PRACTICES FOR EFFECTIVE CONTINUOUS ARCHITECTURE

EOIN WOODS

iSAQB Software Architecture Gathering

November 2022

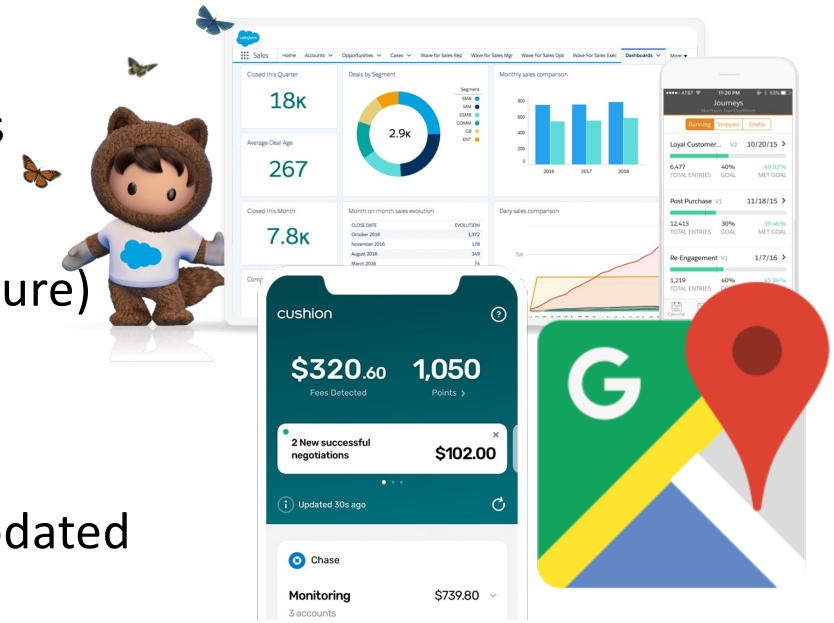
Eoin Woods

- Endava's CTO, based in London (since 2015)
- 10+ years in products - Bull, Sybase, InterTrust
- 10 years in capital markets - UBS and BGI
- Software developer, architect, now CTO
- Author, editor, speaker, community guy

THE CHANGING FACE OF SOFTWARE DEVELOPMENT

Software Development in the age of Digital Platforms

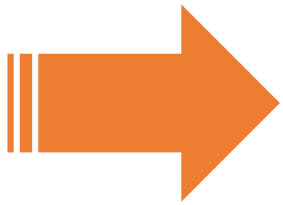
- Platforms are ”**always on**”
- Platforms are built to **evolve** in unpredictable ways
- Some **intelligent behaviour** is becoming expected
- **Continual updates** to the software (and infrastructure)
- **Parallel developments** occur on a platform
- Platforms are **highly connected**
- **Multiple interfaces and audiences** to be accommodated
- Platforms must be **extensible by design**



A multi-dimensional software engineering challenge!

So Software Practice Evolved

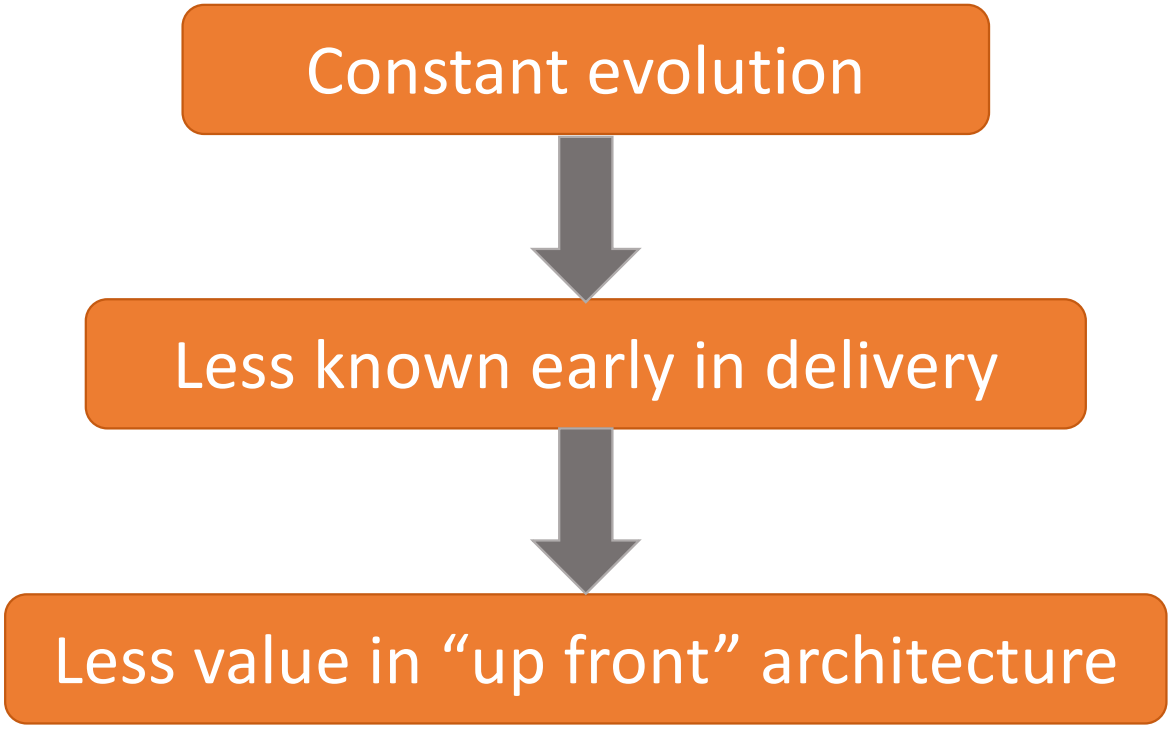
- DevOps (and SRE)
- Reusable Cloud Services
- Microservices and Serverless
- Cloud Infrastructure
- Agile Working



