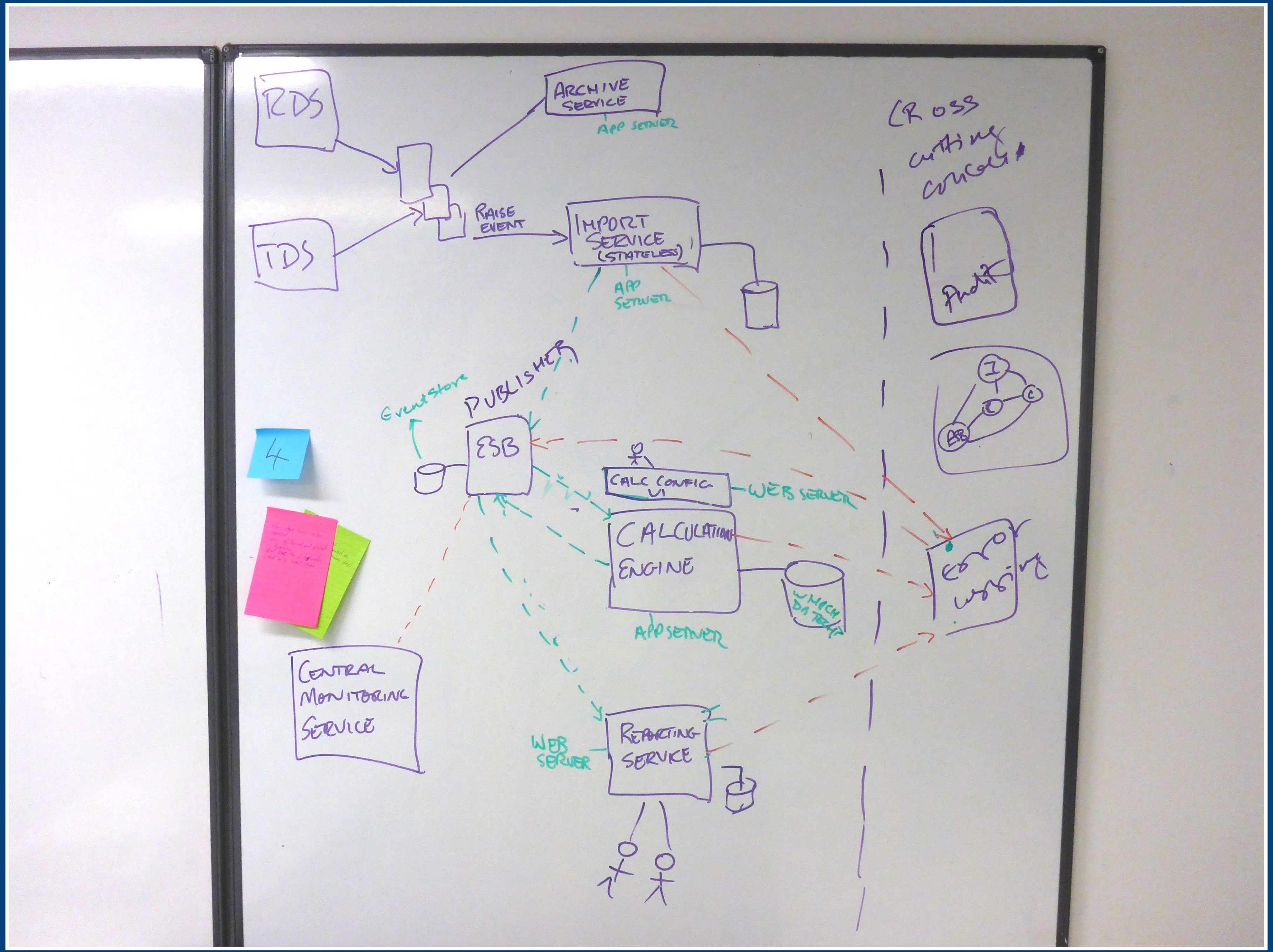
Five things every developer should know about **software architecture**

Simon Brown

1. Is that what we're going to **build**?



2. Is it going to **work**?

1. Software architecture isn't about big design up front

Historically there’s been a tendency towards **big design up front**

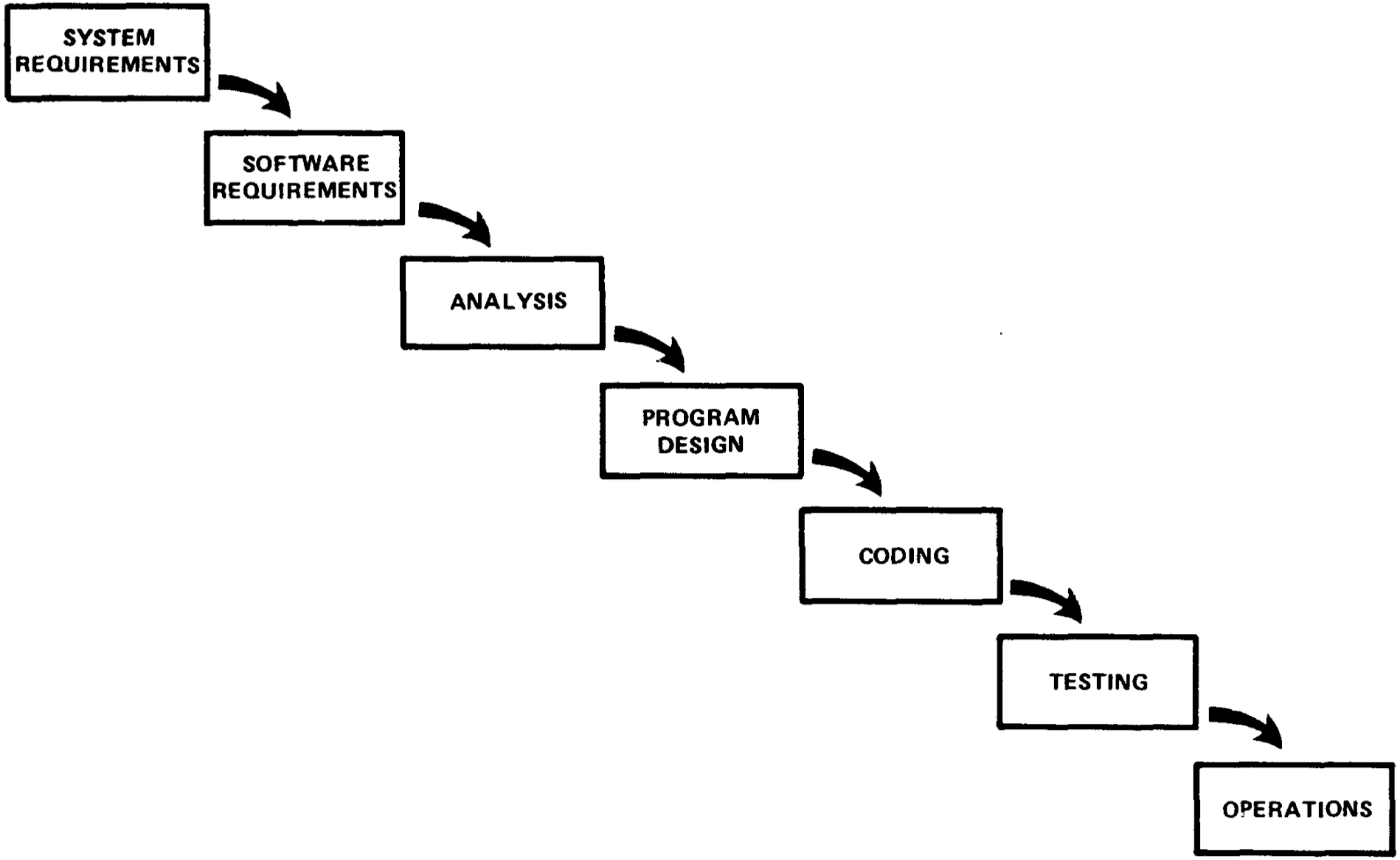


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

I believe in this concept, but the implementation described above is risky and invites failure.

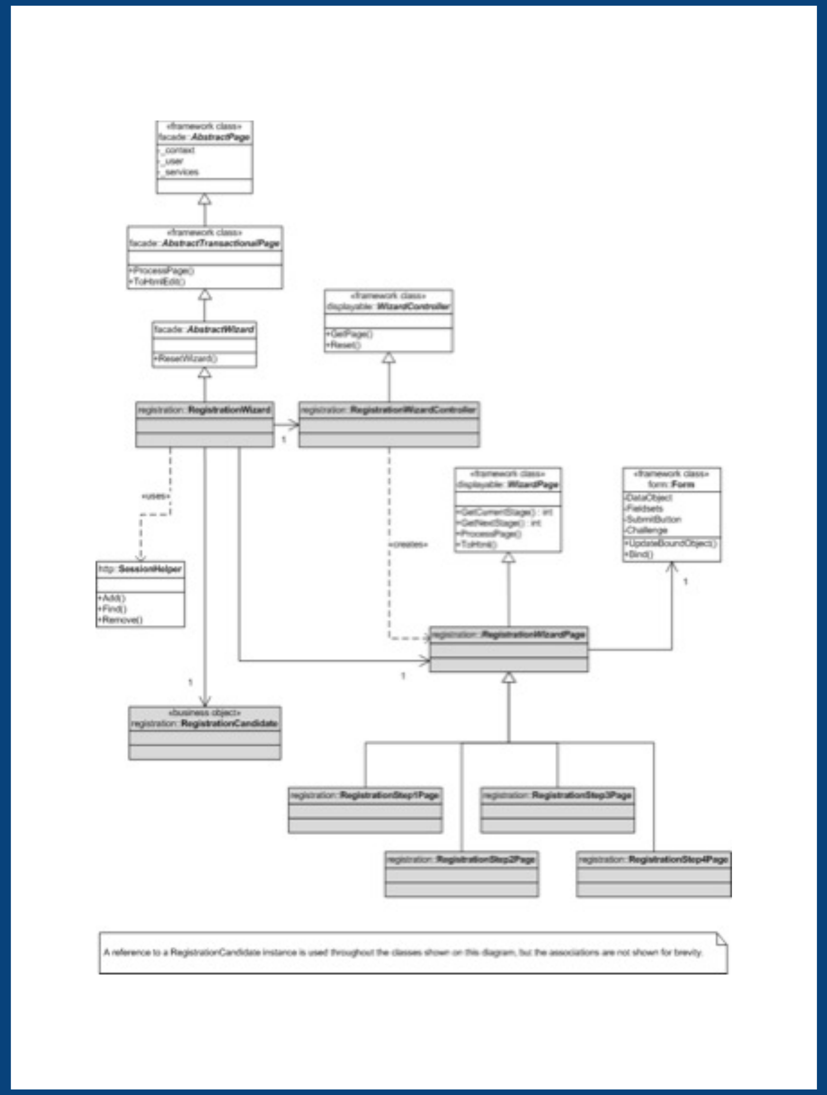
Managing the development of large software systems

