Antonio Martini

Professor of Software Engineering

University of Oslo

Course IN5140
2023-10-11

LARGE-SCALE AGILE AND ARCHITECTURE

Who is Antonio Martini?

- Italian
  - No kebab pizza! ☺
  - 12 years in Scandinavia – survived many winters!

- Previously
  - Worked as a Software Developer
  - PhD in Software Engineering (Chalmers)
  - Postdoc (Chalmers)
  - Independent Consultant
  - Advisory board on a startup (Skuld.ai)
  - My own startup (AnaConDebt)
  - Principal, Strategic Researcher at CA Technologies

- Currently:
  - Professor at UiO

- Hobbies
  - Board games, strategy computer games, pool, etc.
  - Football, volleyball, beach volley, medieval fencing
  - Piano, Drumset, etc.
  - Travel!
  - …and no time for them! ☺

Agenda

- Recap
  - What is software architecture?
  - How to think about architecture?
- Agile and Architecture
  - A complicated relationship
  - Current state of the art
- Agile Architecting
  - Process
  - Product
  - Organization
- Agile and architects
  - Industrial case study
- Summary

- We will use mentimeter during the lecture, participate!
  - Check the following symbol during the lecture

What is Software Architecture?

First question, let's try Menti

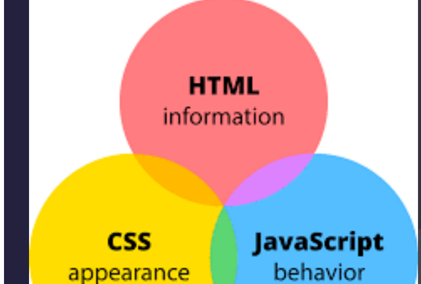
- What is software architecture?

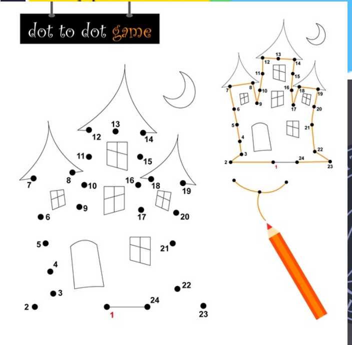
Software architecture is…

- All of the followings:
  - Overall system structure
  - The important stuff – whatever that is
  - Things that people perceive as hard to change
  - A set of architectural design decisions

Software Architecture characteristics

- Multitude of stakeholders
- Quality driven (tradeoff)
- Separation of concerns
- Recurring styles (patterns)
- Conceptual integrity (vision)





Why software architecture?

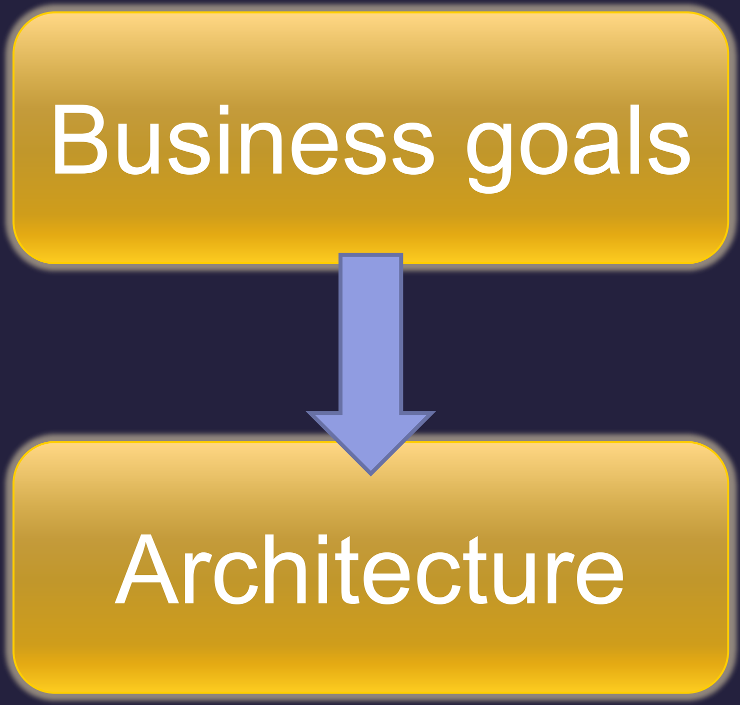
- To get a grasp of a complex system
- Facilitates the communication among the stakeholders about their needs
- Supports decisions about future development and maintenance
  - Reuse
  - Budget
- Analysis of the product before it's built
  - Cost reduction
  - Risk reduction

You can't ignore architecture

- All products HAVE an architecture
  - It can be bad
  - It can be good
- In all projects we SHOULD think about architecture
  - Maybe less in small projects
  - Maybe more in large projects
- Thinking about the architecture is a necessary (and smart) process

How to think about Architecture

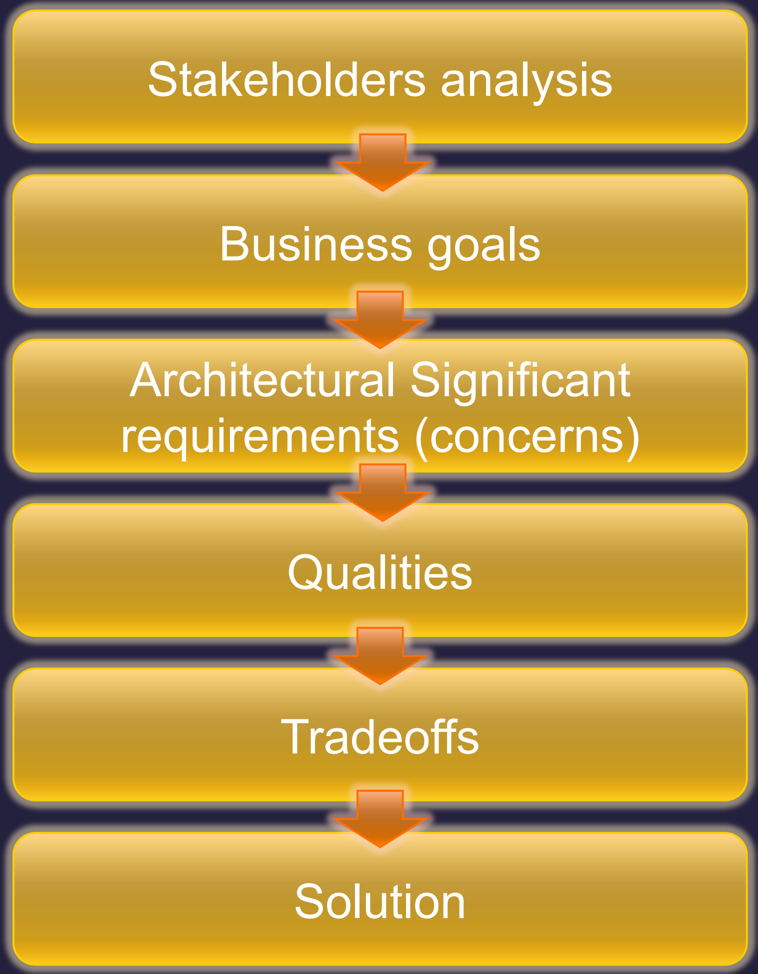
Business drives architecture



Business goals

Architecture

A process to think about architecture



Stakeholders analysis

Business goals

Architectural Significant requirements (concerns)

Qualities

Tradeoffs

Solution

Who?

What do they need?

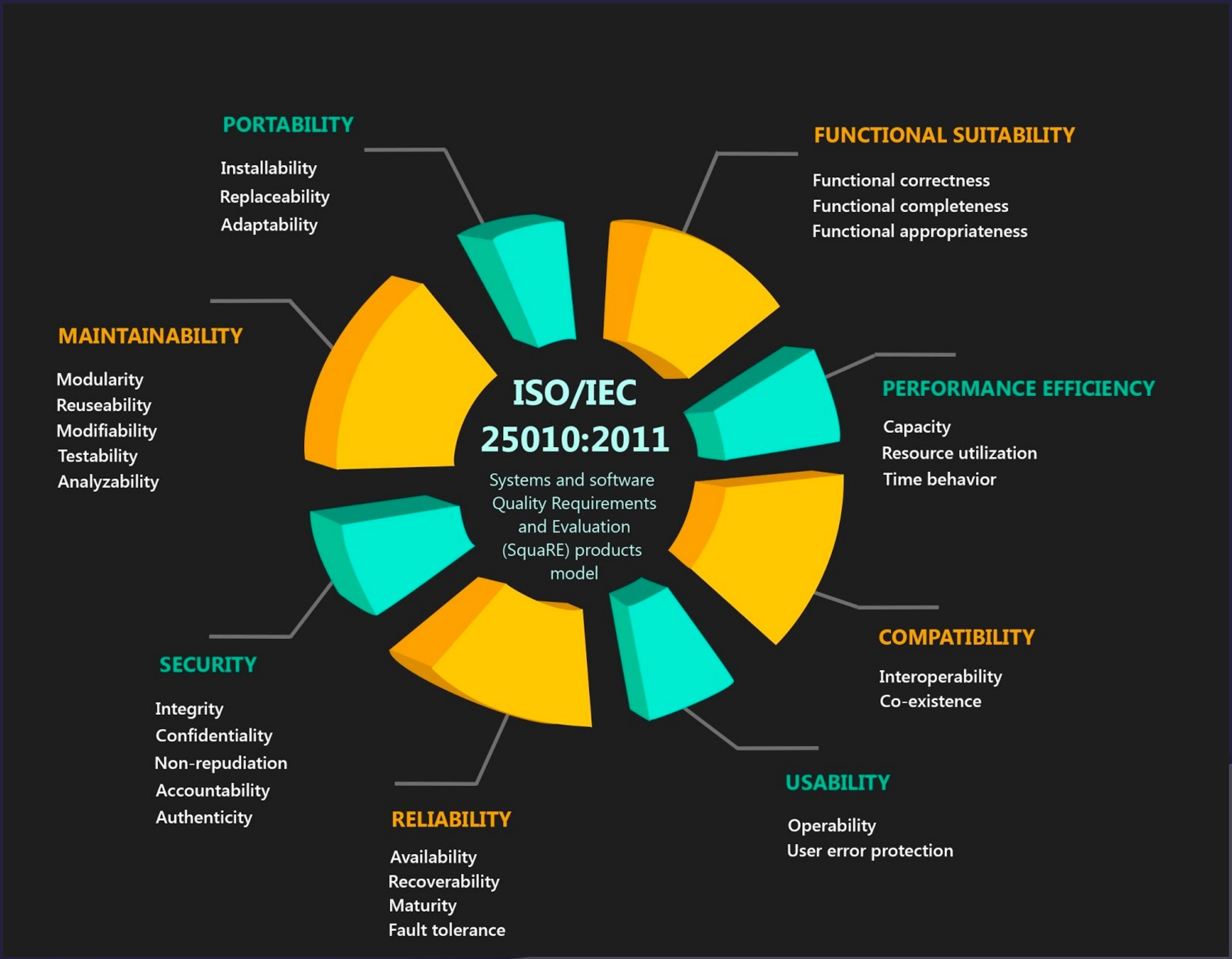
What should the system do?

What qualities are important?

What should we focus on?

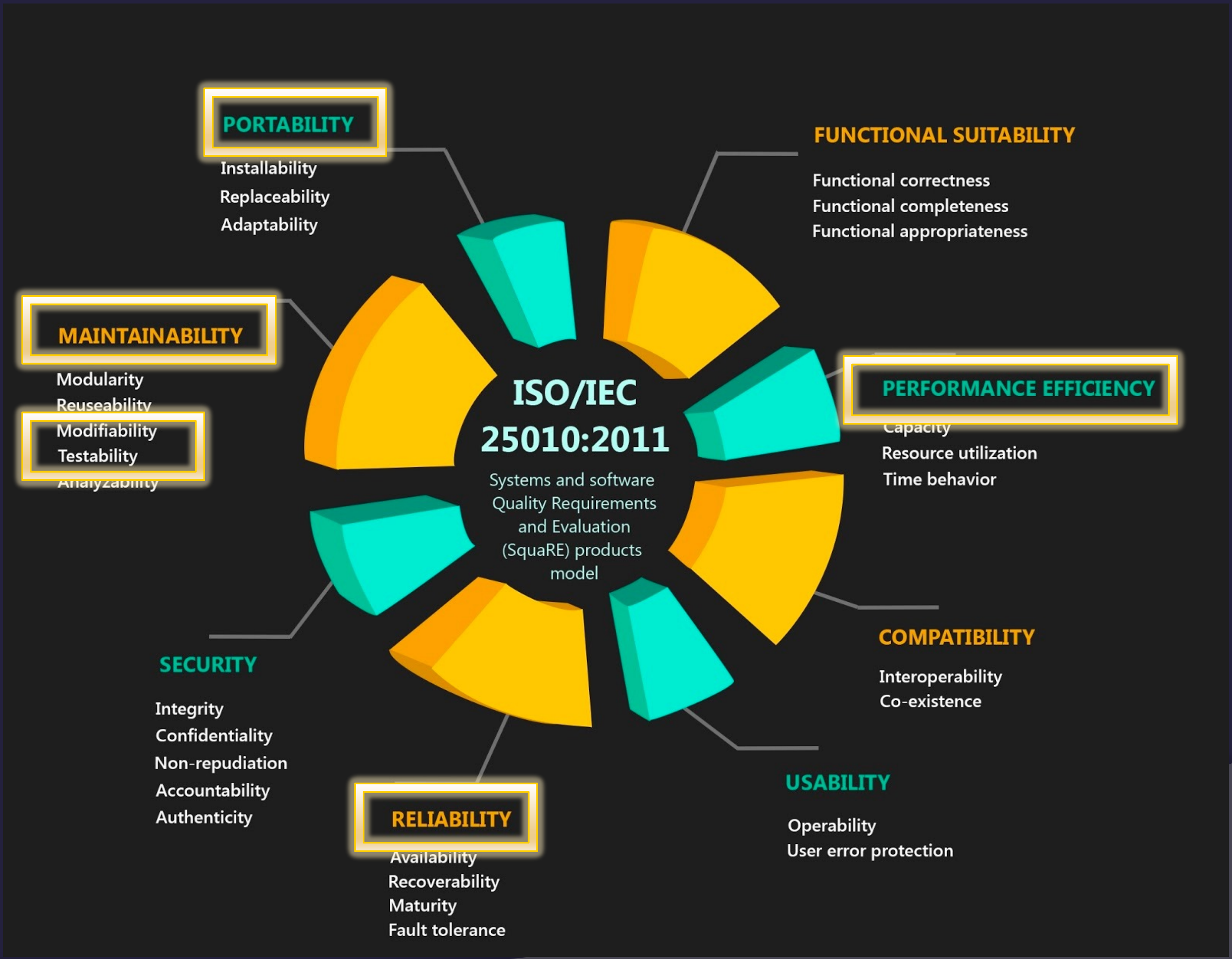
How should we implement it?