FLUID
EVOLVING
ARCHITECTURE

endava

Traditional software architecture is of less value

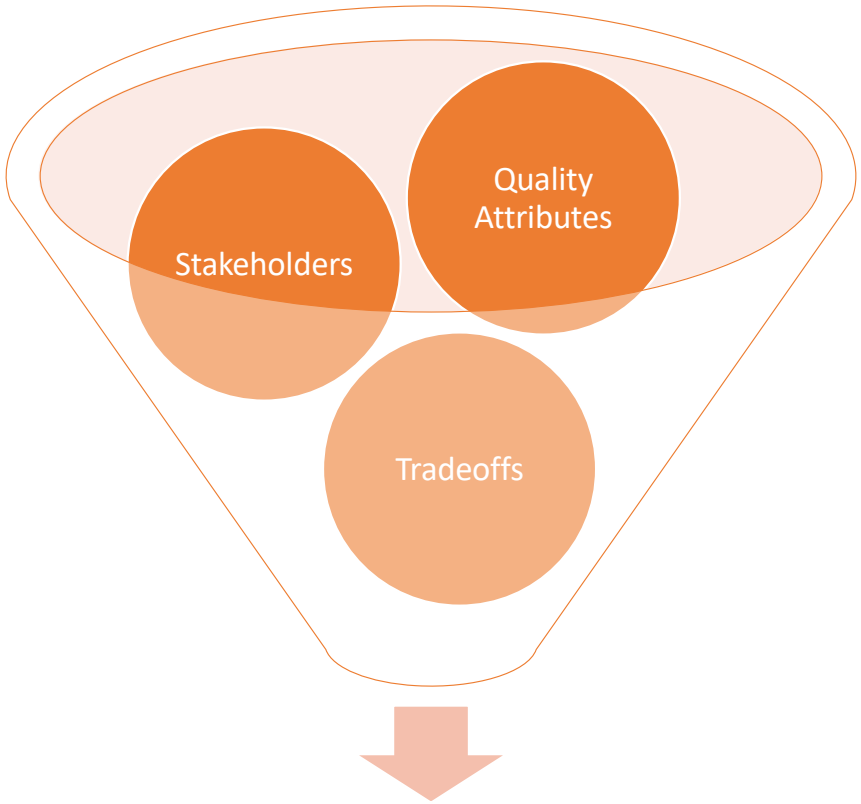


endava

INTRODUCING CONTINUOUS ARCHITECTURE

endava

Is architecture still needed? Yes!

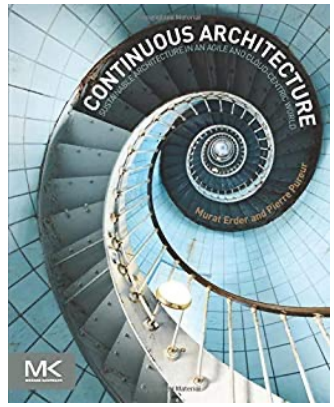


- Achieving **quality attributes**
- Balancing **stakeholder** concerns
- Making complex **tradeoffs**
- Achieving **cross cutting concerns** across many independent parts

But architecture is now a continual **flow of decisions**

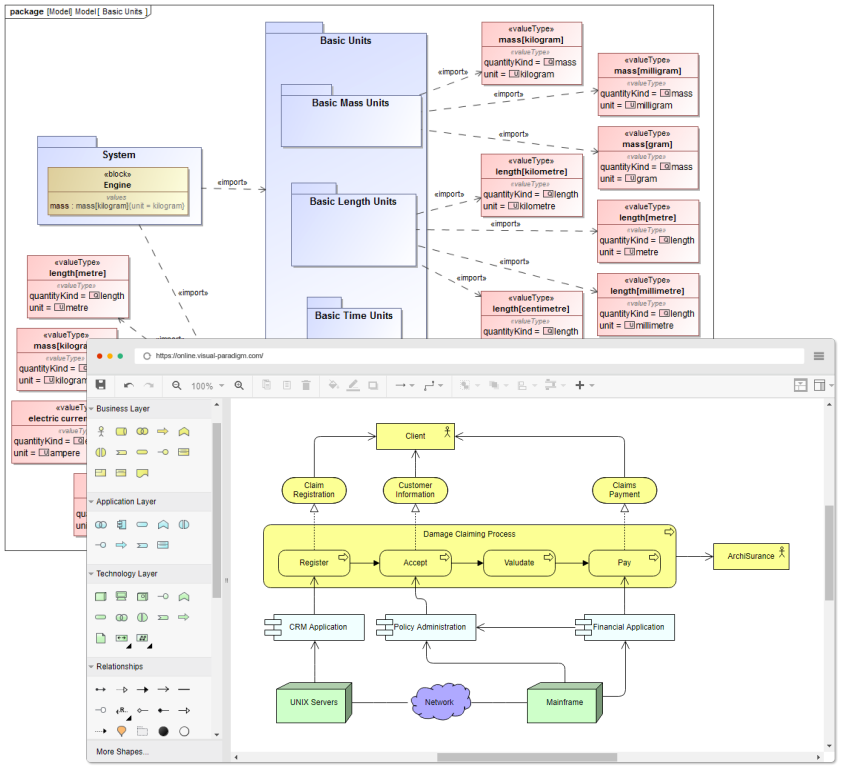
Continuous Architecture Principles

- **Principle 1**: Architect **products**: evolve from projects to products
- **Principle 2**: Focus on **quality attributes**, not functional requirements
- **Principle 3**: Delay **design decisions** until absolutely necessary
- **Principle 4**: Architect for **change** – leverage the "power of small"
- **Principle 5**: Architect for **build, test, deploy and operate**
- **Principle 6**: Model the **organisation** of your teams after the design of the system

