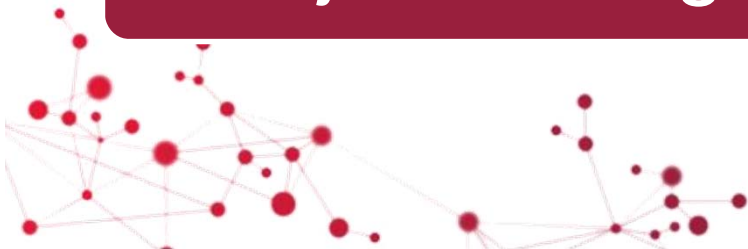
Decisions are your main deliverable

Keep a backlog of architectural concerns

Let economic impact determine your focus

Keep it small

Use just enough anticipation



Principles of Agile Architecting

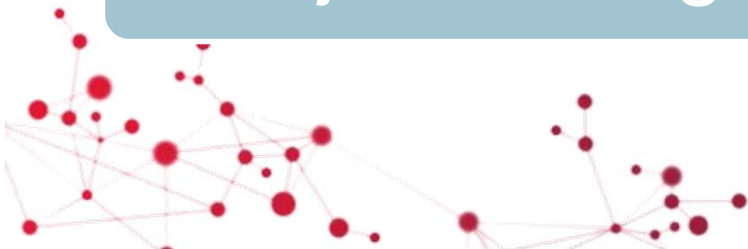
Decisions are your main deliverable

Keep a backlog of architectural concerns

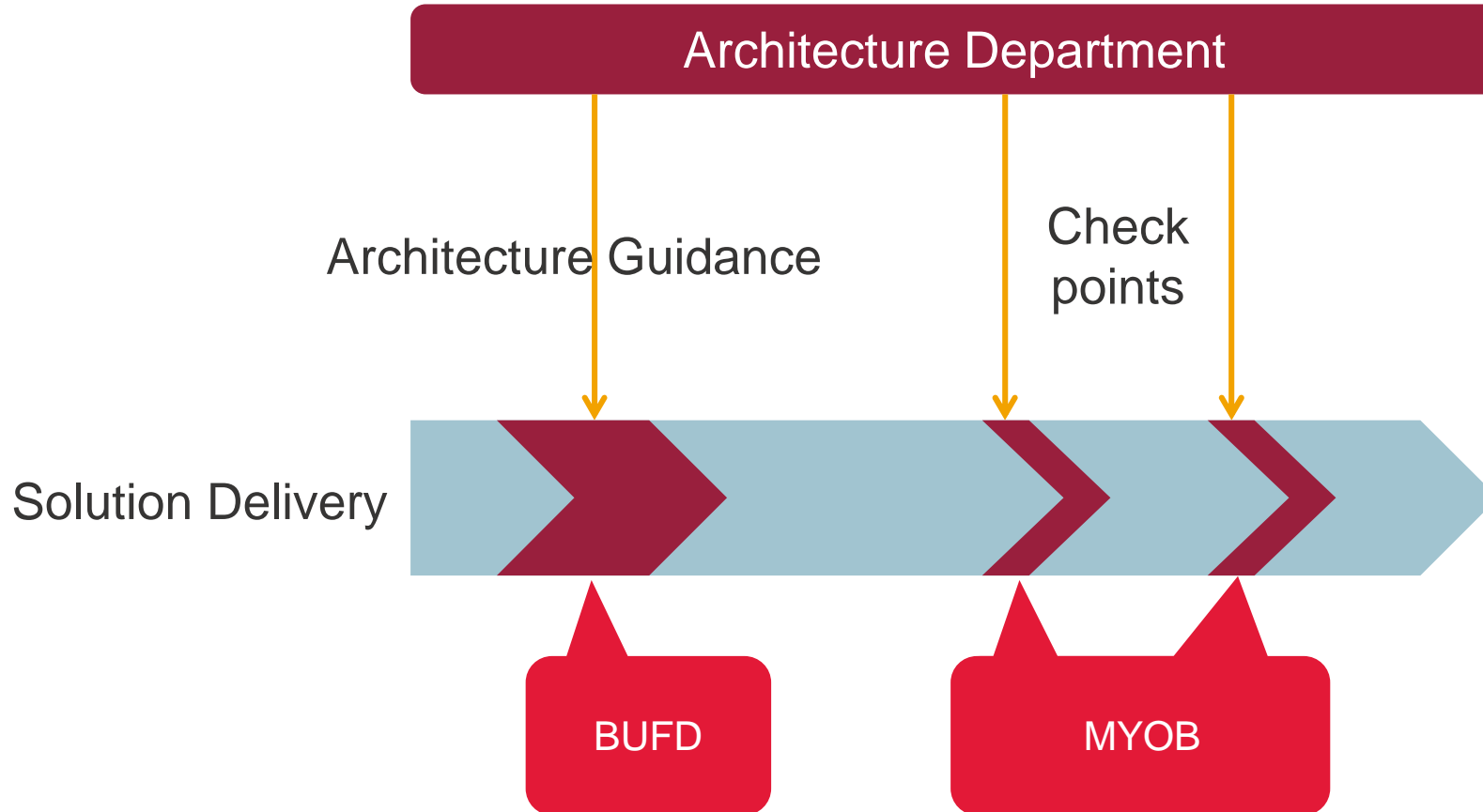
Let economic impact determine your focus

Keep it small

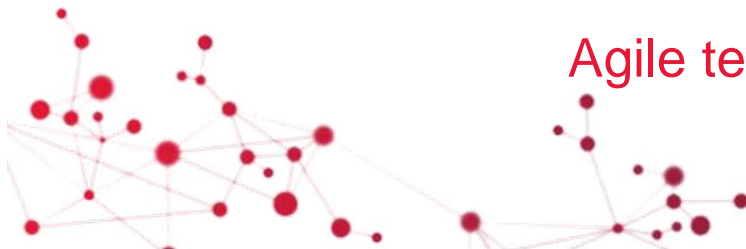
Use just enough anticipation



Traditional architect involvement



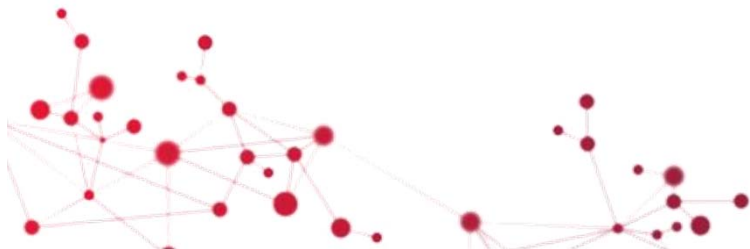
Agile team perception of architect involvement




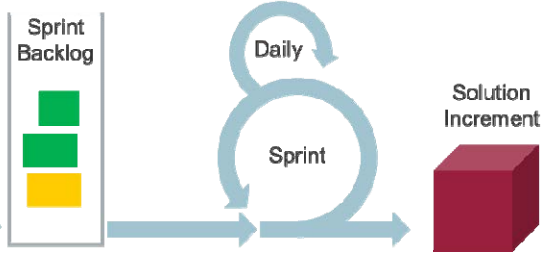

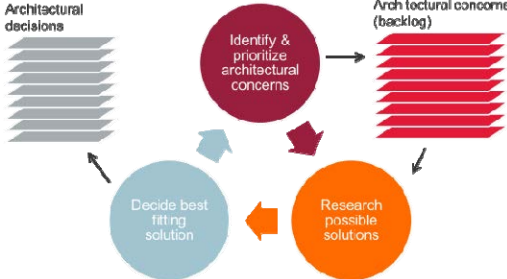
Decisions are your main deliverable

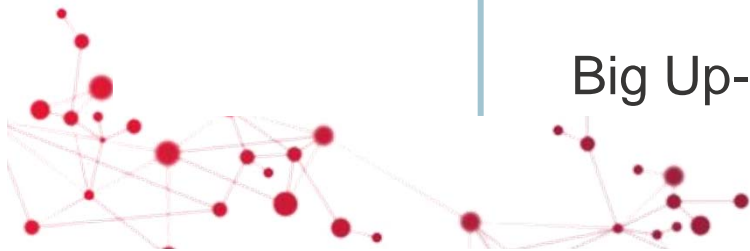
Focus on Architectural Decisions

- Convey change
- Convey implications
- Convey rationale & options
- Ease of traceability
- Agile documentation