System Qualities



https://www.iso.org/standard/22749.html

System Qualities – All stakeholders



ISO/IEC 25010:2011

Systems and software Quality Requirements and Evaluation (SquaRE) products model

PORTABILITY
Installability
Replaceability
Adaptability

FUNCTIONAL SUITABILITY
Functional correctness
Functional completeness
Functional appropriateness

MAINTAINABILITY
Modularity
Reuseability
Modifiability
Testability
Analyzability

PERFORMANCE EFFICIENCY
Capacity
Resource utilization
Time behavior

COMPATIBILITY
Interoperability
Co-existence

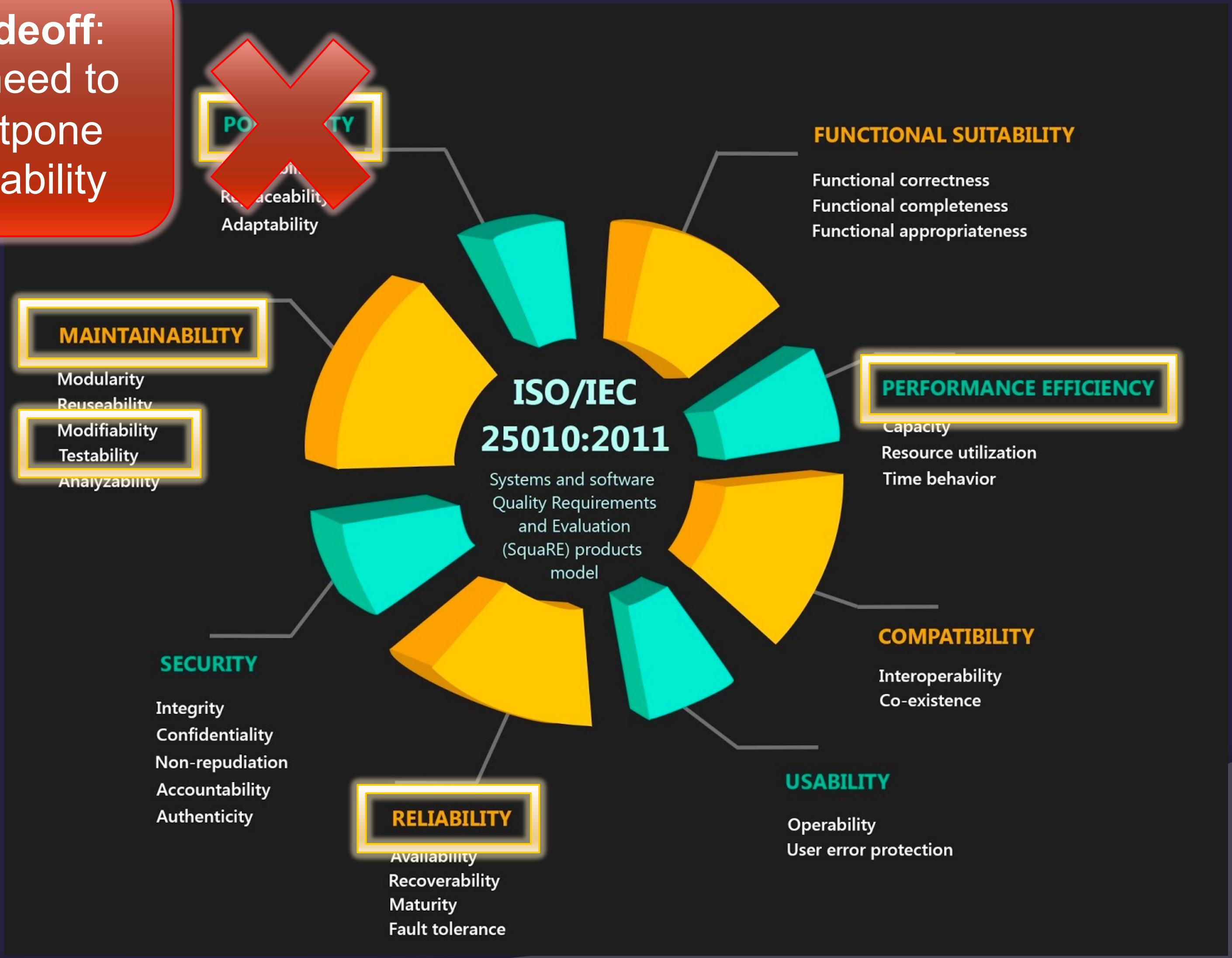
SECURITY
Integrity
Confidentiality
Non-repudiation
Accountability
Authenticity

USABILITY
Operability
User error protection

RELIABILITY
Availability
Recoverability
Maturity
Fault tolerance

System Qualities – Trade-off

Tradeoff: we need to postpone portability



ISO/IEC 25010:2011

Systems and software Quality Requirements and Evaluation (SquaRE) products model

PO TY

Re aceabilit

Adaptability

FUNCTIONAL SUITABILITY

Functional correctness
Functional completeness
Functional appropriateness

MAINTAINABILITY

Modularity
Reuseability
Modifiability
Testability
Analyzability

PERFORMANCE EFFICIENCY

Capacity
Resource utilization
Time behavior

COMPATIBILITY

Interoperability
Co-existence

SECURITY

Integrity
Confidentiality
Non-repudiation
Accountability
Authenticity

USABILITY

Operability
User error protection

RELIABILITY

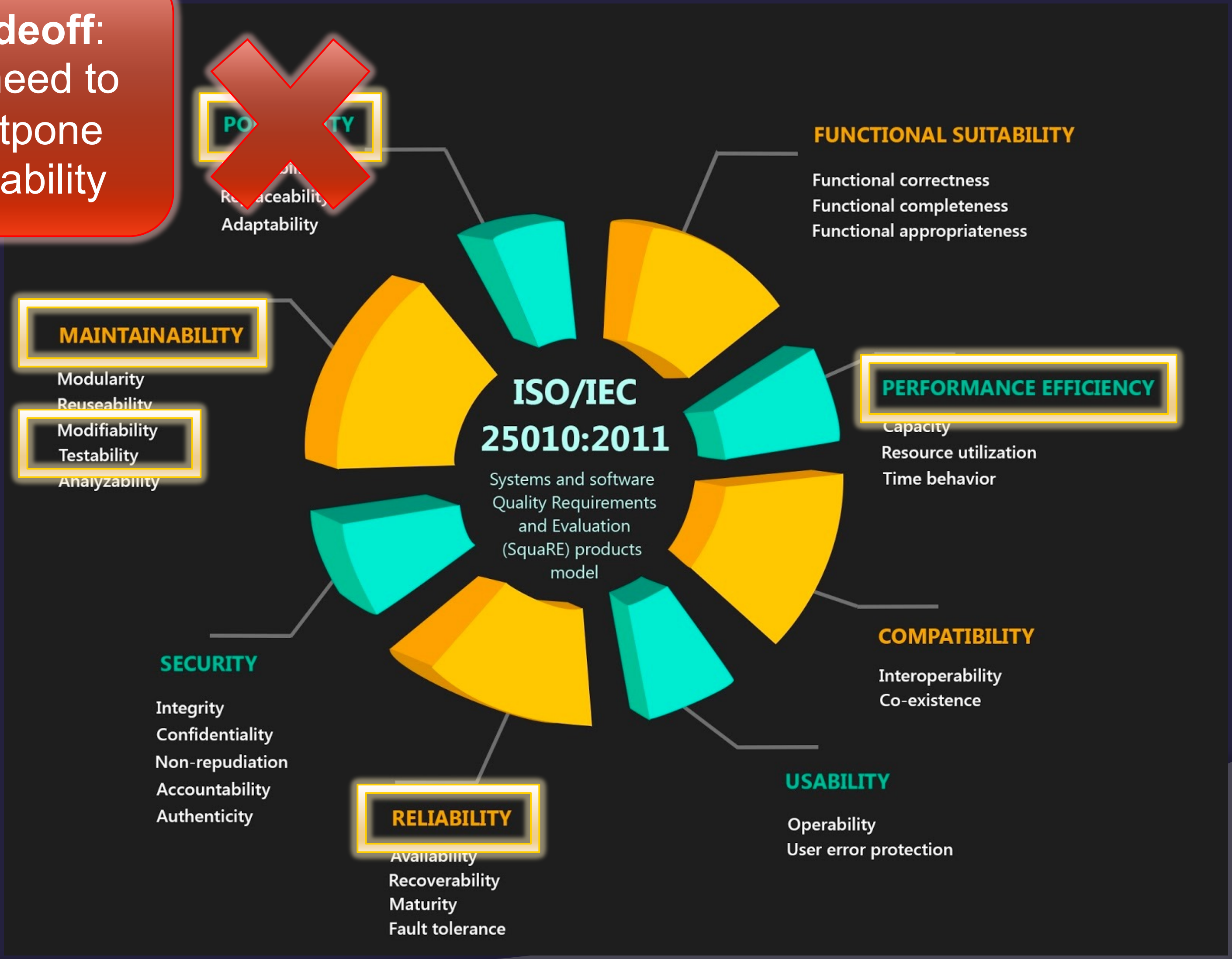
Availability
Recoverability
Maturity
Fault tolerance

Cost/benefits scenarios and analysis (simplified example)

	Benefit: Users short-term	Benefit: User long-term	Cost	Total
Solution 1	-- (vs competitor)	++ (both platforms) - (lack of visibility)	+ (cheaper in total)	0
Solution 2	++ (vs competitor)	+ (visibility) - (no users in one platform)	- (rewrite)	+1

System Qualities – Trade-off

Tradeoff: we need to postpone portability



ISO/IEC 25010:2011

Systems and software Quality Requirements and Evaluation (SquaRE) products model

PO TY

Re aceability
Adaptability

FUNCTIONAL SUITABILITY

Functional correctness
Functional completeness
Functional appropriateness

MAINTAINABILITY

Modularity
Reuseability
Modifiability
Testability
Analyzability

PERFORMANCE EFFICIENCY

Capacity
Resource utilization
Time behavior

COMPATIBILITY

Interoperability
Co-existence

SECURITY

Integrity
Confidentiality
Non-repudiation
Accountability
Authenticity

RELIABILITY

Availability
Recoverability
Maturity
Fault tolerance

USABILITY

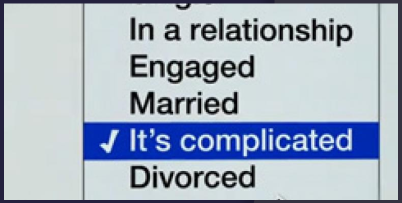
Operability
User error protection

Architecture = Tradeoff

- It will never be perfect
- It's all about doing the best tradeoff

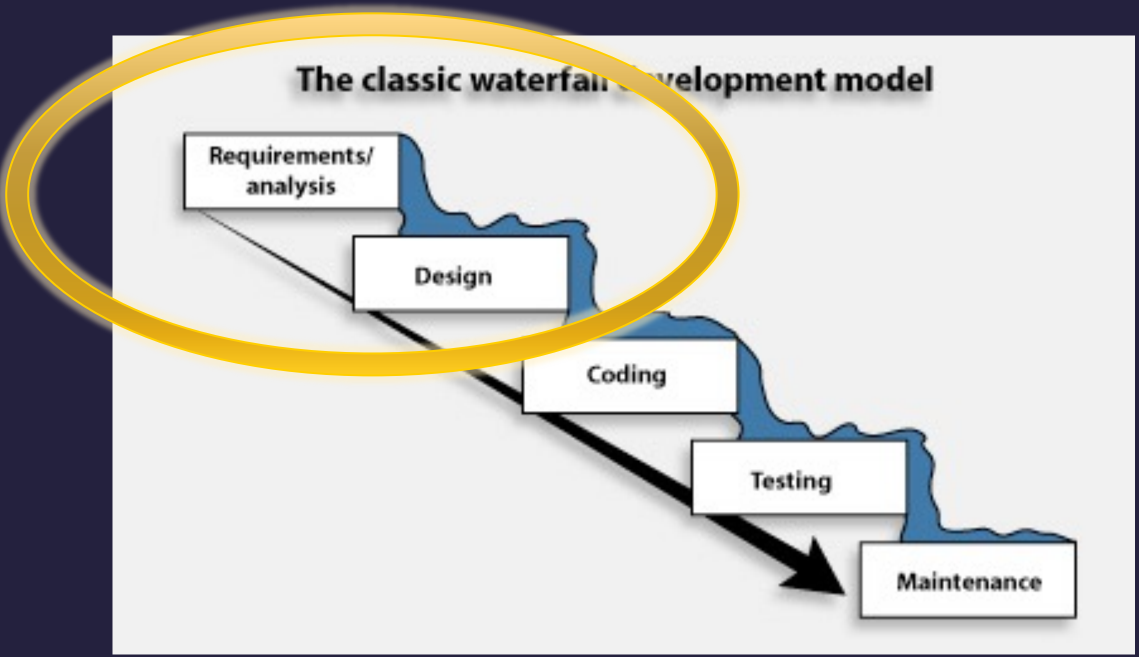
- But what if the best tradeoff is a... moving target?

Agile and Architecture: a complicated relationship



In a relationship
Engaged
Married
✓ It's complicated
Divorced

Once upon a time it was Waterfall…



Architecture

The enemy: Big Upfront Anything*

- Requirement engineering
  - Should we do upfront requirement engineering?
    - Yes
      - We need to understand what the users want
      - We need to understand the domain and its constraints
    - No
      - Too much documents and time spent are a waste
      - Requirements change anyways
  - There is a middle ground, we don't have to be extreme

* Book Chapter 3 (up to 3.3 included)

Architecture in waterfall: 3 problems

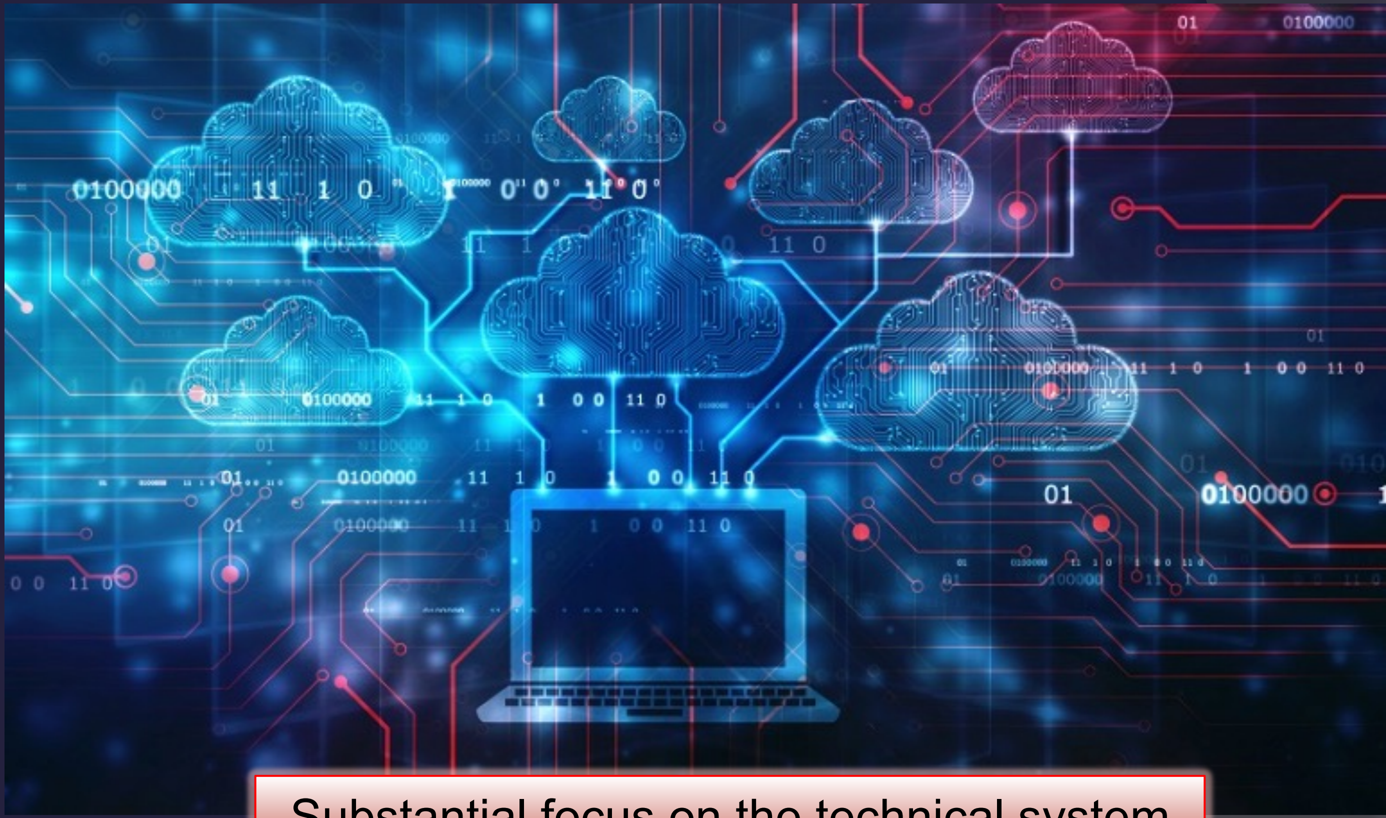
- Upfront design
  - assumes a "perfect" architecture is already known
- Is this advisable/possible/realistic?
  - Problem 1: Arch. / Reqs. / Impl. first?
    - Some Architecture Reqs emerge from implementation*
    - The earlier the Arch decision, the more the probability that is wrong because of lack of information**
  - Problem 2: Architects are humans
    - Several bias: some decisions are not rational*
      - Anchoring
        - Let's use microservices! Let's design OO!
        - First decisions have a strong effect on other decisions

* Van Vliet, Tang: *Decision making in software architecture,* JSS, 2016

** Poort, van Vliet: *Architecting as a Risk- and Cost Management Discipline*

Problem 3: Focus on the Wrong Outcome

Little delivered value!