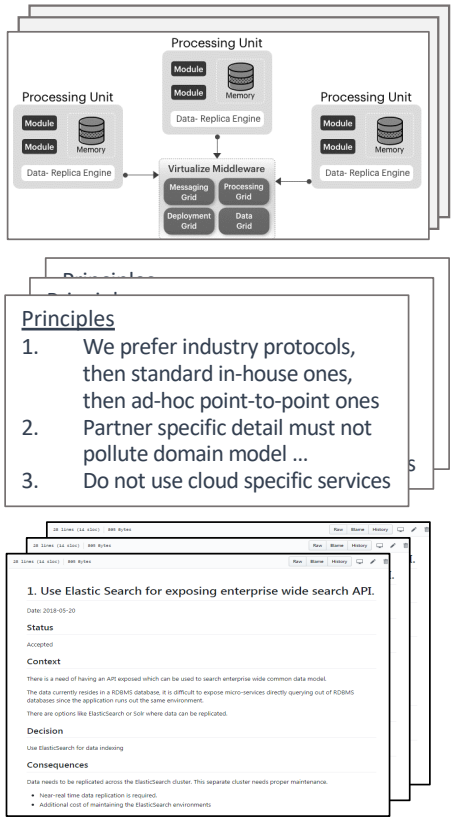


Murat Erder &
Pierre Pureur, 2015

Moving to Continuous Architecture



Top Down Prescriptive Design



Styles & Patterns

Principles

Decisions

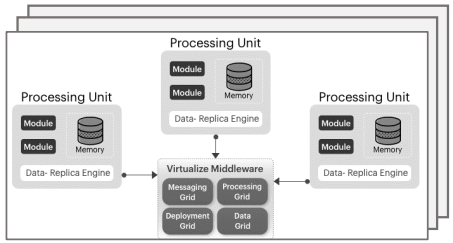
Evolving Shared Design

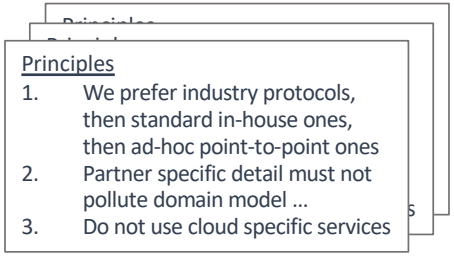
Artifacts of Continuous Architecture

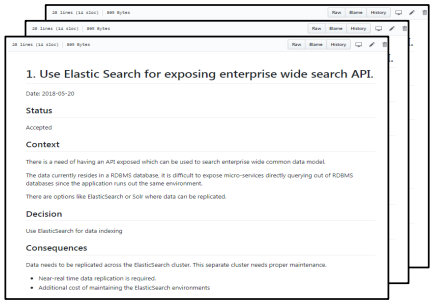
Styles & Patterns:
Common solutions to repeating problems

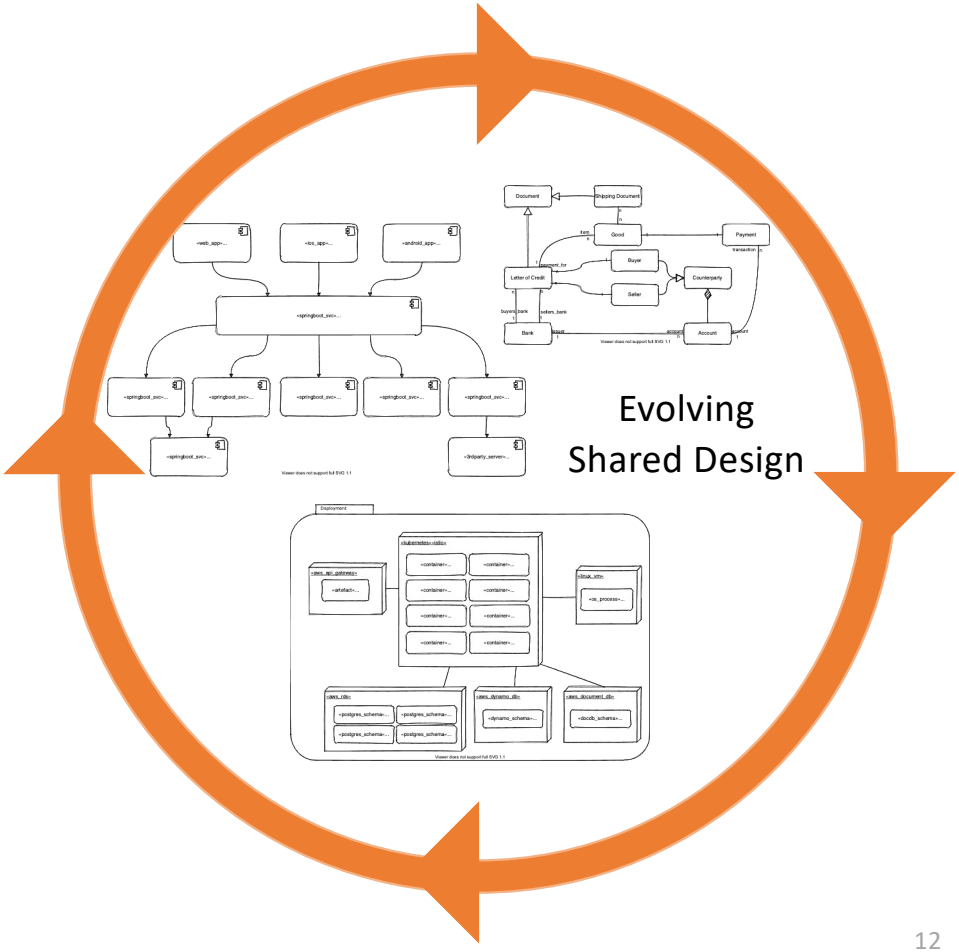
Principles:
Guidance to achieve aligned design decisions

Decisions:
Understanding what we did, when and why









THE ACTIVITIES OF CONTINUOUS ARCHITECTURE

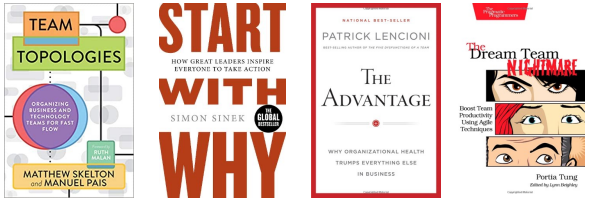
endava

Essential Continuous Architecture Activities

Provide Leadership

- **Leading** rather than ”managing”
  - Get the team doing the architecture work
- Lead the **resolution** of technical concerns
  - What are the key architecture principles?
  - Which option to choose when none is perfect?
  - What can be deferred, what needs to be done now? (Tech debt)
  - What are the options when something ”impossible” is needed?
- Constant progress towards **technical excellence**
  - What are the right ways of working?
  - What are the long-term consequences of decisions?
  - ”the conscience of the team”
- Represent the **technical view** ”upwards”
  - “difficult conversations”

https://www.pexels.com/@fauxels



Leadership: a Technical Leadership Group

- **Form a team** to own the architecture & technical concerns
  - Technical Leadership Group, Tech Leads Group, …
- Spreads **knowledge** across the team
- **Empowers** others to take responsibilities with support
  - Topic champions for security, performance, resilience, …
- Provides technical **growth** opportunities
- Allows **multiple perspectives** for decision making
- Frees up **your** time
  - But there does need to be a decision making mechanism (usually you)