Dr Winston W. Royce

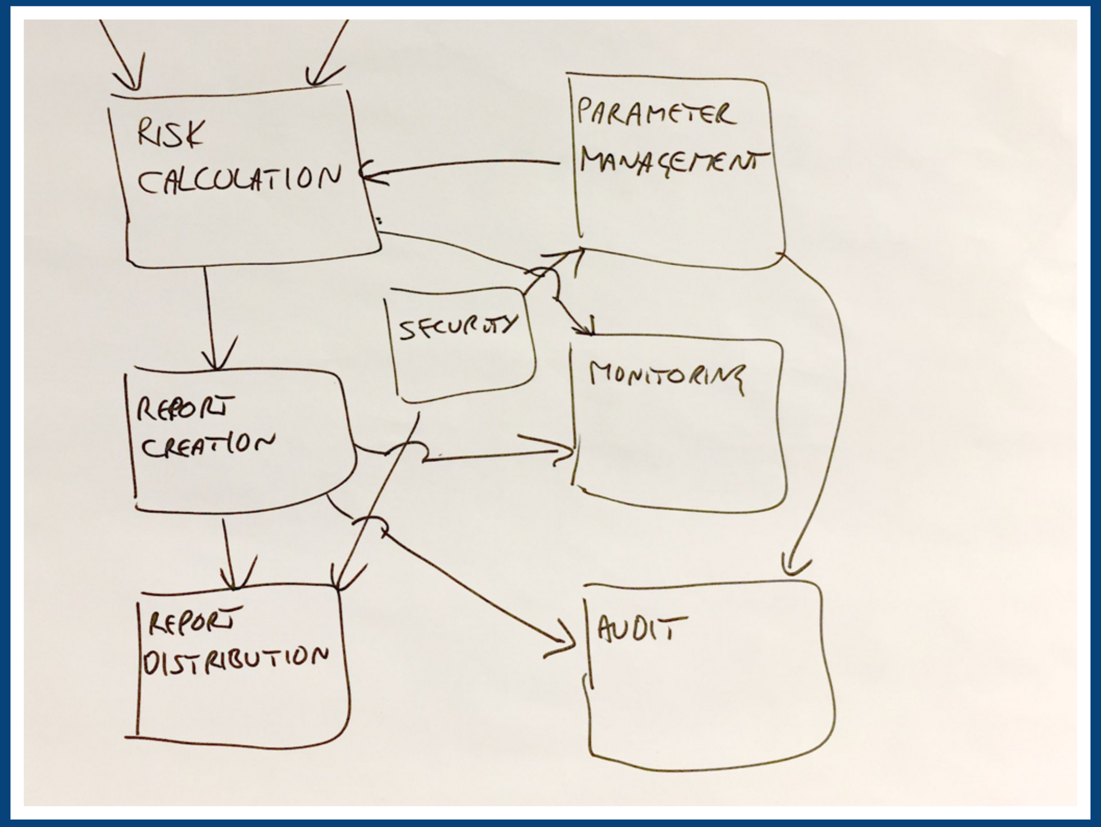
Responding to change
over
following a plan

Big design up front

Software Architecture Document



vs



No design up front

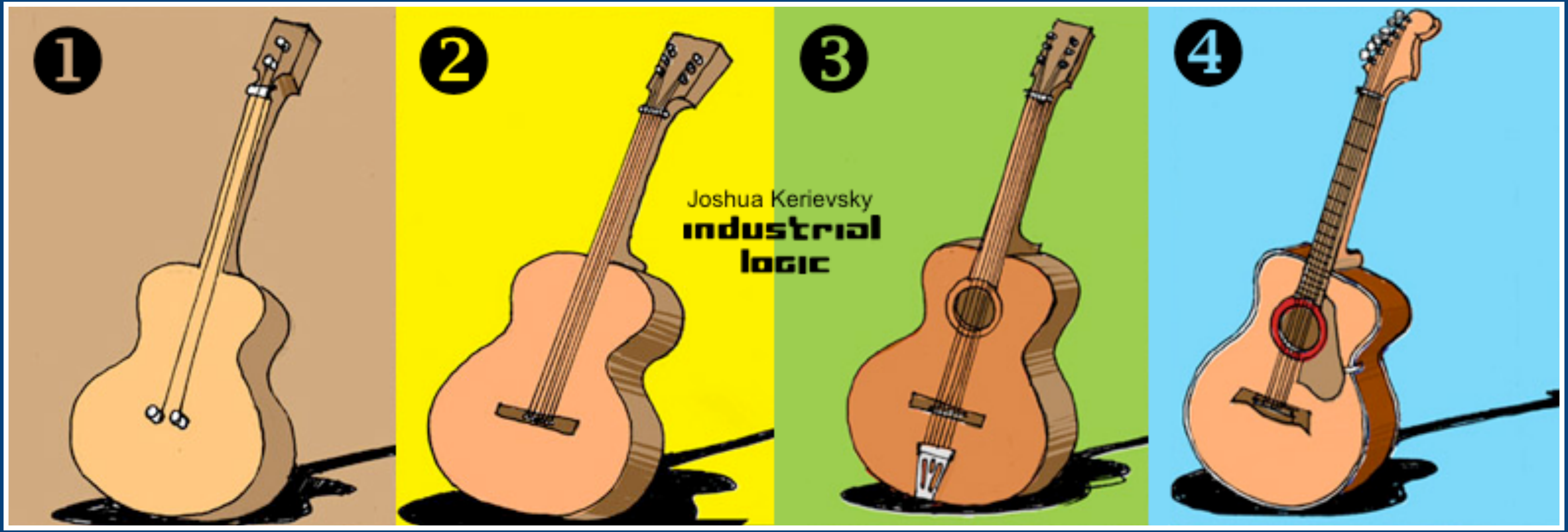
Big design up front is dumb. Doing no design up front is even dumber.

Dave Thomas

How much **up front design** should you do?

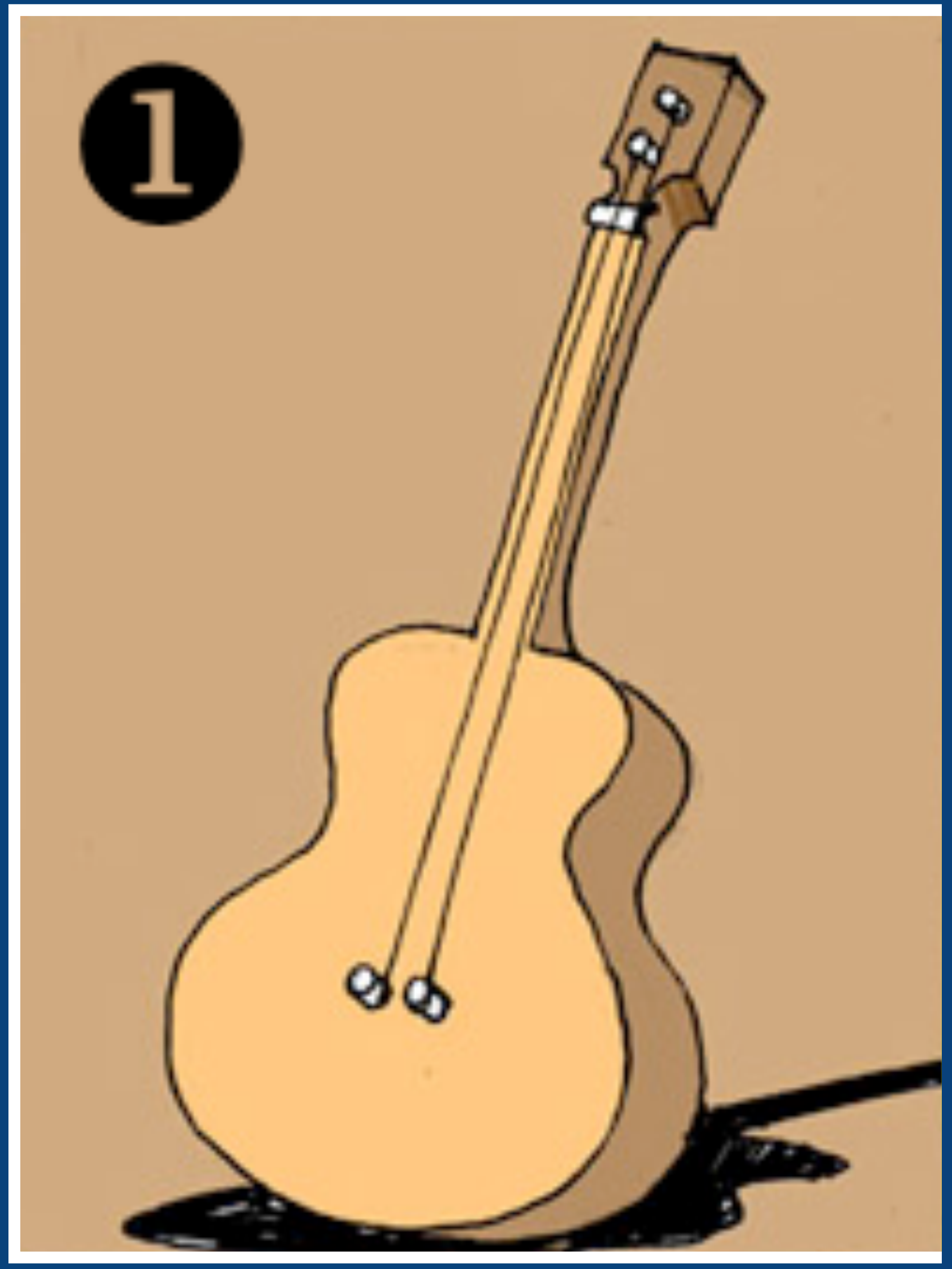
Sometimes requirements are known, and sometimes they aren’t

(enterprise software development vs product companies and startups) just enough



Evolutionary Design

Beginning With A Primitive Whole



Evolutionary Design

Beginning With A Primitive Whole

Architecture represents the **significant decisions**, where significance is measured by **cost of change**.