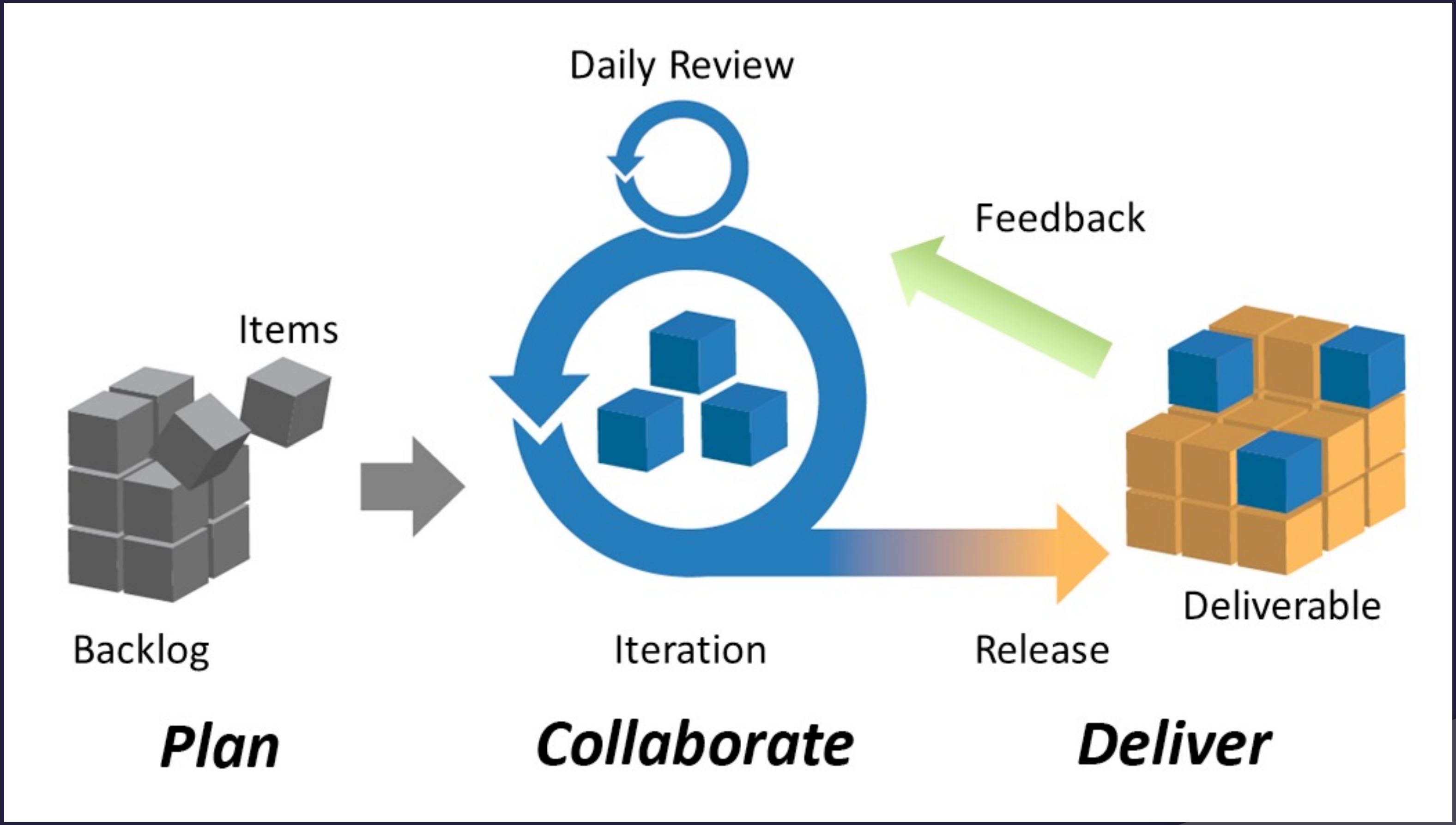
Substantial focus on the technical system

But the solution of problem 3 is not just doing the opposite! You need a balance…

Too much focus on delivering short-term value!

Little focus on the technical system

…then the Agile revolution happened



Spot what can affect architecture in the agile manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Spot what can affect architecture

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

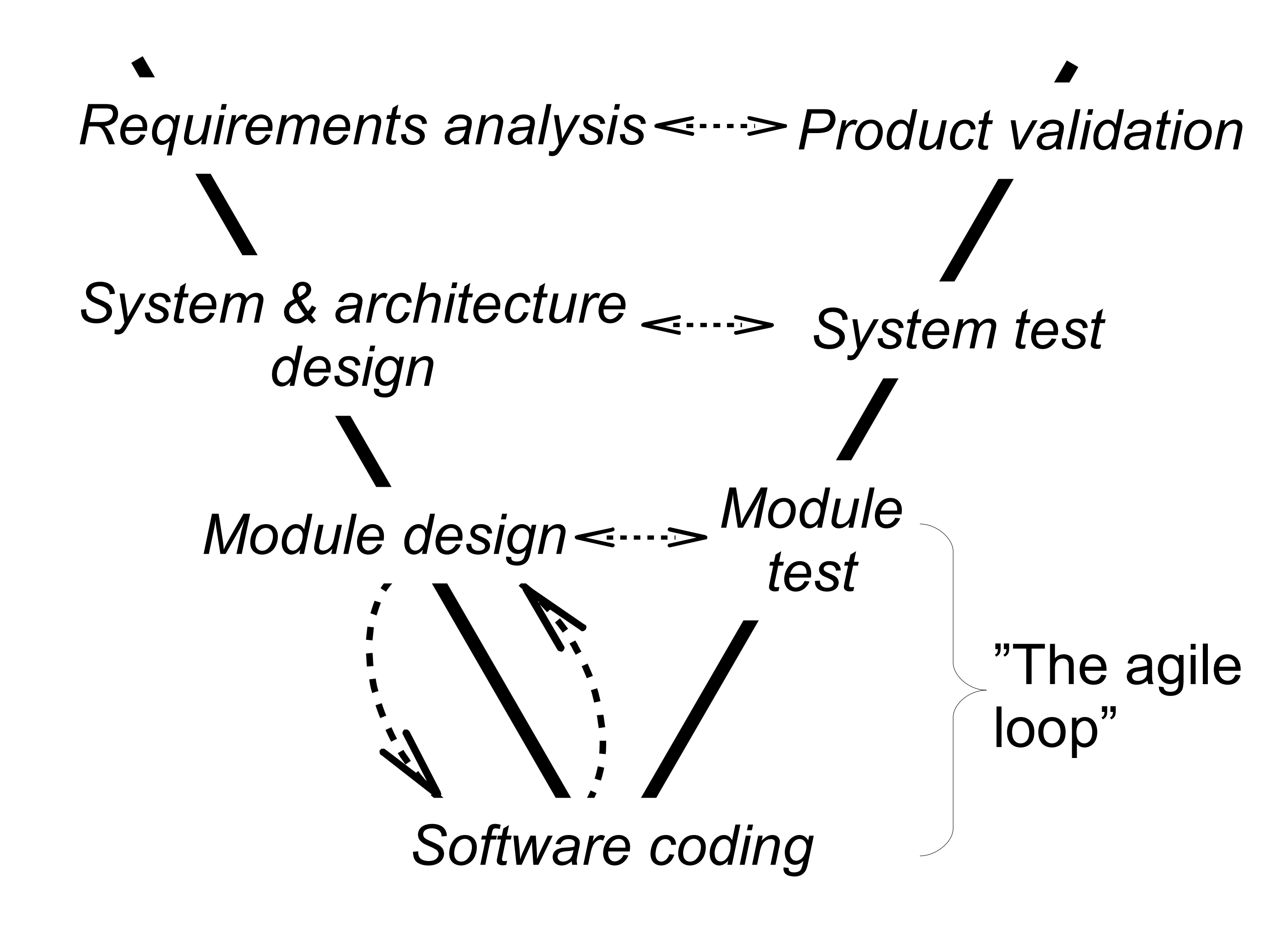
Necessary but not sufficient

- The best architectures, requirements, and designs emerge from self-organizing teams

Different architecture management in different contexts…

- Small projects:
  - 1 team can
    - understand few stakeholders
    - manage their concerns
    - handle complexity
    - grasp the overall view
- Large projects:
  - Many teams need to coordinate on
    - understanding many stakeholders
    - tradeoff among many concerns
    - handle high complexity
    - share the same view

Meanwhile, in practice…



Requirements analysis <----> *Product validation*

System & architecture design <----> *System test*

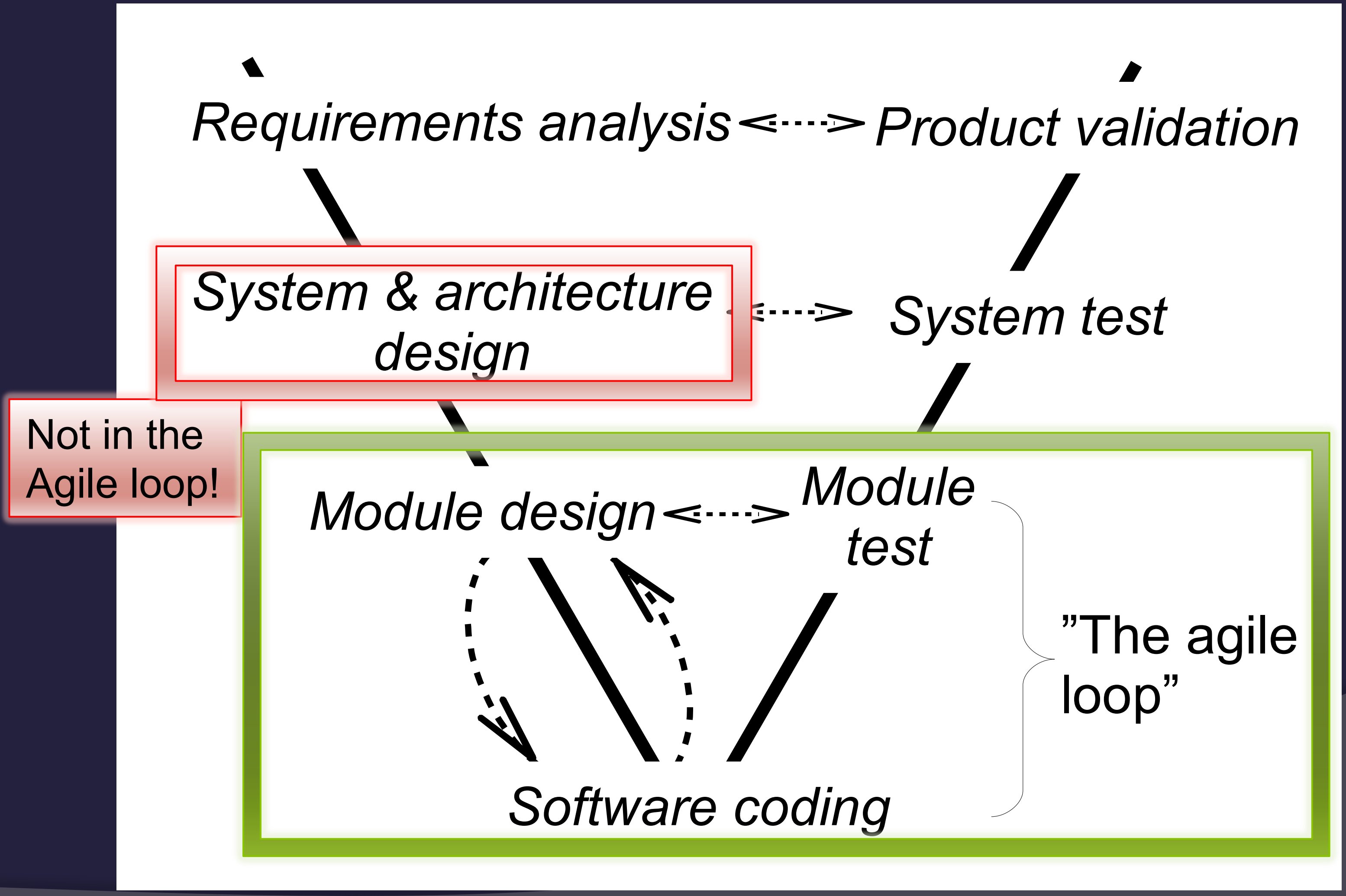
Module design <----> *Module test*

Software coding

”The agile loop”

* Eklund, Olsson, Strøm - *Industrial Challenges of Scaling Agile in Mass-Produced Embedded Systems,* 2014

What about architecture?



Requirements analysis <···> Product validation

System & architecture design <···> System test

Module design <···> Module test

Software coding

”The agile loop”

Not in the Agile loop!

* Eklund, Olsson, Strøm - *Industrial Challenges of Scaling Agile in Mass-Produced Embedded Systems*, 2014

Architecture in Agile?

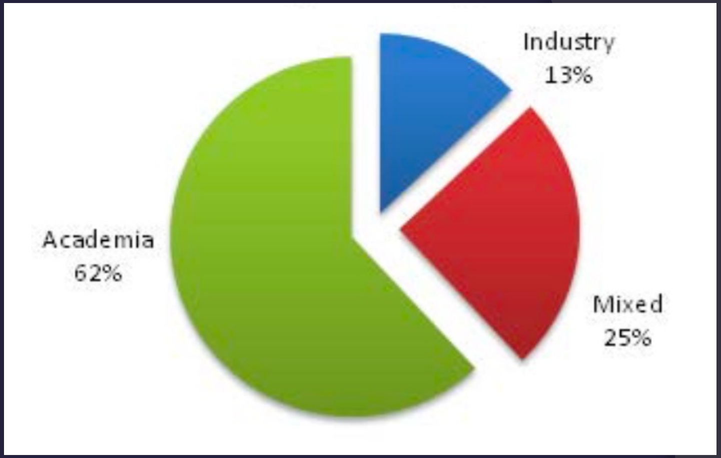
- Not emphasized in Agile practices
- "Just enough architecture/design"

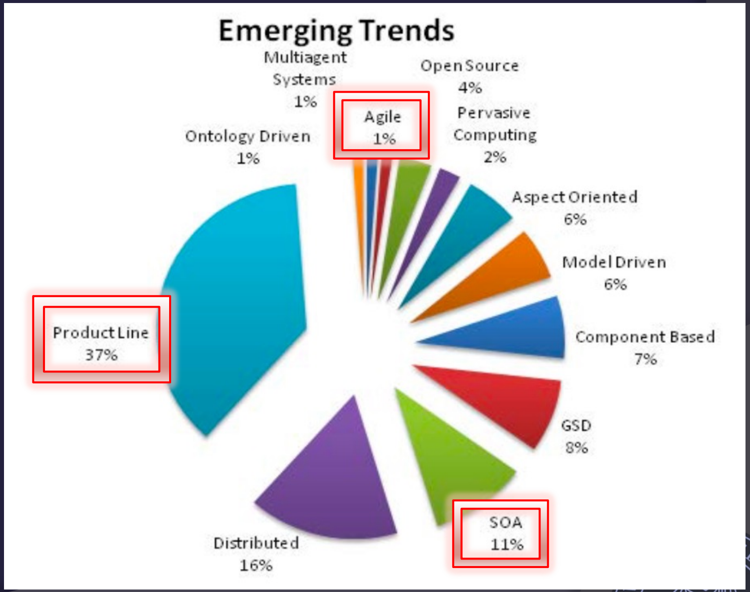
- But what does that mean?
- Are there studies?

Agile and Architecture: what do we know?

Current studies

- Most of the papers are not based on industrial experience or evaluation
- Not much on “Continuous” and “Agile”
  - 1% Agile
  - 11% SOA
  - 37% Software Product Lines
    - require upfront design

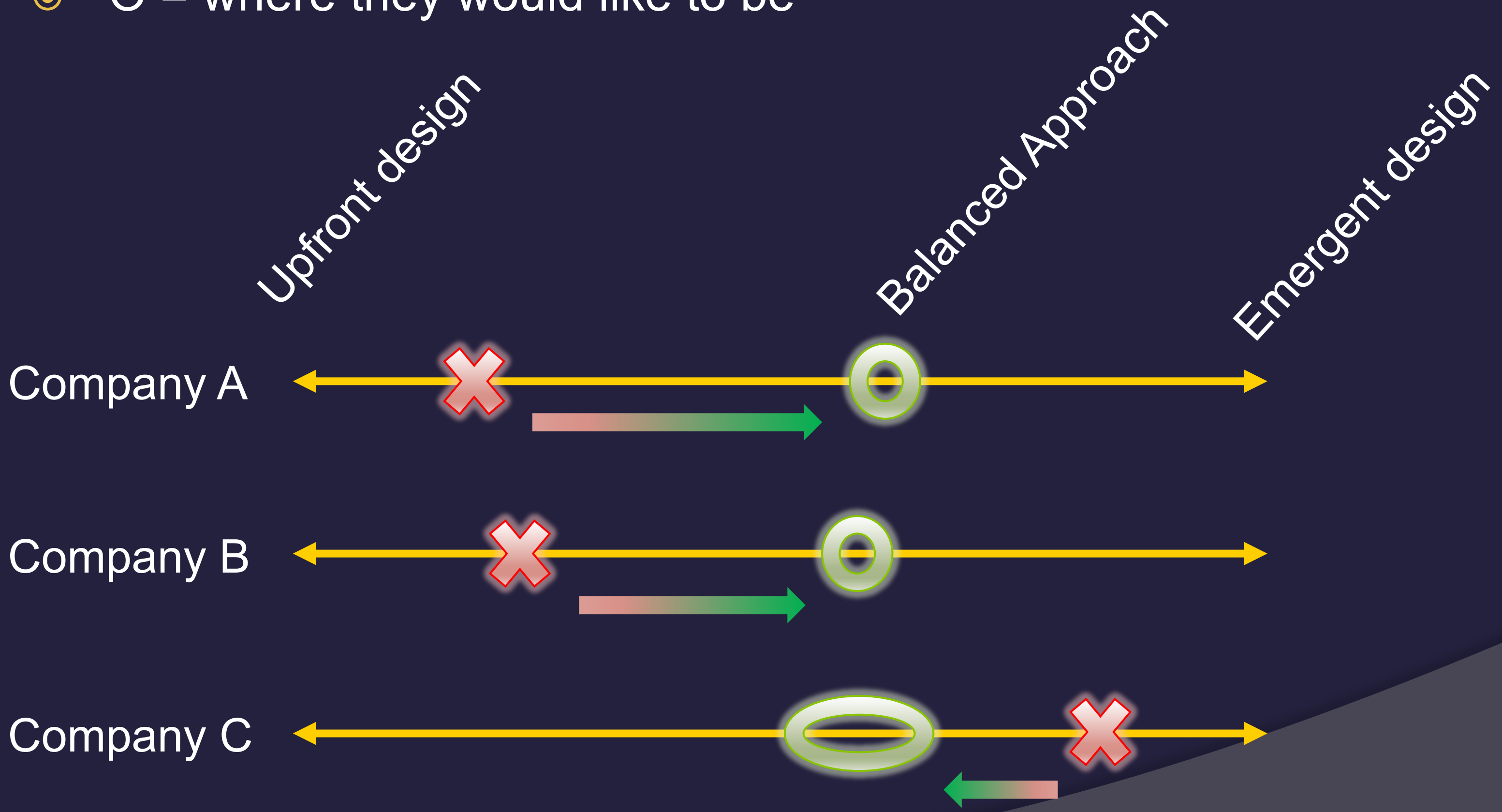




* Qureshi, Usman, Ikram: *Evidence in Software Architecture, a Systematic Literature Review,*