https://www.pexels.com/@fauxels

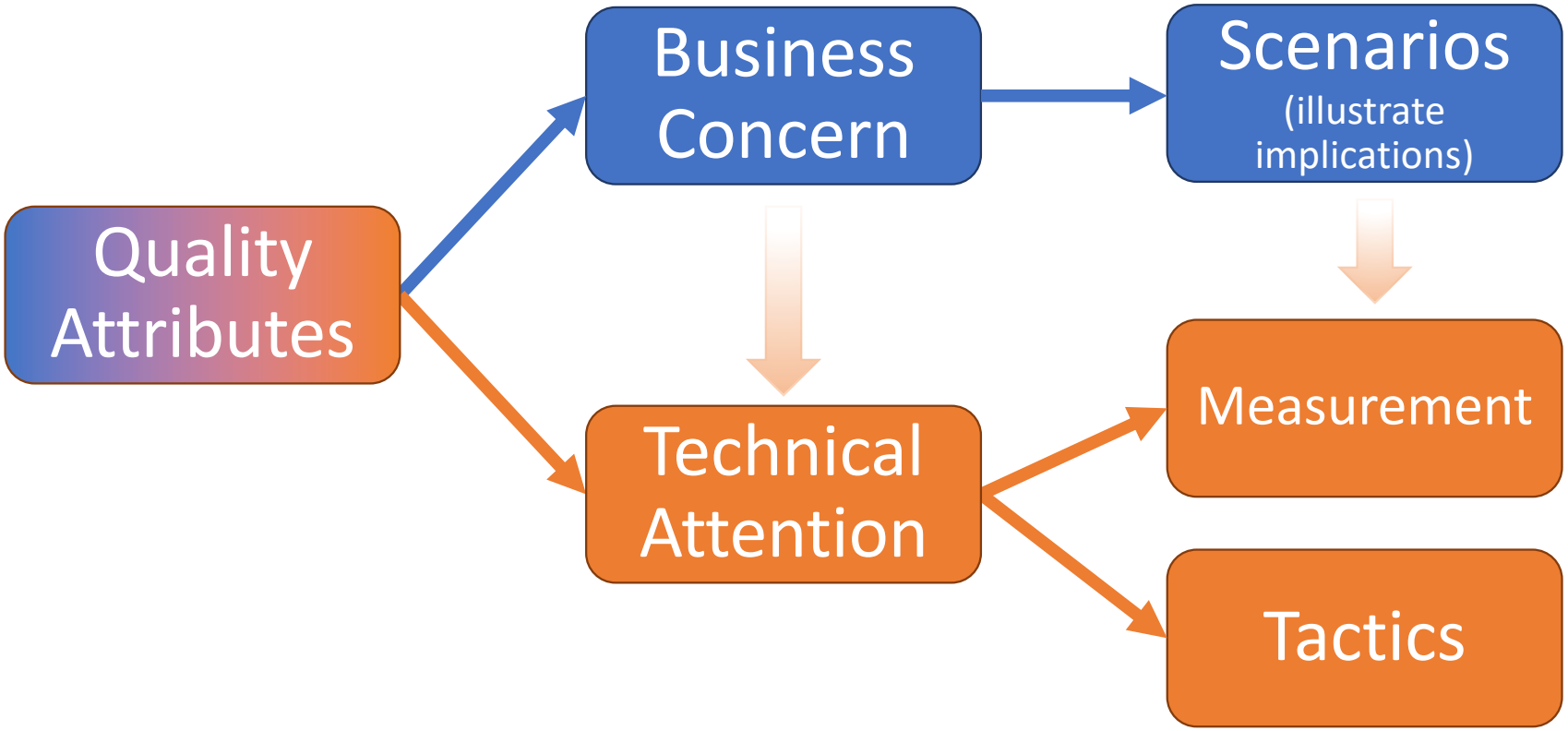
Focus on Quality Attributes

- Reminder probably unnecessary for quality attributes!
  - “the architect’s obsession”
  - Constant process of balancing tradeoffs
- **Cross-cutting concerns** that need system level attention
  - The question is how to get attention for them?
- Most **teams and product owners** are obsessed by **features**
  - ”how many stories in this sprint?” (meaning ”features”)
- Quality attributes often **not needed ”this sprint”**
  - Stores up potential problems for later
- How do we **integrate** them into Continuous Architecture?

https://pixabay.com/users/publicdomainpictures-14

endava

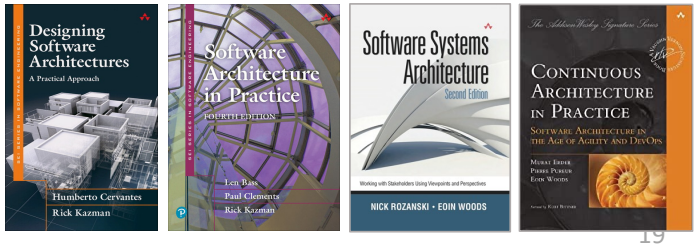
Focus on Quality Attributes



Quality Attributes Architectural Approach

- Get attention using **scenarios** – make the need clear
  - Stimulus + Response + Measurement (+ implications if needed)
  - Consider using a Quality Attribute Tree to organise them
- **Prioritisation** is key and an important architectural activity
  - Working across stakeholder groups to find an **acceptable balance**
  - **Security**, **performance**, **scalability**, **resilience** are normally important
  - Evolution (maintainability, flexibility) is normally assumed
- Help the team to break down these huge **goals into stories**
  - Implementation of architectural **tactics, styles & patterns**
- Make sure the stories get **into the backlog** – PO conversations
- **Own** some of the **difficult** ones (e.g. resilience & BCP)
  - Find people to be "**champions**" for the others

https://pixabay.com/users/publicdomainpictures-14



Drive Architectural Decisions

- Less detailed models means **decisions and principles** become more important – new **first-class artefacts** of architecture
- Architectural decisions are how we …
  - achieve **quality properties** (via tactics)
  - make **tradeoffs**
  - manage **technical debt**
  - achieve **sustainable delivery**
  - maximise **value**

https://www.pexels.com/@karolina-grabowska

Actually they always were …
we used to just hide them in our models!

Drive Architectural Decisions

- **Making**, **validating**, **managing** and **implementing** decisions is core to doing architecture continuously
- We must ensure that **good decisions** are made
  - Practical, logical, balanced, well-argued, well-communicated
- Ensure decisions **align with our architecture principles**
  - Or cause the principles to evolve
- Decisions must be **captured**, **understood**, **implemented** and **curated**

https://www.pexels.com/@karolina-grabowska

Drive Architectural Decisions

- Use the technical leadership group to:
  - **Make** decisions
  - **Validate** decisions
  - **Communicate** decisions
  - **Implement** decisions
- **Capture** decisions in an **accessible** way
  - ADRs in Git appear to have become something of a standard
    - https://adr.github.io/ and https://github.com/npryce/adr-tools
  - Simple wiki pages with a well defined format also work well
- **Curating** decisions over time is important
  - Control the number & organise the catalogue
  - Revalidate and remove obsolete decisions
  - Feedback into the architecture principles

Remember: you need to make sure good decisions are made, not make all the decisions!

https://www.pexels.com/@karolina-grabowska

Manage Technical Debt

Technical debt is a well established yet nebulous concept …

… very context specific

One person's "debt" is another person's "simplest thing possible"

Hard coded validation rather than a chain-of-responsibility of validators.
Debt? Or simple and effective?