Grady Booch

Architecture

Design

Implementation

Programming languages
Technologies and platforms
Monolith, microservices or hybrid approach

Curly braces on the same or next line
Whitespace vs tabs

We’re not trying to make every decision

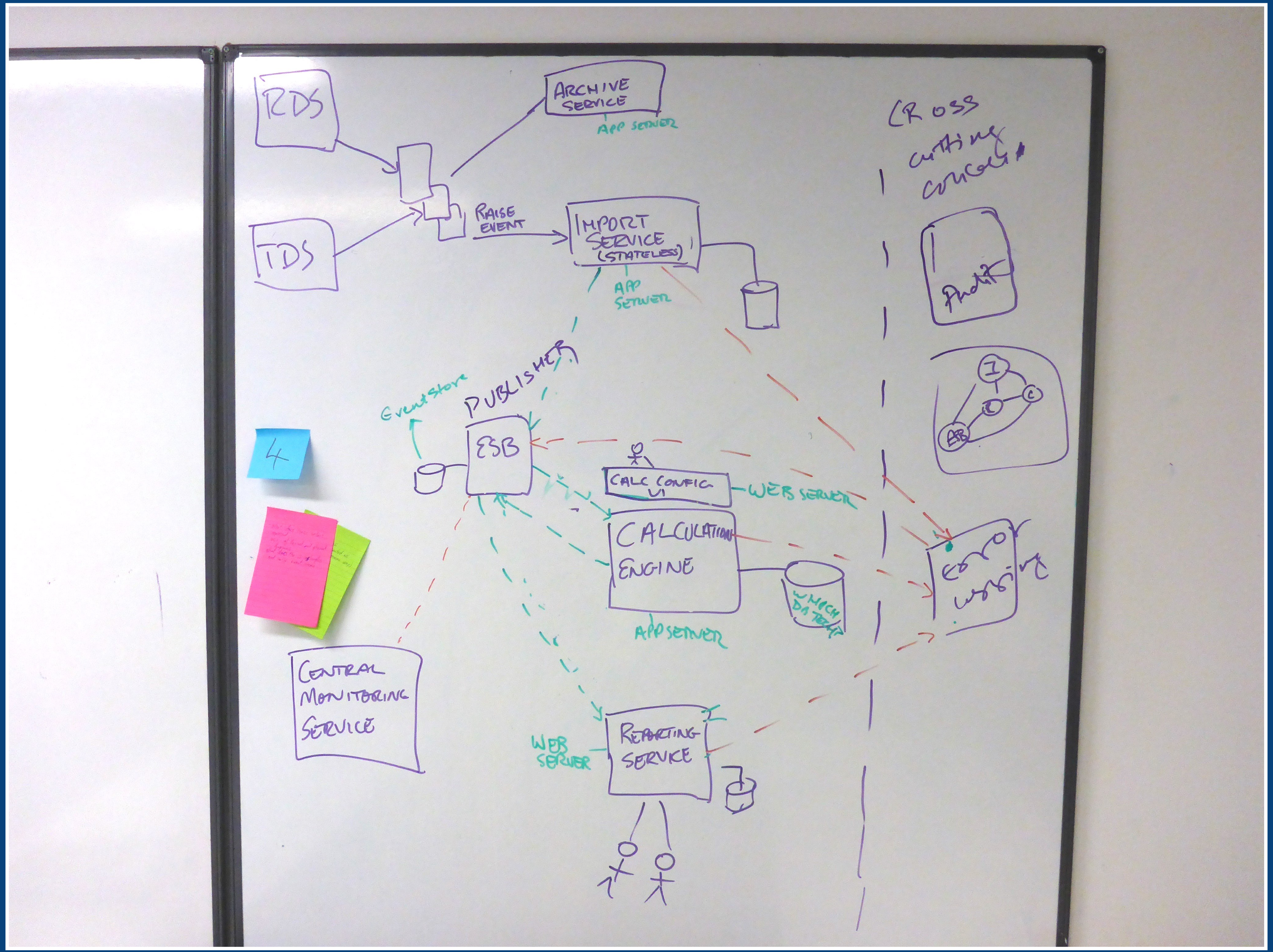
I think there is a role for a broad starting point architecture. Such things as stating early on how to layer the application, how you'll interact with the database (if you need one), what approach to use to handle the web server.

Martin Fowler

https://martinfowler.com/articles/designDead.html

A **starting point** adds value

1. Is that what we're going to **build**?



2. Is it going to **work**?

Base your architecture on requirements, travel light and prove your architecture with concrete experiments.

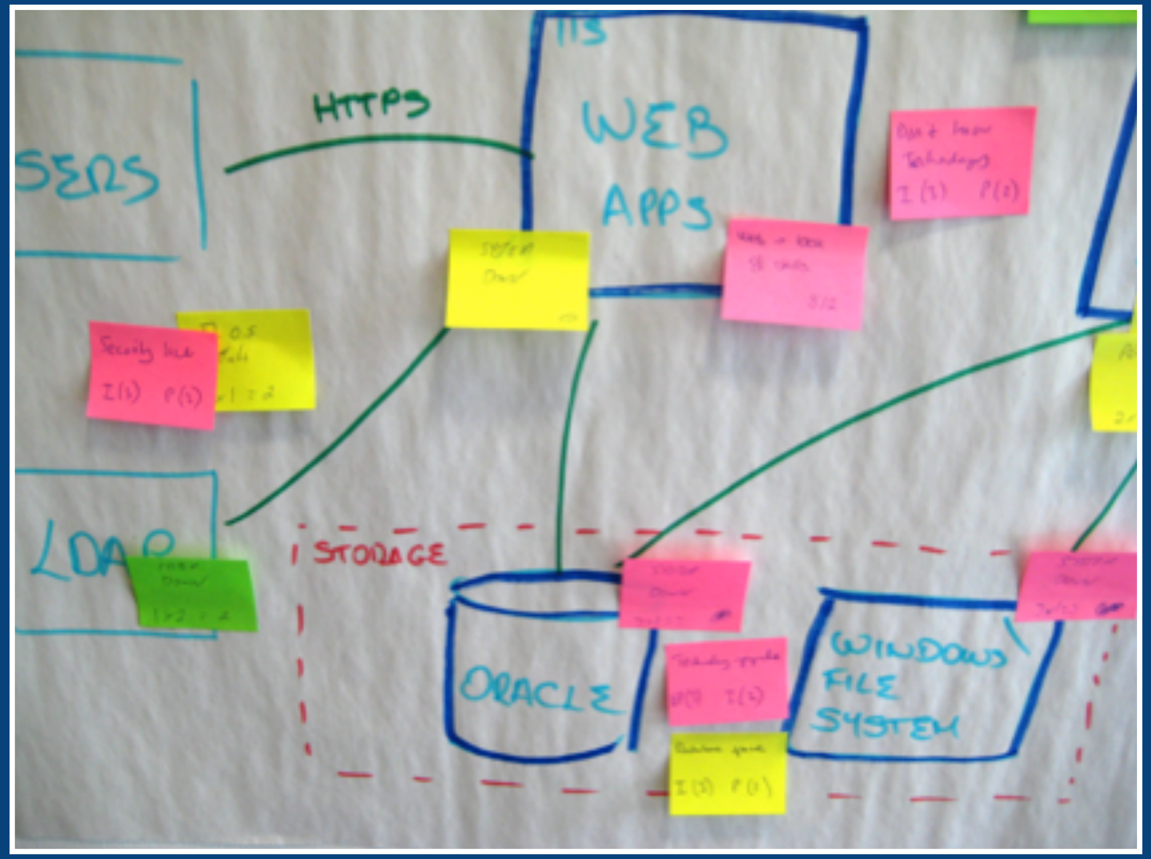
Agile Architecture: Strategies for Scaling Agile Development