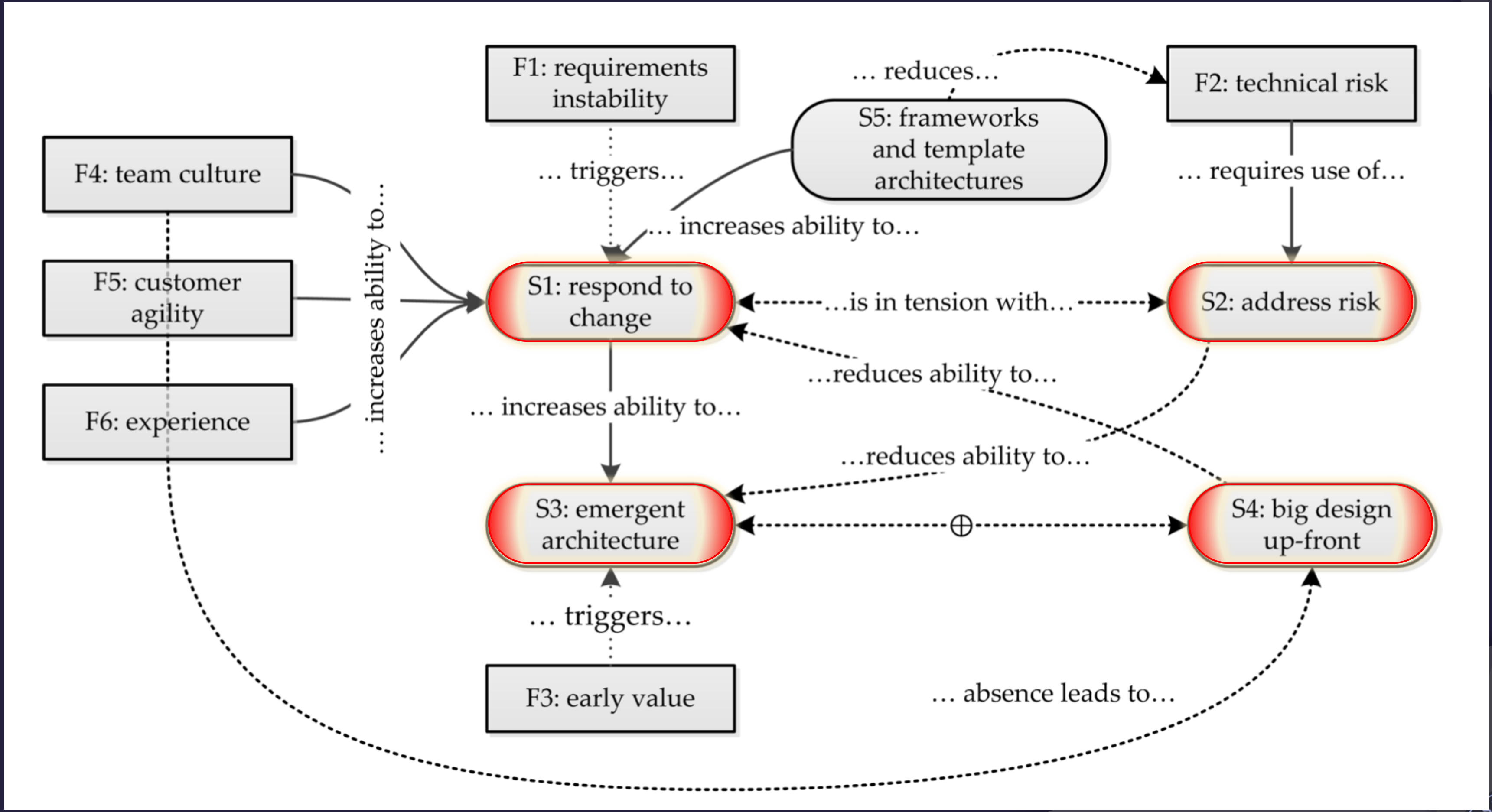
What do companies want? A balanced approach*

- X = where they are now
- O = where they would like to be



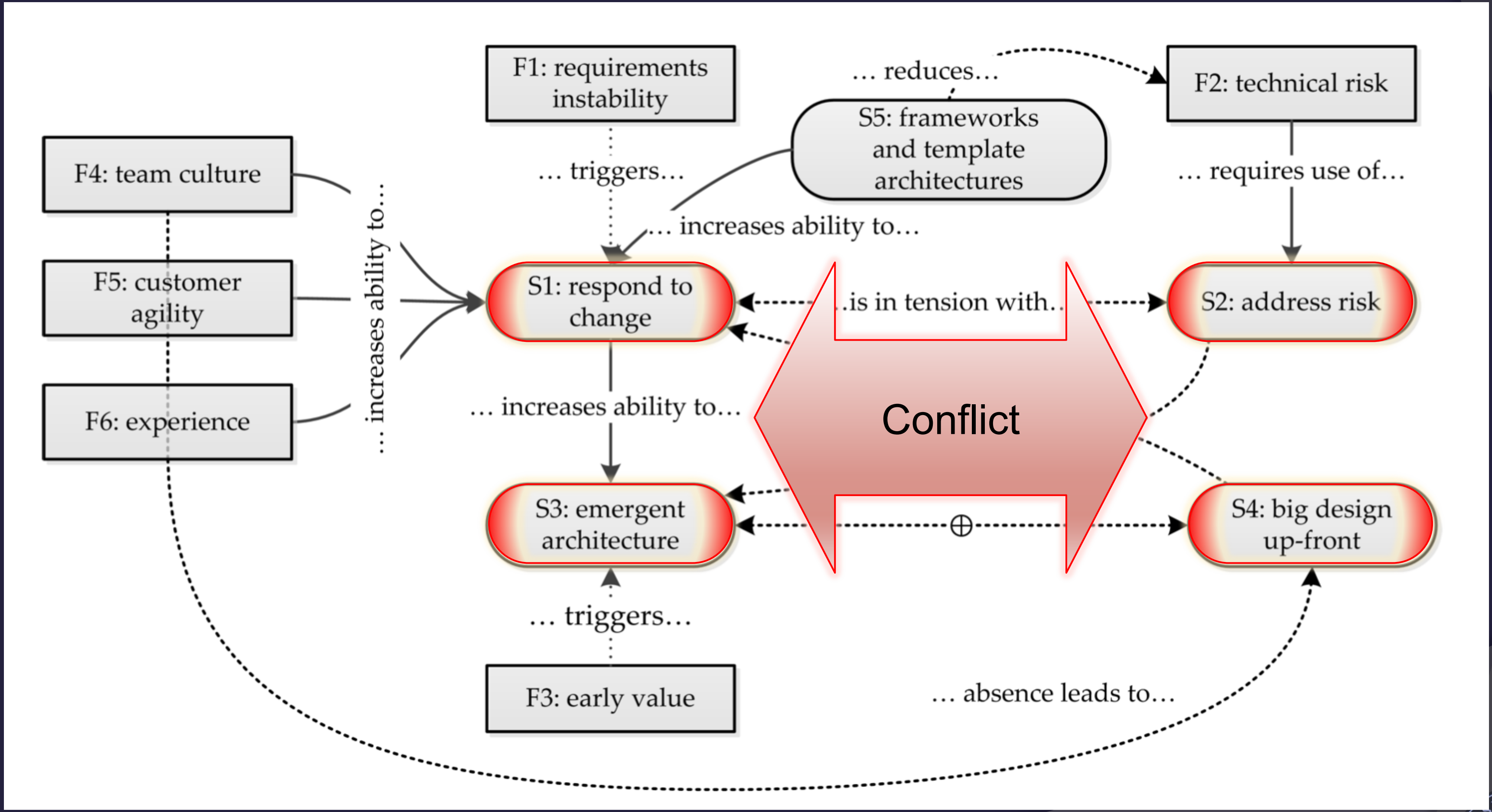
T. Mårtensson, D. Ståhl, A. Martini and J. Bosch, "*Continuous Architecture: Towards the Goldilocks Zone and Away from Vicious Circles*" 2019 IEEE International Conference on Software Architecture (ICSA) 2019

Why is it so difficult to balance the right upfront design?



* Waterman, Noble, Allan - *How Much Up-Front? A Grounded Theory of Agile Architecture,* ICSE, 2015

Why is it so difficult to balance the right upfront design?



* Waterman, Noble, Allan - *How Much Up-Front? A Grounded Theory of Agile Architecture,* ICSE, 2015

Frameworks embed architectural decisions and help agility

- *"many of the architectural decisions are embedded in the framework, and hence architectural changes can be made with a lot less effort"*

- Examples:
  - Apache Kafka – message streaming
  - JUnit – Java unit testing framework
  - Django – python web applications
  - …

Prioritizing architectural significant concerns (backlog)

- Architecture is a risk- and cost management activity*
- Significance depends on Risk (Decision)
- Risk:
  - Prob (Failure) x Impact (Failure)
    - An example of impact is the necessity of re-architecting because of a probable failure

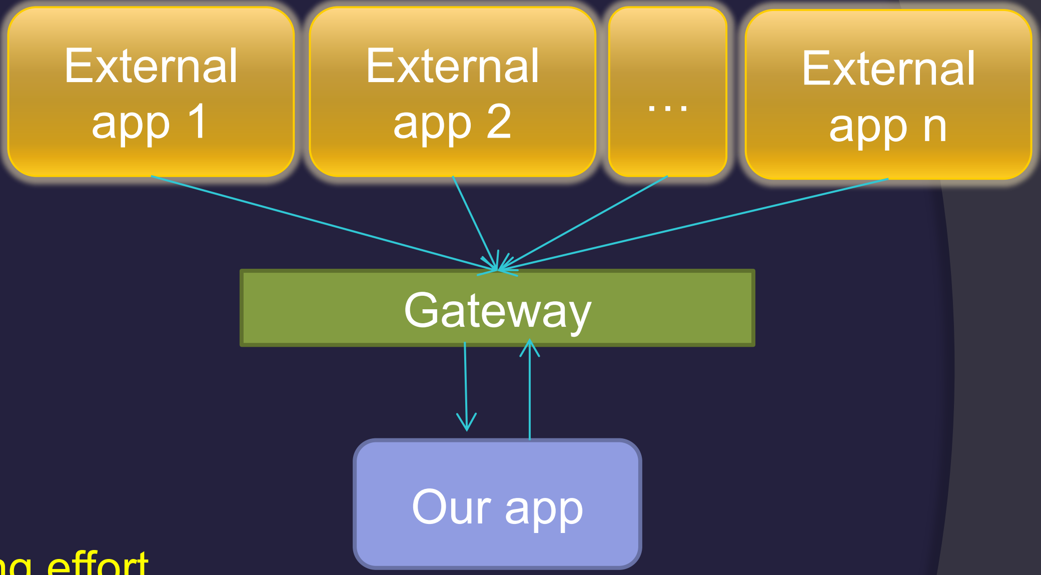
* Poort, van Vliet: *Architecting as a Risk- and Cost Management Discipline*

Example of risk

- Is it higher the riski to suffer from a flu or from malaria taking a bus in Oslo?
  - Disclaimer: numbers are just examples, they are not real
  - Impact is represented as 1-5 (5 = worst)
- Malaria
  - Prob (Malaria) = 0.0000001
  - Impact (Malaria) = 4
  - Risk = 0 x 4 = 0.0000004
- Flu
  - Prob (Flu) = 0.6
  - Impact (Flu) = 1
  - Risk = 0.6 x 1 = 0.6
- When taking the subway in Oslo, one should be more worried to get a flu than to get malaria
  - 0.6 >> 0.0000004

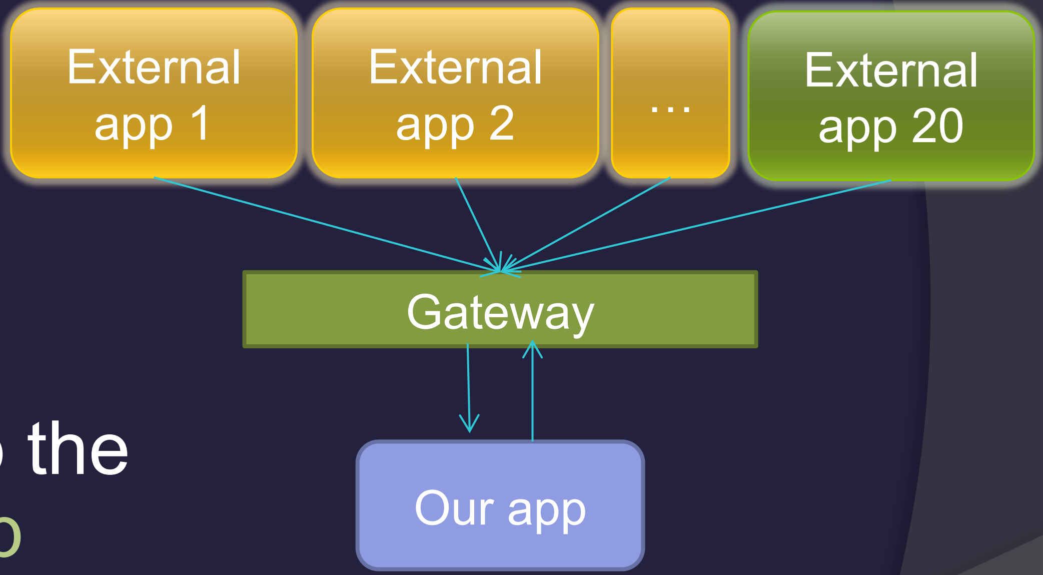
Architectural example (1)

- We build an app that supports other apps
- We need to develop a gateway
- Architectural concern: scalability
  - Significant?
- We don't know how many apps we will need to support
  - Is it going to be 1 or 100?
- We might waste a lot of engineering effort in planning a scalable gateway that is not used
- The risk of not being able to support apps (unhappy customers) has
  - A high impact (4)
  - A low probability at this stage (0.3)
- Result: at the moment the risk of this concern is low
  - 0.3 (probability of failure) x 4 (high impact) = 1.2



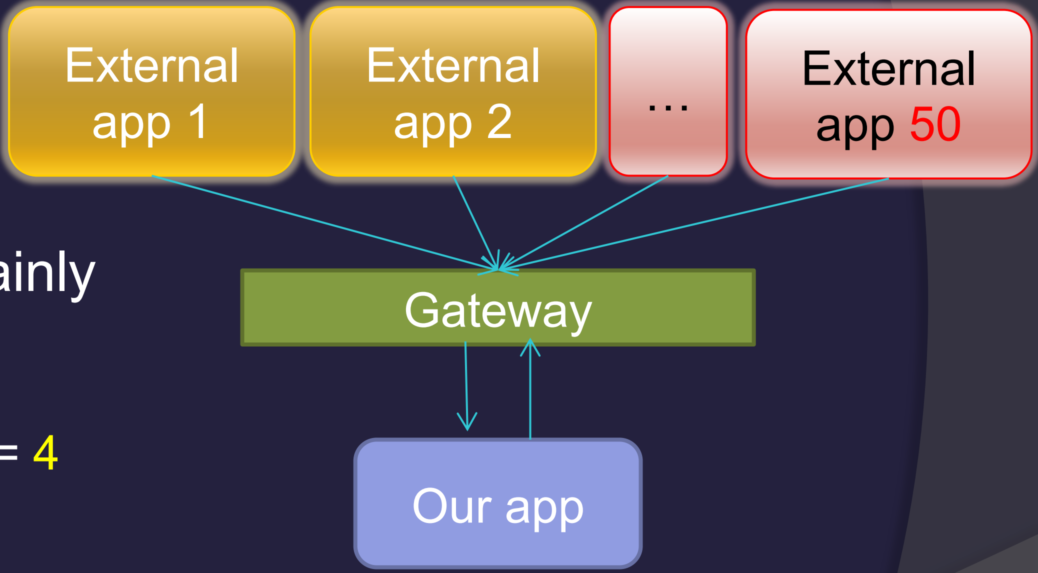
Architectural example (2)

- The risk is low, so the scalability concern is not significant
- We decide to develop the gateway to support up to 20 apps
  - Architectural decision that costs less and doesn't address the scalability concern



Architectural example (3)