Continuous stream of architectural decisions

	Traditional	Agile
Development	 <p>Big Bang</p>	 <p>Continuous stream of improvements</p>
Architecture	 <p>Big Up-Front Design</p>	 <p>Continuous stream of Architectural Decisions</p>



Principles of Agile Architecting

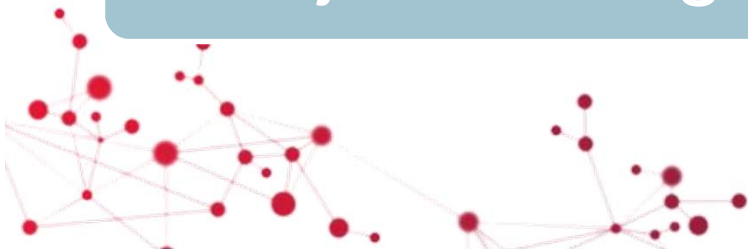
Decisions are your main deliverable

Keep a backlog of architectural concerns

Let economic impact determine your focus

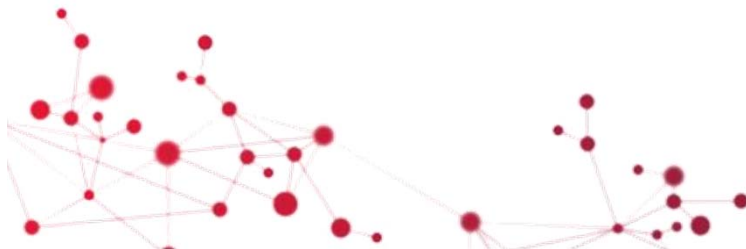
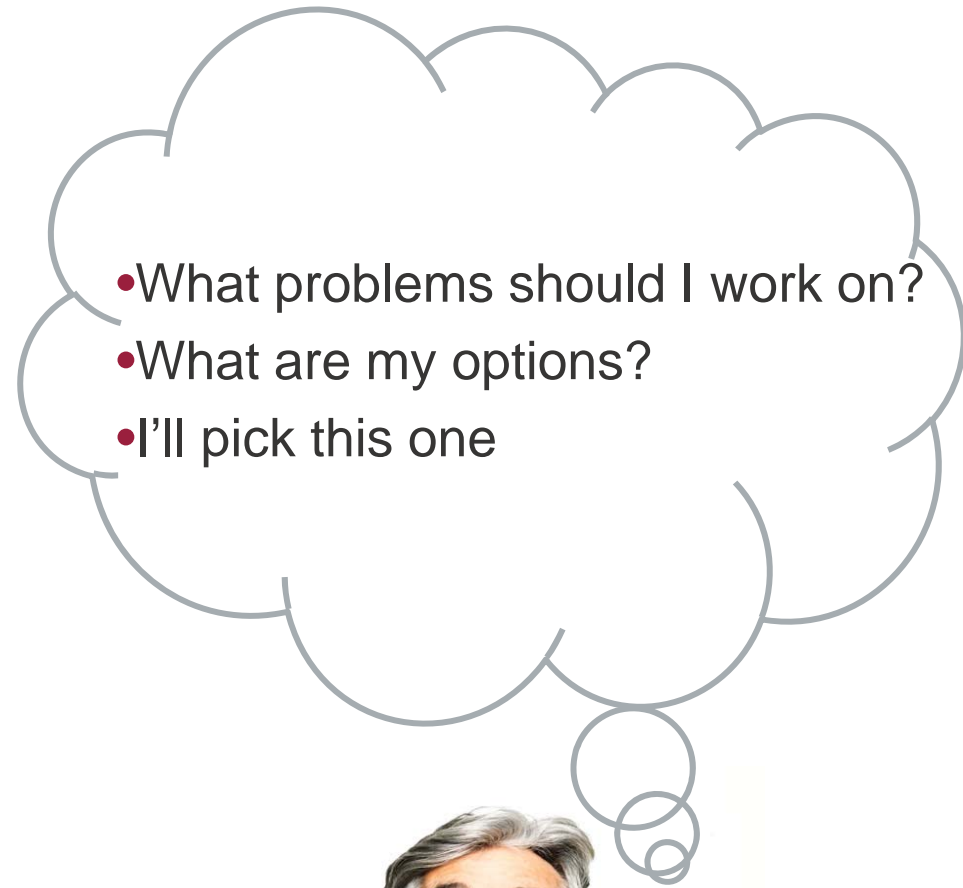
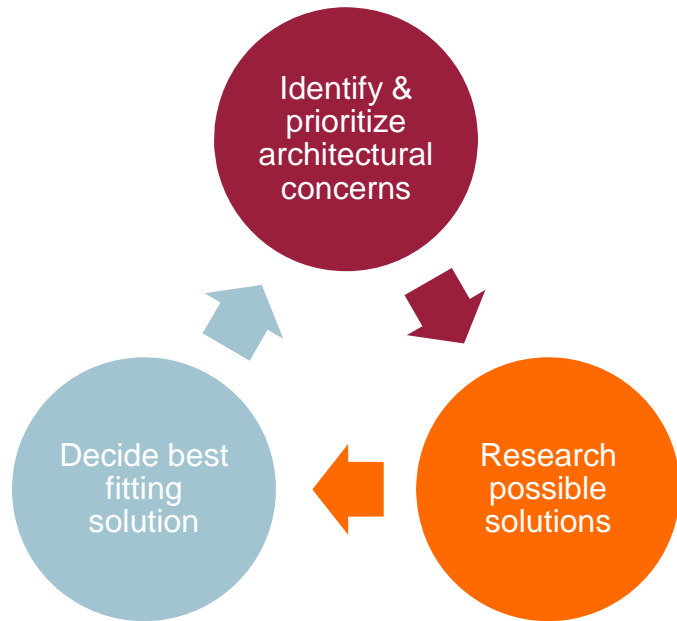
Keep it small

