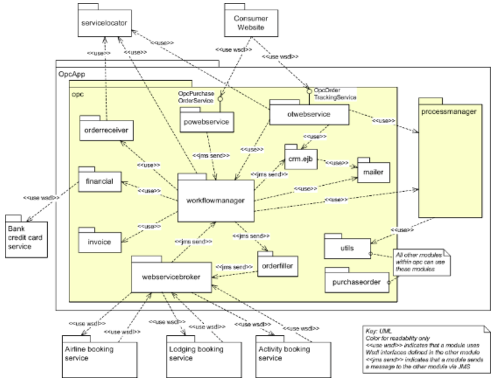
The Life and Times of an Architecture

Finding the common ground between traditional and agile architecting

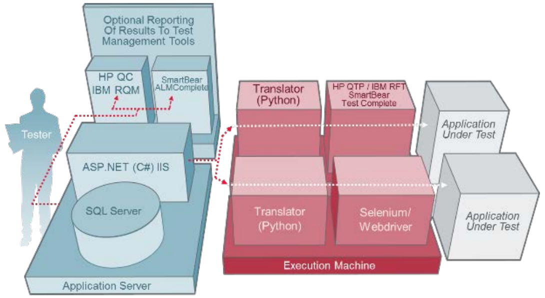
Eltjo Poort
SATURN 2017

CGI
Experience the commitment®

What do we architect?



Internal structure of software

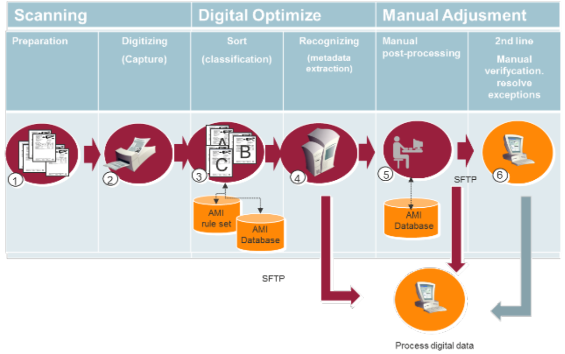


Software CI/CD pipelines

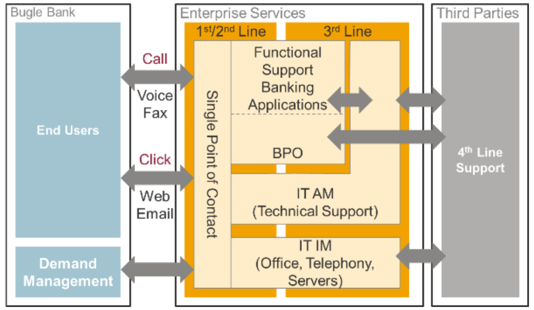
IT infrastructure for application landscape

Embedded systems

Integrated software systems



Enterprise IT systems



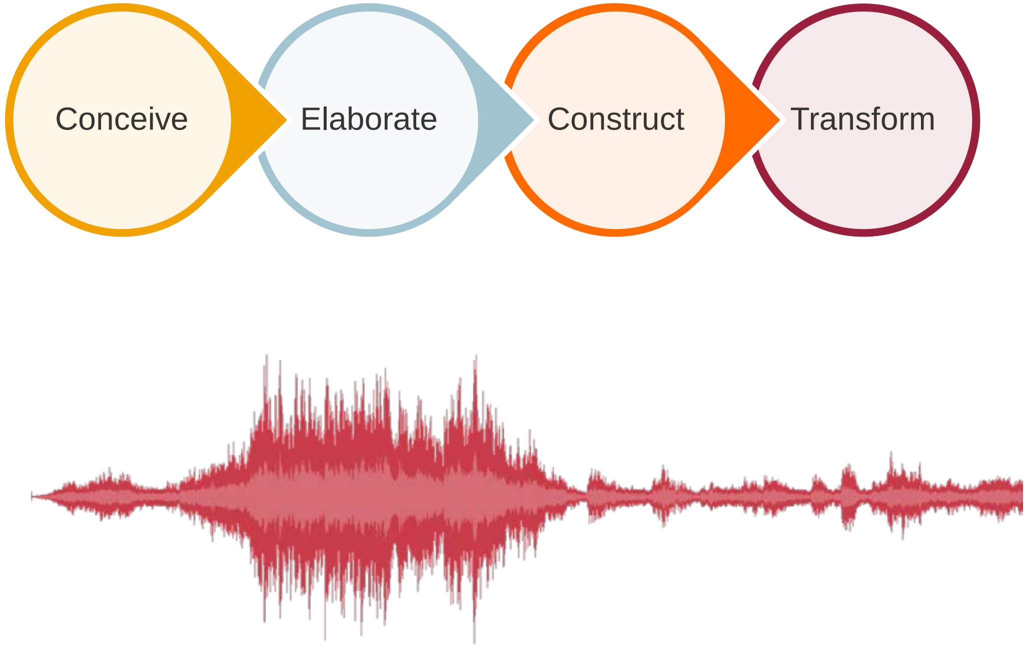
Outsourcing services

The world’s largest robot

When do we architect?

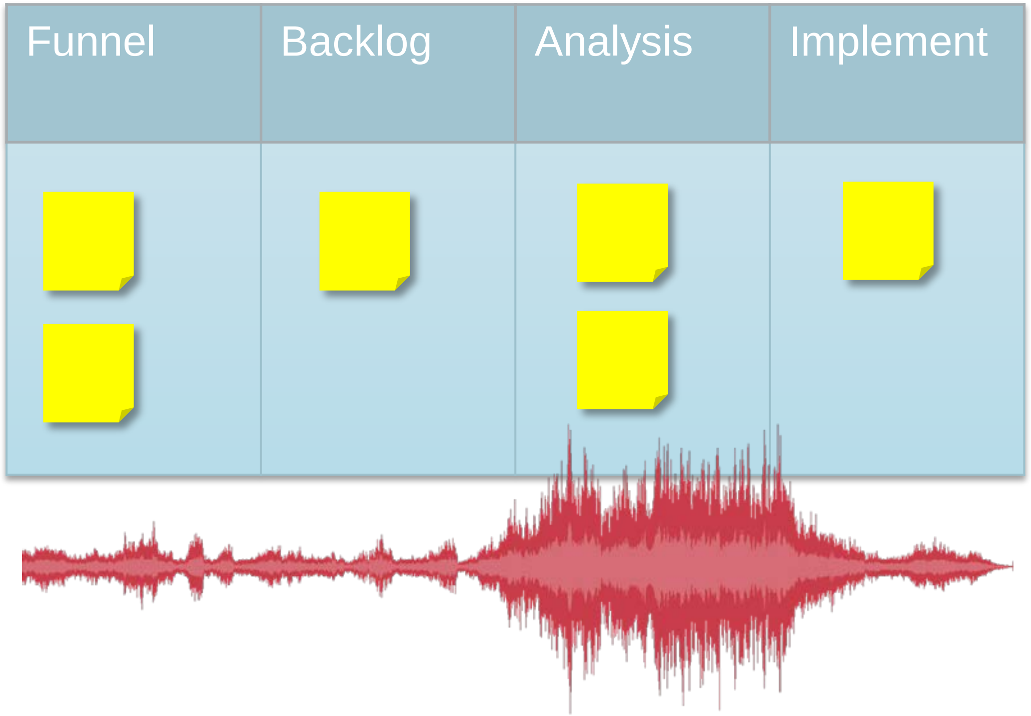
Traditional governance

Project



Agile governance

Architectural Epic



Definition of Solution

***Solution**: a coherent set of changes delivered to address a defined set of stakeholder needs*

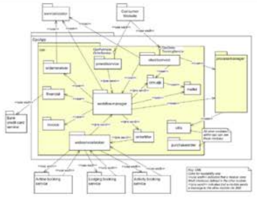
Changes: solution elements are created, modified or removed

Delivered: coordination depends on governance model:

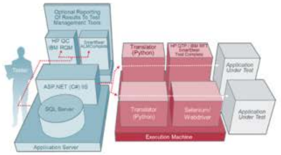
- agile or traditional
- value stream, program or project
- contractual or otherwise

Defined: depends on governance model:

- Epic / set of (user) stories
- Program / project definition
- Contract
- Change request



Internal structure of software

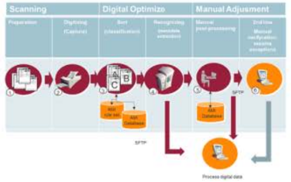


Software CI/CD pipelines

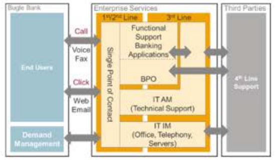
IT infrastructure for application landscape

Embedded systems

Integrated software systems



Enterprise IT systems



Outsourcing services

The world's largest robot

How do we architect?

Architecture as a stream of design decisions lowers the cost of change:

- Convey rationale and options
- Convey changes and implications
- Deal with changing context

Knowing the “Why?” behind an architectural decision is key to revisiting that decision when things change

Why do we architect?

Architecture’s primary business goal is risk and cost reduction, and it works.

Projects that apply architecture practices have:

- Reduced uncertainty in feasibility of solution
- Reduced risk of delivery troubles
- Better predictability of solution cost
- Less budget overrun

Source: Raymond Slot, PhD Thesis, 2010.

design

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

Who Needs an Architect?

Martin Fowler

Wandering 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 schizophrenia 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 architect.) 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:

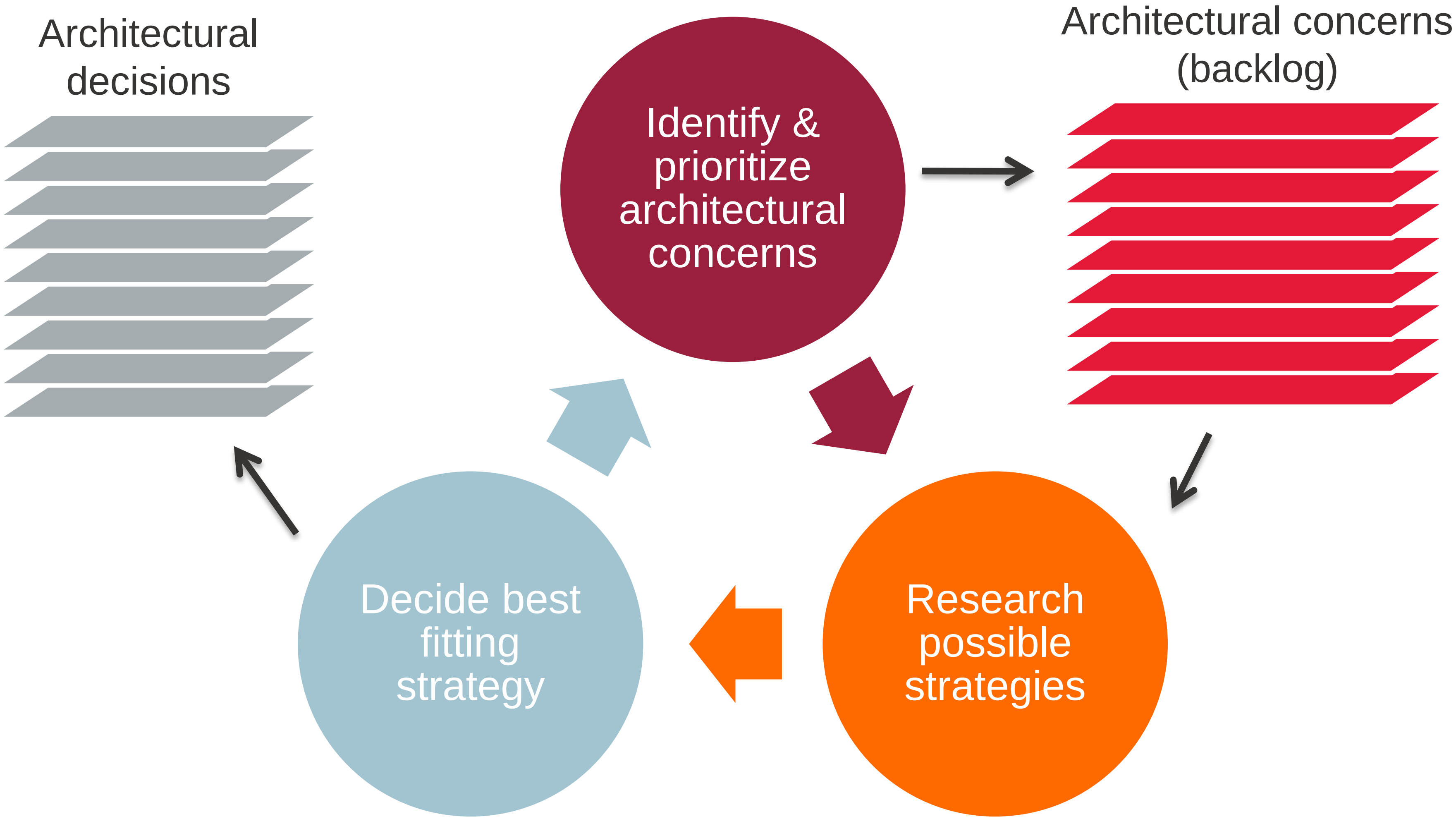
> I was a reviewer on the IEEE standard that used that, and I argued uselessly that this was clearly a completely bogus definition. There is no highest level concept of a system. Customers have a different concept than developers. Customers do not care at all about the structure of significant components. 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 piece. 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