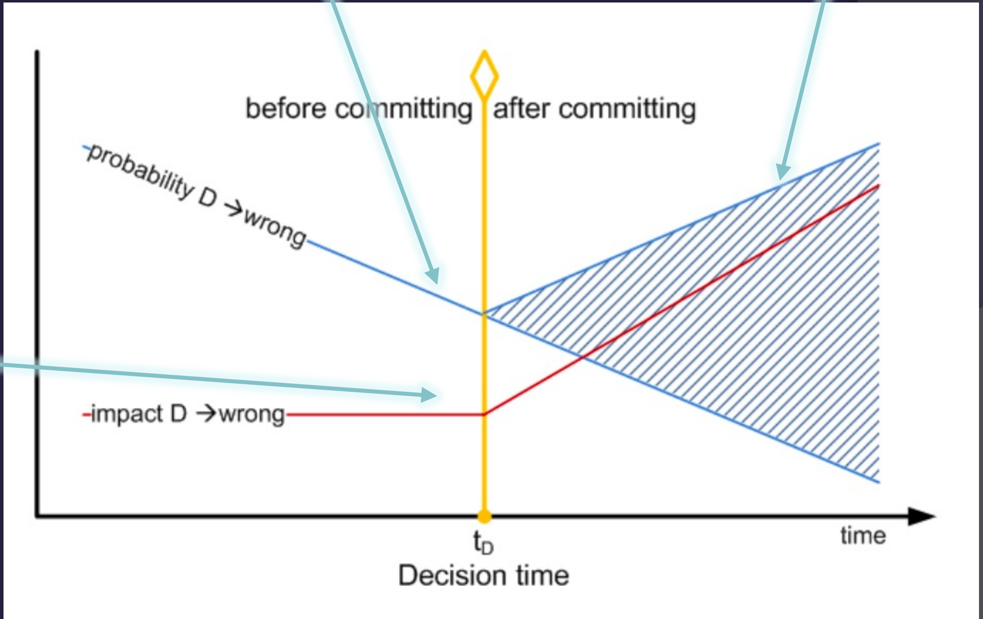
- After some time, we receive new information: now more than 30 new apps want to use our gateway



- What happens now?
- Now our solution will be certainly wrong (failure)
- The risk is high
  - 1 (prob of failure) x 4 (impact) = 4
  - 4 >> 1.2 (previous risk)
  - The concern is significant
- We need to re-architect as soon as possible to address the scalability concern
  - The more we wait the more re-architecting costs
  - The more we wait the more customers could experience connectivity issues

Other considerations *

- The probability of an architectural decision D to be wrong decreases over time
  - More information can become available
- At the same time, the probability of D to be wrong can increase over time after we have taken a decision
  - More information can become available from implementation
- The impact of failure increases over time after we have taken a decision D
  - If we need to change such decision, it has a cost of re-architecting

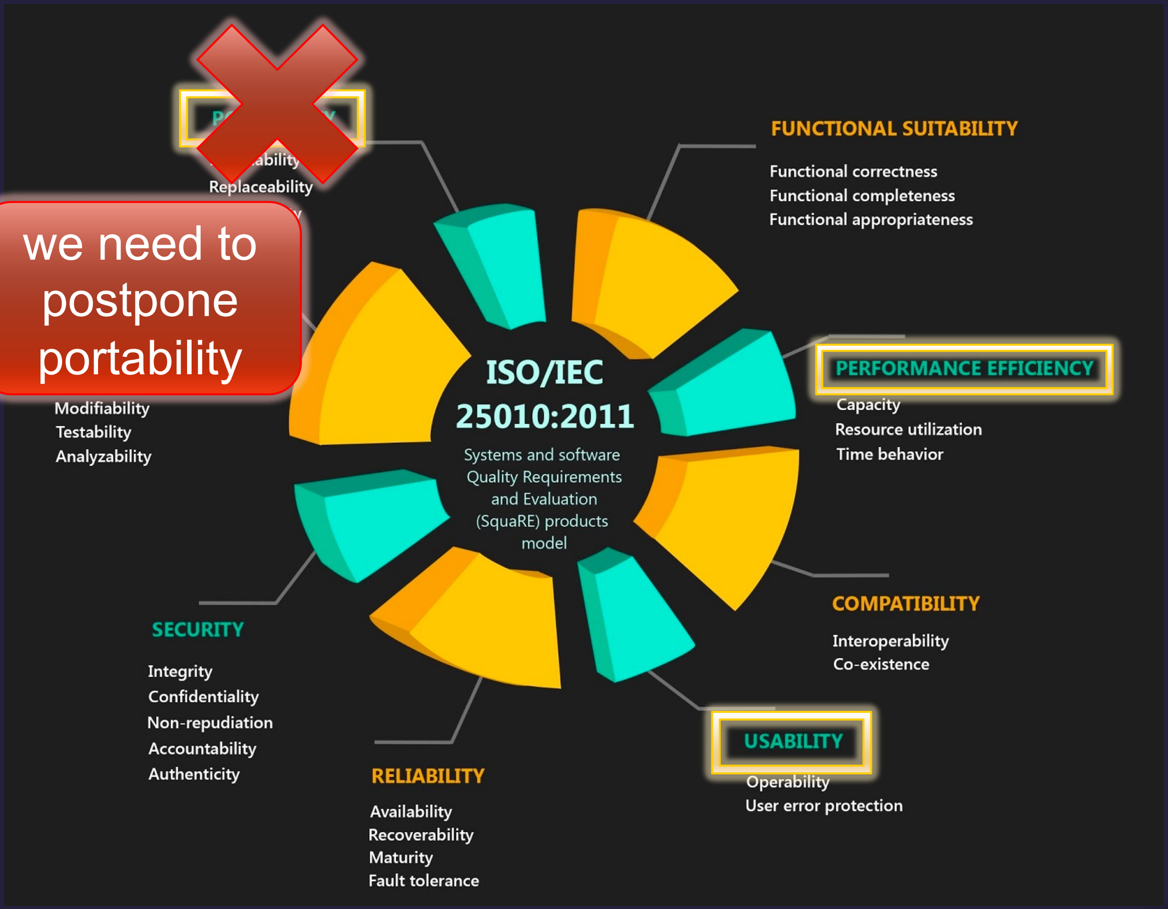


[*] Poort, van Vliet: *Architecting as a Risk- and Cost Management Discipline*

So what should we do?

- Postponing architectural decisions can increase the probability of being correct
  - Only if possible and in case there is uncertainty
- More information can change our perception of the risks
  - Architectural decisions need to be monitored and revisited
- Architectural Backlog should be continuously re-prioritized based on risk (and cost)

Re-prioritize your architecture backlog



we need to postpone portability

Concern	Rank
Scalability	1
Maintainability	2
Performance	3
Portability	4
…	…

Continuous Re-prioritization

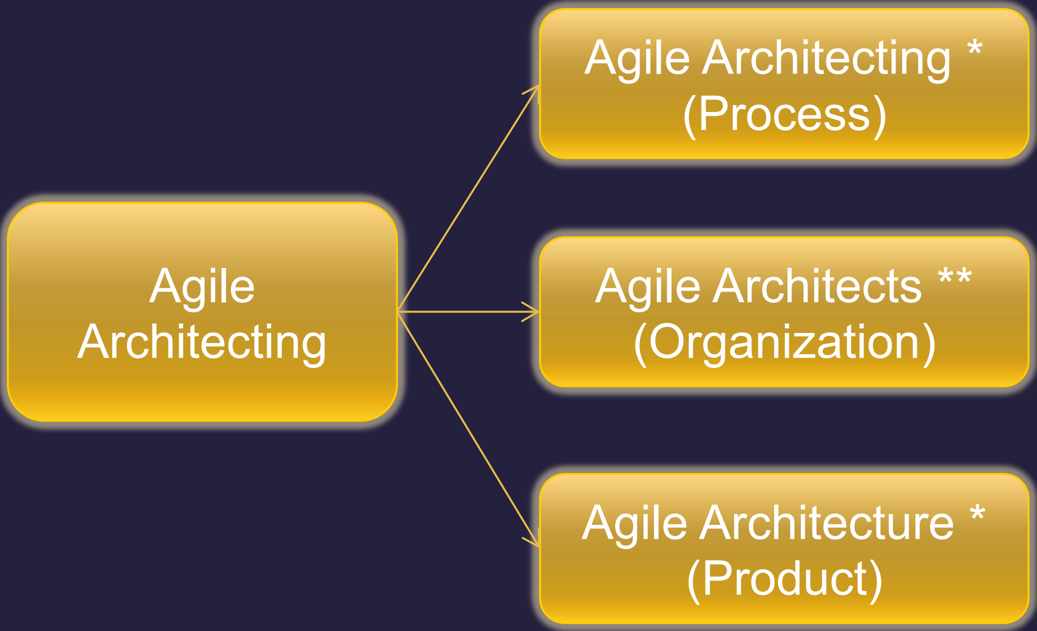
Agile Architecting

What is Agile Architecting?

- Architecting that supports Continuous Delivery *
  - Continuous Delivery is:
    - *"Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way."*

* Murat Erder, Pierre Pureur "*Continuous Architecture*", Elsevier, 2015
* various blogs online, https://continuousdelivery.com/

Agile Architecting: 3 dimensions

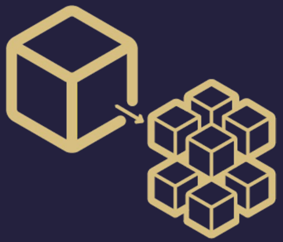


Agile Architecting

Agile Architecting *
(Process)

Agile Architects **
(Organization)

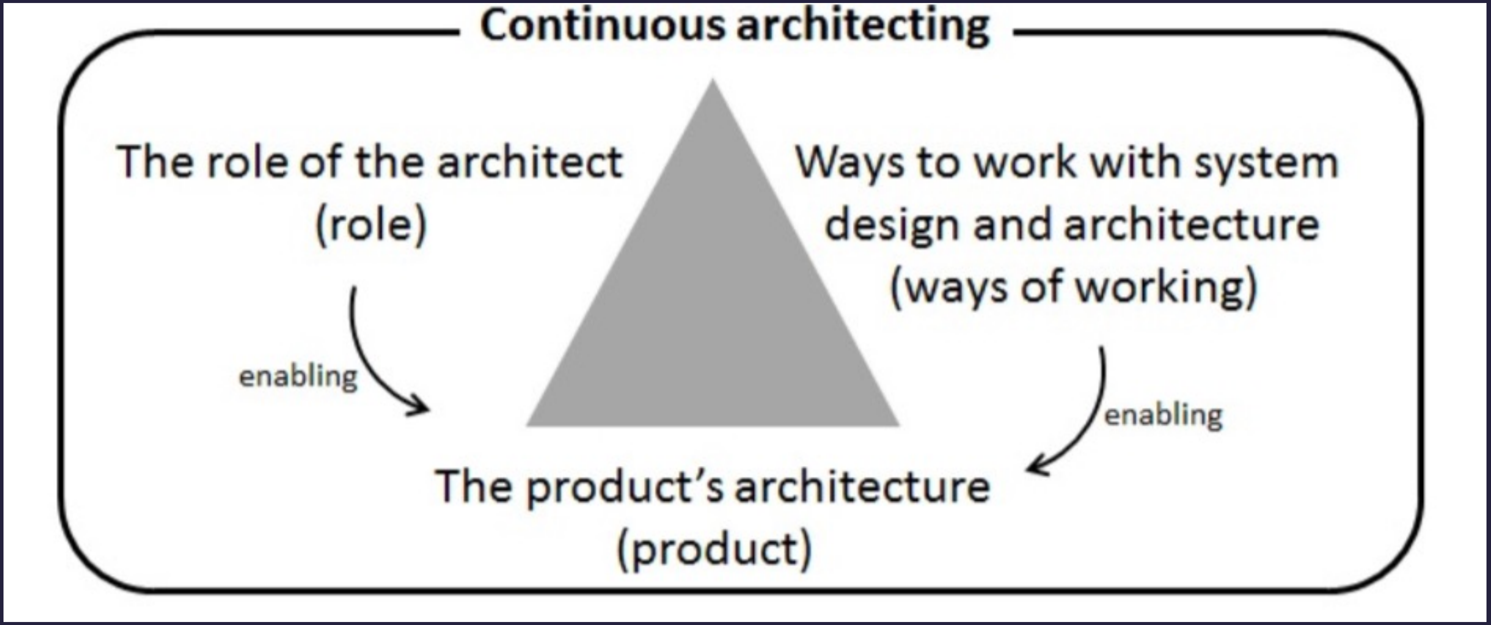
Agile Architecture *
(Product)



* Bellomo, Kruchten, Nord, Ozkaya: *How to Agilely Architect an Agile Architecture*

** A. Martini and J. Bosch, *“A Multiple Case Study of Continuous Architecting in Large Agile Companies: Current Gaps and the CAFFEA Framework,”*

Product architecture needs support



Continuous architecting

The role of the architect (role)

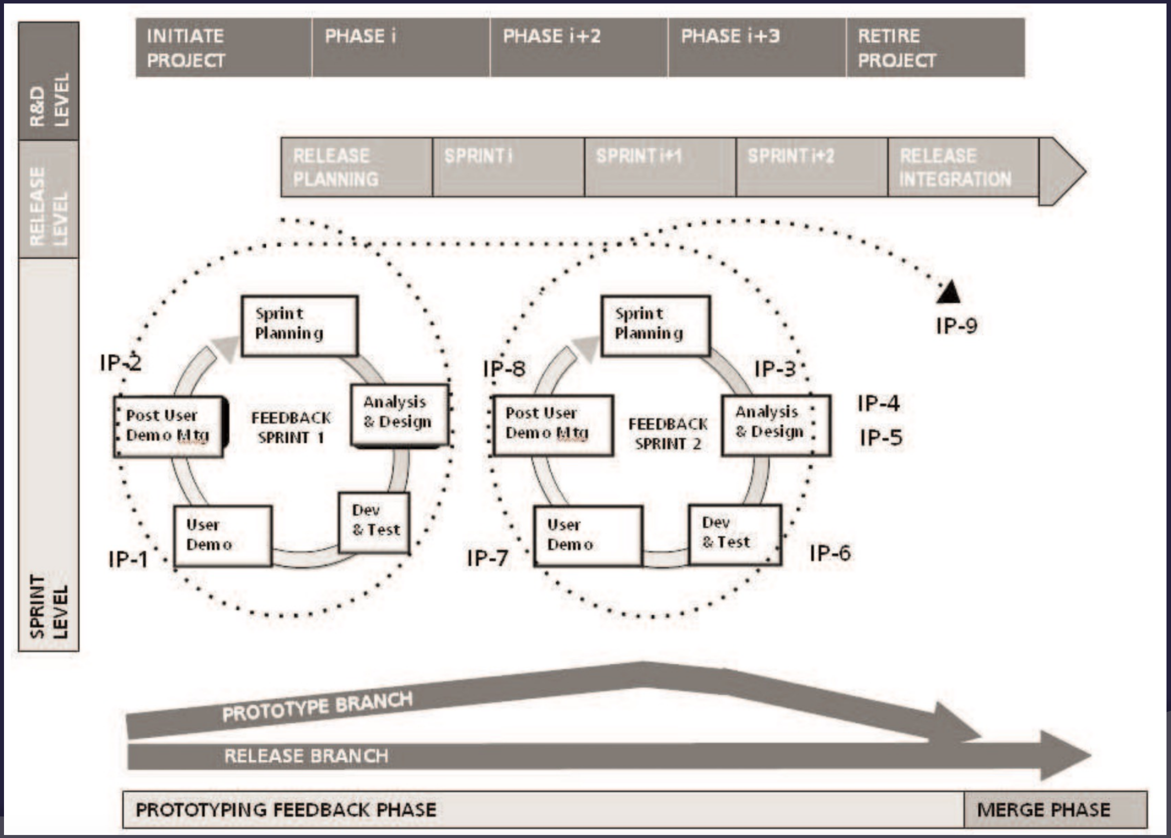
Ways to work with system design and architecture (ways of working)

The product's architecture (product)

T. Mårtensson, D. Ståhl, A. Martini and J. Bosch, "*Continuous Architecture: Towards the Goldilocks Zone and Away from Vicious Circles*" 2019 IEEE International Conference on Software Architecture (ICSA) 2019

Agile Architecting – Process*

- An Agile way to define an architecture, using an iterative lifecycle, allowing the architectural design to tactically evolve over time, as the problem and the constraints are better understood



* Bellomo, Kruchten, Nord, Ozkaya: *How to Agilely Architect an Agile Architecture*

Continuous Re-prioritization

Agile Architecting – Product*

- Product Architecture enables Agility
  - Layered architecture
  - Service Oriented Architecture
    - Microservices
  - Frameworks
  - Other tactics*

* Bellomo, Kruchten, Nord, Ozkaya: *How to Agilely Architect an Agile Architecture*

Organization: who is in charge of the overall design?