Use just enough anticipation

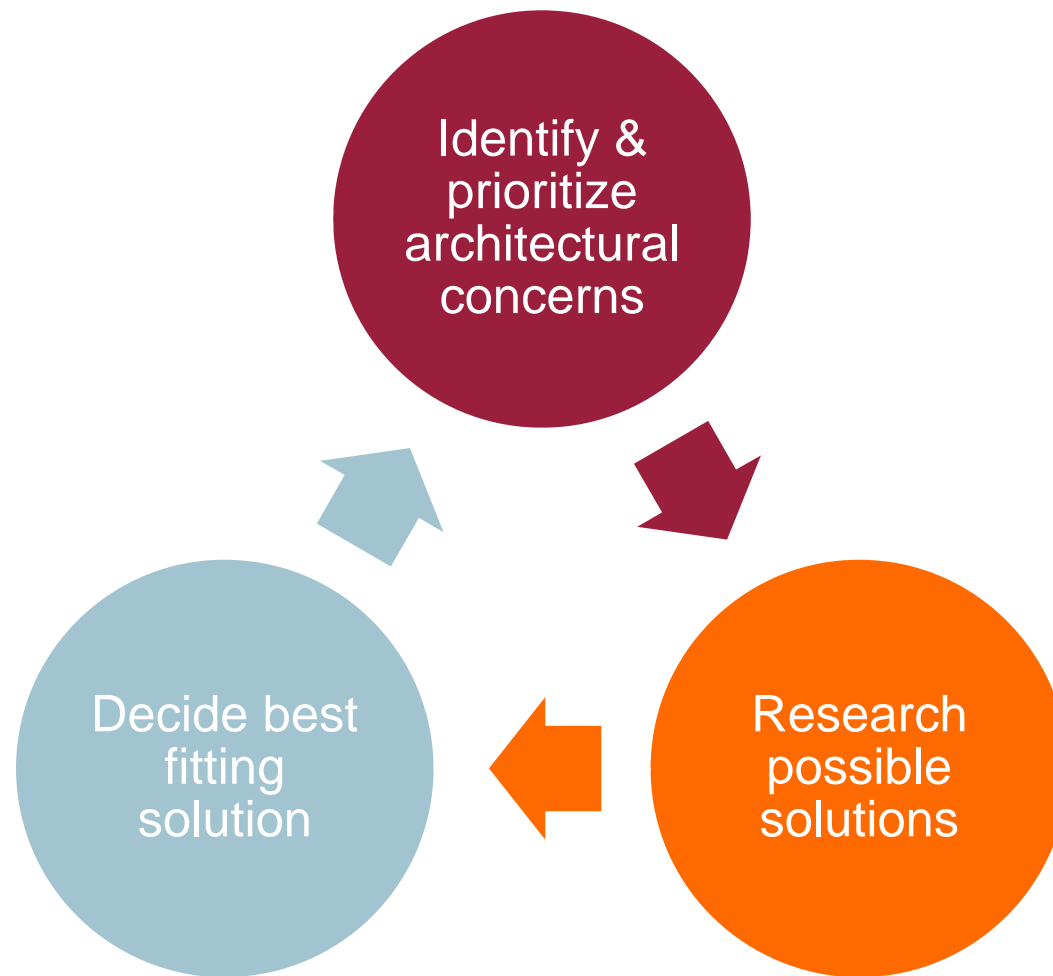


The Architect's Daily Job

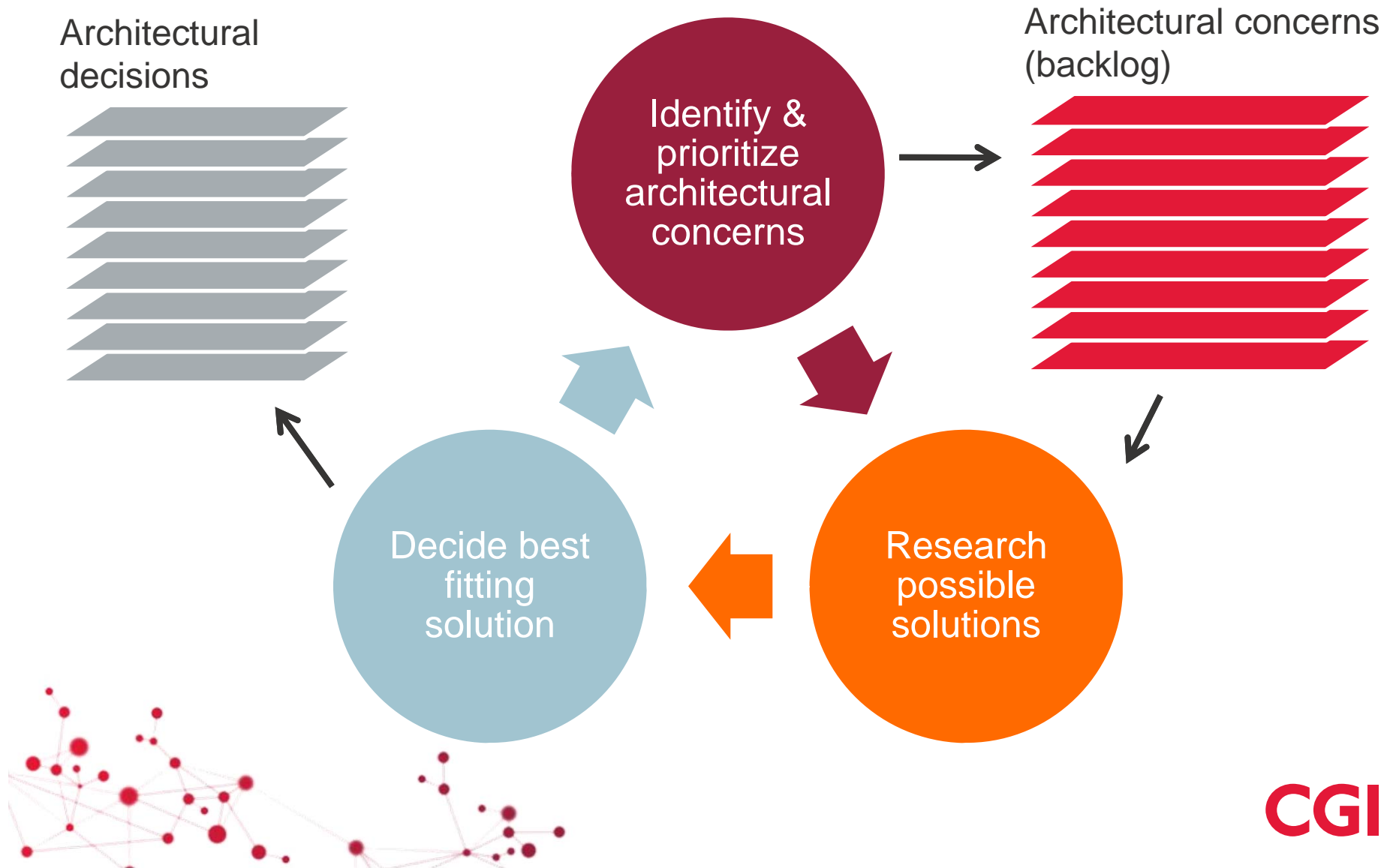
Architecting Microcycle



The Architecting Microcycle



The Architecting Workflow



Principles of Agile Architecting

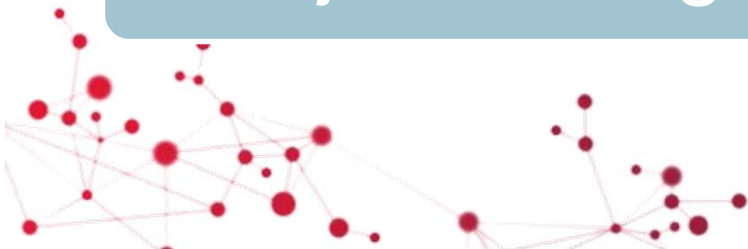
Decisions are your main deliverable

Keep a backlog of architectural concerns

Let economic impact determine your focus

Keep it small

Use just enough anticipation



What is architecture about?

“**Fundamental** concepts or properties of a system in its environment embodied in its elements, relationships, and in the **principles** of its design and evolution”.

[ISO/IEEE]

“Architecture is about the **important stuff**. Whatever that is.”

[Fowler]

After talking to architects and stakeholders on dozens of projects, we have come to equate the “important stuff” with the stuff that has most impact on **risk** and **costs**.

Important \leftrightarrow *high risk and cost*

design

Editor: Martin Fowler @ ThoughtWorks @ fowler@acm.org

Who Needs an Architect?

Martin Fowler



Walking down our corridor a while ago, I saw my colleague Dave Rice in a particularly grumpy mood. My brief question caused a violent statement, “We shouldn’t interview anyone who has ‘architect’ on his resume.” At first blush, this was an odd turn of phrase, because we usually introduce Dave as one of our leading architects.

The reason for his title schizomania is the fact that, even by our industry’s standards, “architect” and “architecture” are terribly overloaded words. For many, the term “software architect” fits perfectly with the smug controlling image at the end of *Matrix Reloaded*. Yet even in firms that have the greatest contempt for that image, there’s a vital role for the technical leadership that an architect such as Dave plays.

What is architecture?

When I was fretting over the title for *Patterns of Enterprise Application Architecture* (Addison-Wesley, 2002), everyone who reviewed it agreed that “architecture” belonged in the title. Yet we all felt uncomfortable defining the word. Because it was my book, I felt compelled to take a stab at defining it.

My first move was to avoid fuzziness by just letting my cynicism hang right out. In a sense, I define *architecture* as a word we use when we want to talk about design but want to puff it up to make it sound important. (Yes, you can imagine a similar phenomenon for ar-

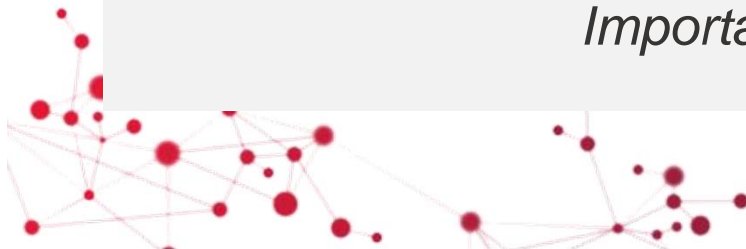
chitect.) However, as so often occurs, inside the blighted cynicism is a pinch of truth. Understanding came to me after reading a posting from Ralph Johnson on the Extreme Programming mailing list. It’s so good I’ll quote it all. A previous posting said

The RUP, working off the IEEE definition, defines architecture as “the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces.”

Johnson responded:

I was a reviewer on the IEEE standard that used that, and I argued weakly that this was clearly a completely bogus definition. There is no highest level concept of a system. Customers have a different concept than developers. Customers do not care at all about the structure of significant components. So, perhaps an architecture is the highest level concept that developers have of a system in its environment. Let’s forget the developers who just understand their little pieces. Architecture is the highest level concept of the expert developers. What makes a component significant? It is significant because the expert developers say so.