2 IEEE SOFTWARE *Published by the IEEE Computer Society* 0740-7459/03/$17.00 © 2003 IEEE

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

The Architecting Microcycle

Workflow for architectural decision making


Architectural decisions
Identify & prioritize architectural concerns
Architectural concerns (backlog)
Decide best fitting strategy
Research possible strategies

Cone of Uncertainty

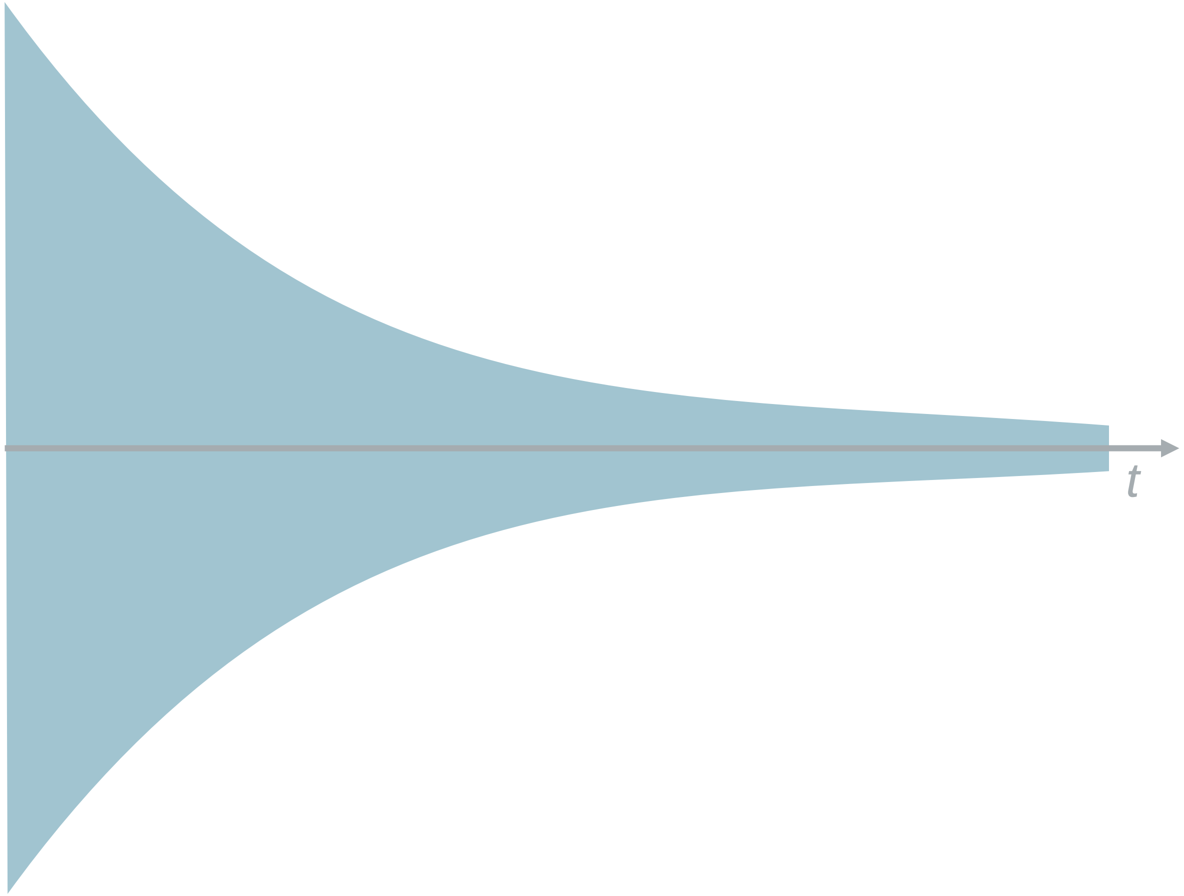
Evolution of amount of uncertainty

(in solution life cycle)

Cone is narrowed by:

- Research
- Decision making

Architecture narrows the cone by researching strategies to architectural concerns and making architectural decisions.



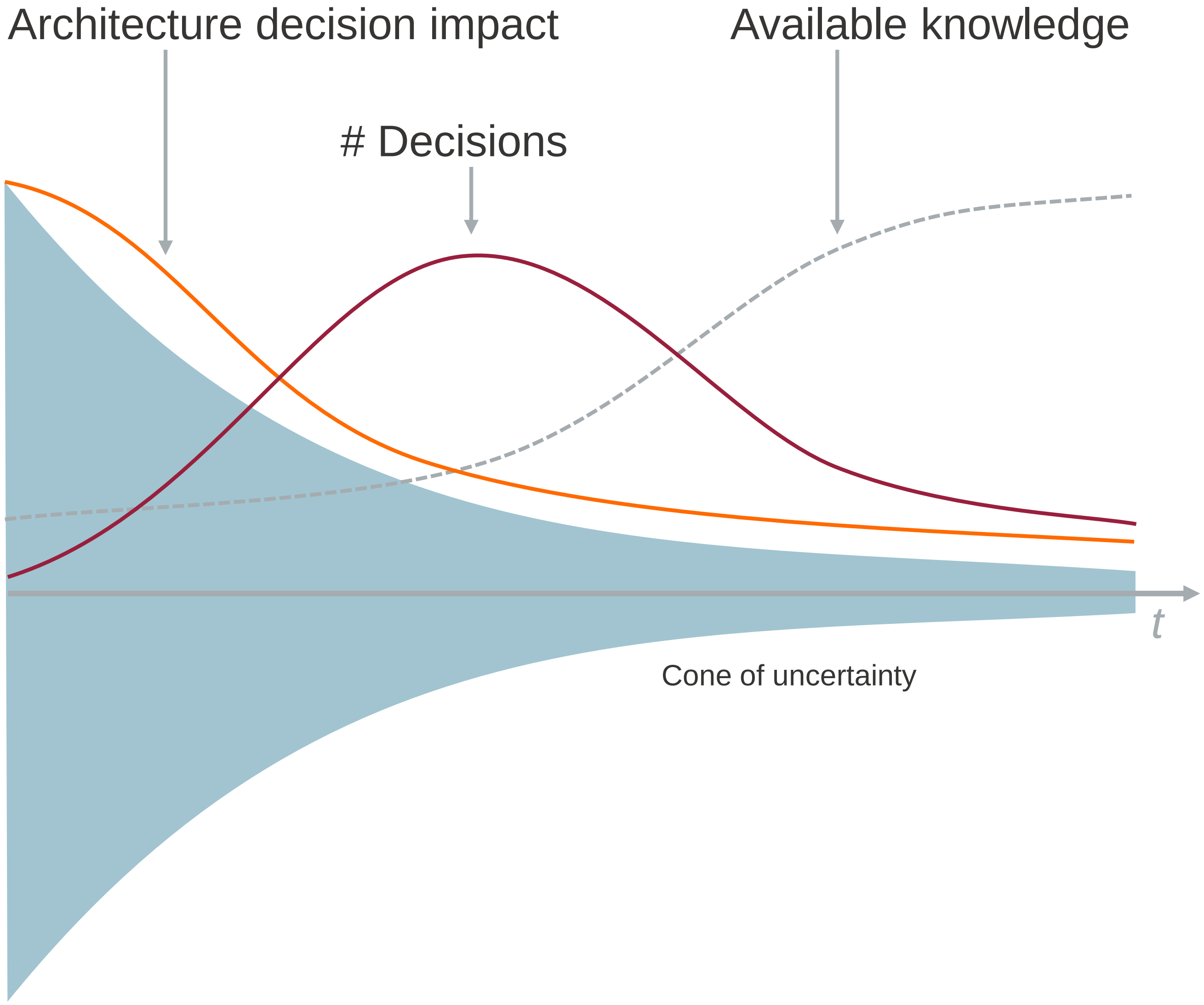
Source: Steve McConnell, Barry Boehm

Architectural Decisions
throughout the life cycle

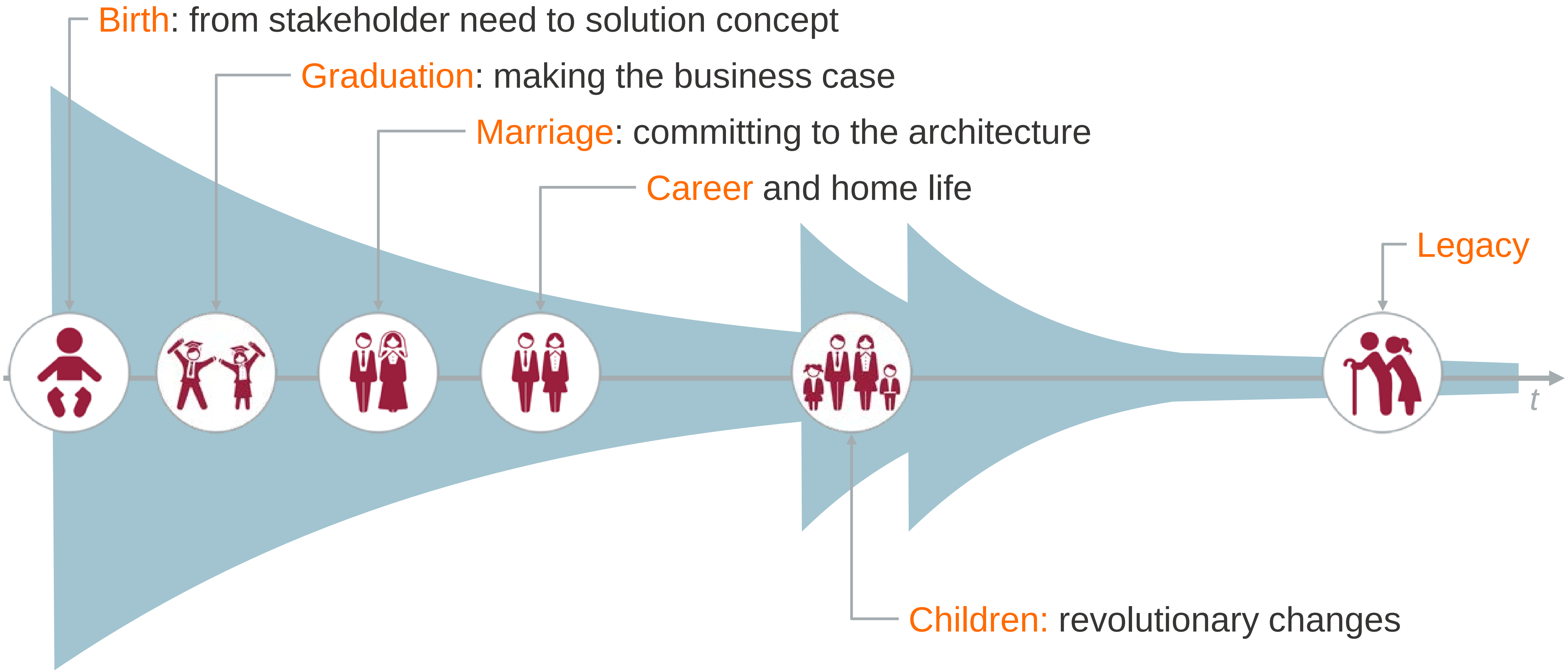
All architectural decisions are based on incomplete information…

…and the highest impact decisions are taken while the least factual knowledge is available.