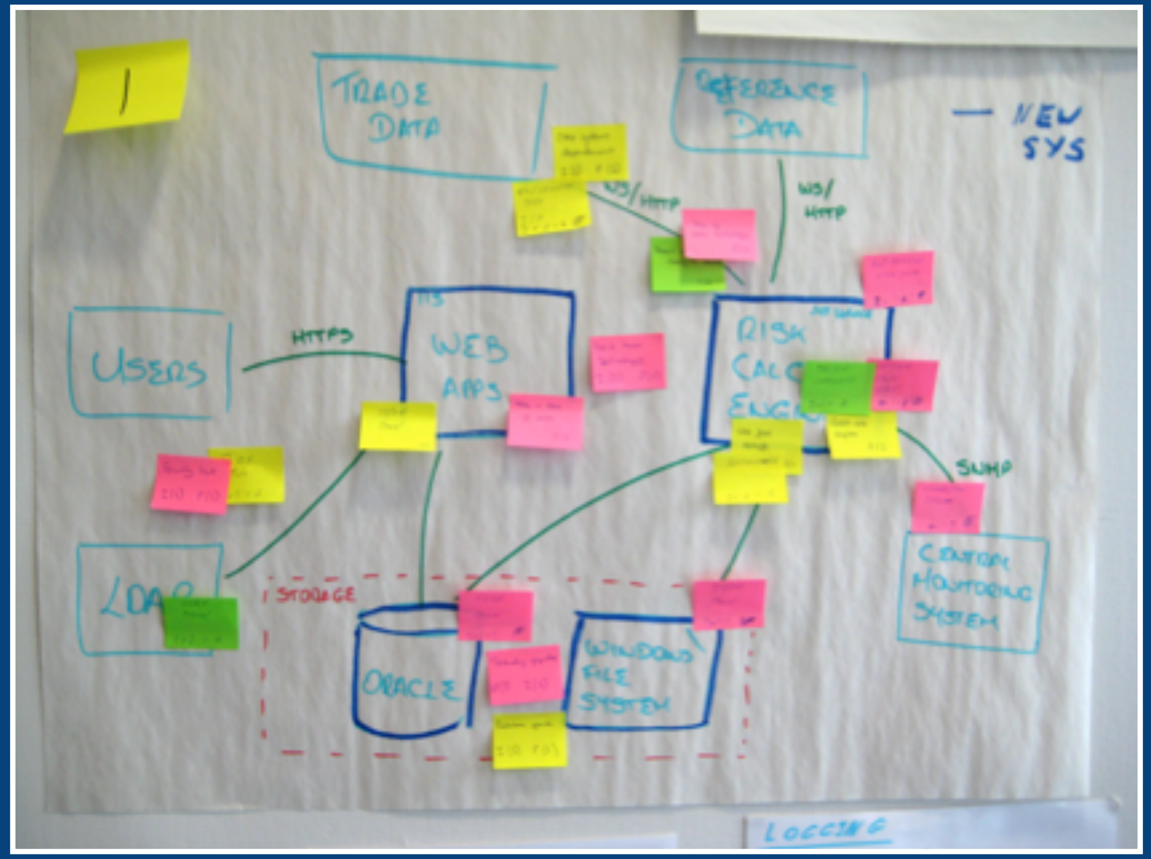
Scott Ambler

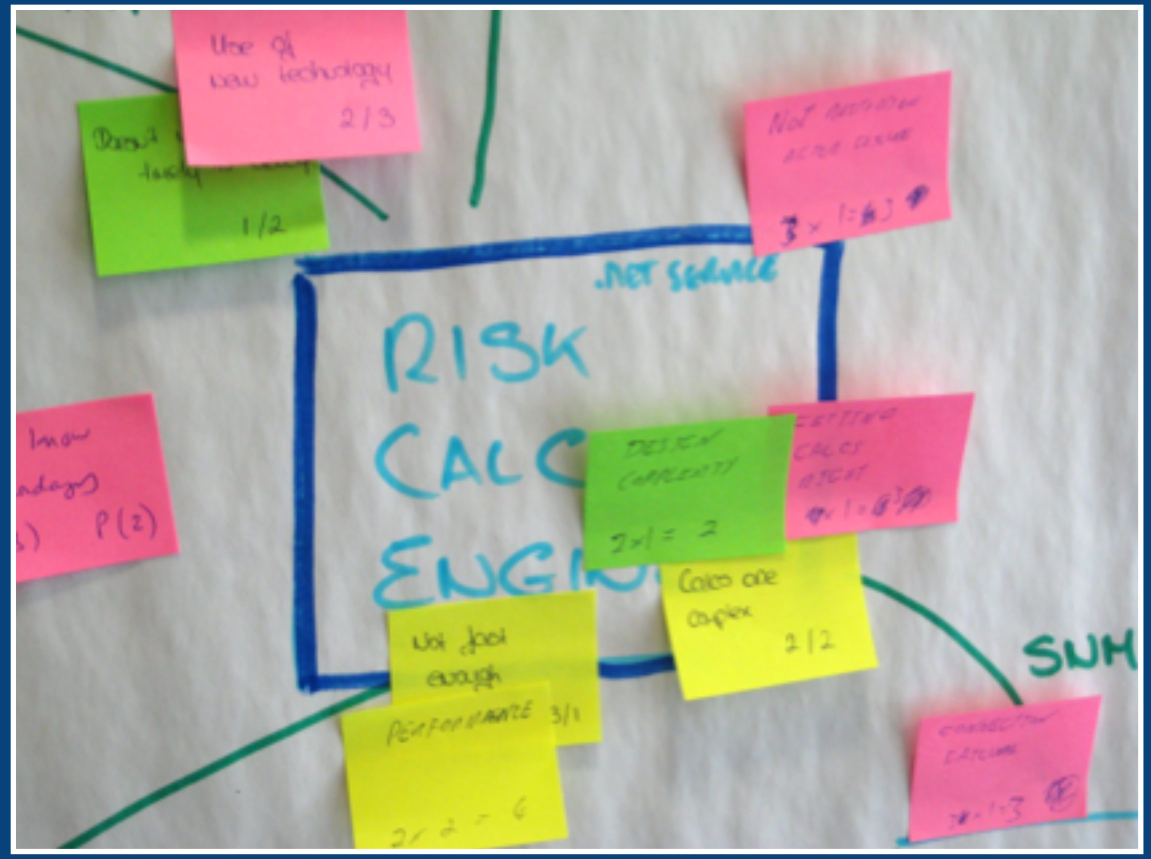
Concrete experiment

Proof of concept, prototype, spike, tracer, vertical slice, walking skeleton, executable reference architecture, …

Identify and **mitigate** your **highest priority risks**







Risk-storming

A visual and collaborative technique for identifying risk

Threat modelling

(STRIDE, LINDDUN, Attack Trees, etc)

How much up front design should you do?

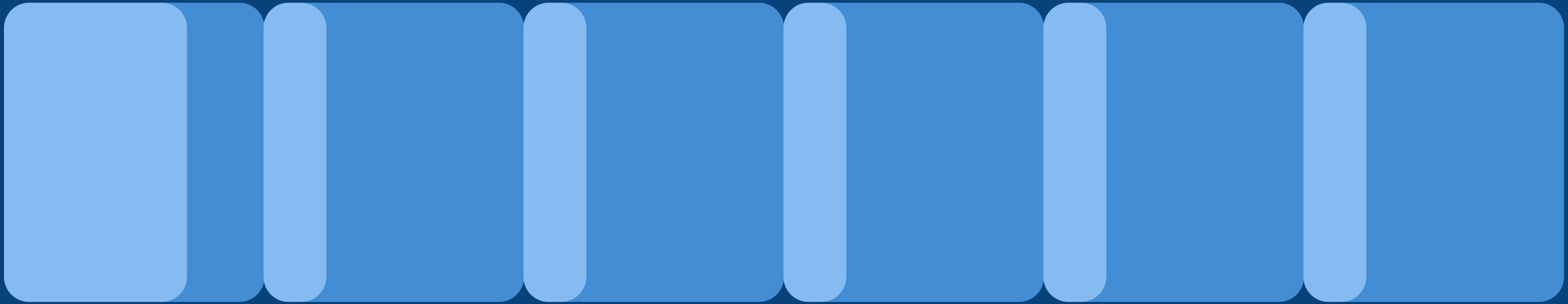
Up front design is an iterative and incremental process; stop when:

- ✓ You understand the significant architectural drivers (requirements, quality attributes, constraints).
- ✓ You understand the context and scope of what you're building.
- ✓ You understand the significant design decisions (i.e. technology, modularity, etc).
- ✓ You have a way to communicate your technical vision to other people.
- ✓ You are confident that your design satisfies the key architectural drivers.
- ✓ You have identified, and are comfortable with, the risks associated with building the software.

Techniques: Workshops, interviews, Event Storming, Impact Mapping, domain modelling, OOAD, CRC, DDD, architecture reviews, ATAM, architecture dry runs, Risk-storming, concrete experiments, C4 model, ADRs, etc.

Up front design is not necessarily about creating a perfect end-state or complete architecture

Enough up front design
to create a good
starting point and **direction**



Some Design Up Front + Evolutionary Design

2. Every team needs technical leadership

Software development teams don’t need architects

Software development teams do need technical leadership

Chaos

Big ball of mud, spaghetti code, inconsistent approaches to solving the same problems, quality attributes are ignored, deployment problems, maintenance issues, etc

The software architecture role

(technical leadership, and responsible for the technical success of the project/product)

Architectural drivers

Understanding the goals; capturing, refining and challenging the requirements and constraints.

Designing software

Creating the technical strategy, vision and roadmap.

Technical risks

Identifying, mitigating and owning the technical risks to ensure that the architecture "works".

Technical leadership

Continuous technical leadership and ownership of the architecture throughout

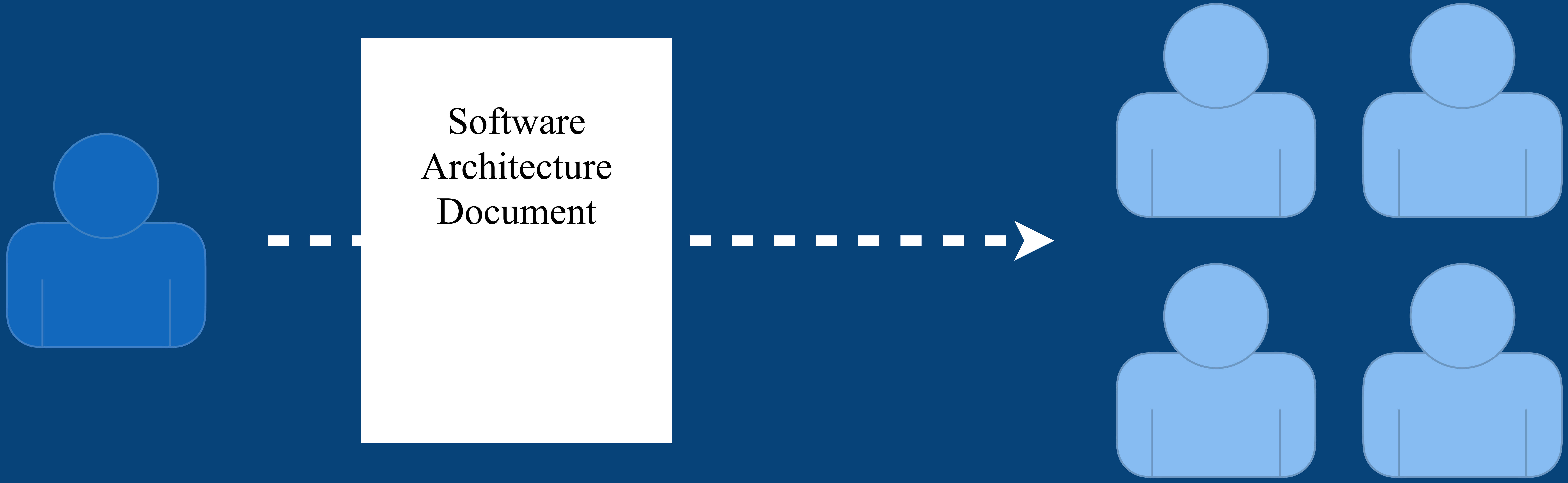
Quality assurance

Introduction and adherence to standards, guidelines, principles, etc.

Every team needs technical leadership

3. The software architecture role is about coding, coaching and collaboration

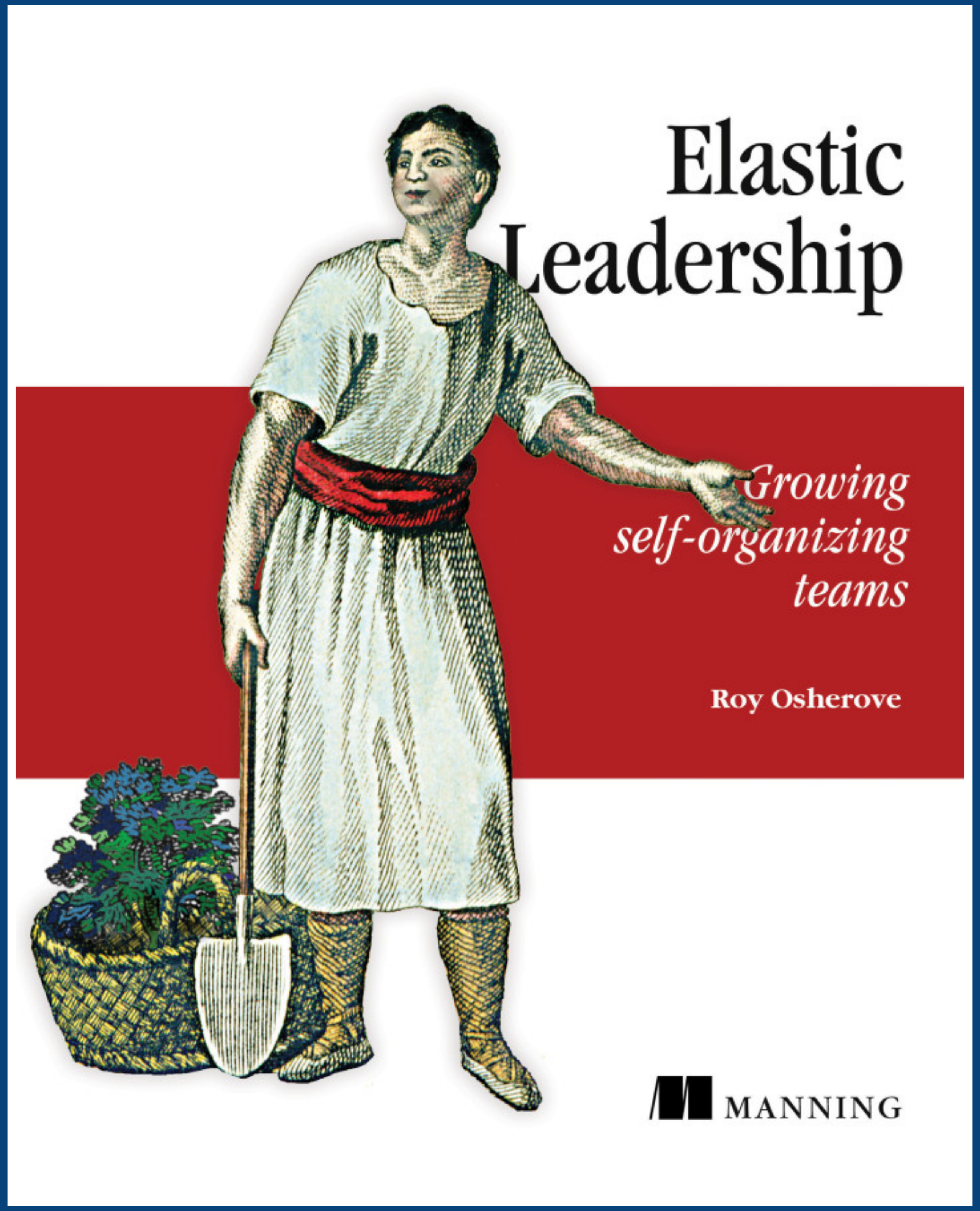
Software development is not a relay sport



AaaS

Architecture as a Service

Continuous technical leadership


Elastic Leadership

Growing self-organizing teams

Roy Osherove

MANNING

Different types of teams need different leadership styles

Pair architecting

Soft skills

(leadership, communication, presentation, influencing, negotiation, collaboration, coaching and mentoring, motivation, facilitation, political, etc)

Should software architects write **code**?