Agile and Architects: a case study*

* A. Martini and J. Bosch, “A Multiple Case Study of Continuous Architecting in Large Agile Companies: Current Gaps and the CAFFEA Framework”, Working International Conference on Software Architecture, 2016, Venice, Italy

Research Design – gaps and framework

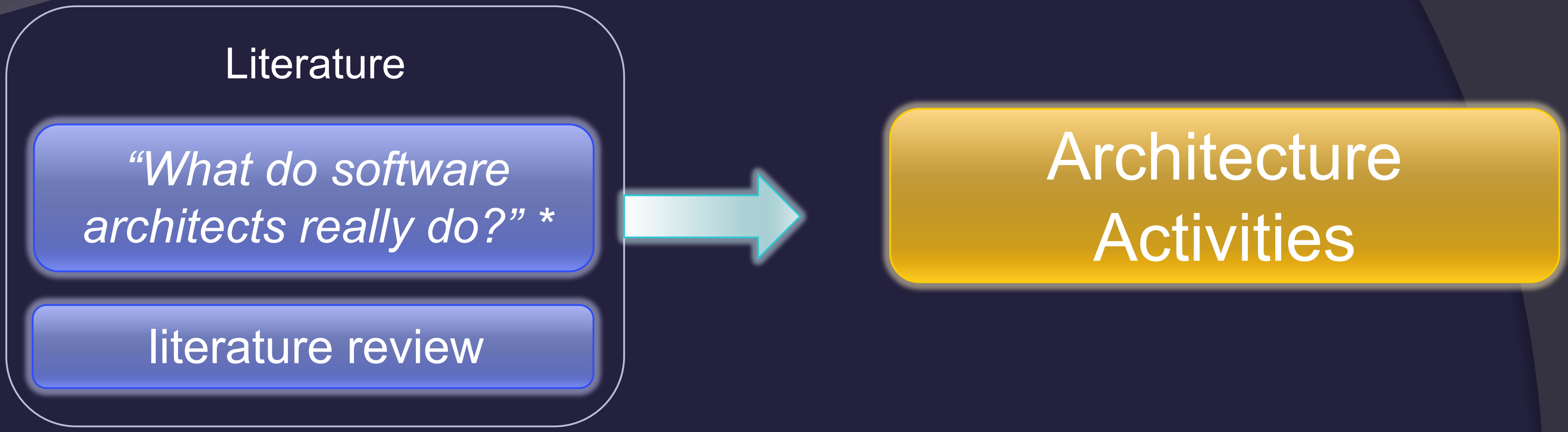
Literature

“What do software architects really do?” *

literature review

* P. Kruchten, “What do software architects really do?” *Journal of Systems and Software*, Dec. 2008

Research Design – Gaps and Solution



Literature

“What do software architects really do?” *

literature review

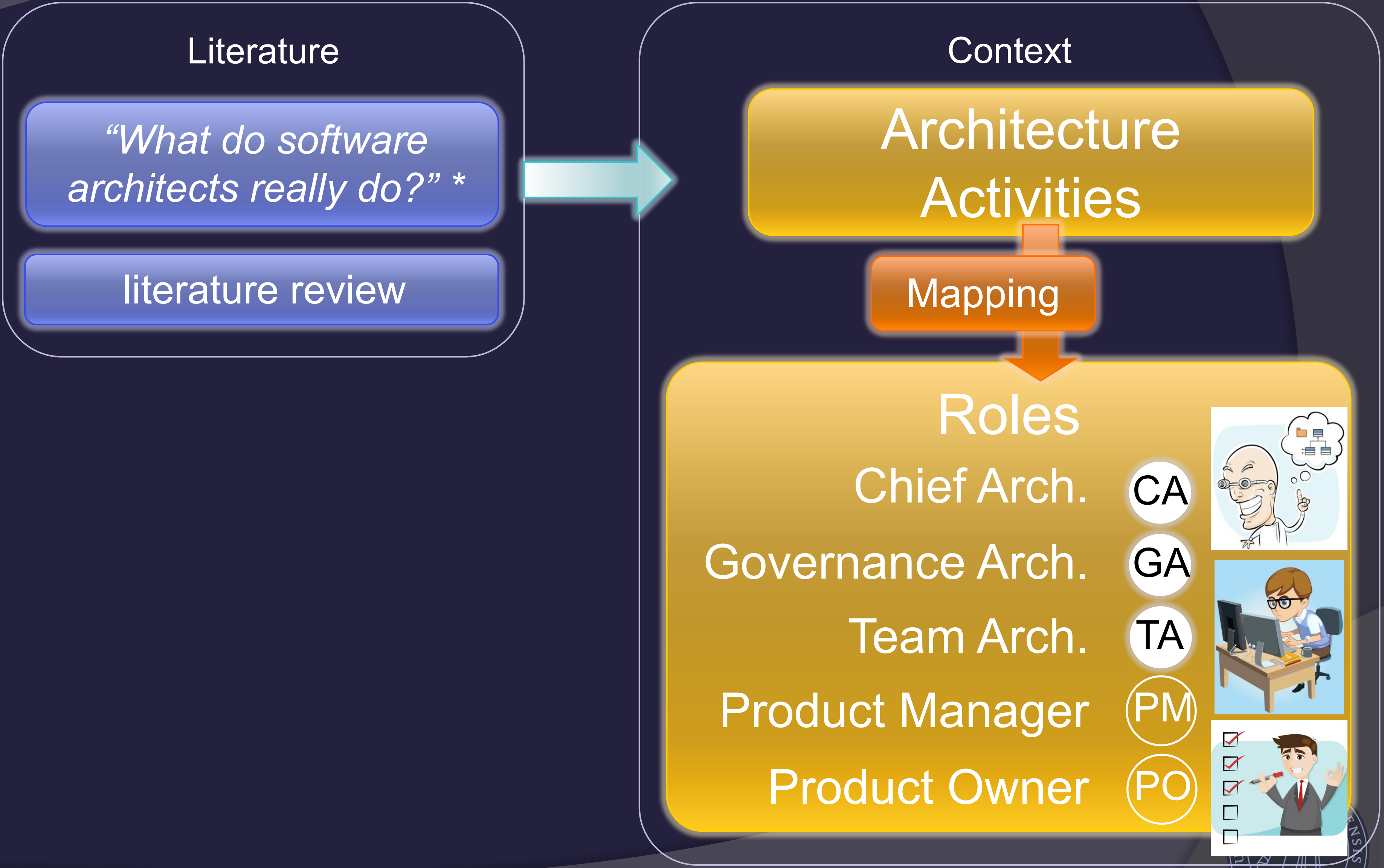
Architecture Activities

* P. Kruchten, “What do software architects really do?” *Journal of Systems and Software*, Dec. 2008

Architecture activity examples

Activity: Architecture Knowledge Production	
Sub-Activities	**Meaning**
Architect	Examines concerns and context in order to compe up with architectureally significant requirements.
Integrate	Integrates software engineering and knowledge engineering tools and repositories into the development process.
Share	Shares the AK with implementers to facilitate their understanding.
Trace	Creates necessary links (fwd., bwd., formal, informal) between reasoning-, design-, genera-, and context knowledge.
Synthesize	Extends or modifies the design knowlede through creation of detailed design for the architecture.
Distill	Examines design to turn patterns therein into general knowledge for future reuse.
Apply	Uses existing solutions, patterns, templates (general knowledge) to solve problems at hand.

Research Design – Gaps and Solution



Literature

*“What do software architects really do?” **

literature review

Context

Architecture Activities

Mapping

Roles

Chief Arch. CA

Governance Arch. GA

Team Arch. TA

Product Manager PM

Product Owner PO

* P. Kruchten, “What do software architects really do?” *Journal of Systems and Software*, Dec. 2008

Interactive mapping in 5 companies

- Who is responsible now?
- Is this activity missing?
- Who should do it?

Activity	Meaning
Architect	Examines concerns and context in order to compe up with architectureally significant requirements. a A A*
Integrate	Integrates software engineering and knowledge engineering tools and repositories into the development process. A* A
Share	Shares the AK with implementers to facilitate their understanding. a A
Trace	Creates necessary links (fwd., bwd., formal, informal) between reasoning-, design-, genera-, and context knowledge. a A
Synthesize	Extends or modifies the design knowlede through creation of detailed design for the architecture. a A
Distill	Examines design to turn patterns therein into general knowledge for future reuse. A A*
Apply	Uses existing solutions, patterns, templates (general knowledge) to solve problems at hand. a

Relationship

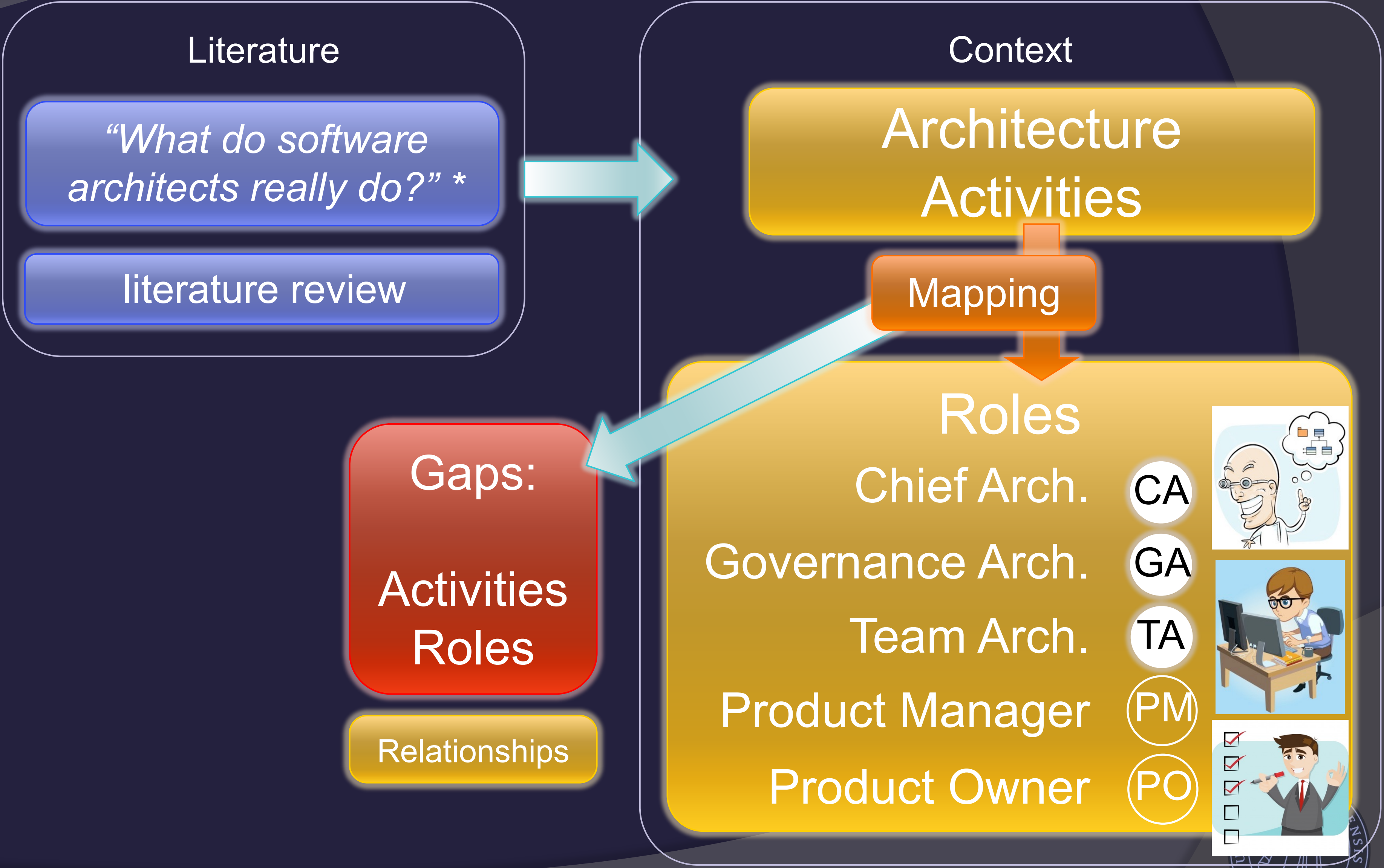
A

a A*

Gap

a A

Gaps and Solution



Literature

*“What do software architects really do?” **

literature review

Context

Architecture Activities

Mapping

Roles

Chief Arch. CA

Governance Arch. GA

Team Arch. TA

Product Manager PM

Product Owner PO

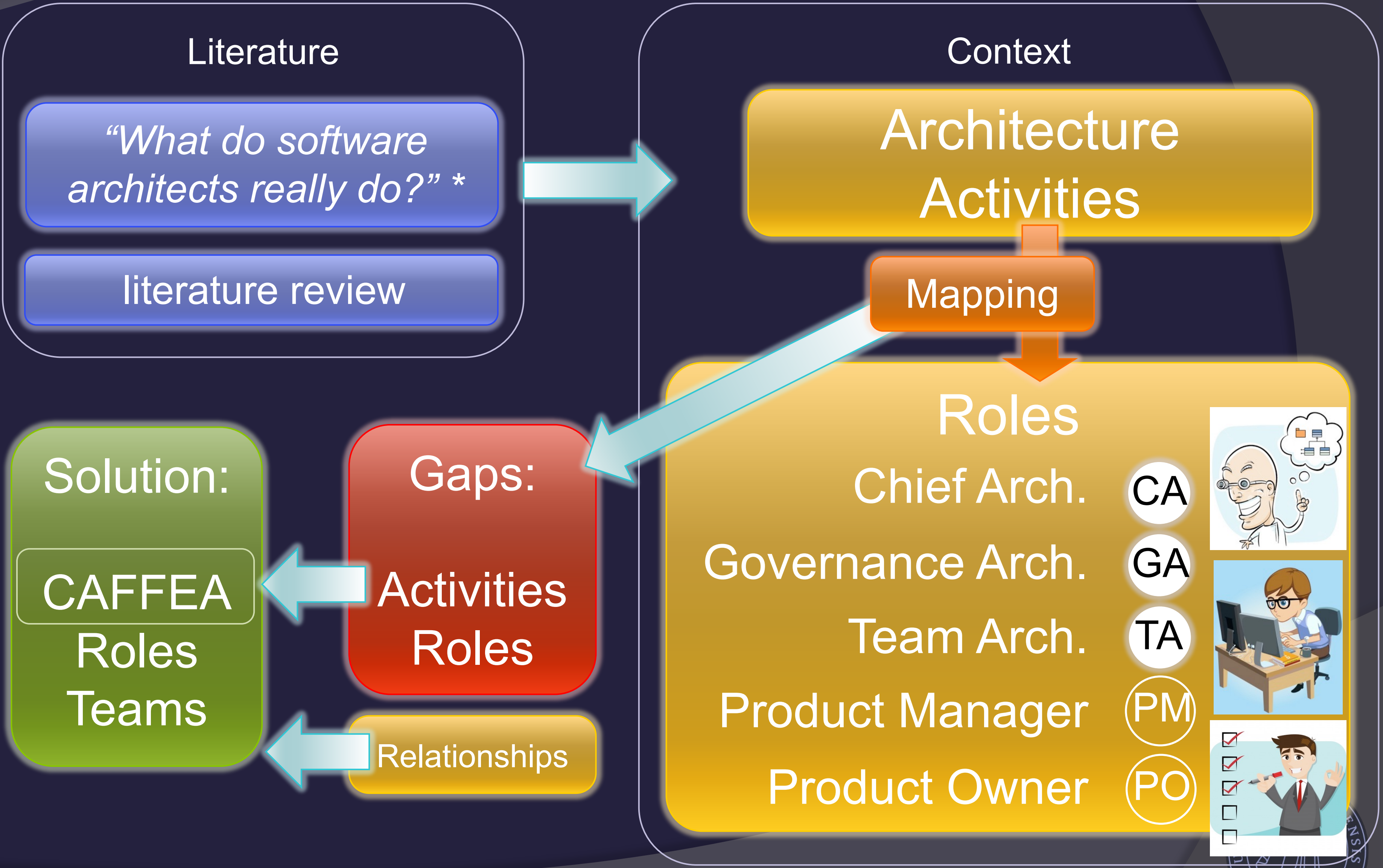
Gaps:

Activities
Roles

Relationships

* P. Kruchten, “What do software architects really do?” *Journal of Systems and Software*, Dec. 2008

Gaps and Solution



Literature

*"What do software architects really do?" **

literature review

Context

Architecture Activities

Mapping

Roles

Chief Arch. CA

Governance Arch. GA

Team Arch. TA

Product Manager PM

Product Owner PO

Solution:

CAFFEA

Roles

Teams

Gaps:

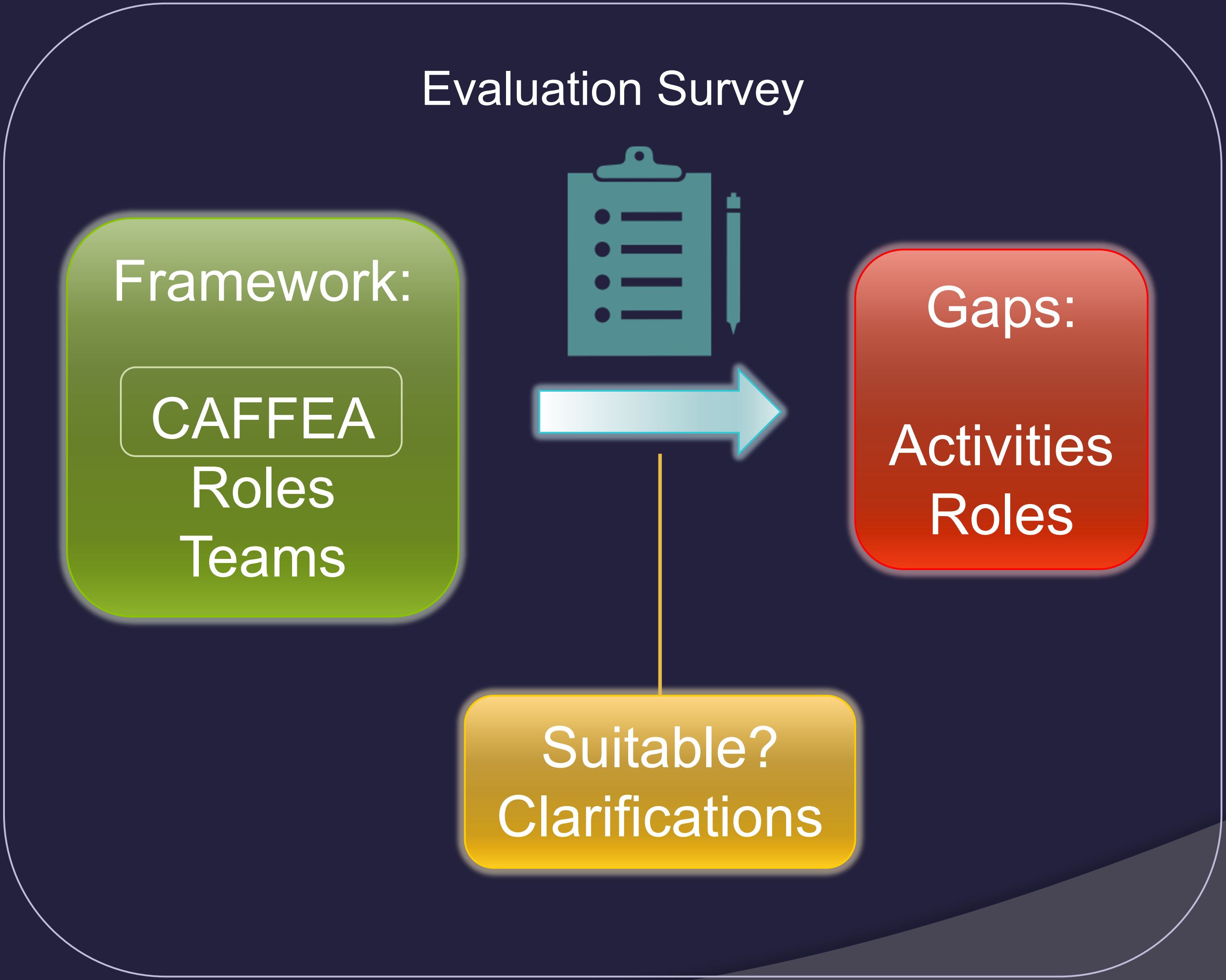
Activities

Roles

Relationships

* P. Kruchten, "What do software architects really do?" *Journal of Systems and Software*, Dec. 2008