Source: Philippe Kruchten



The life and times of an architecture



Birth: from stakeholder need to solution concept

Graduation: making the business case

Marriage: committing to the architecture

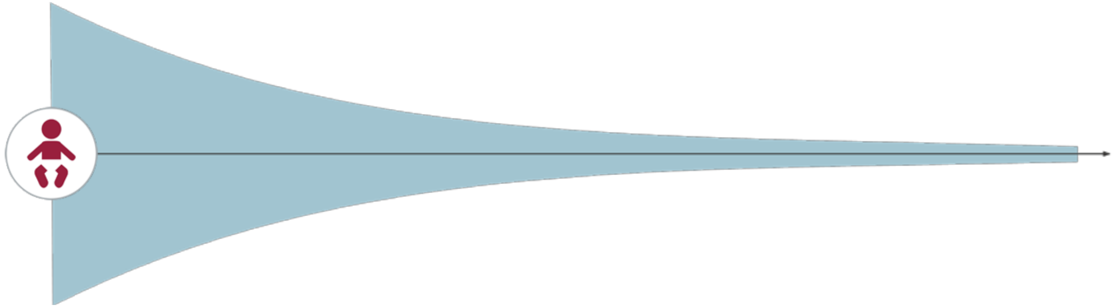
Career and home life

Legacy

Children: revolutionary changes

Birth

Solution concept



Typical concerns

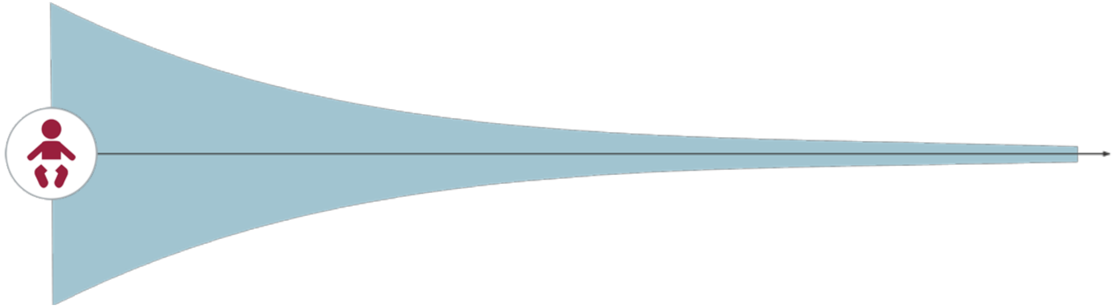
- What do the stakeholders need?
- How can we give it to them?

Typical output

- Business goals
- Scope
- First solution concept

Birth

Solution concept



Typical concerns

- What do the stakeholders need?
- How can we give it to them?

Traditional

- Project Brief
- Inception
- Solution Outline

Typical output

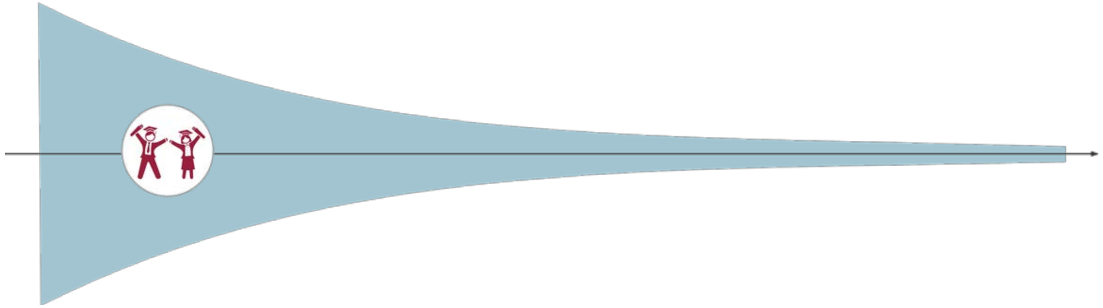
- Business goals
- Scope
- First solution concept

Agile

- Capture
- Funnel

Graduation

Business case

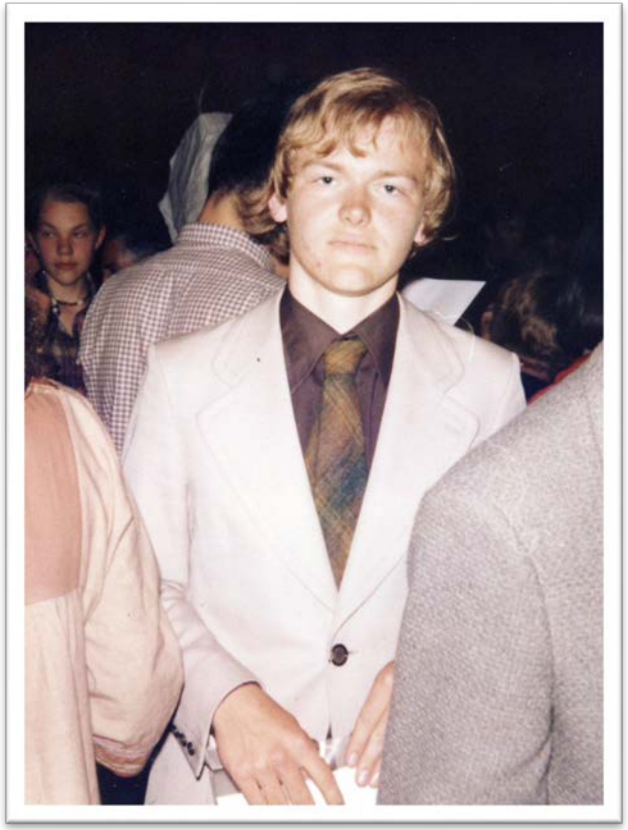


Typical concerns

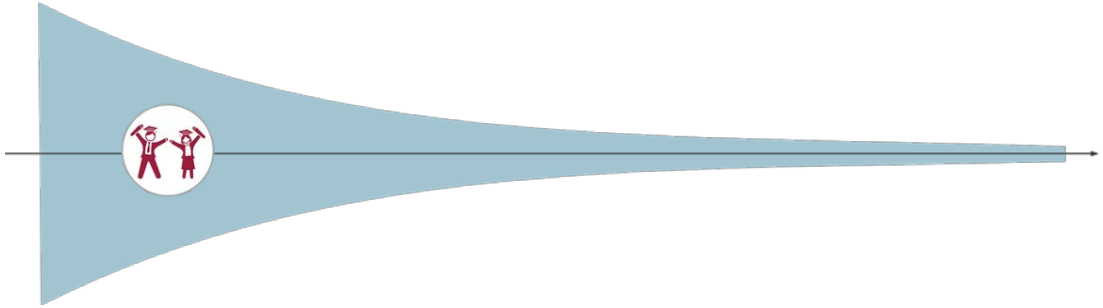
- Is it worth doing this?
- Are we confident risks and costs are under control?

Typical output

- Business case
- Solution design
- Delivery concept



Graduation
Business case



Typical concerns

- Is it worth doing this?
- Are we confident risks and costs are under control?

Traditional

- Project Initiation Documentation
- Elaboration
- Architectural Design

Typical output

- Business case
- Solution design
- Delivery concept

Agile

- Value statement, ROI
- Refine understanding
- Analysis

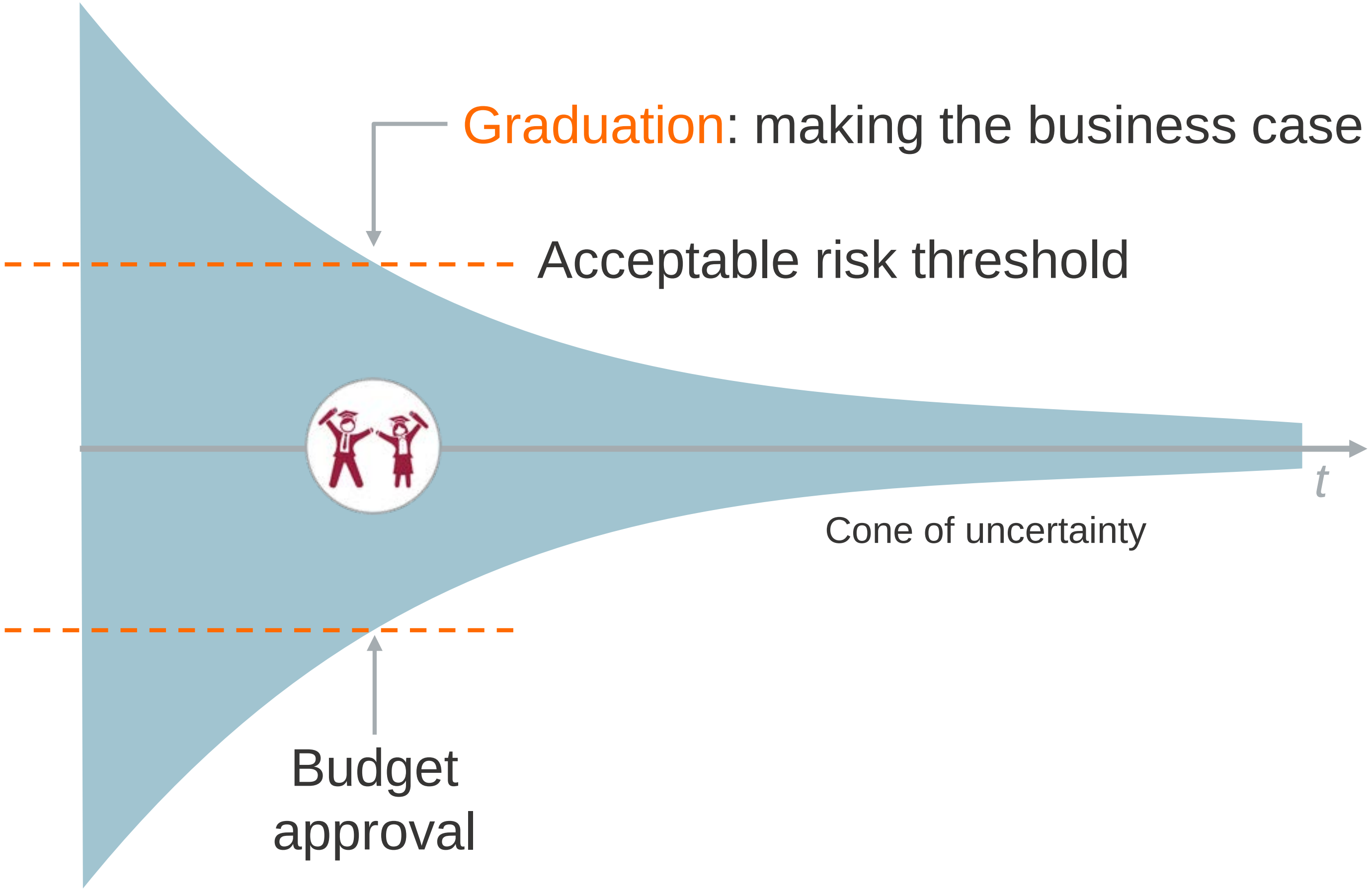
CGI

Graduation

- Stakeholder(s) funding solution (project/epic) agree that uncertainty in risk and cost is below a threshold
- Architecture is (part of) evidence that threshold is reached