So, a better definition would be “In most successful software projects, the expert developers working on that project have a shared understanding of the



CGI

Architecture as a Risk- and Cost Management Discipline

Managing Cost and Risks is architecture's **primary business goal**

Cost and Risks are **prioritizing factors** determining architect's concerns

Architect should be an expert on costing and risk mitigation

Architecture as a risk mitigation mechanism

- Reduce uncertainty in feasibility of solution
- Reduce troubled projects

Architecture as a cost control mechanism

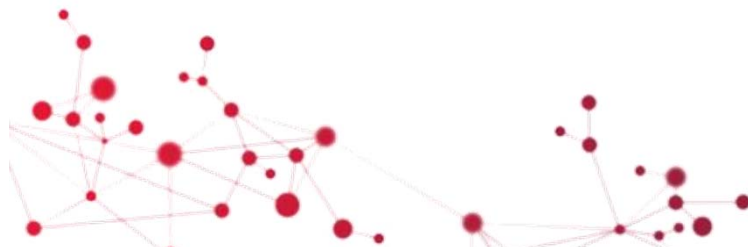
- Better predictability of solution cost
- Less budget overrun



RCDA: Risk and Cost Driven Architecture

Solution architecting principles and practices based on a view of architecture as a risk and cost management discipline

- Applicable in agile and traditional engagements
- Highly scalable and pragmatic
- Architectural decision making based on economic trade-offs
- Architecture communication in economic terms
- Traceability from requirements to cost



CGI

Architecture as a Risk- and Cost Management Discipline

Managing Cost and Risks is architecture's **primary business goal**

Cost and Risks are **prioritizing factors** determining architect's concerns

Architect should be an expert on costing and risk mitigation

Architecture as a risk mitigation mechanism

- Reduce uncertainty in feasibility of solution
- Reduce troubled projects

Architecture as a cost control mechanism

- Better predictability of solution cost
- Less budget overrun



Principles of Agile Architecting

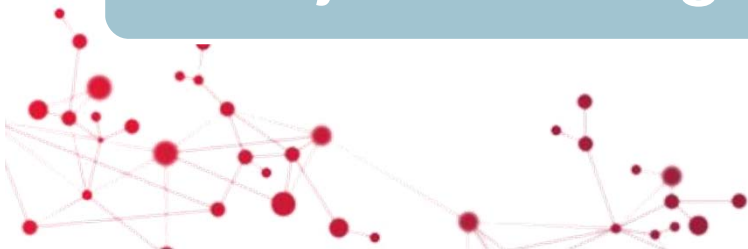
Decisions are your main deliverable

Keep a backlog of architectural concerns

Let economic impact determine your focus

Keep it small

Use just enough anticipation

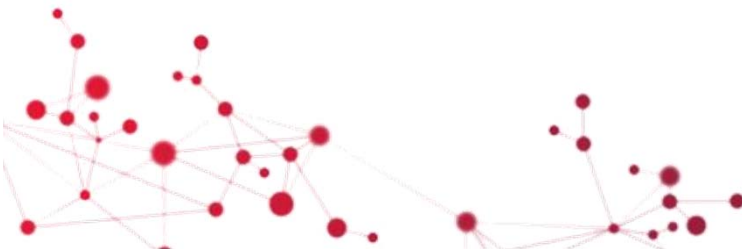


Architecture should be minimal

Avoid trap of “architecting everything”

- Architectures are hard to change
- Big architectures obstruct agility
- Give designers/developers as much freedom as they can handle
- Give yourself chance to keep total overview
- Three factors determine optimal amount of up-front design:

Business criticality + Size - Volatility



Principles of Agile Architecting

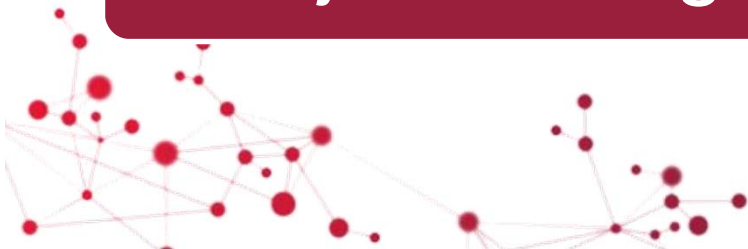
Decisions are your main deliverable

Keep a backlog of architectural concerns

Let economic impact determine your focus

Keep it small

Use just enough anticipation



Just Enough Anticipation

Flow of architectural decisions ahead of development

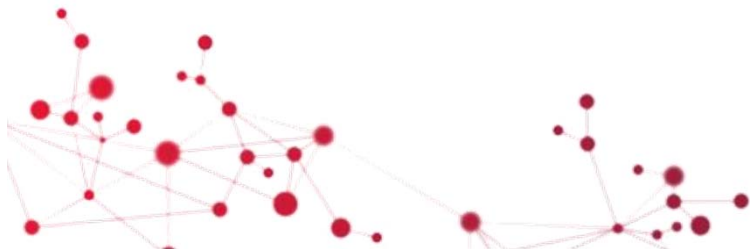
Metaphor: Runway extended while in operation

- Just long enough to accommodate anticipated airplanes



Key tools to determine right amount of anticipation:

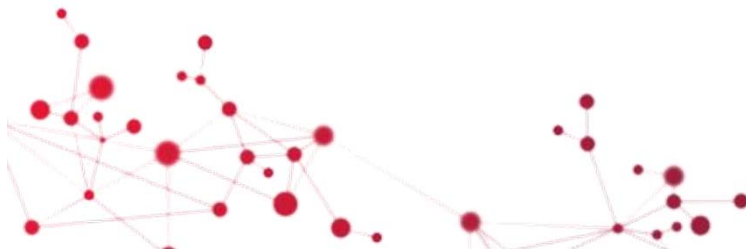
- Dependency analysis
- Technical debt control
- Economic trade-off: Net Present Value, Real Options Analysis



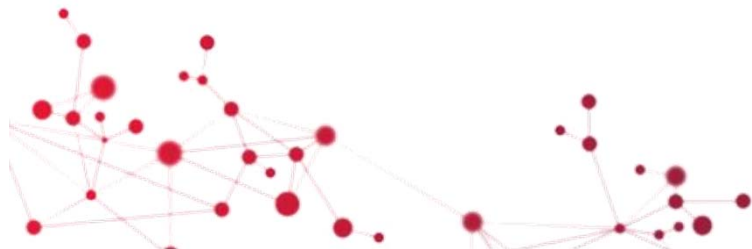
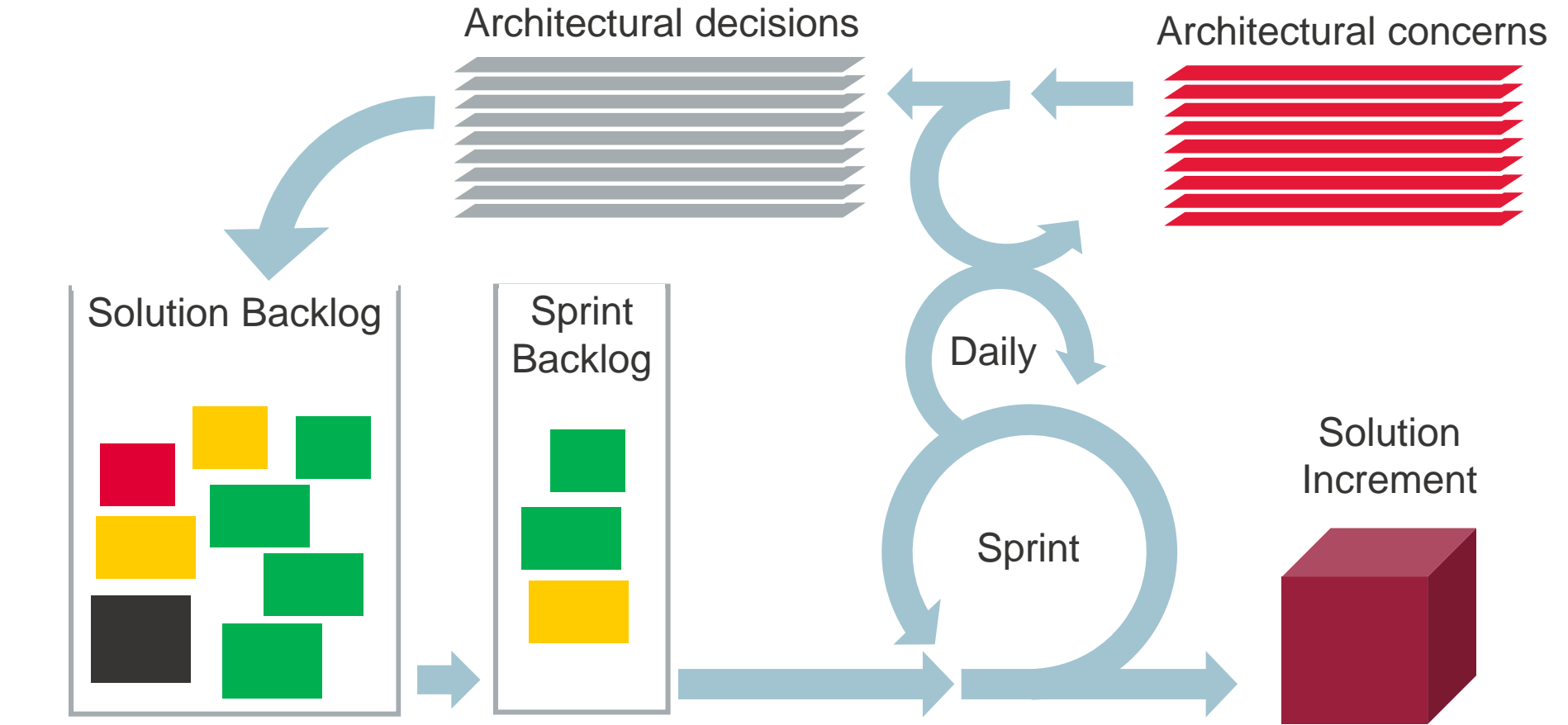
Balance your backlog