Did it work? – First Evaluation



Evaluation Survey

Framework:

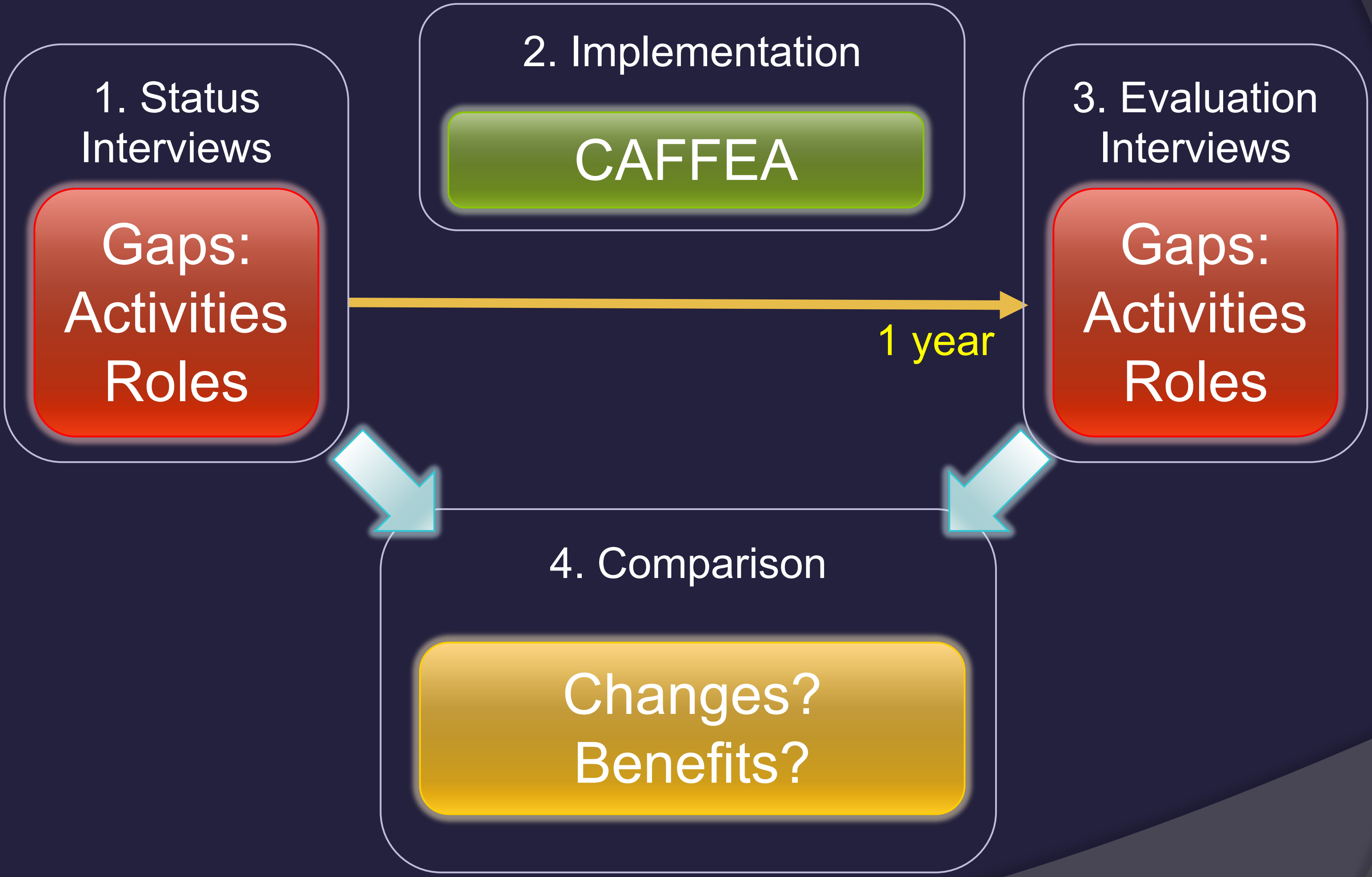
CAFFEA
Roles
Teams

Gaps:

Activities
Roles

Suitable?
Clarifications

Did it work? – Empirical Evaluation



1. Status Interviews

Gaps:
Activities
Roles

2. Implementation

CAFFEA

3. Evaluation Interviews

Gaps:
Activities
Roles

1 year

4. Comparison

Changes?
Benefits?

What we found– Gaps and CAFFEA

Findings: challenges in activities

- Risk management
- Architectural decisions and changes
- Providing architectural knowledge
- Monitor the current status of the system

Short-term or long-term value delivery?

What is worth changing in our architecture?

What qualities are really important?

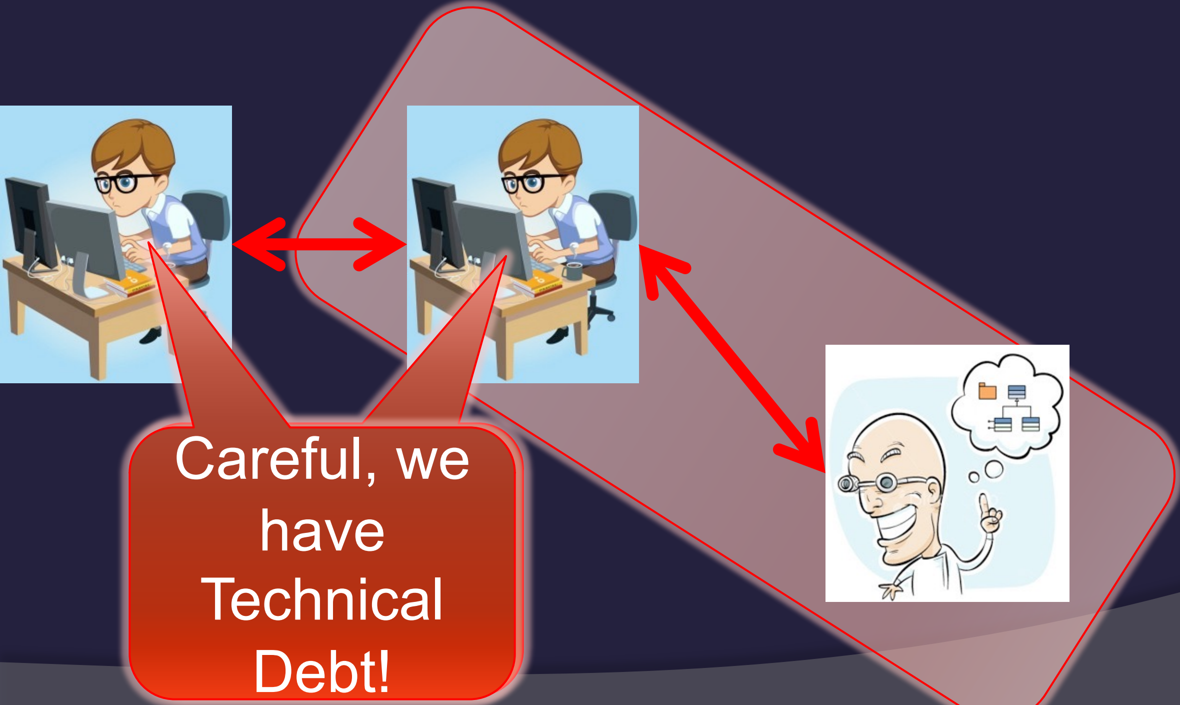
How much technical debt do we have?

Findings: roles in CAFFEA

- Missing activities (and needed!) due to:
  - Roles not present in the organization
  - Roles overloaded with activities
  - Roles not aware of the need for the activities
- Needed roles:
  - Chief Architect CA
  - Governance Architect GA
  - Team Architect TA

Findings: Teams in CAFFEA

- Risk management
- Architectural decisions and changes
- Providing architectural knowledge
- Monitor the current status of the system

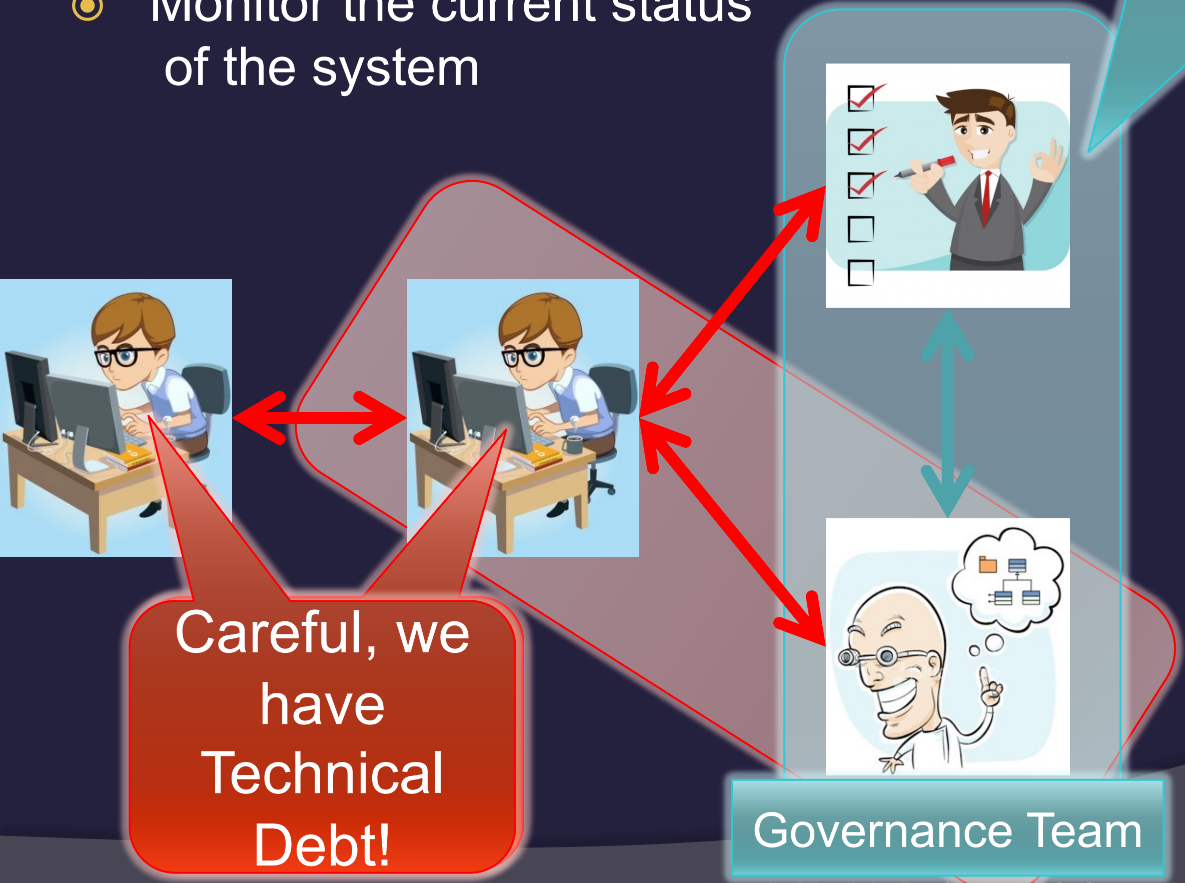


Careful, we have Technical Debt!

Findings: Teams in CAFFEA

- Risk management
- Architectural decisions and changes
- Providing architectural knowledge
- Monitor the current status of the system

What to do next? Refactoring or features?

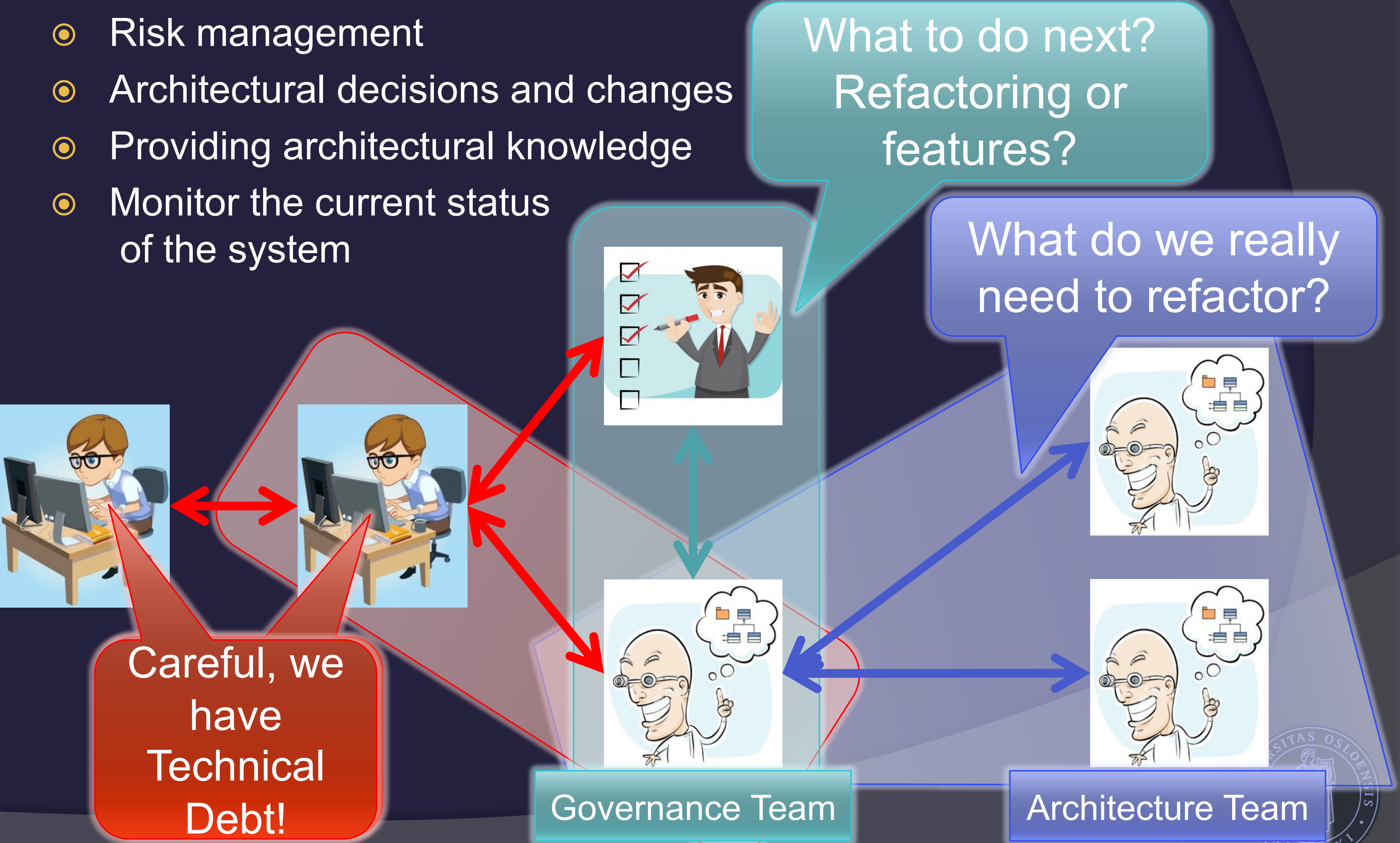


Careful, we have Technical Debt!