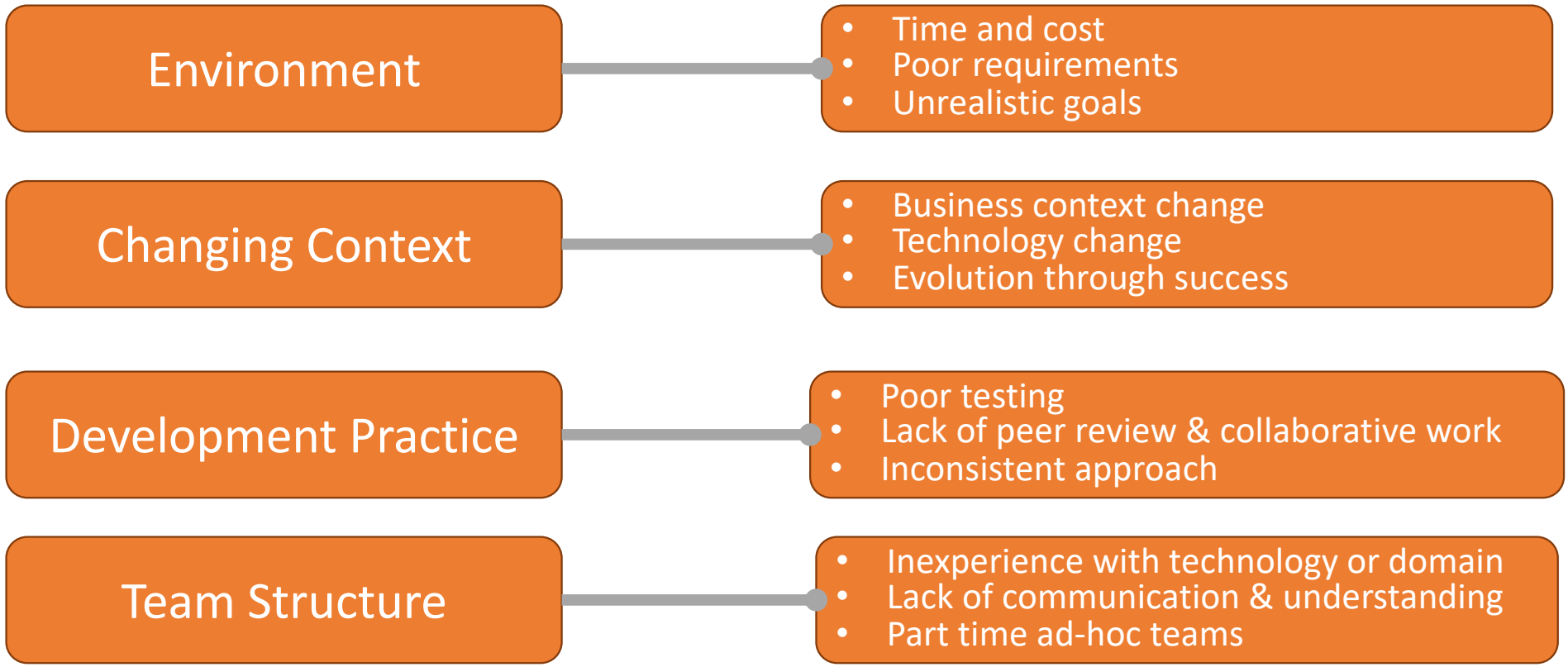
Does Technical Debt Matter?

Technical debt matters when it stops us doing something …

… it is now too expensive to make a change
… we are too slow to react to a need
… our team is too inefficient to be valuable
… it is too risky to update our technology

It is these situations that the architect needs to be looking ahead for, to predict and avoid