Tip: document this evidence in architectural views that address uncertainty-related concerns:

- Operational View
- Delivery Breakdown View


Graduation: making the business case
Acceptable risk threshold
t
Cone of uncertainty
Budget approval

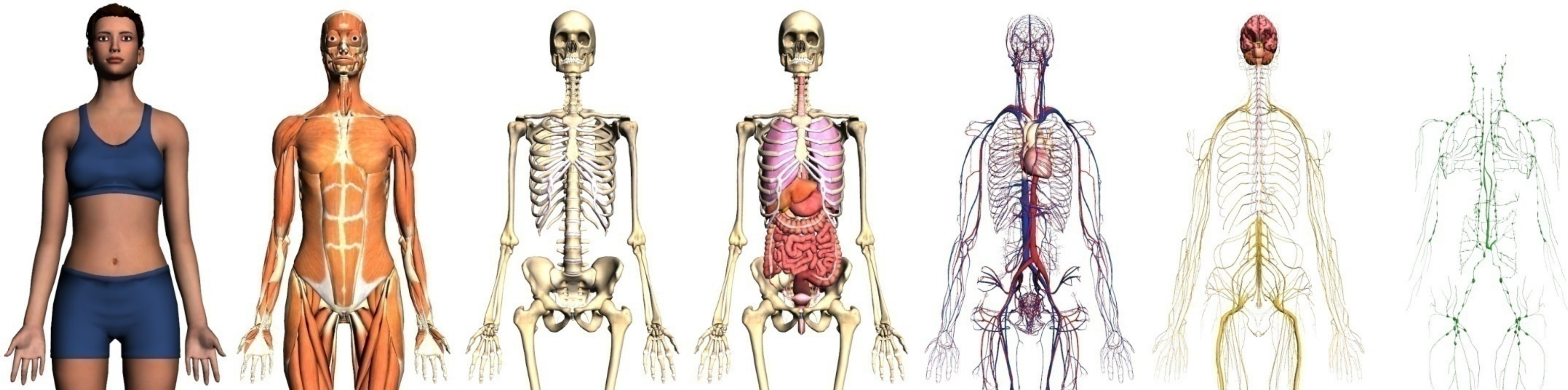
Graduation

Views and Viewpoints

All architecture documentation methods use views

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

- Viewpoints address concerns per stakeholder (group)
- Budget go/no-go concerns are common across traditional and agile contexts



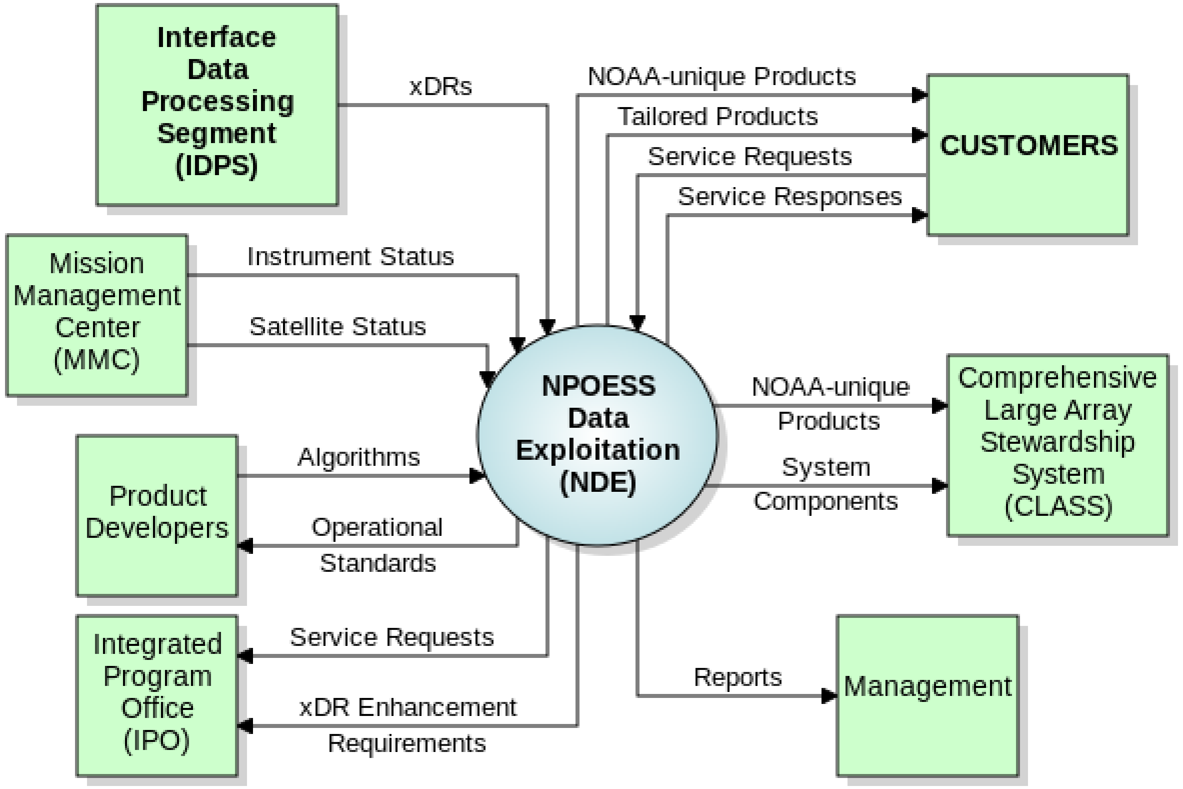
Graduation Views

Operational View: Context Diagram

Context Diagram:

Solution in its operational environment

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



Source: Wikimedia

Graduation Views

Operational View: Operational Decomposition

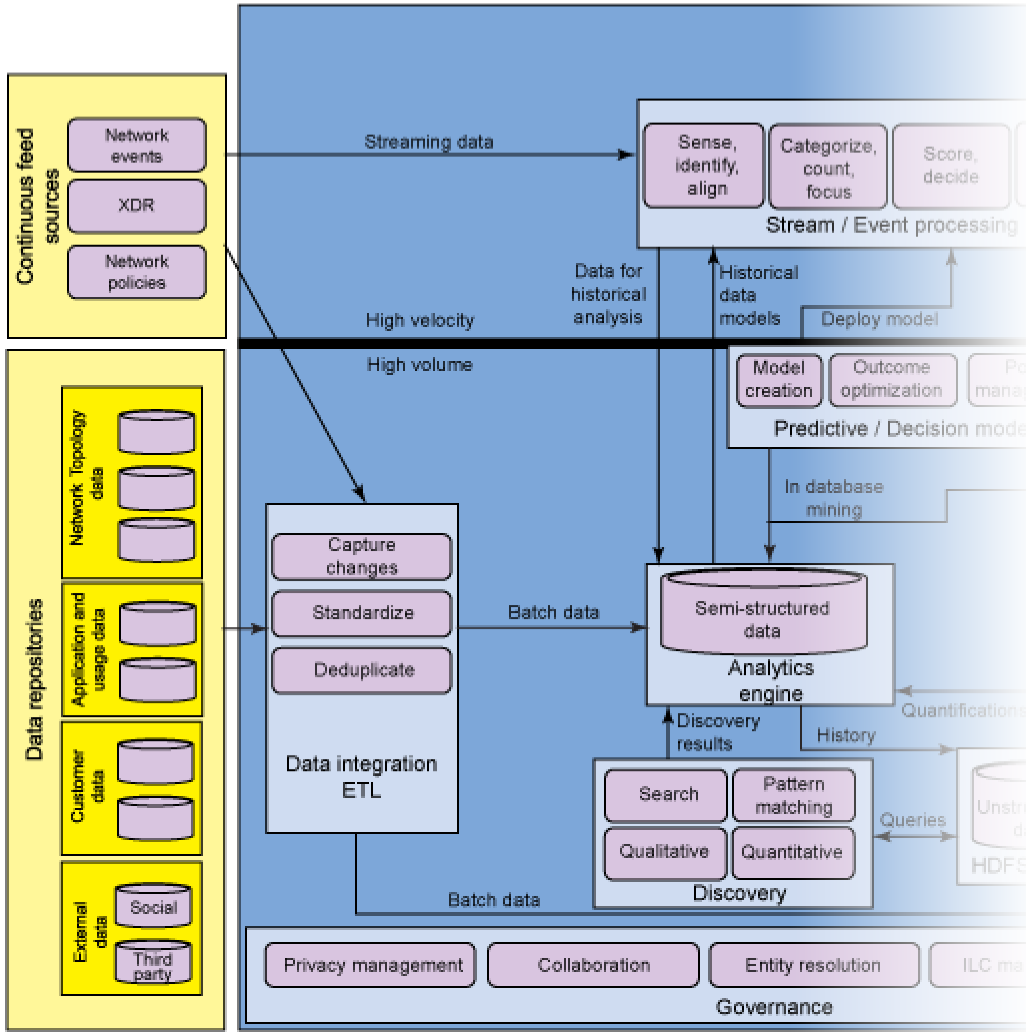
Operational Decomposition:

Components interact in running solution

- “How do run-time elements depend on each other?”
- “How does information flow through the solution?”

Transient Solutions:

- “How is the system operated now?” → CMO, As-is
- “How will it run after transformation?” → FMO, To-be



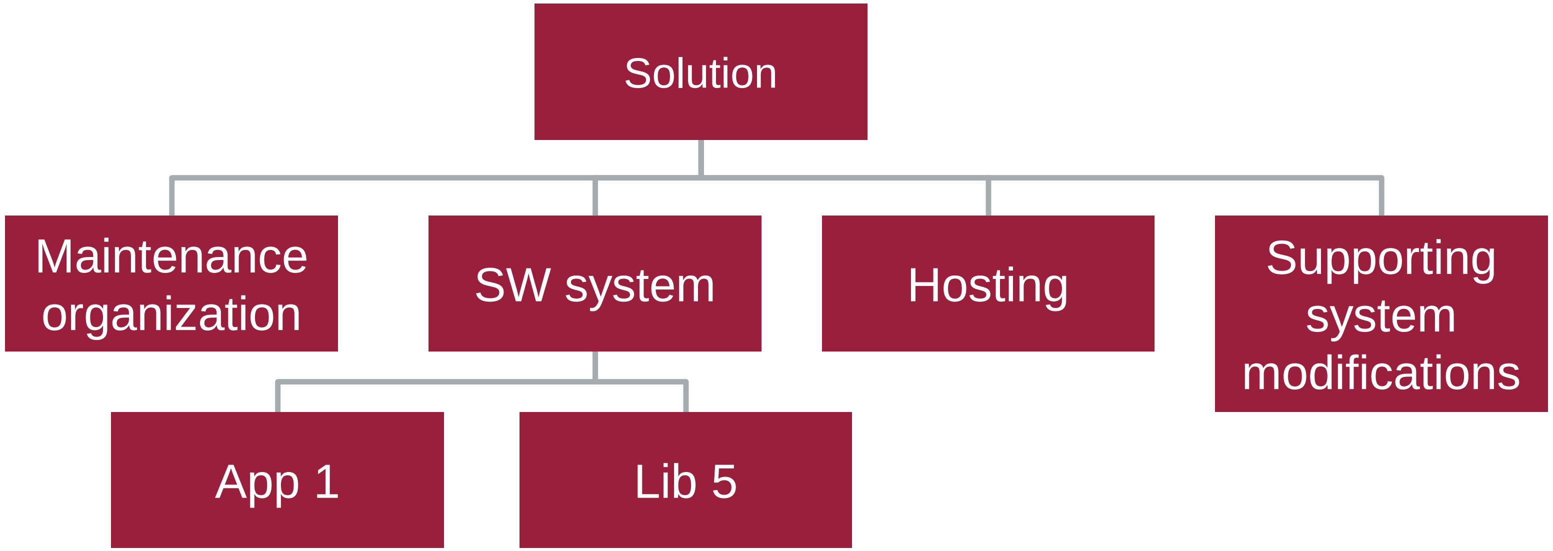
Source: ibm.com

Graduation Views

Solution Breakdown Structure

“How can the solution be decomposed to manage delivery?”