Governance Team

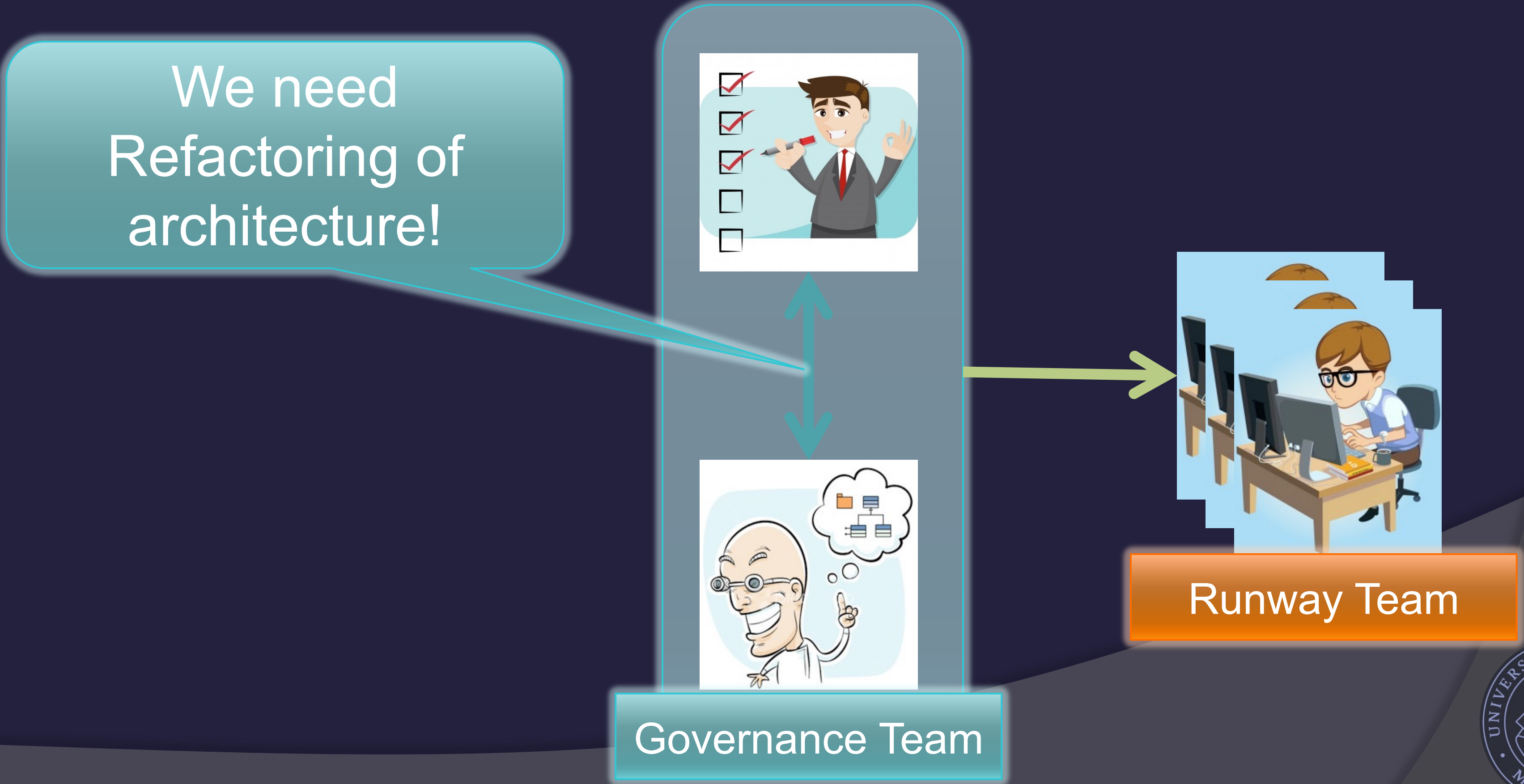
Findings: Teams in CAFFEA

- Risk management
- Architectural decisions and changes
- Providing architectural knowledge
- Monitor the current status of the system


What to do next? Refactoring or features?
What do we really need to refactor?
Careful, we have Technical Debt!
Governance Team
Architecture Team

Findings: Runway Team in CAFFEA

- A feature team dynamically appointed when the architecture needs improvement

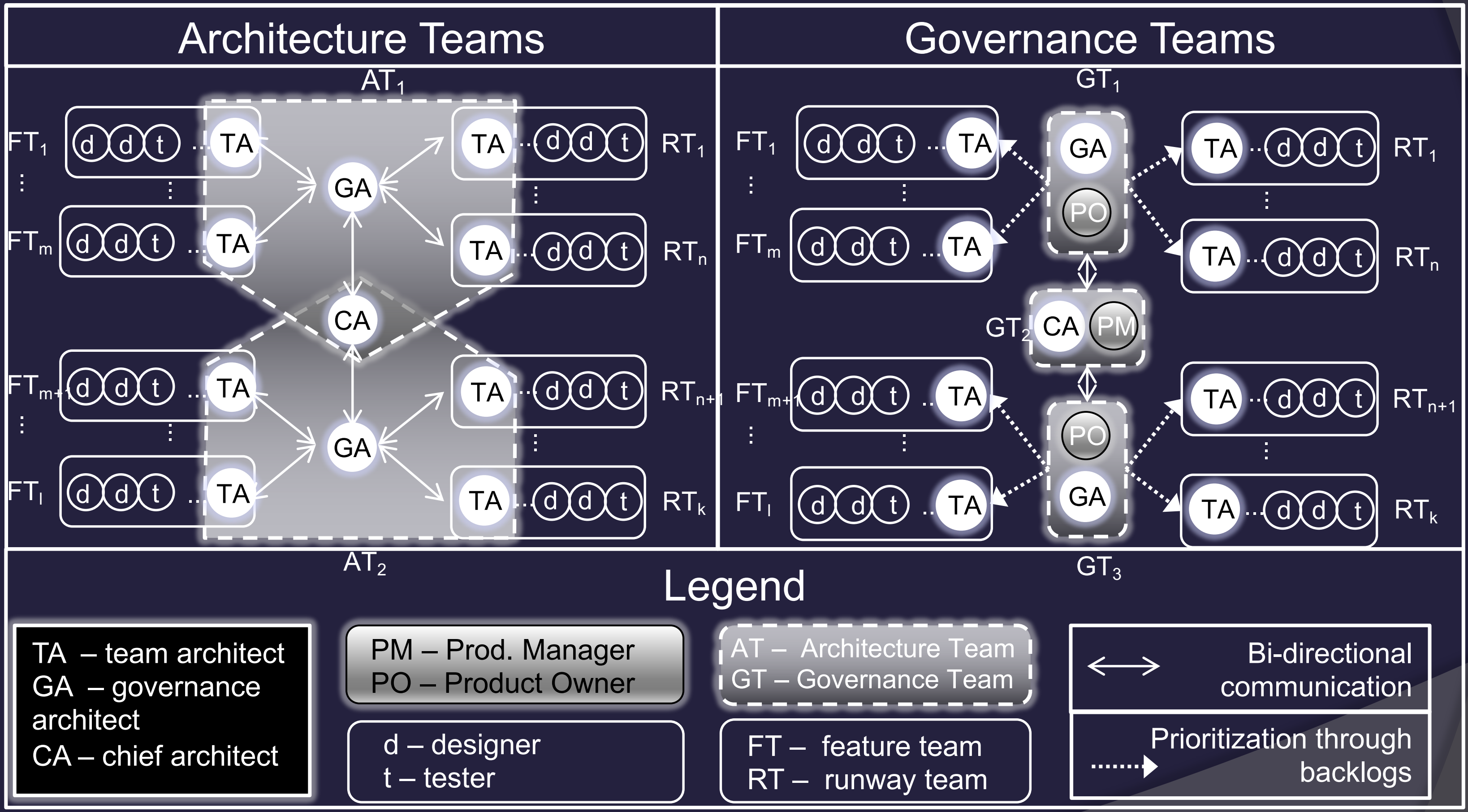


We need Refactoring of architecture!

Runway Team

Governance Team

Overall CAFFEA framework



Architecture Teams

AT₁

FT₁ d d t … TA

FTₘ d d t … TA

GA

TA … d d t RT₁

TA … d d t RTₙ

CA

FTₘ₊₁ d d t … TA

FTₗ d d t … TA

GA

TA … d d t RTₙ₊₁

TA … d d t RTₖ

AT₂

Governance Teams

GT₁

FT₁ d d t … TA

FTₘ d d t … TA

GA

PO

TA … d d t RT₁

TA … d d t RTₙ

GT₂ CA PM

FTₘ₊₁ d d t … TA

FTₗ d d t … TA

PO

GA

TA … d d t RTₙ₊₁

TA … d d t RTₖ

GT₃

Legend

TA – team architect
GA – governance architect
CA – chief architect

PM – Prod. Manager
PO – Product Owner

AT – Architecture Team
GT – Governance Team

⟷ Bi-directional communication

d – designer
t – tester

FT – feature team
RT – runway team

┈▶ Prioritization through backlogs

* A. Martini and J. Bosch, "A Multiple Case Study of Continuous Architecting in Large Agile Companies: Current Gaps and the CAFFEA Framework", Working International Conference on Software Architecture, 2016, Venice, Italy

Did it work? – Evaluation

Improvement in risk management

- Architectural Technical Debt discovered and managed
  - Architecture Team
- Long-term perspective
  - Governance Team
  - Allocation of Runway Team

Improvement in managing decisions

- Informal tracking of decisions during meetings
- Conflicts discovered and solved
  - Between architects related to different teams and different views
- Follow up on “bad” decisions

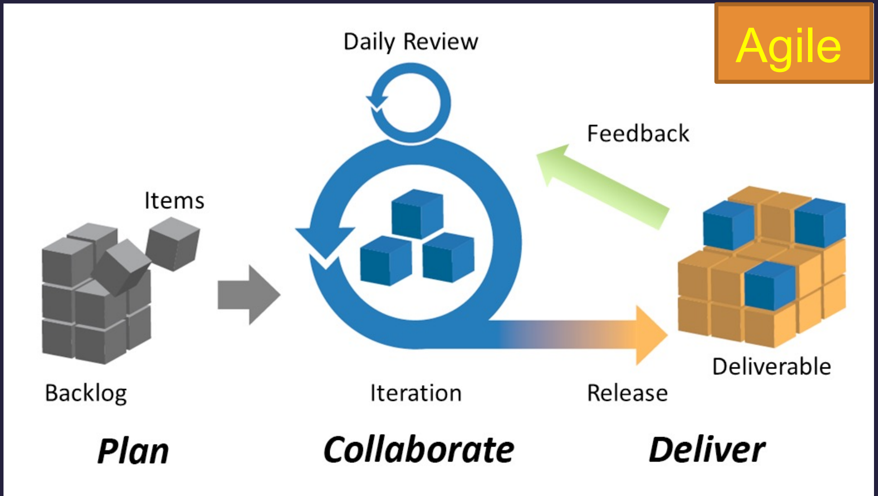
Improved communication

- Necessary inter-team socio-technical communication facilitated
  - Architecture Team as opportunity to share
    - Improvements
    - Needs
- Better communication about the current status of the system
- Enforcement and capillary distribution of Architectural Knowledge

Other improvements

- The formalized framework in place provided:
  - Clear knowledge references
  - Clear architecture responsibilities
  - Architecture activities not overlooked

Summary



Changes in

- Practices
- Organization

Lack of focus

- Architecture

Compliant

CAFFEA Framework

- Roles
- Teams
- Responsibilities

Improves

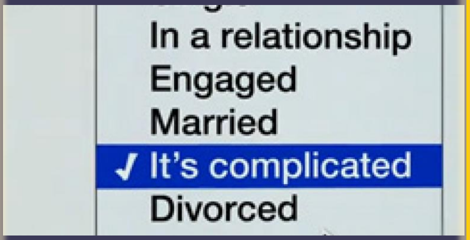
Gaps in Architecture Practices

- Risk management
- Architectural decisions and changes
- Providing architectural knowledge
- Monitor the current status of the system

Take aways

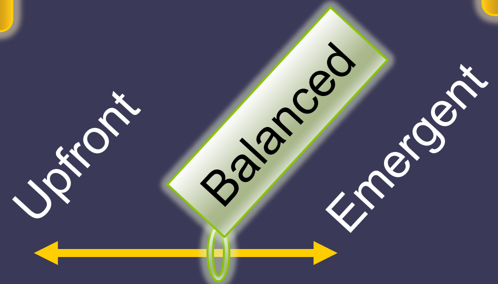
Summary

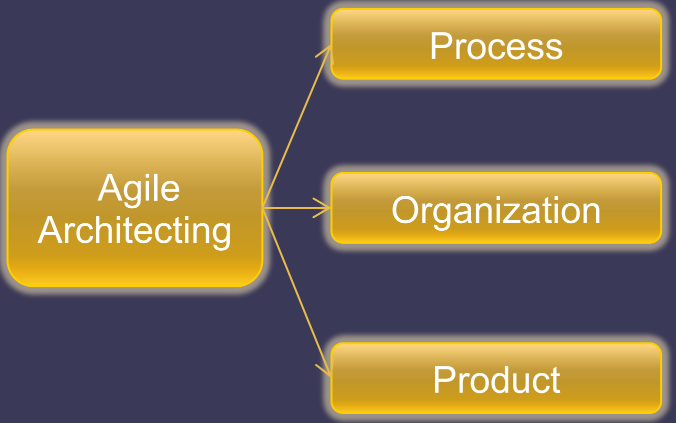
Architecture

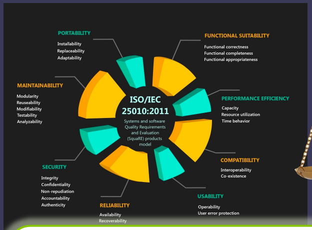


In a relationship
Engaged
Married
✓ It's complicated
Divorced

Agile

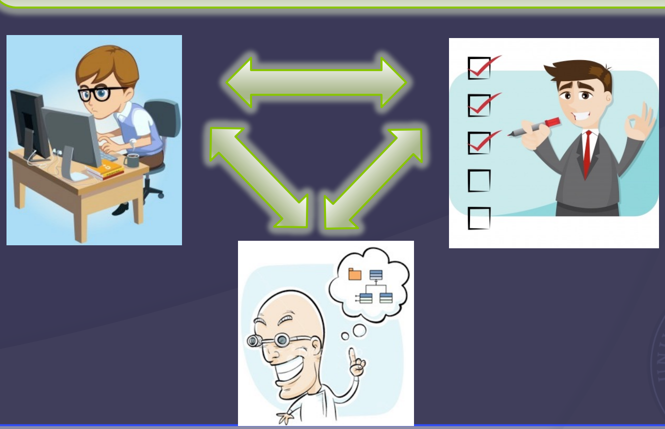




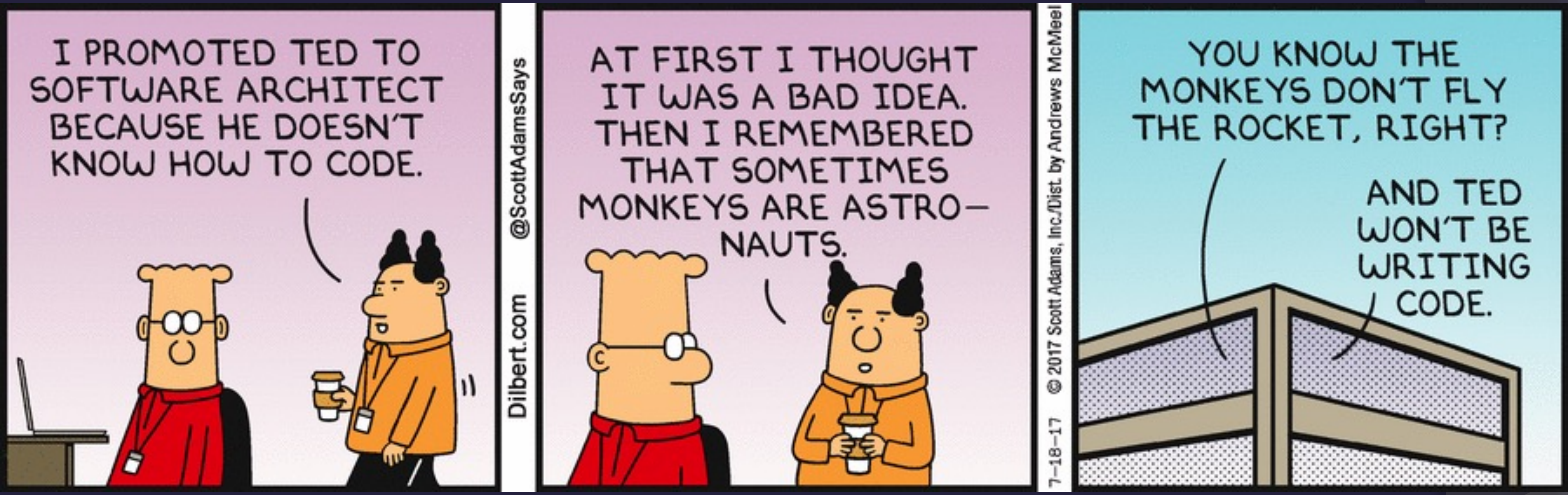


- Continuously re-prioritize architectural concerns according to risks
- Use frameworks

Communication among roles and stakeholders is key



Don’t underestimate architecture (and architects)…


I PROMOTED TED TO SOFTWARE ARCHITECT BECAUSE HE DOESN'T KNOW HOW TO CODE.
Dilbert.com
@ScottAdamsSays
AT FIRST I THOUGHT IT WAS A BAD IDEA. THEN I REMEMBERED THAT SOMETIMES MONKEYS ARE ASTRO– NAUTS.
7-18-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel
YOU KNOW THE MONKEYS DON'T FLY THE ROCKET, RIGHT?
AND TED WON'T BE WRITING CODE.

Questions?

Comments?

- antonio.martini@ifi.uio.no