Architecture and other solution improvements

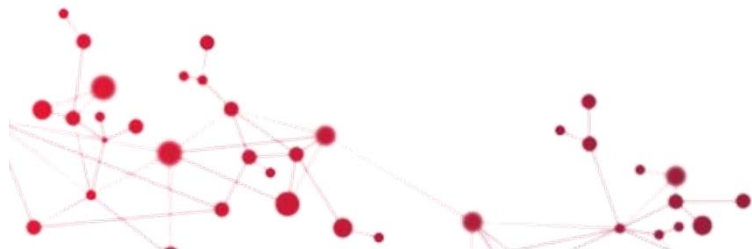
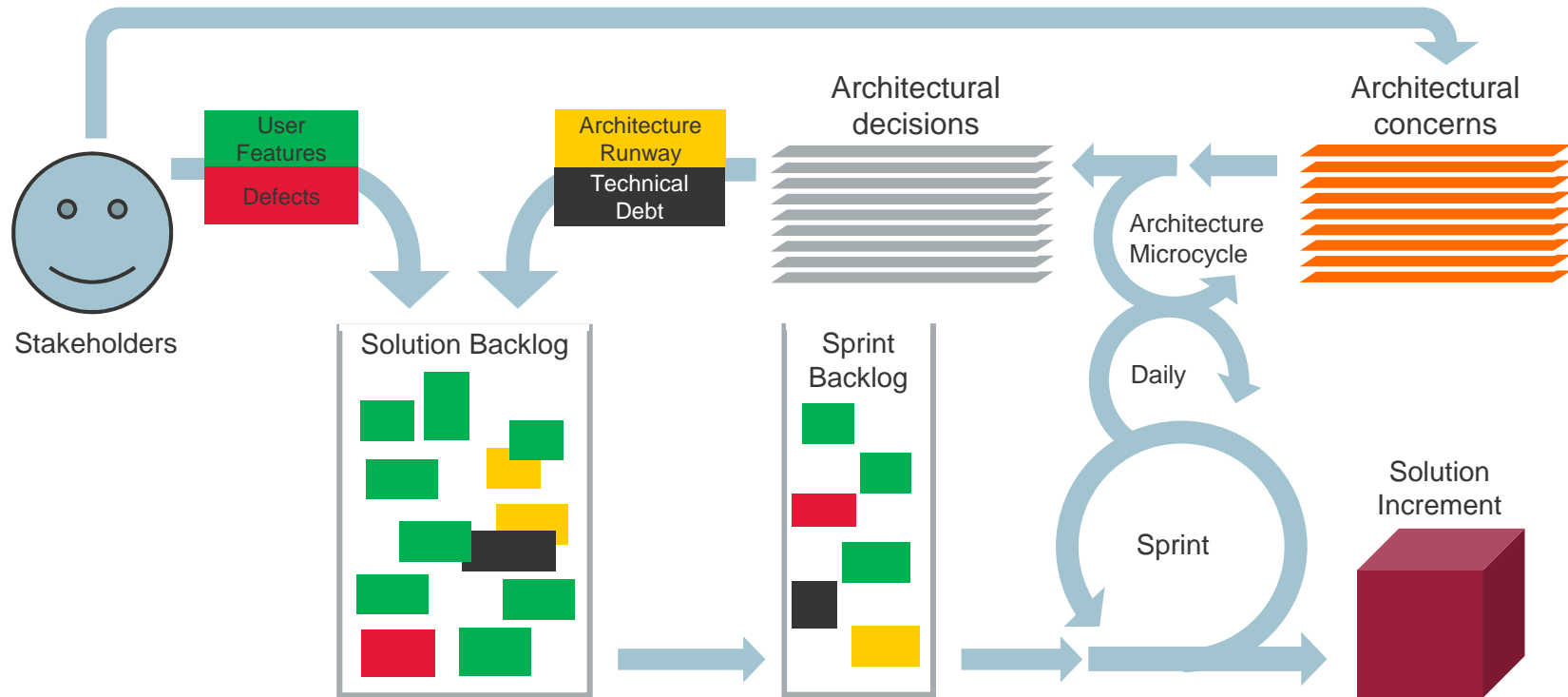
	Visible	Invisible
Positive Value	New features Added functionality	Architectural, Structural features
Negative Value	Defects	Technical Debt



SCRUM and the Architecture Microcycle



SCRUM and the Architecture Microcycle



Architecting the Time dimension

Issues with time-agnostic architectures

- Limited usefulness of architecture documents
 - perpetually “almost finished”
 - already obsolete when they’re issued
- Risk of development based on revoked architectural decisions
- Difficulty planning ahead



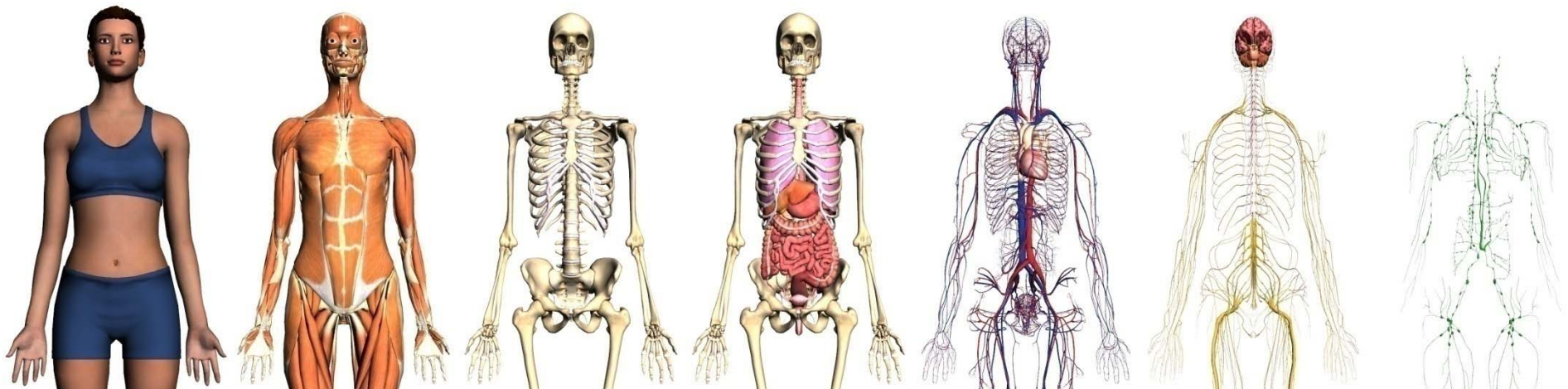
Architecting the Time dimension

Evolution Viewpoint

All architecture documentation methods use **views**

- ISO 42010, TOGAF, Archimate, 4 + 1, 'Views and Beyond'
- Viewpoints address concerns per stakeholder (group)

What if we added a viewpoint for timing concerns?