Basis for organizational allocation, solution costing, project planning / story decomposition



Solution

Maintenance organization

SW system

Hosting

Supporting system modifications

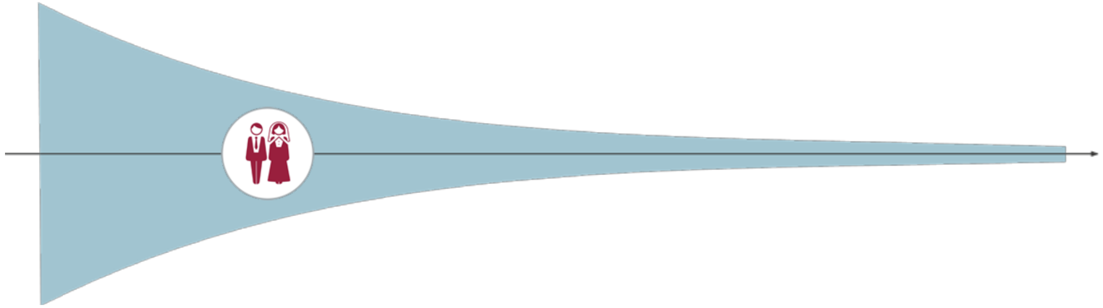
App 1

Lib 5

Solution Breakdown Structure

Marriage

Committing to the architecture



Typical concerns

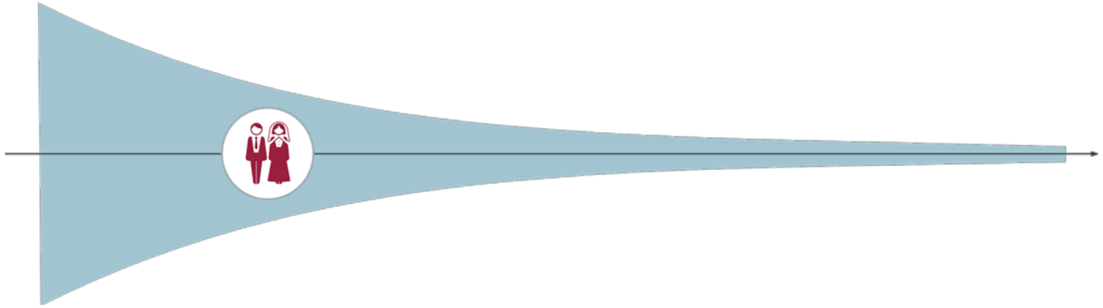
- How confident are we about this architecture?
- What are the alternatives?

Typical output

- Comparative / trade-off analysis
- Elaborated solution design
- Design review

Marriage

Committing to the architecture



Typical concerns

- How confident are we about this architecture?
- What are the alternatives?

Traditional

- Project Initiation Documentation
- Elaboration
- Critical Design Review

Typical output

- Comparative / trade-off analysis
- Elaborated solution design
- Design review

Agile

- Refine understanding
- Review, evaluation
- Evaluation

Marriage

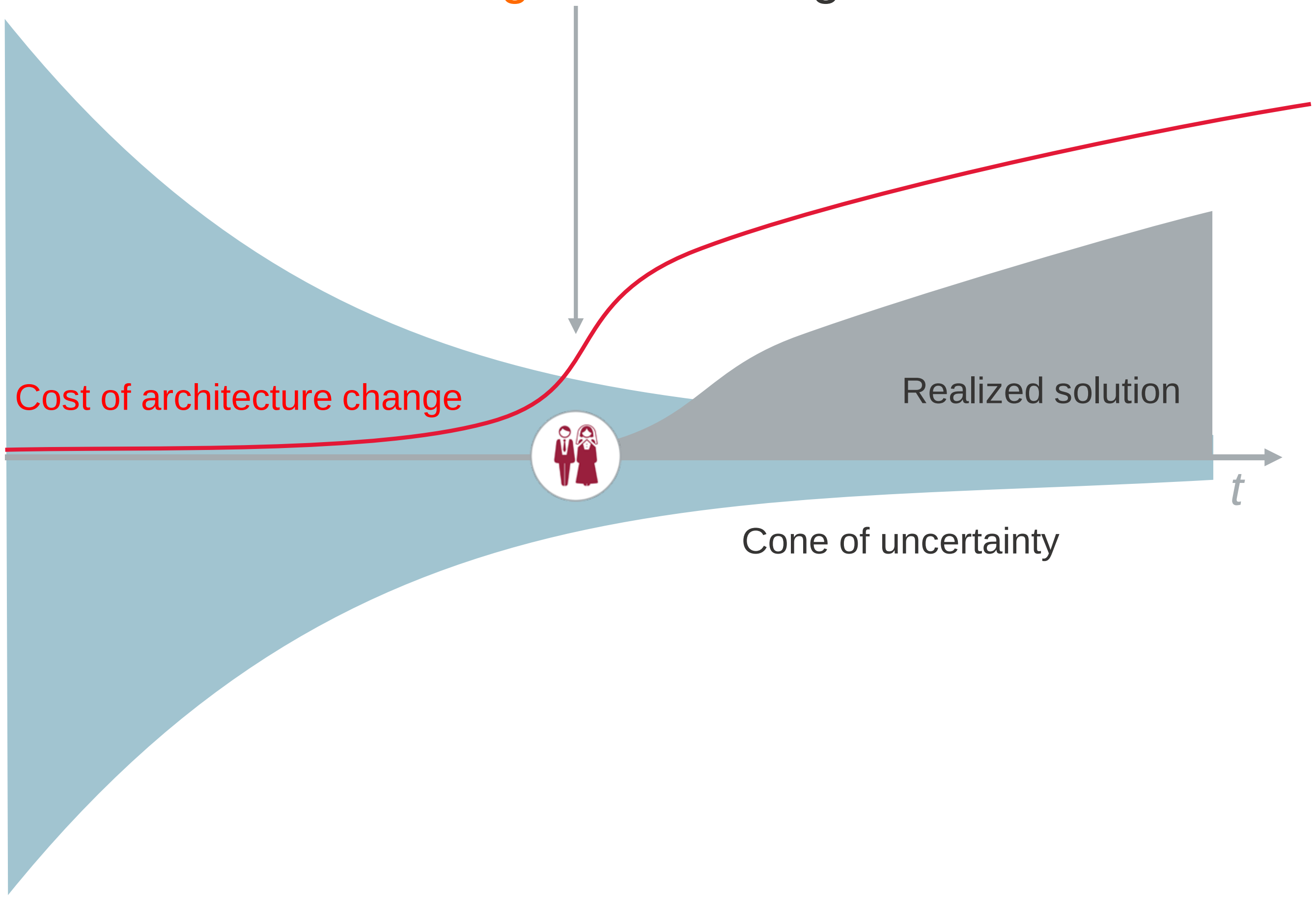
Cost of change to architecture quickly grows after a point in time, e.g.:

- Size of realization → refactoring
- Divestments in training, development environment, third party products/commitments

Tip: organize independent architecture assessment before crossing this line

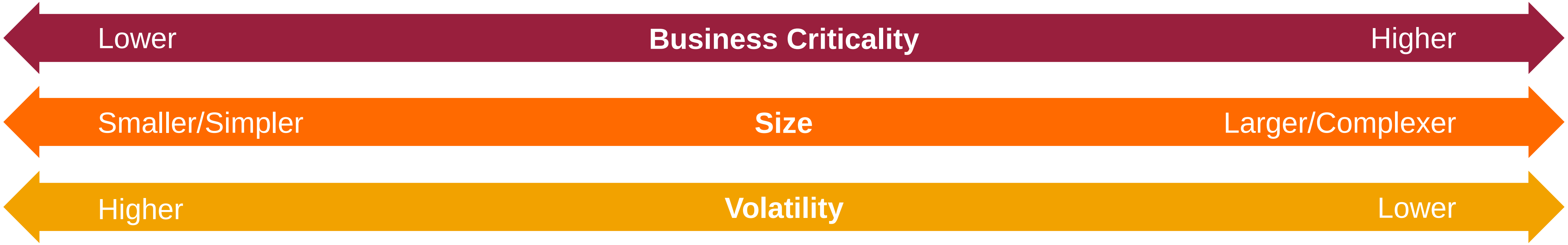
- Cf. Kahneman “Outside View”
- E.g. ATAM

Marriage: committing to the architecture



Marriage preparation

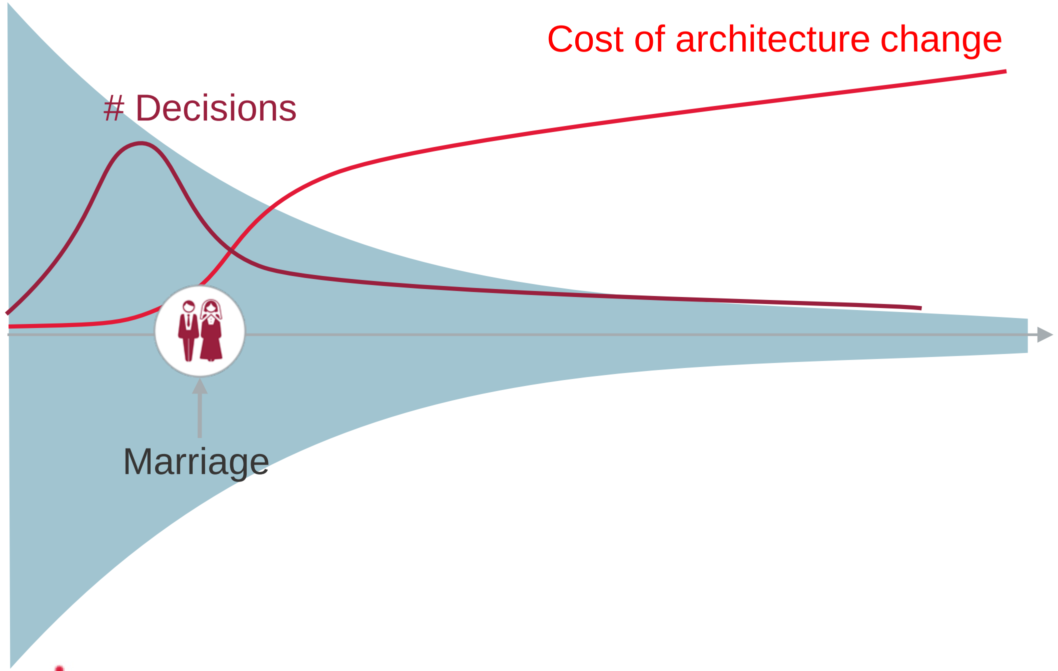
How many architectural decisions before commitment?