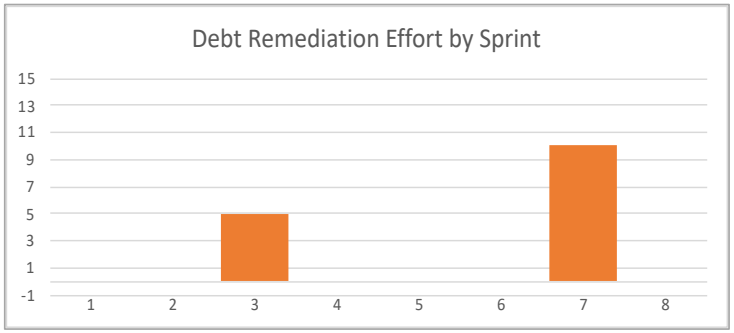
Sources of Technical Debt

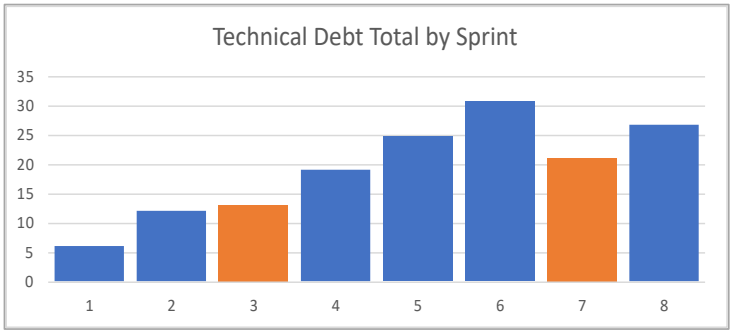


Source: Managing Technical Debt, Kruchten, Nord, Ozkaya

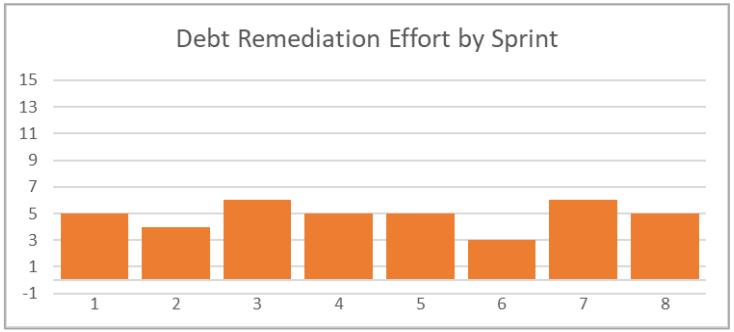
Dealing With Technical Debt

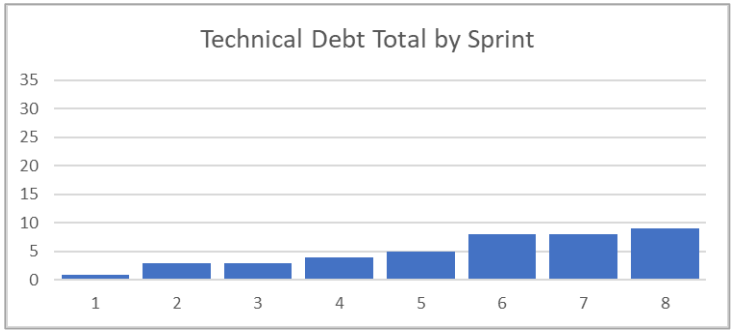
Just-in-Time



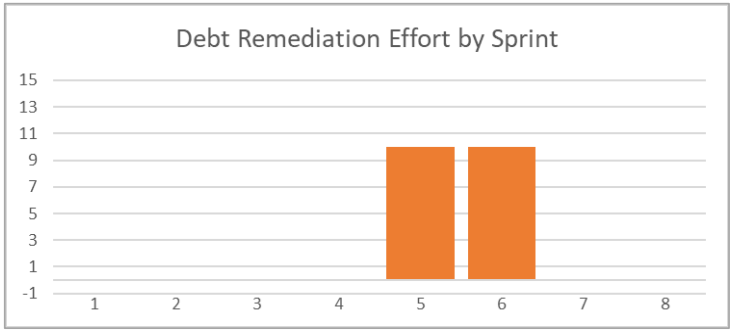


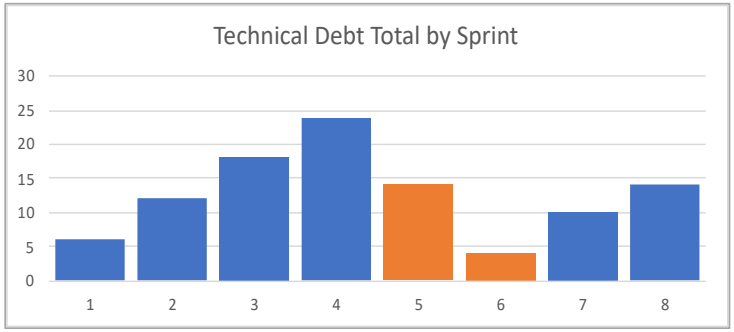
Little and Often



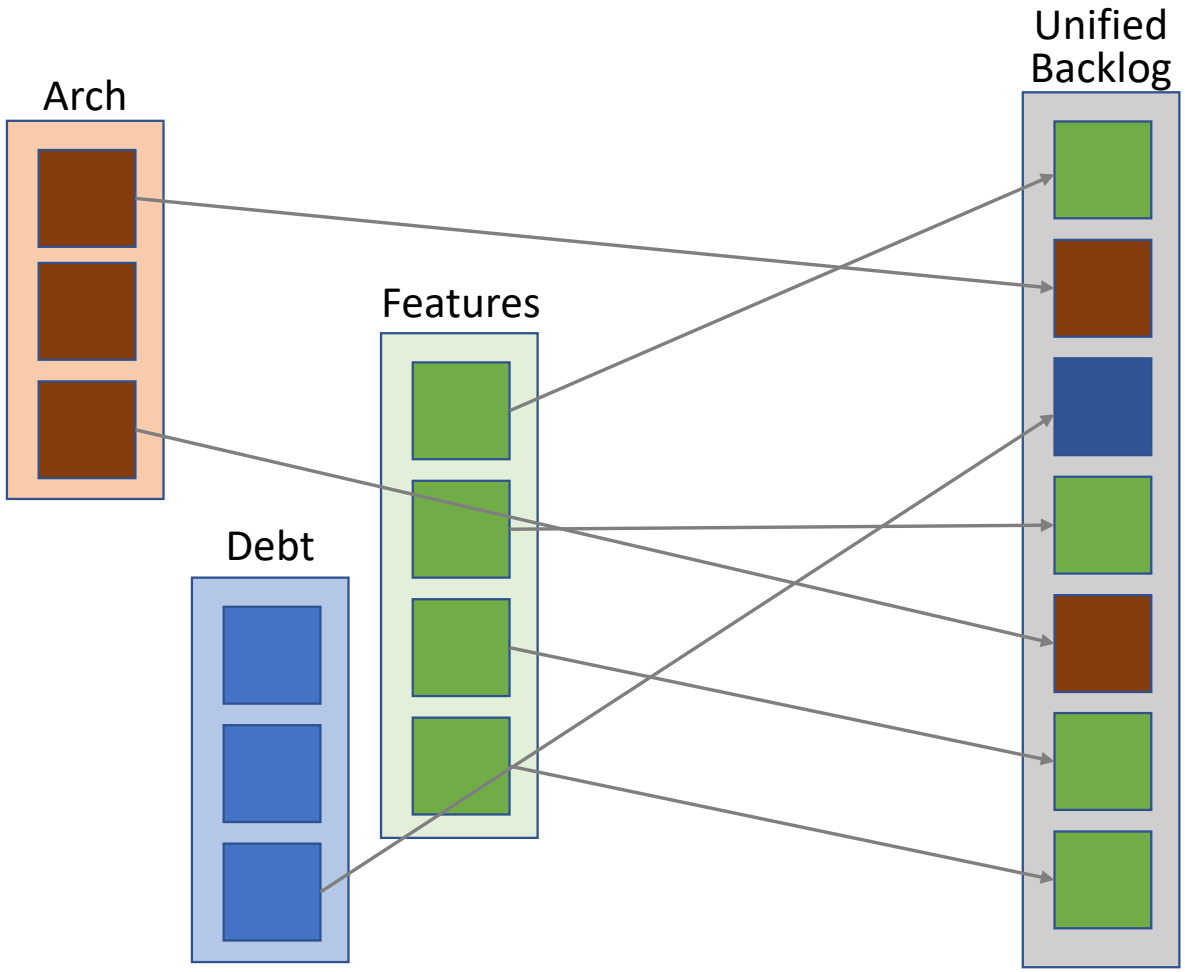


Cleanup

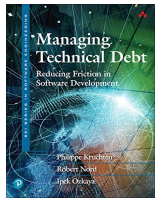




Dealing With Technical Debt

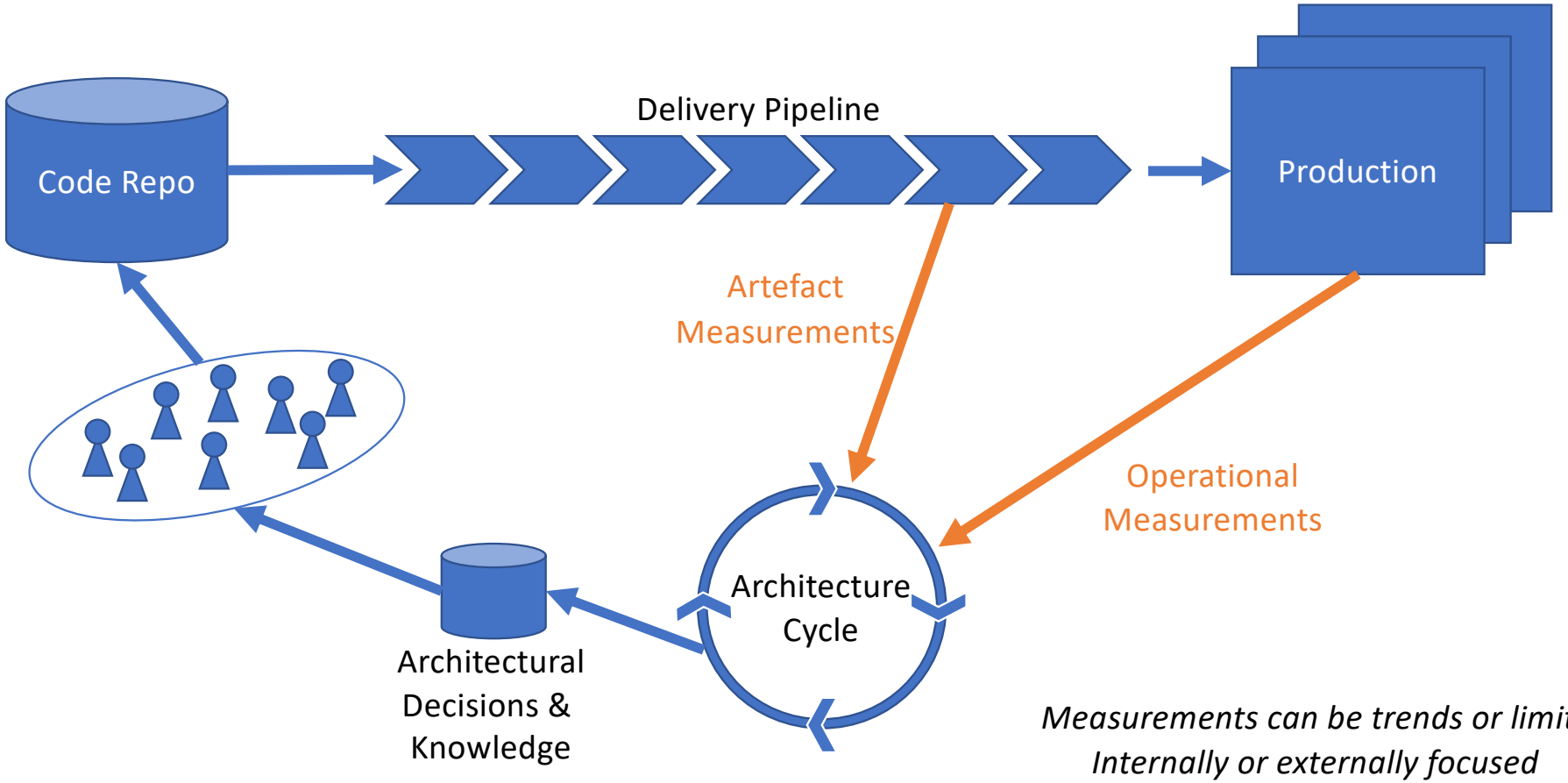


Unified backlog for visibility and prioritisation



endava

Implement Feedback Loops

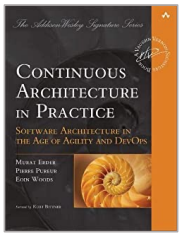


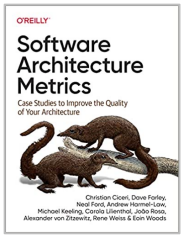
Measurements can be trends or limits
Internally or externally focused
Good ones provide architectural "reality checks"

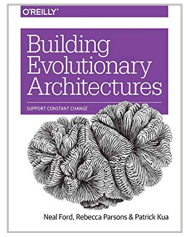
Implement Feedback Loops

- Feedback loops are your ”**architectural reality check**”
- **Automated**, **semi-automated** and **manual** all have their place
- Typically measure **quality attributes** but can be functional
- **Internal** (e.g. code complexity) and **external** (e.g. API response time) are both important
- Start **small and simple**, targeting biggest risks or concerns
- Over time the implementation can become **complex**
  - Don’t fall in love with your feedback loop implementation!