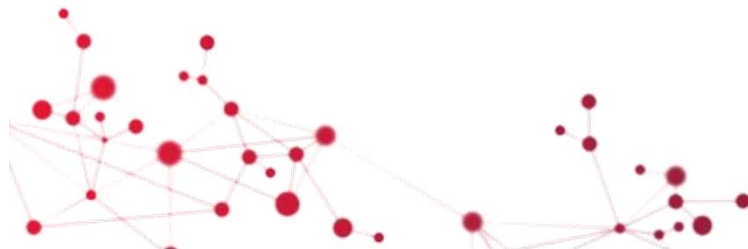


Architecting the Time dimension

Evolution Viewpoint

Step 1: Identify events with architectural impact

Event	When expected	Impact type	Impact
Competitor releases next generation product	Q4/2017	Business value + Risk	Our own product will be harder to sell if we do not match their new features, which would cause us to lose revenue.
Microsoft Windows XP support discontinued	4/2014	Risk	Vulnerabilities no longer patched; implies security risk, e.g. risk of intrusion and data leaks.
Corilla license contract expires	5/2017	Cost	Opportunity for cost reduction by switching to open source alternative.
New version of IBM WebSphere	11/2015	Cost	Opportunity for maintenance cost reduction by using new features announced for next version.
Project to build System Y finishes	Q1 2017	Business value + Risk	System Y (which is interdependent with ours) will require interface features that are currently not supported by our solution. We need to build these features or our solution will lose its business value.



Architecting the Time dimension

Evolution Viewpoint

Step 2: Identify backlog items for solution roadmap

project backlog

user stories

use cases

functional requirements

feature wish-list

acceptance criteria

change request log

solution blueprint

architectural concerns

architectural decisions

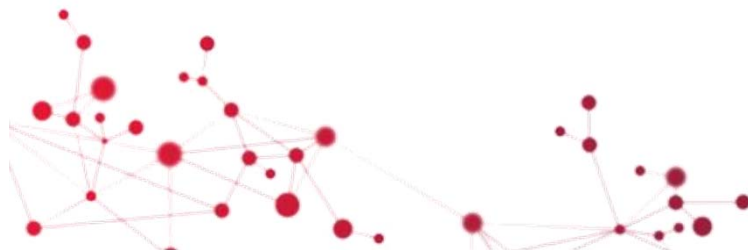
part list

	Visible	Invisible
Positive Value	New features Added functionality	Architectural, Structural features
Negative Value	Defects	Technical Debt

defect database

architectural concerns

risk list



Architecting the Time dimension

Evolution Viewpoint

Step 3: Dependency Analysis

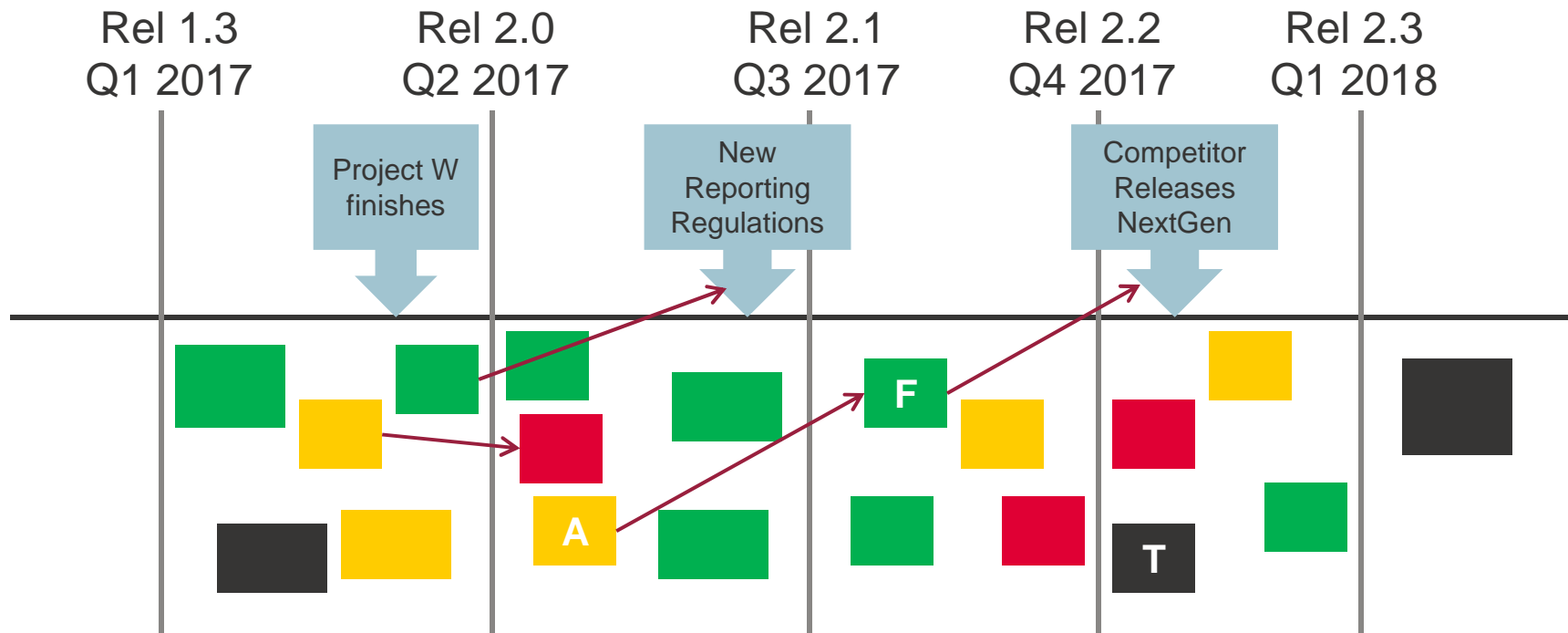
	Logon	GPS	I/F-A	Session	Cache	Pub/Sub	DataPers	RuleEng
UC1				X			X	
UC2				X			X	
UC5			X			X	X	
UC6			X			X		X
UC7	X	X		X				X
UC8	X			X				
UC9		X				X		
AT3					X	X		
AT4						X		
AT5					X			
AT6	X							X
AT7	X							
AT8				X	X		X	
AT9							X	