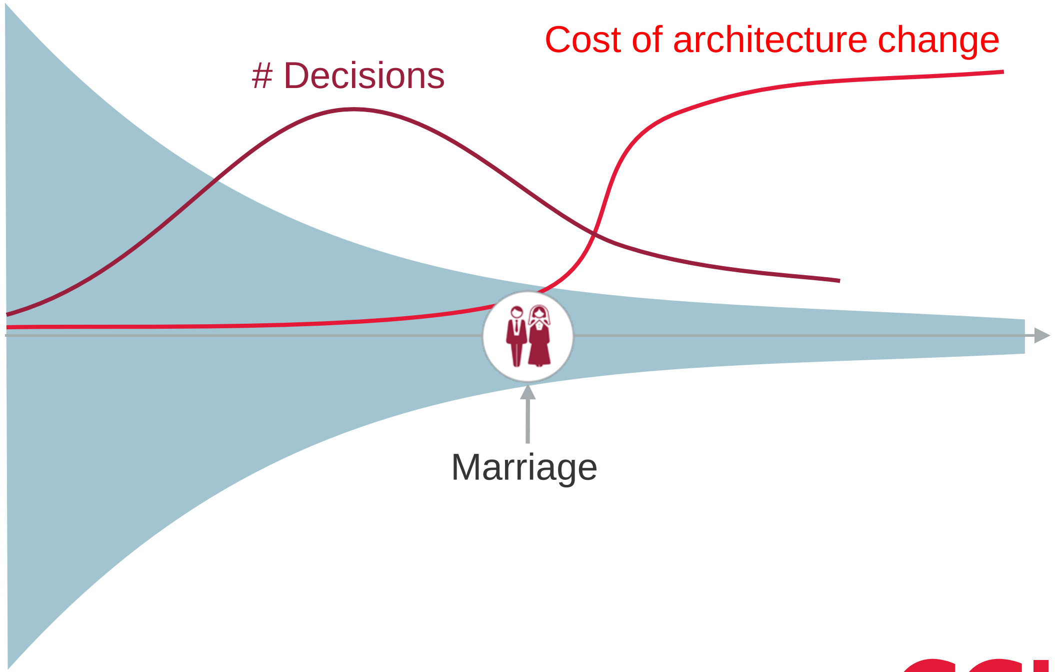


Lower | **Business Criticality** | Higher

Smaller/Simpler | **Size** | Larger/Complexer

Higher | **Volatility** | Lower

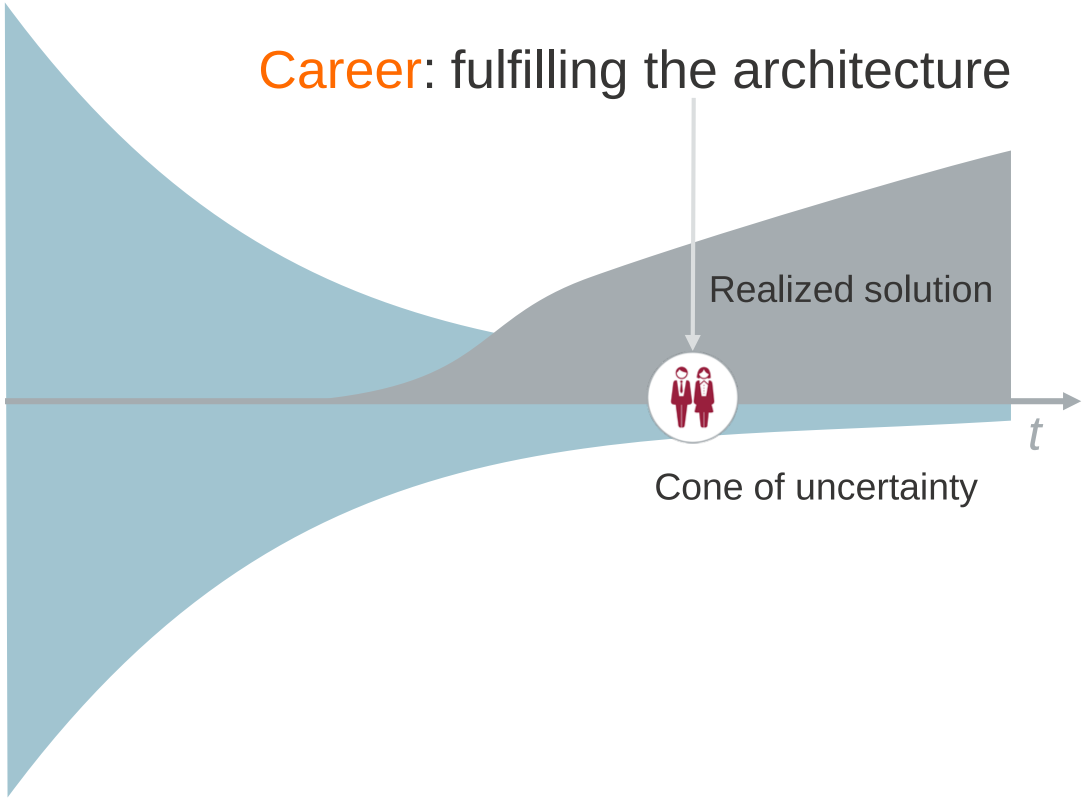




Source: Barry Boehm

Career and Home Life

Fulfilling the architecture



Career and Home Life

Fulfilling the architecture

Completeness

Backlog/ Project plan

- Ensure realization of *all* elements
- Based on Delivery Breakdown View

Testing

Def.of Done/ Project plan

- Verify NFRs and other architectural requirements

Coherence

Team organization

- Architect involved in integration
- Minimize inter-team dependencies

Communication

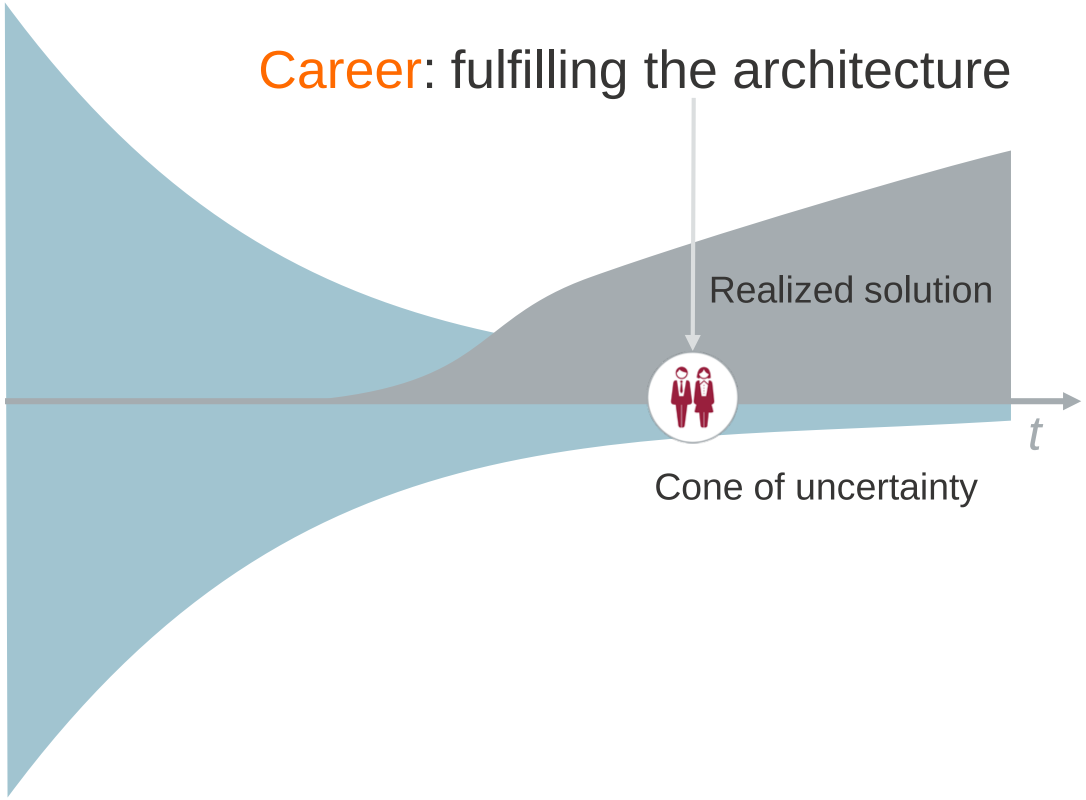
Team organization

- Architecture views, wallpaper, telling, training

Risk Management

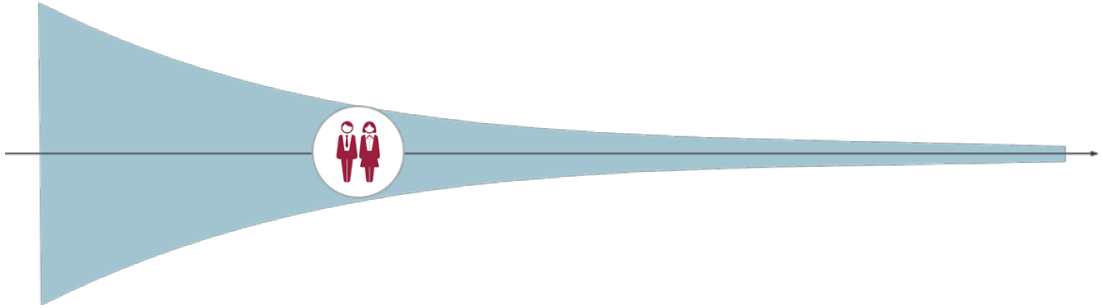
Risk log
Concern register

- Identify & manage (new) architectural concerns



Career and home life

Health and Debt Control



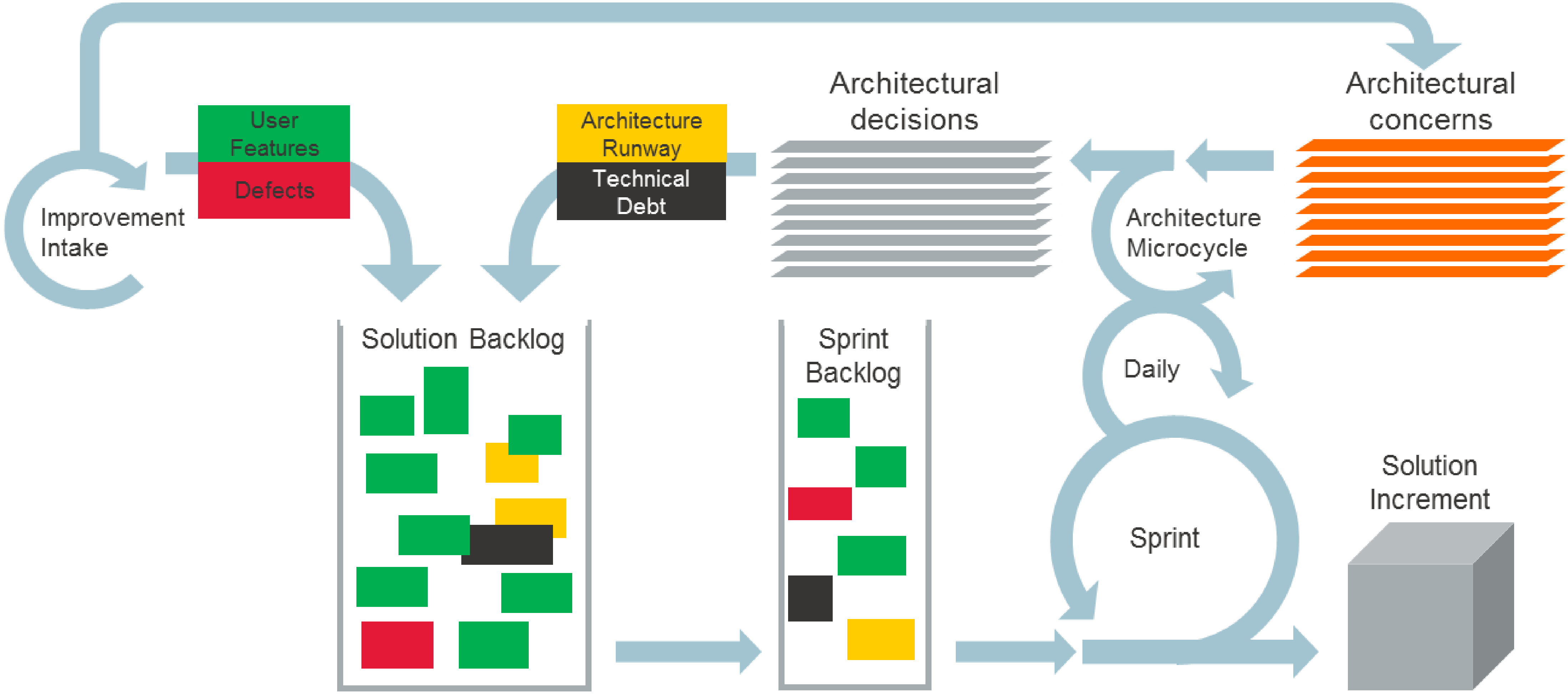
	Visible	Invisible
Positive Value	New features Added functionality	Architectural, Structural features
Negative Value	Defects	Technical Debt

Source: Philippe Kruchten

What’s in your backlog?