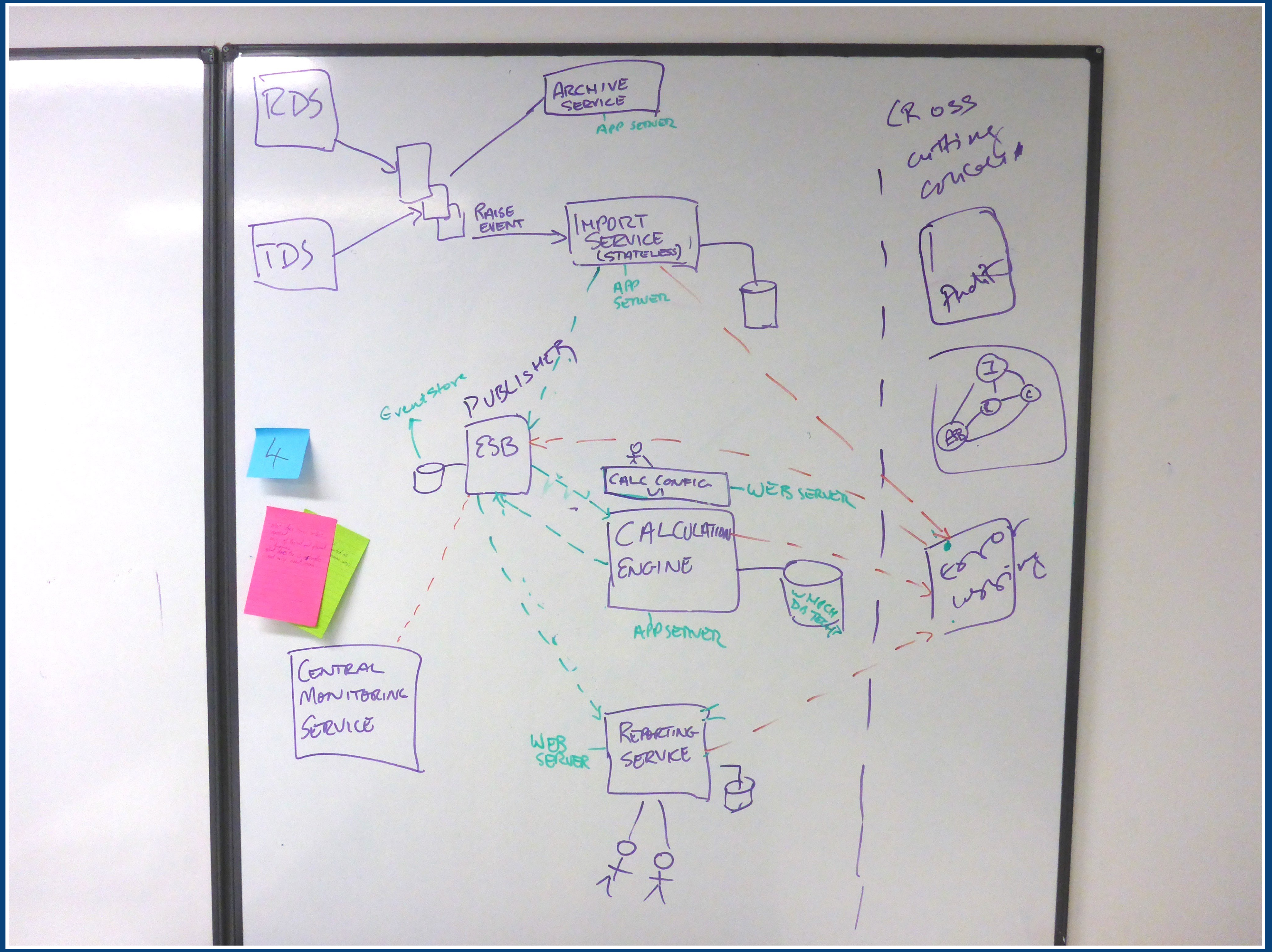
Production code, prototypes, frameworks, foundations, code reviews, experimenting, etc

Good software architects
are typically
good software developers

The people designing software must understand technology …

all decisions involve trade-offs

1. Is that what we're going to **build**?



2. Is it going to **work**?

The software architecture role

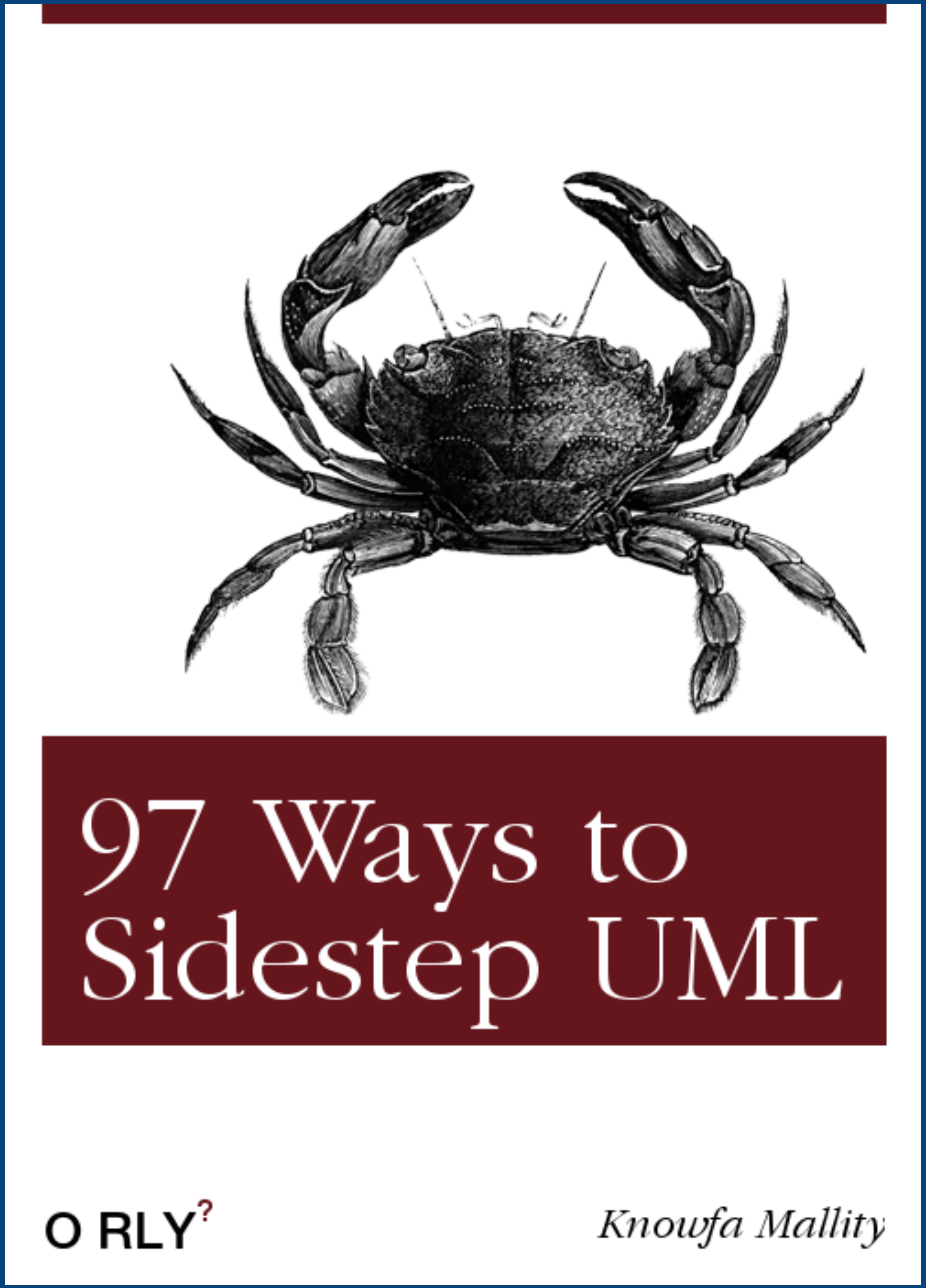
is **multi-faceted**

(technical depth, technical breadth, soft skills)

4. You don't need to use UML

In my experience, optimistically,

1 out of 10 people use UML



#2 "Not everybody else on the team knows it."

#3 "I'm the only person on the team who knows it."

#36 "You'll be seen as old."

#37 "You'll be seen as old-fashioned."

#66 "The tooling sucks."

#80 "It's too detailed."