Architecting the Time dimension

Evolution Viewpoint

Step 4: Visual Timeline



Legend

Dependency



User feature



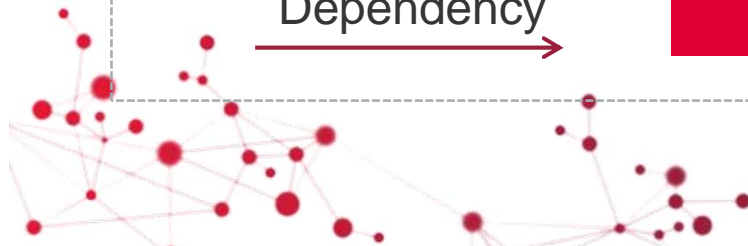
Architectural improvement



Defect removal

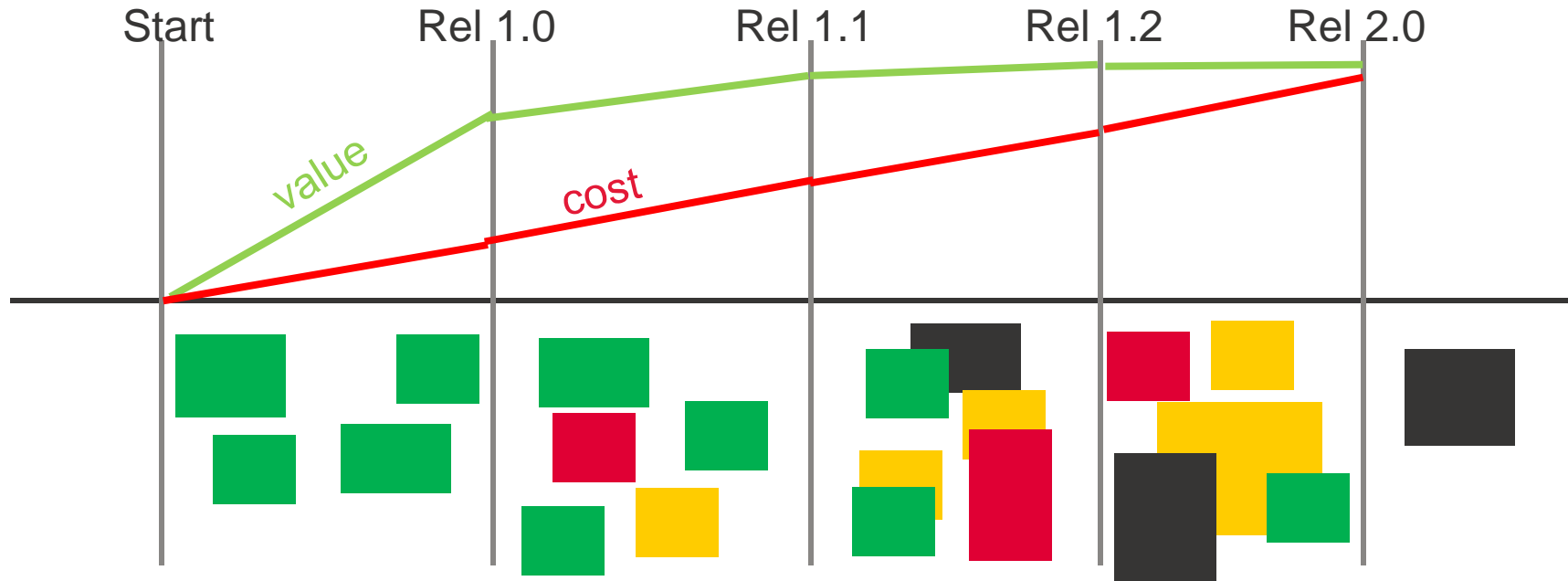


Technical debt reduction

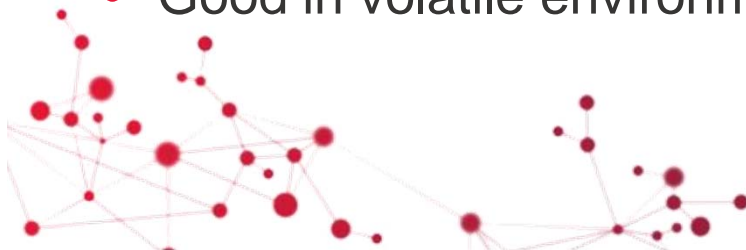


Architecture Roadmapping

Release strategy 1: value-first

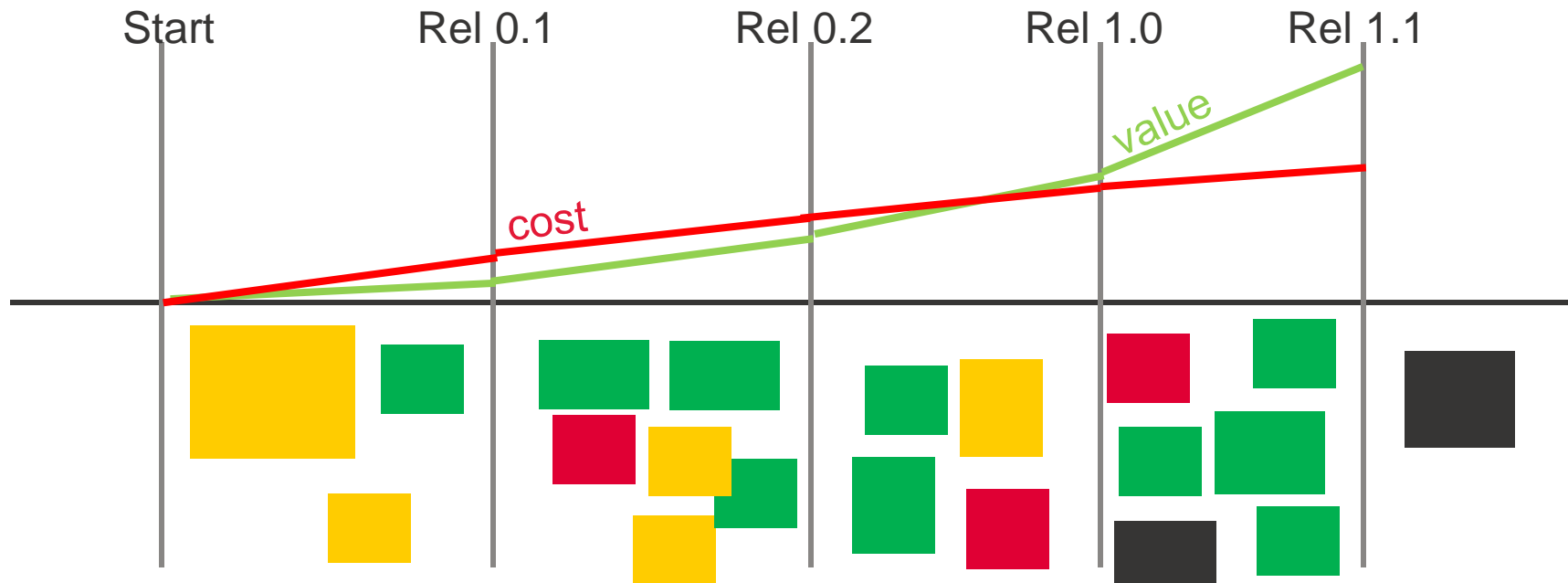


- In line with Agile philosophy
- May increase TCO (more refactoring)
- Too “greedy” algorithm may run project into wall (complete rebuild)
- Good in volatile environments



Architecture Roadmapping

Release strategy 2: architecture-first



- 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

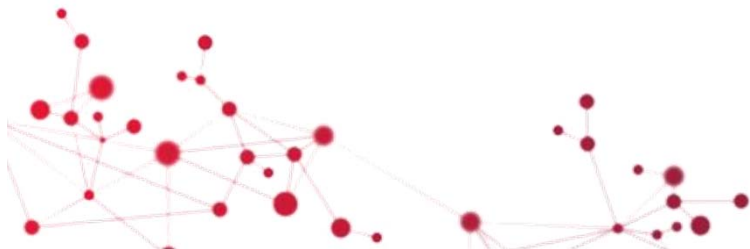


Architecture Roadmapping

Real-life experiences (1/3)

Typically found architecturally significant events:

- Project or process milestones, such as delivery and approval deadlines; also deadlines in dependent projects
- Product version/infrastructure upgrades
- Business changes
 - Changing agreements (KPIs, SLAs), mergers/take-overs, legislative/policy
- Changes in availability of resources, e.g. availability of expertise

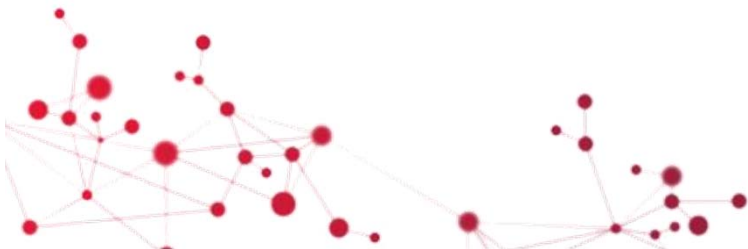


Architecture Roadmapping

Real-life experiences (2/3)

Lessons learned

- Anticipation documents often informal
 - “roadmap”
 - “decision support”
 - “strategy document”
- Need stakeholders to identify significant future events!

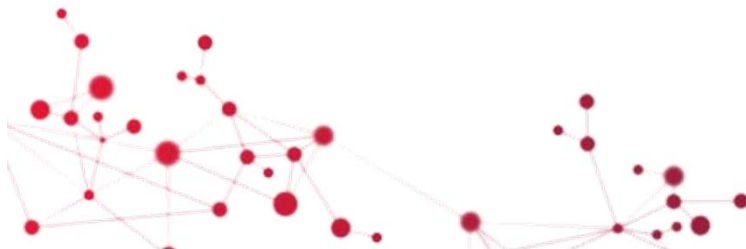


Architecture Roadmapping

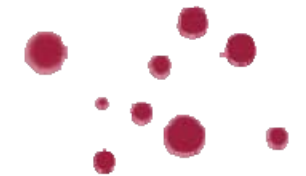
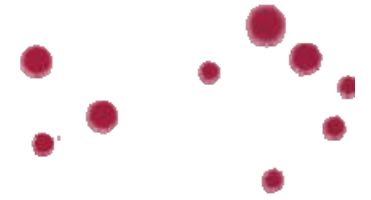
Real-life experiences (3/3)

Significant benefits observed

- Improved (more realistic) stakeholder expectations
- Better prioritization of required architectural improvements
- Helps architects articulate business impact of roadmapping scenarios
- Helps architects discuss timing of architectural improvements
 - based on business impact rather than generic (dogmatic) “rules” like YAGNI



Questions or Comments?