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

Health and Debt Control using SCRUM & RCDA



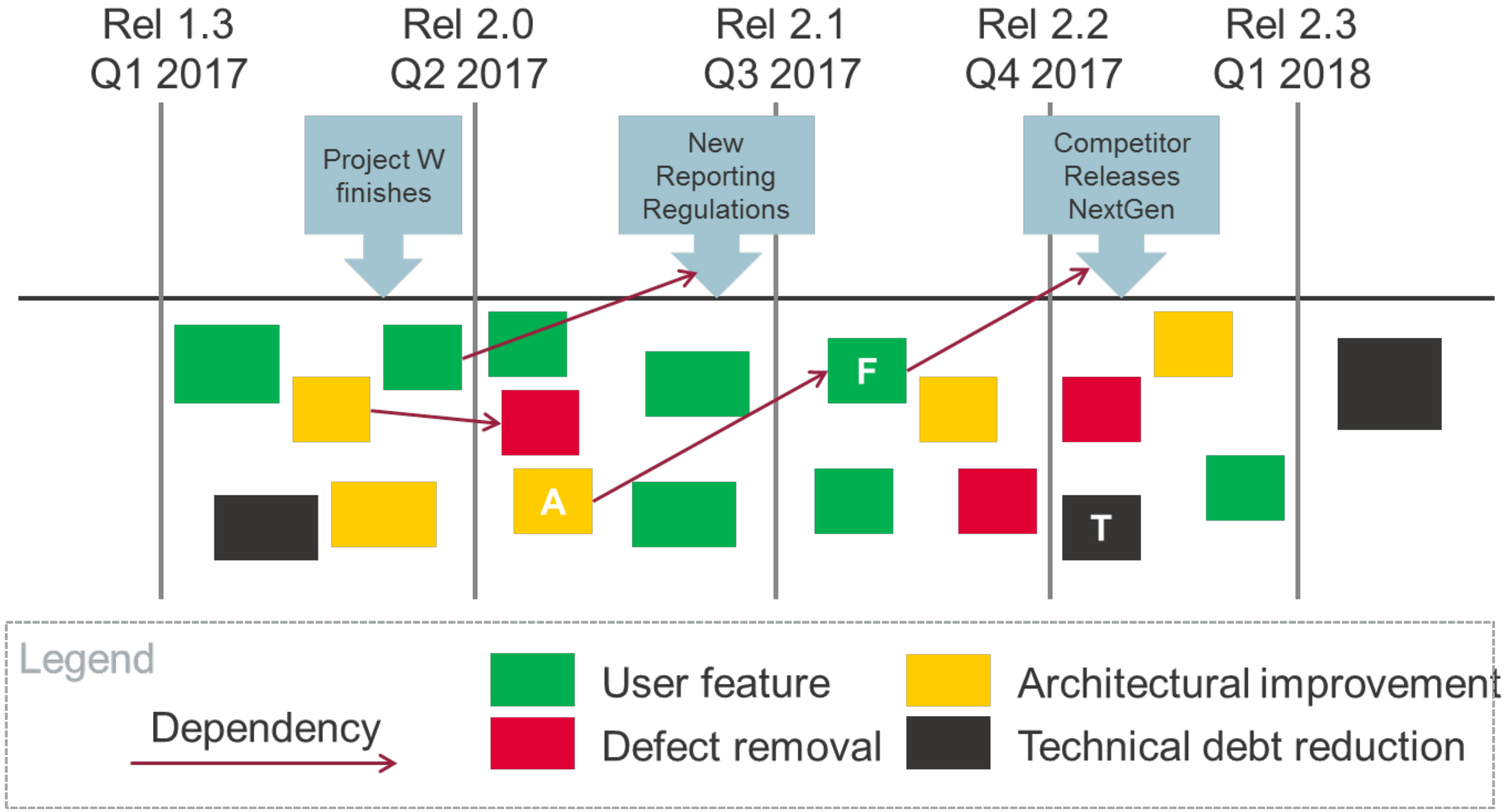
Health and Debt Control

Architecture Roadmapping with Just Enough Anticipation

Just Enough Anticipation achieved by:

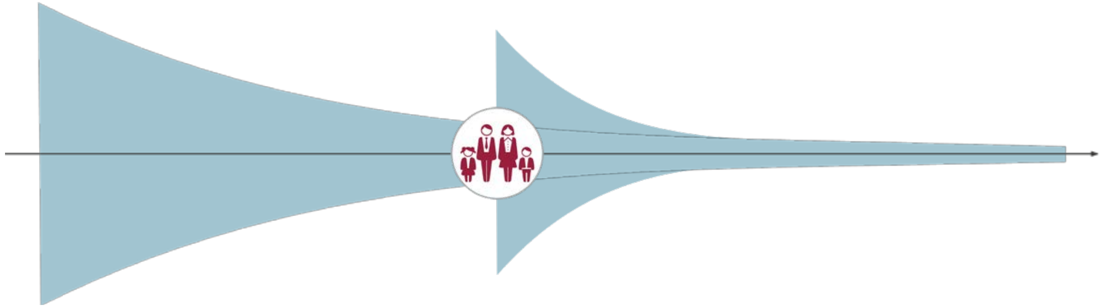
- Dependency Analysis
- Technical Debt Control
- Economic Reasoning

Source: Nanette Brown, Rod Nord, Ipek Ozkaya



Children

Revolutionary Changes



Typical concerns

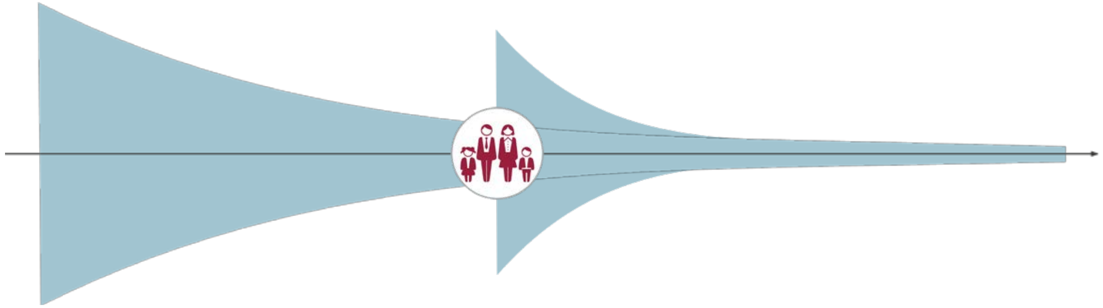
- What has changed since the previous generation?
- Prevent damage to existing status quo

Typical output

- Business goals
- Scope
- First solution concept

Children

Revolutionary Changes



Typical concerns

- What has changed since the previous generation?
- Prevent damage to existing status quo

Traditional

- Project Brief
- Inception
- Solution Outline

Typical output

- Business goals
- Scope
- First solution concept

Agile

- Capture
- Funnel

Our commitment to you

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

Spare slides follow

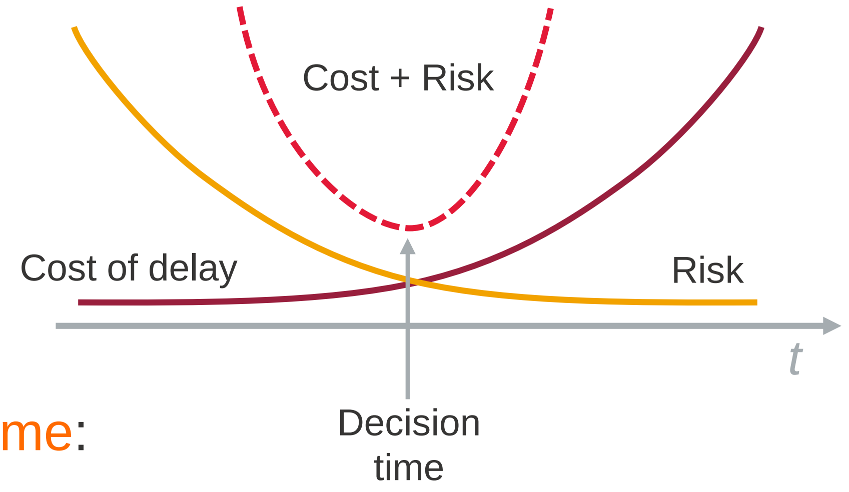
Experience the commitment®

Architectural Decision Making

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

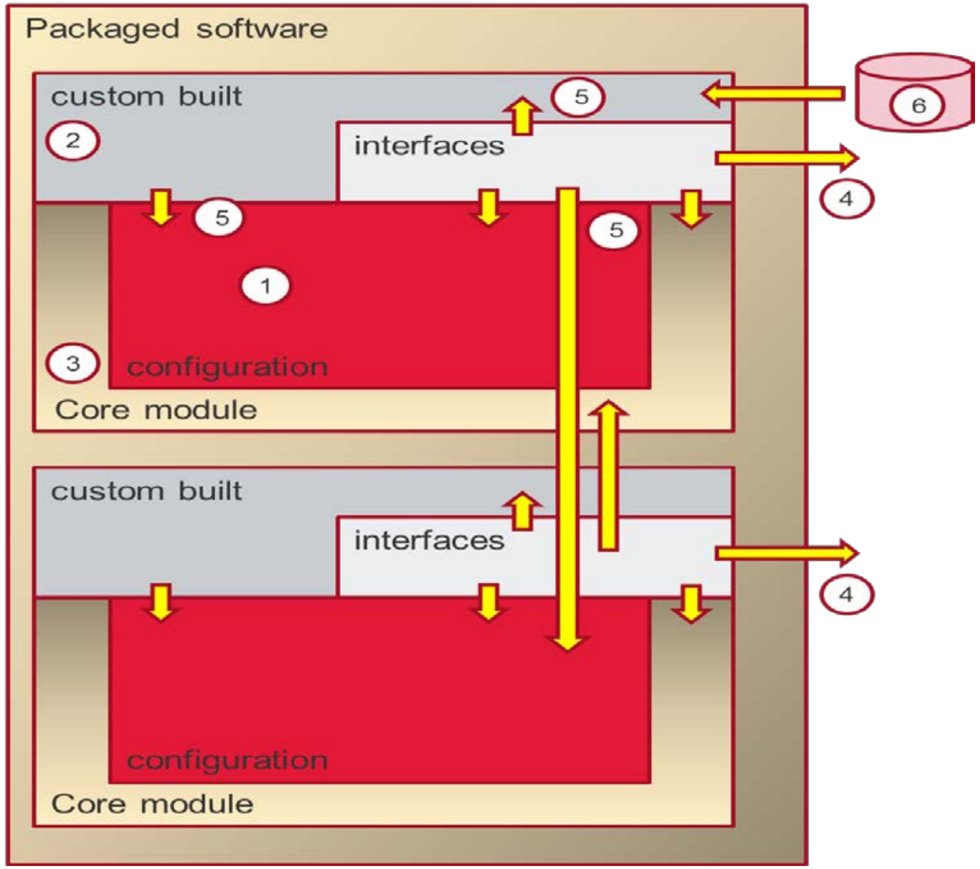
Key skills of Solution Architect:

- timing of architectural decisions
- making decisions based on incomplete information
- dealing with the resulting risks

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

Solution Costing

Example: Package Implementation



1. Configuration
2. Custom built functionality
3. Core module(s) / standard functionality
4. External interfaces
5. Internal interfaces
6. Data