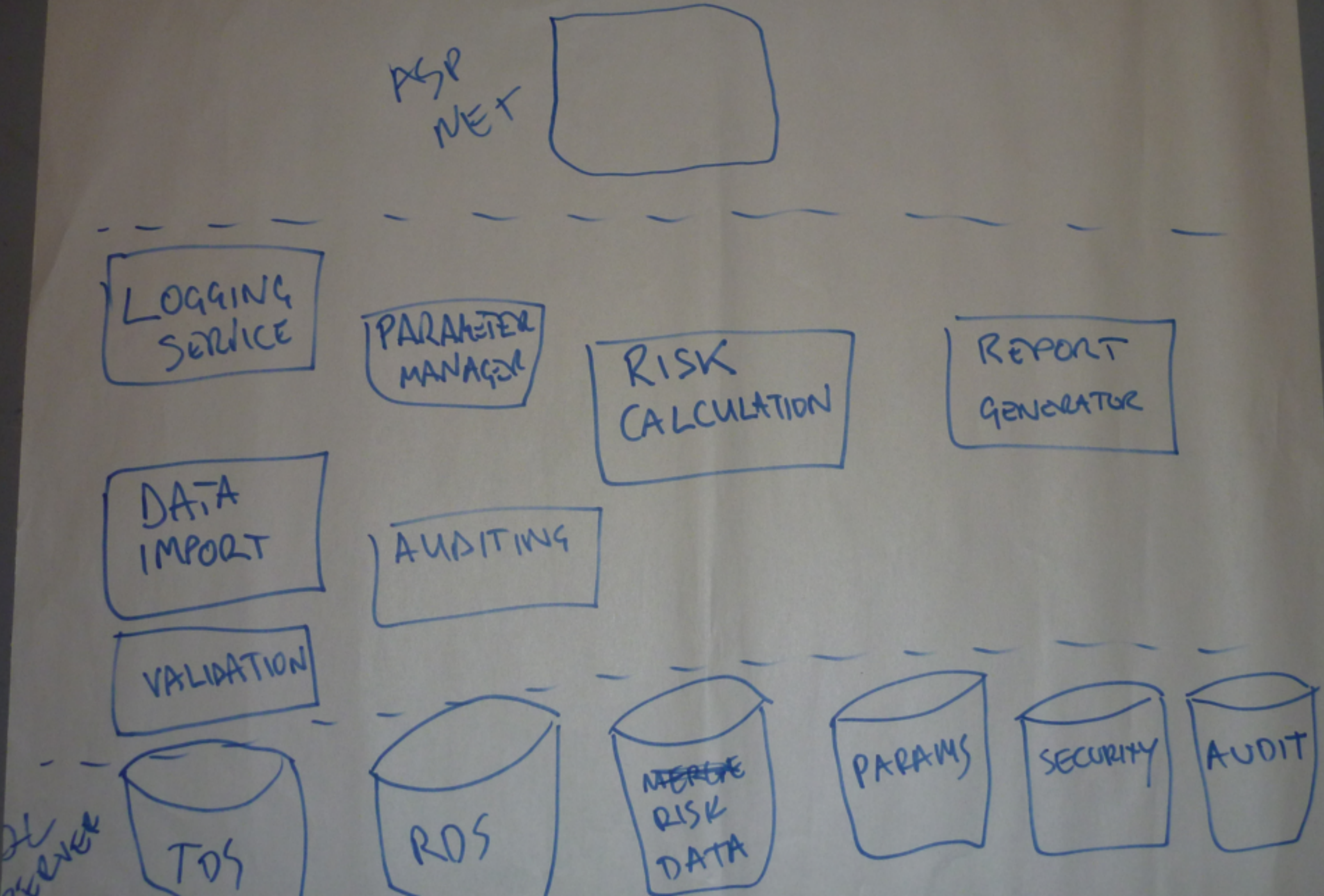
#81 "It's a very elaborate waste of time."

#92 "It's not expected in agile."

#97 "The value is in the conversation."

UML, Cucumber and modeling reality - MPJ's Musings - Fun Fun Function

UML, Cucumber and modeling reality - MPJ's Musings - Fun Fun Function
Just use a whiteboard!
1:42 / 12:48


ASP .NET
LOGGING SERVICE
PARAMETER MANAGER
RISK CALCULATION
REPORT GENERATOR
DATA IMPORT
AUDITING
VALIDATION
TDS
RDS
~~MERGE~~ RISK DATA
PARAMS
SECURITY
AUDIT

Import
mixed
storage
RISK
USER
Git UI
config
ORM
CALCULATION
RAW
REPORT
AUDIT
purging

TDS
RDS
trade report
ref reports
RS
TRANSPORT
ARCHIVE
action
data
error
error
AUDIT
MS
BUS. LOGIC
action
error
ERROR
MONITOR
REPORT GEN
action
error
scheduler
config

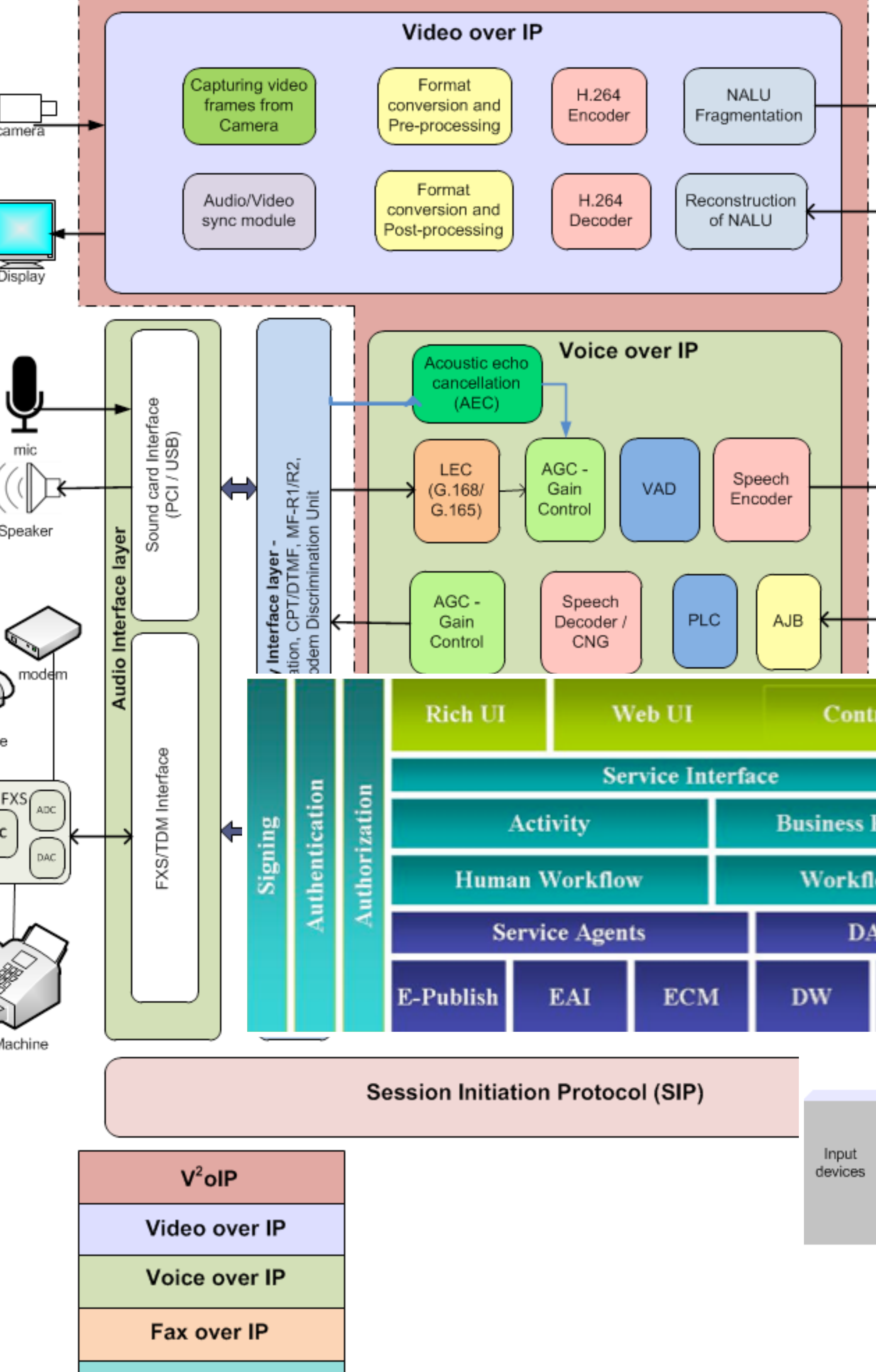
2
SERVICE/BATCH
IMPORT
JOIN
CALC
EXPORT/STORE
REPORT
SNMP
CENTRAL
MONITORING
SERVICE
3
WEBAPP
•GET REPORT
•EDIT
RISK RULES
•AUTH
•AUDIT
RISK RULES

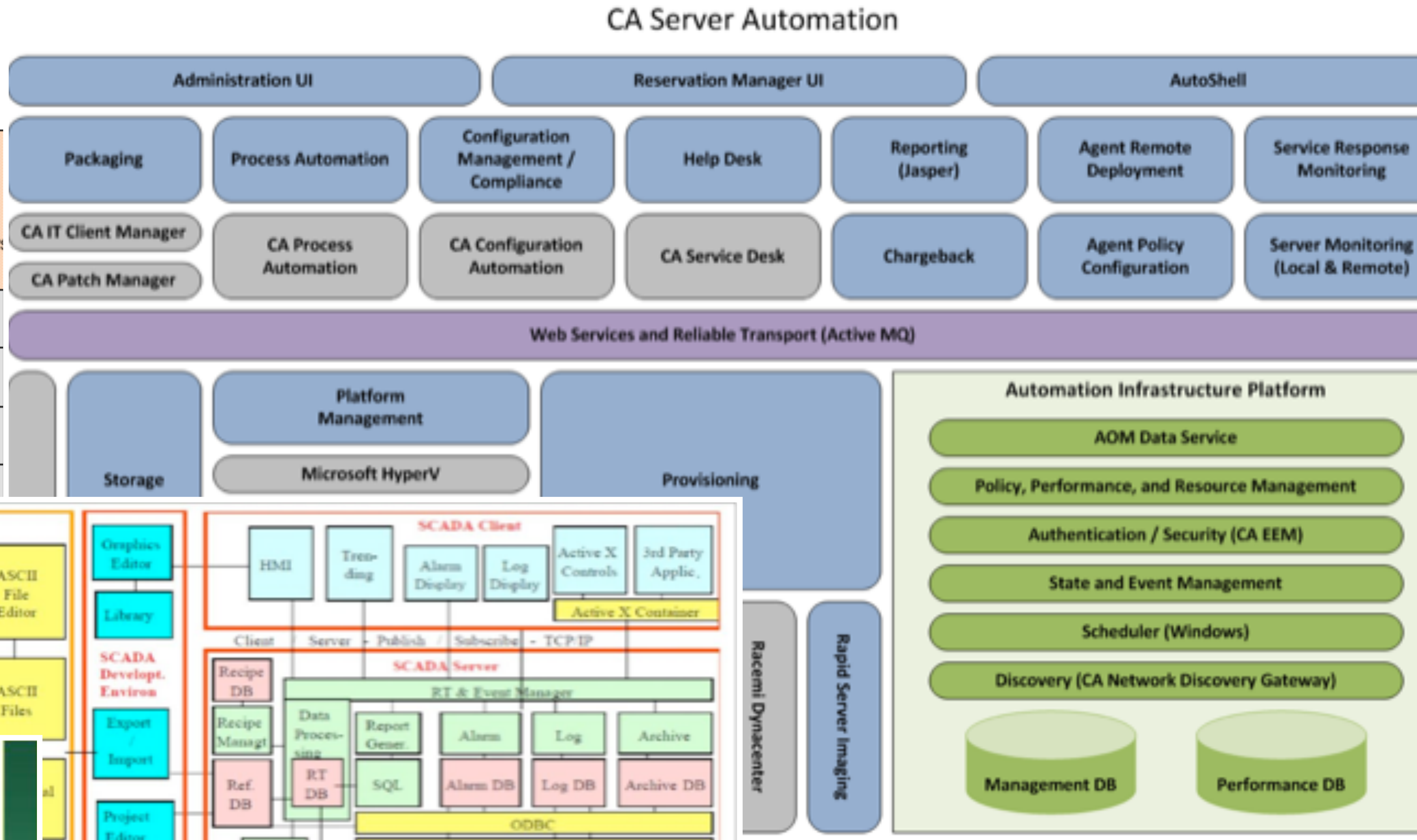
WIN SERVER
APPLICATION
CONSOLE - C#
SCHEDULER
CONSOLE - C#
DATA LOADER
C#
CALCULATION MANAGER
C#
NOTIFICATION MANAGER
C#
EXTERNAL SYSTEMS
SNMP SERVICE
DATA MERGE
C#
XPath
XML READER
C#
RDS
XML
TDS
RISK CALCULATION
C#
parameter
NOTIFICATION SERVICE
C#
SMTP
LOGGER
nLog
SNMP TRAP GENERATOR
DATA ACCESS
SQL SERVER