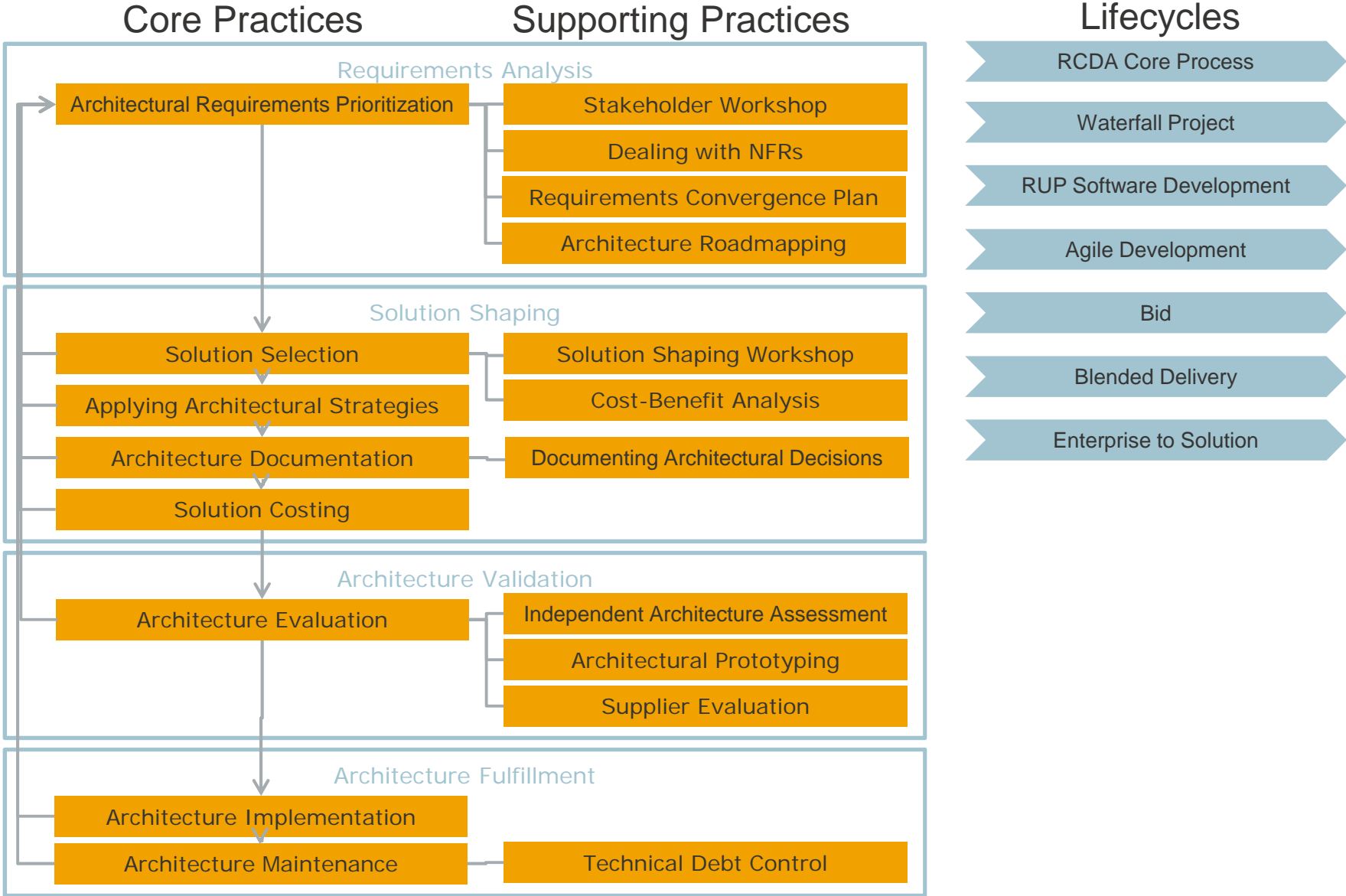


Spare slides follow



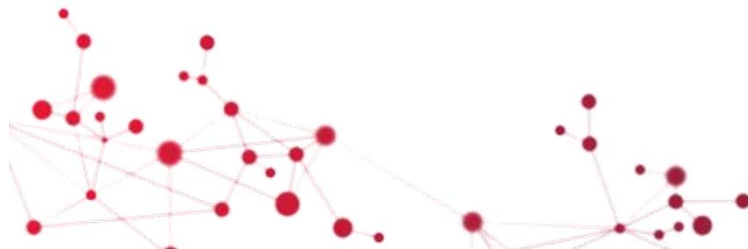
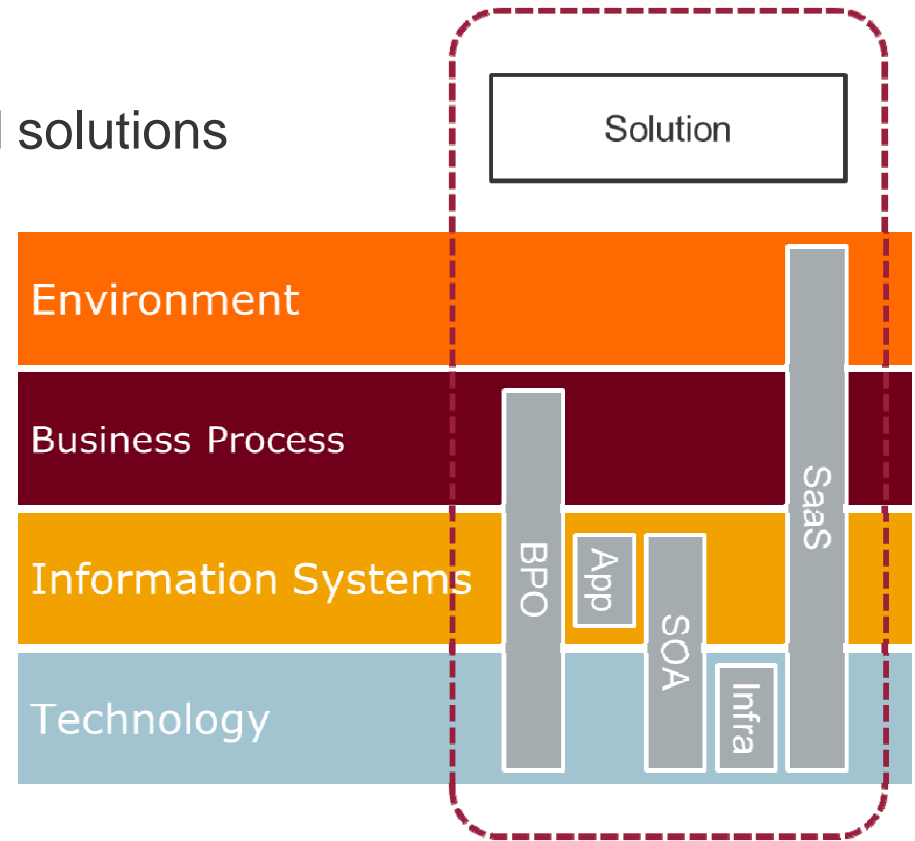
Experience the commitment®

RCDA Practices



“Solution” Architecture?

- RCDA covers all types of IT-based solutions
 - software application
 - system of systems
 - BPO solution
 - service solution
 - systems integration
 - embedded system
 - software as a service
 - ...
- Highly scalable
 - from 2-week architecture for short-deadline bids...
 - ...to 2-year architecture for long-term engagements



References

- Abrahamsson, P., Babar, M. A., & Kruchten, P. (2010, March/April). Agility and Architecture: Can They Coexist? *IEEE Software*.
- Boehm, B. (2010). Architecting: How Much and When? In A. Oram, & G. Wilson, *Making Software: What Really Works, and Why We Believe It*. O'Reilly Media.
- Brown, N., Nord, R. L., & Ozkaya, I. (2010, November/December). Enabling Agility Through Architecture. *CrossTalk*.
- 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*.
- Malan, R., & Bredemeyer, D. (2002, september/oktober). Less is More with Minimalist Architecture. *IT Pro*, pp. 46-48.
- 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*.
- Slot, R. (2010). *A method for valuing architecture-based business transformation and measuring the value of solutions architecture*. Amsterdam.