Solution Costing

Example: Package Implementation Decisions

D1 – Choice of ERP Vendor: TBQ

D2 – Core modules selected: Finance & Order

D3 – Extend Order module with bespoke Order Routing functionality

D4 – Extend Finance module with bespoke Internal Clearing functionality

D5 – Build custom functionality using Mill platform

D6 – Payments will be handled by partner PayUp

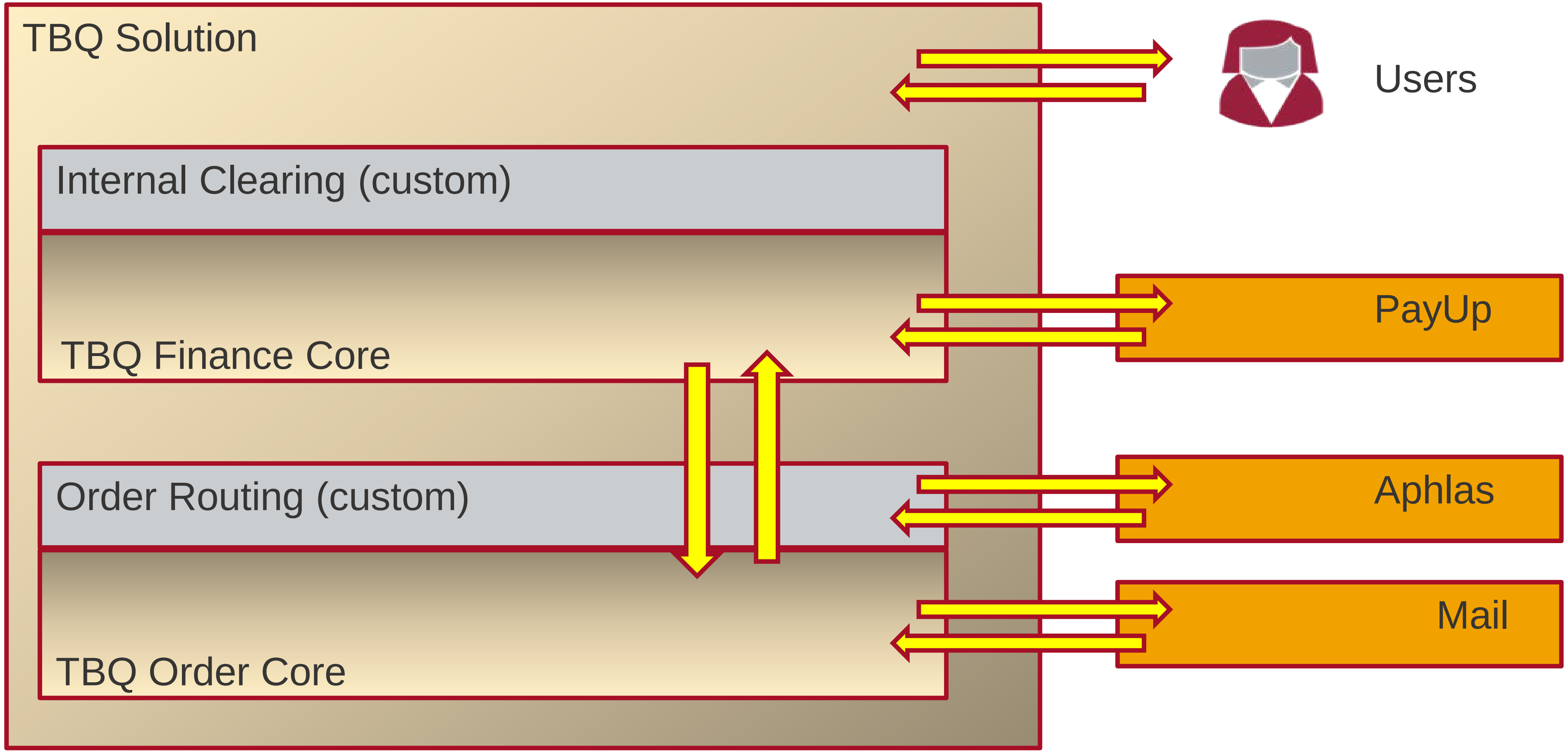
D7 – Order related e-mails sent through existing mail distribution server

D8 – Order Routing module will interface with Fleet mgt partner Aphlas

D9 – Data to be converted: Ledger, Catalog, Stock (no open orders)

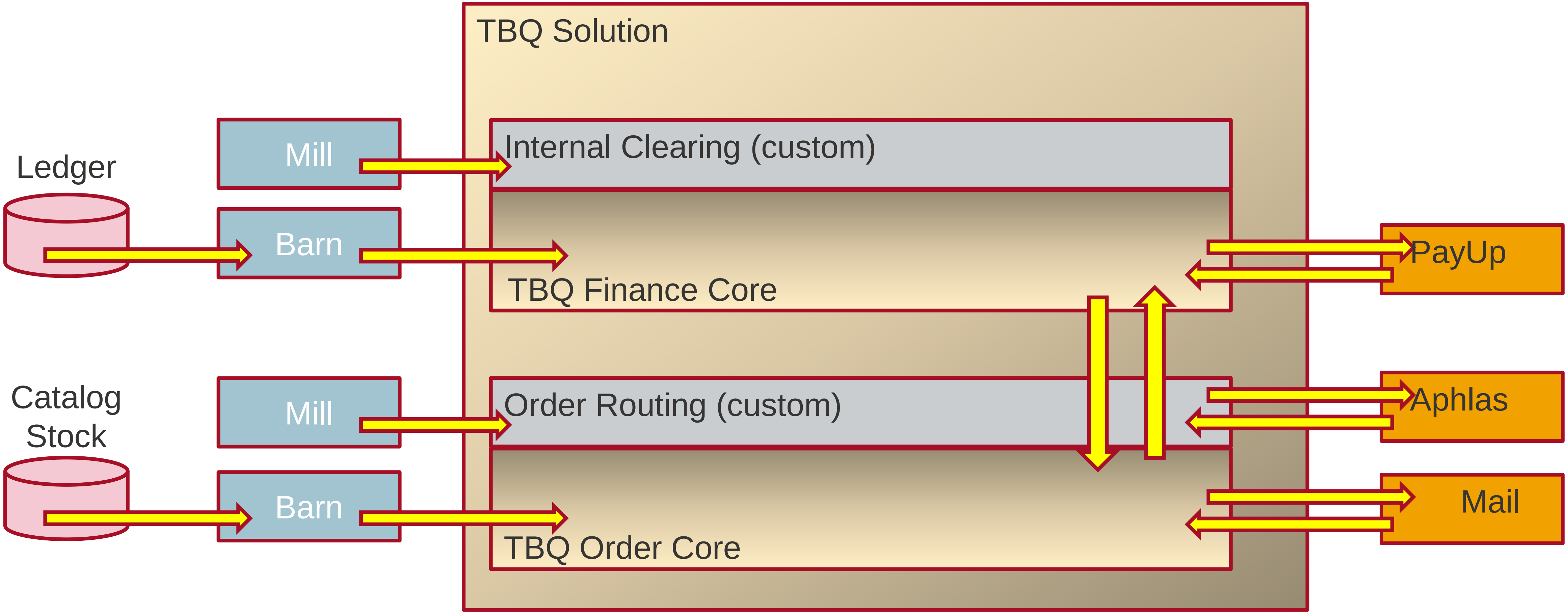
D10 – Selected ETL tool: Barn

Solution Costing
Example: Operational View



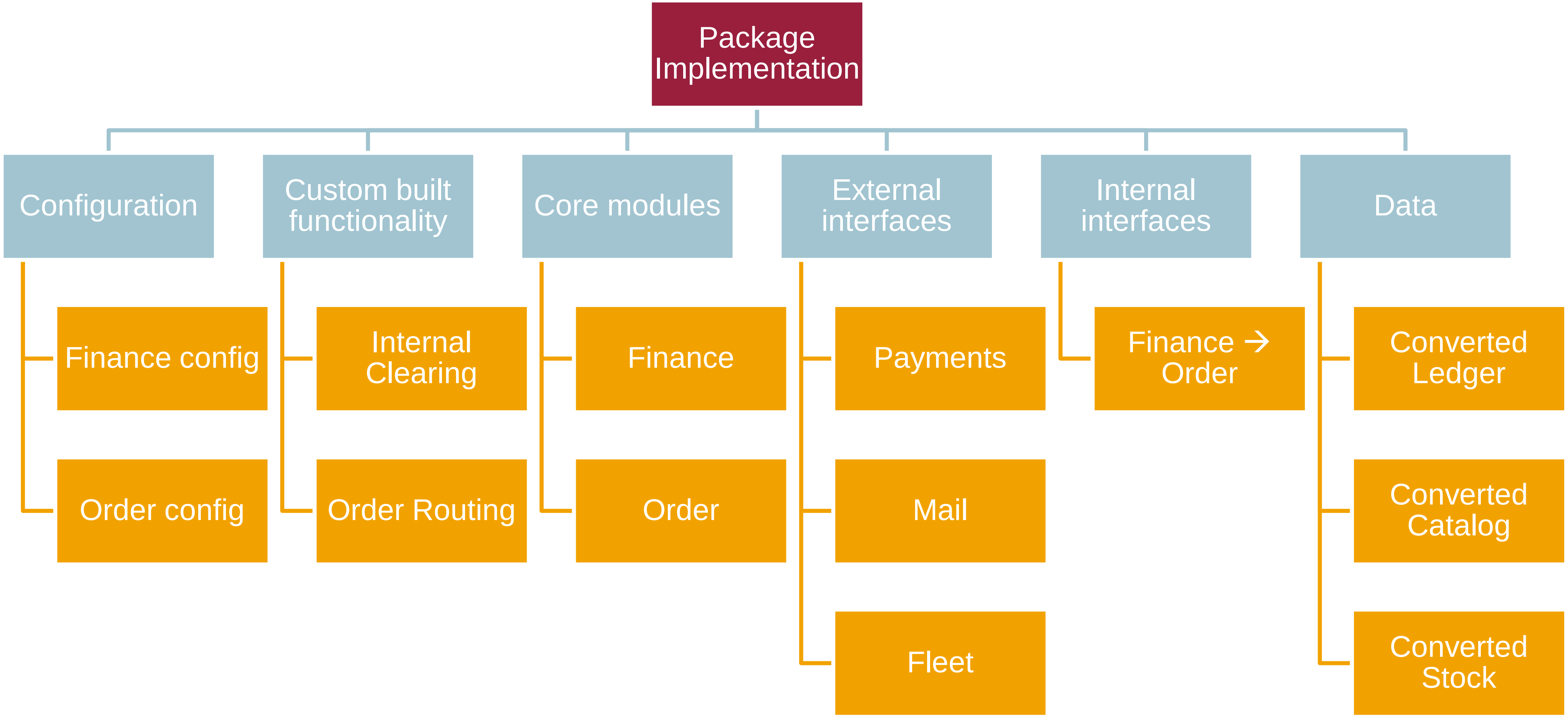
Solution Costing

Example: Construction View



Solution Costing

Example: Package Implementation SBS



Package Implementation

- Configuration
  - Finance config
  - Order config
- Custom built functionality
  - Internal Clearing
  - Order Routing
- Core modules
  - Finance
  - Order
- External interfaces
  - Payments
  - Mail
  - Fleet
- Internal interfaces
  - Finance → Order
- Data
  - Converted Ledger
  - Converted Catalog
  - Converted Stock

Solution Costing

Example: Package Implementation Cost Drivers

Deliverable Elements	Typical cost drivers (realization)	Parameter examples
Configuration	Solution complexity	#Config parameters
	Organization complexity	#Stakeholder workshops
Custom built functionality	Functional size	#Use cases, function pts
	Implementation technology	API calls Programming language
Core module(s) / standard functionality	Vendor IP pricing	Modules, options #Users, #Transactions…
External interfaces	Interface complexity	I/F protocol, technology Non-functional reqs
	Commercial availability	Vendor pricing
Internal interfaces	Interface complexity	I/F protocol, technology Non-functional reqs
Data	Data size	#TB, #tables, #records
	Data quality	Pollution, redundancy
	Data compatibility	(ETL) tooling availability