https://unsplash.com/photos/DKSWyxtcPVQ

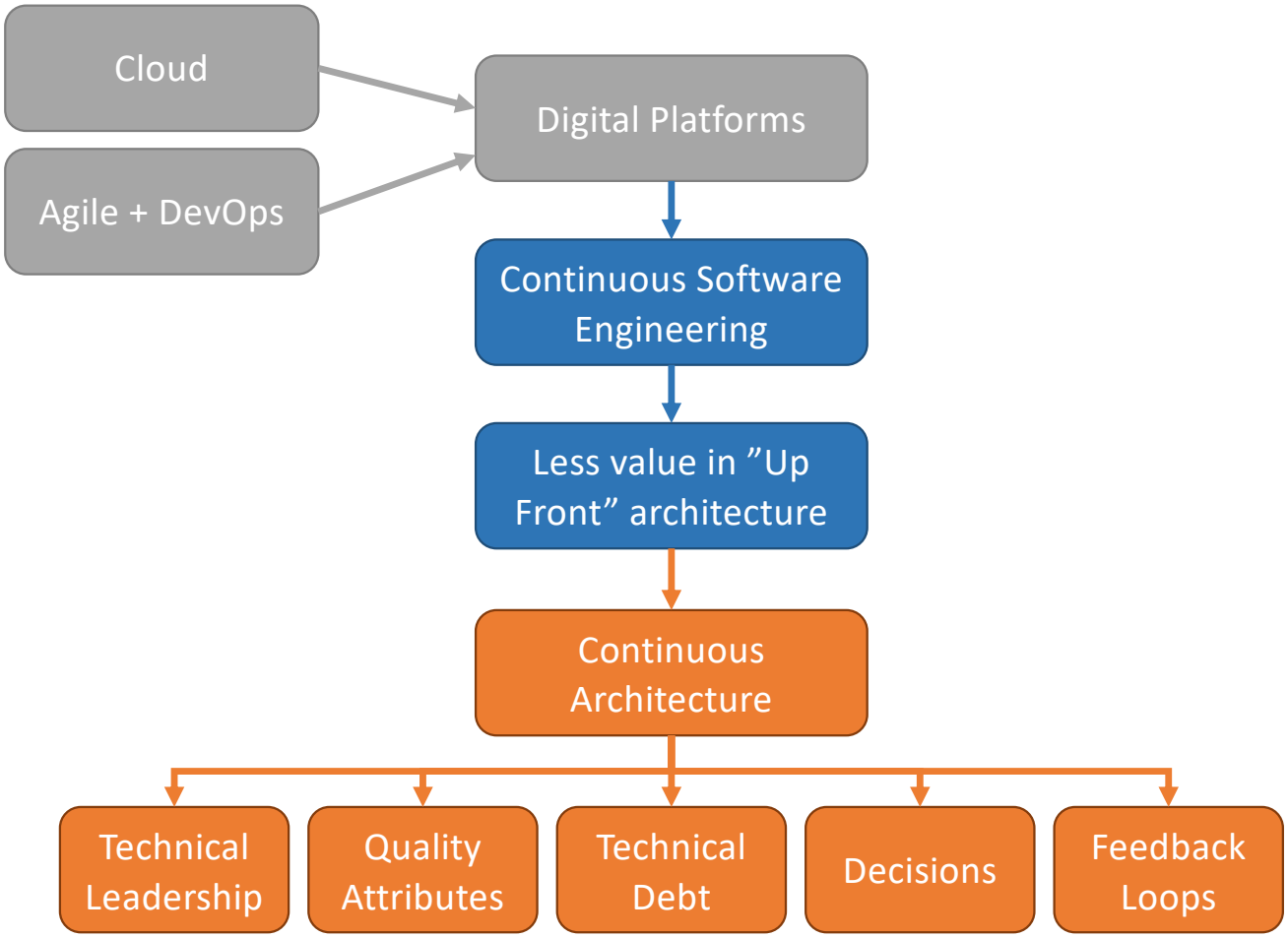






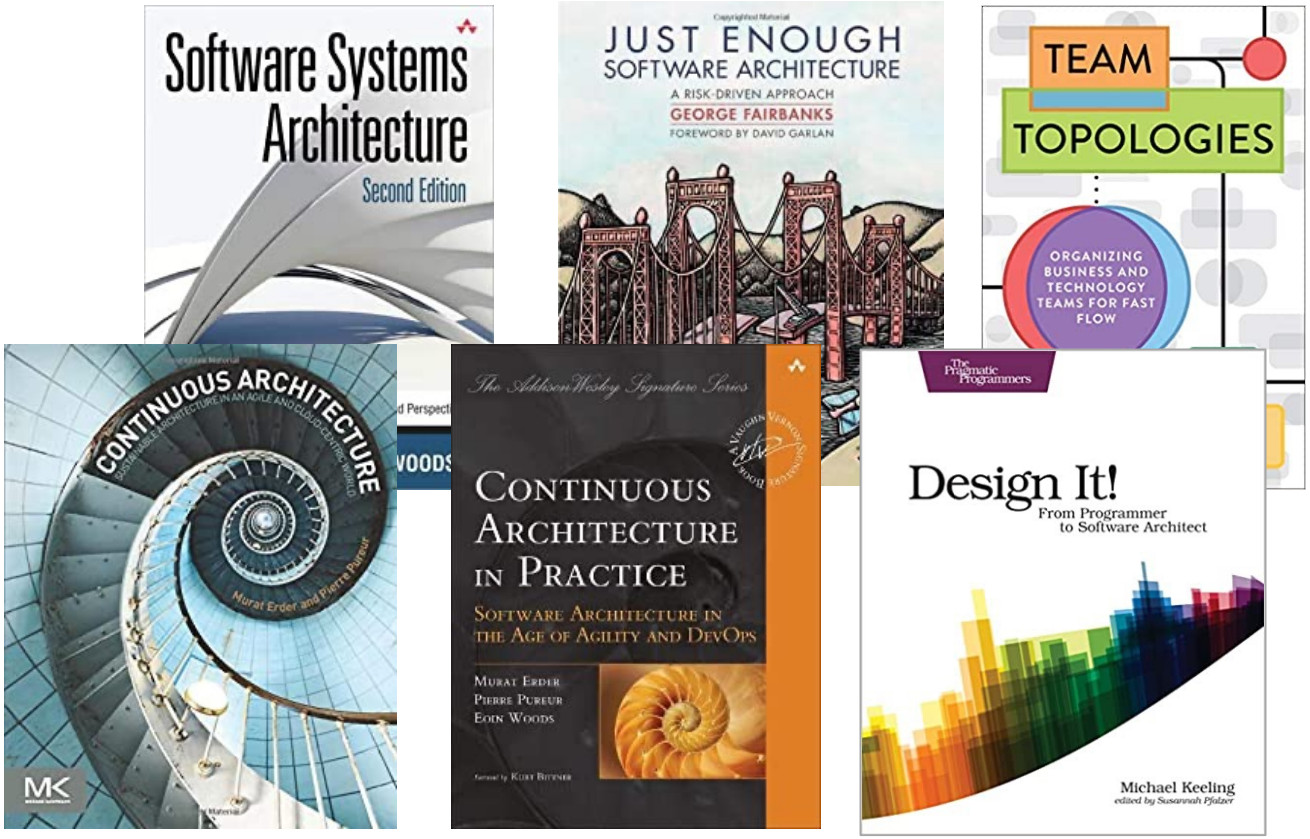
TO CONCLUDE

In Conclusion



https://unsplash.com/photos/NOBZdtTTGrg

To Find Out More



continuous-architecture.com

continuousarchitecture.info

endava

THANK YOU … QUESTIONS?

Eoin Woods
Endava
eoin.woods@endava.com
@eoinwoodz and @eoinwoods@mastodonapp.uk