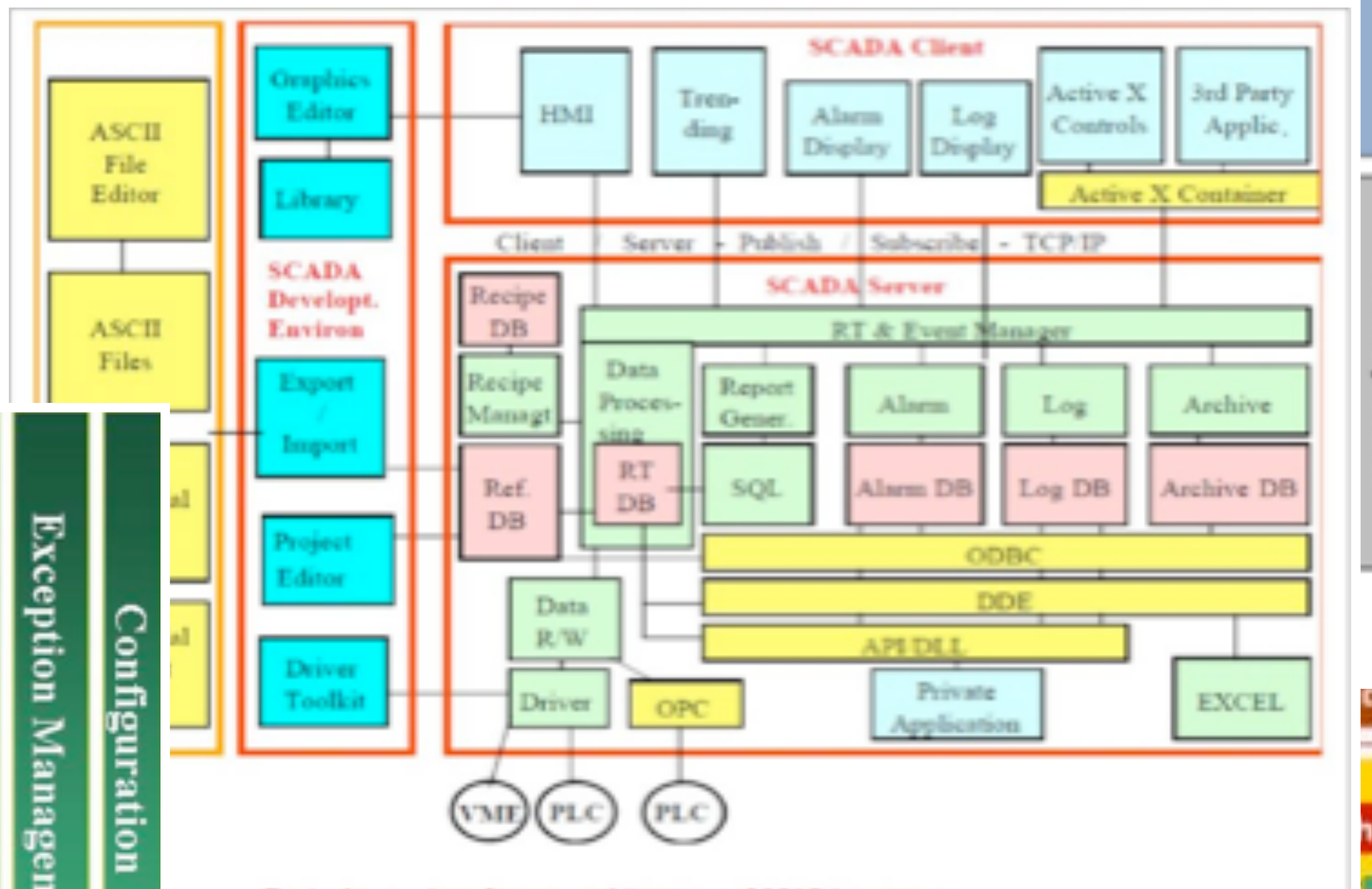
DATA POLL SERVICE
TRADES
AUTO RE-START.
REF DATA.
ARCHIVE
DATA READER TRADES
DATA VALIDATOR
reject
DATA READER REF DATA
DATA VALIDATOR.
reject.
FAIL EVENTS
MONITOR SERVICE SMTP TRAP
DATA STORE
DATA TRANSFORMATION INTEGRITY VALIDATION
rejects
tolerence.
FAIL EVENT
COMPLETE EVENT
STOP
STOPPED EVENT
COMPLETION EVENT
CALCULATION ENGINE
COUNT
A + B = C
PARMS TOLER.
UI
SECURITY
AUDIT
reject.
FAIL EVENT
COMPLETE EVENT
OUTPUT STORE
FAILURE

ENGINE
WHICH DB TECH?
APPSERVER
RING
REPORTING SERVICE
WEB SERVER

Params
Calcs
ret_curr
Risk outputs
Error
App Log
EH?
App
Error Log
Date
- Risk cel
- calcs
- Par
Batch
Params_Distlit
Detail
B_id:

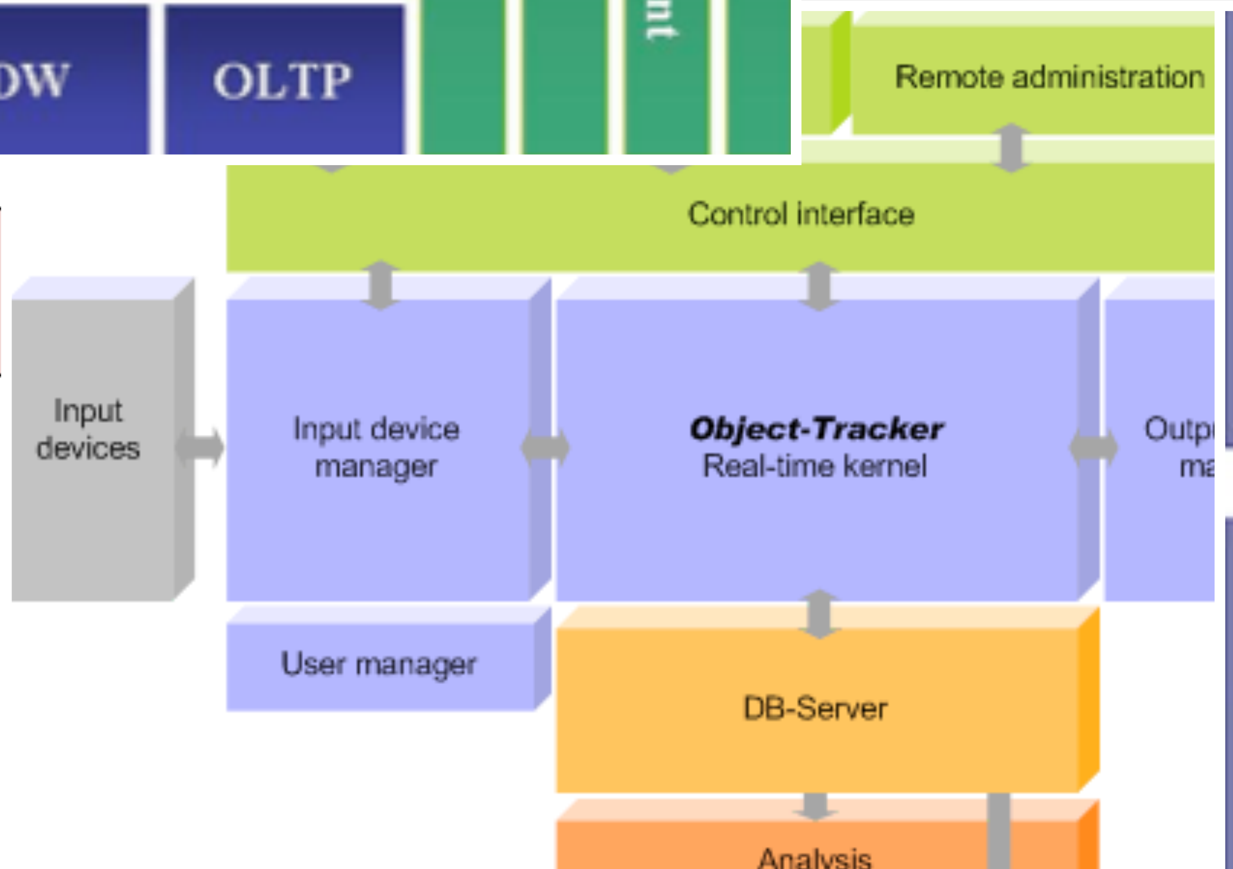


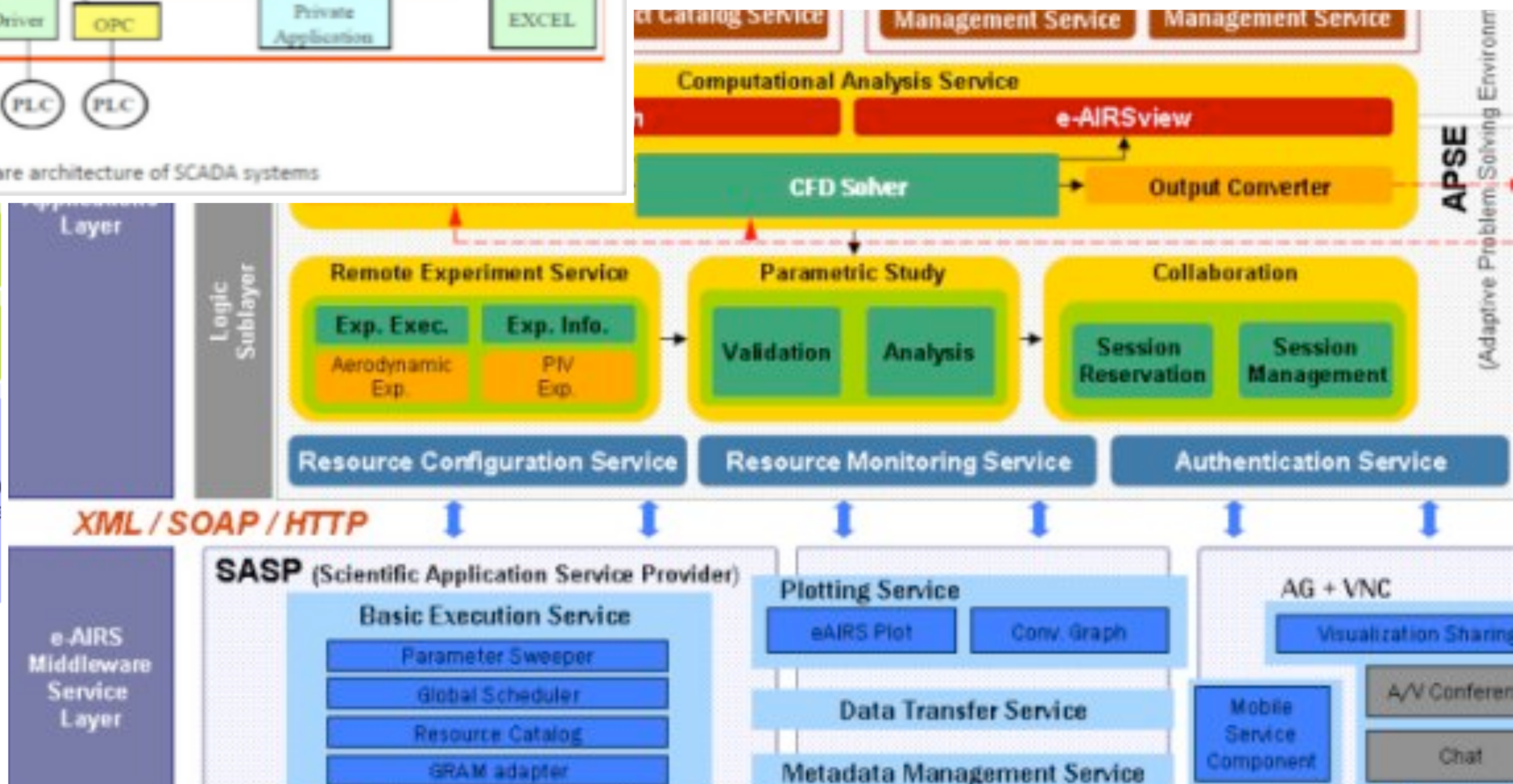




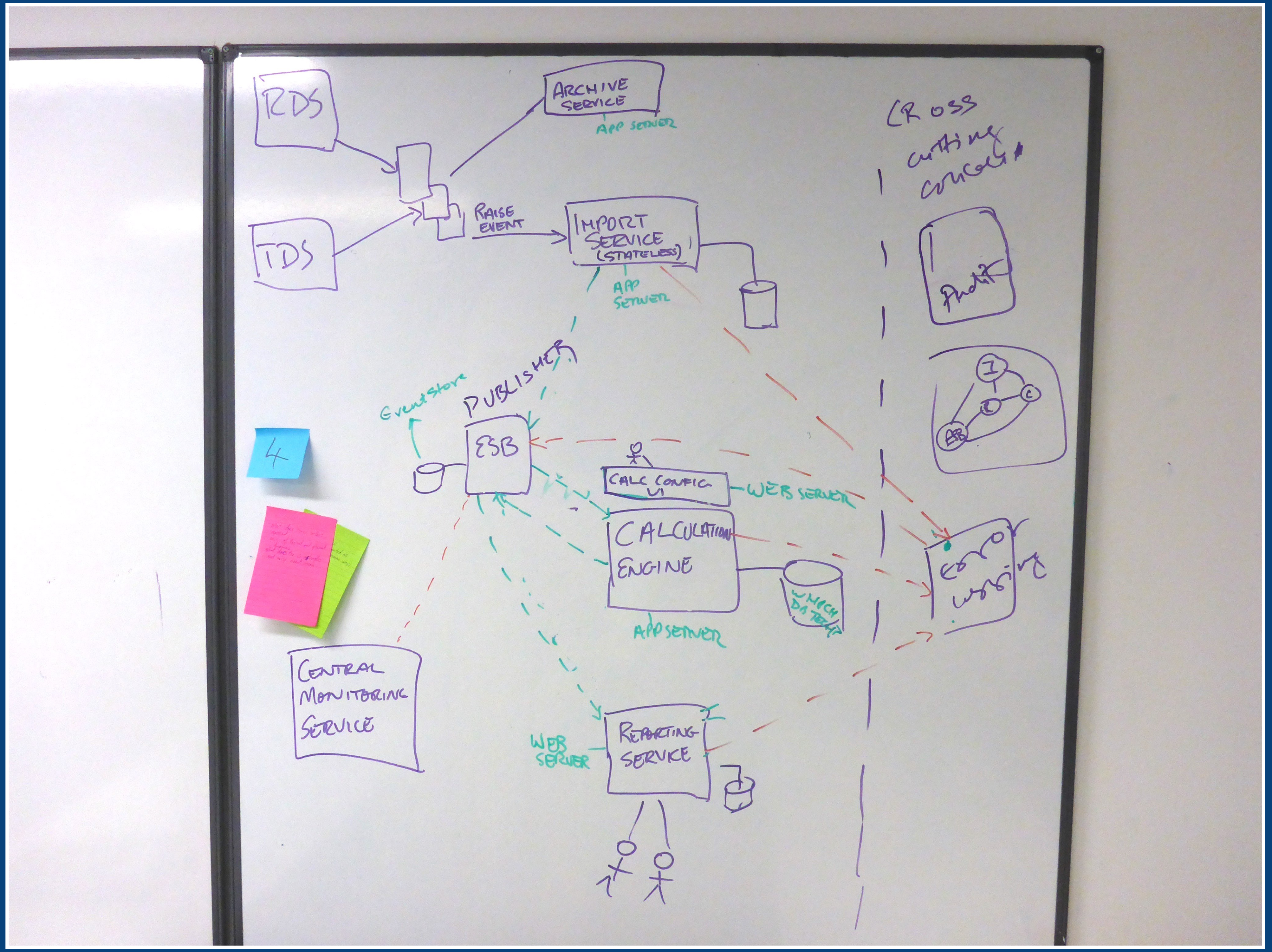
Typical generic software architecture of SCADA systems

Signing
Authentication
Authorization
Rich UI
Web UI
Controls
Service Interface
Activity
Business Rules
Human Workflow
Workflow
Service Agents
DAL
E-Publish
EAI
ECM
DW
OLTP
Monitoring
Log & Trace
Exception Management
Configuration





1. Is that what we're going to **build**?

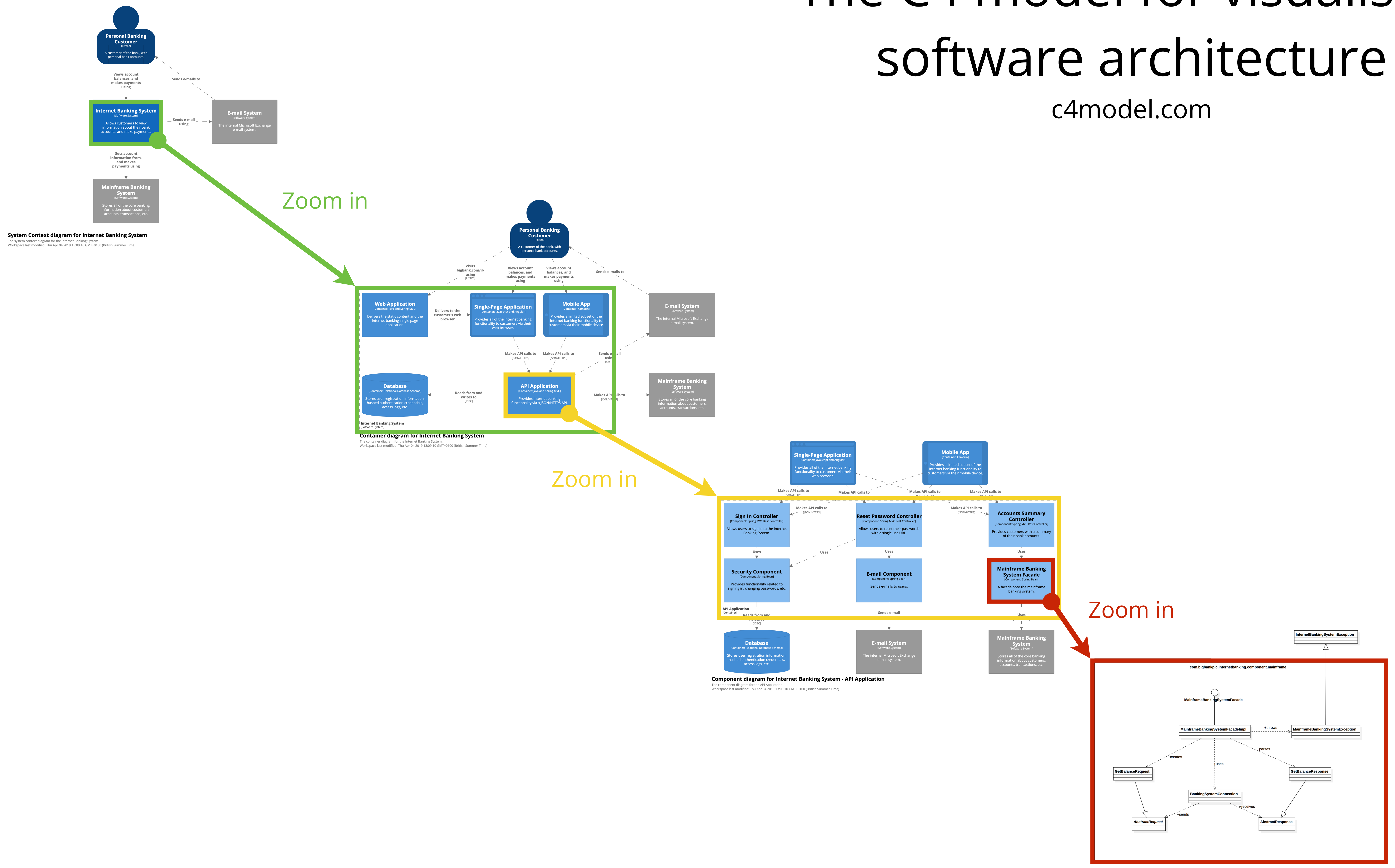


2. Is it going to **work**?

Teams need a **ubiquitous language** to communicate effectively

The C4 model for visualising software architecture

c4model.com

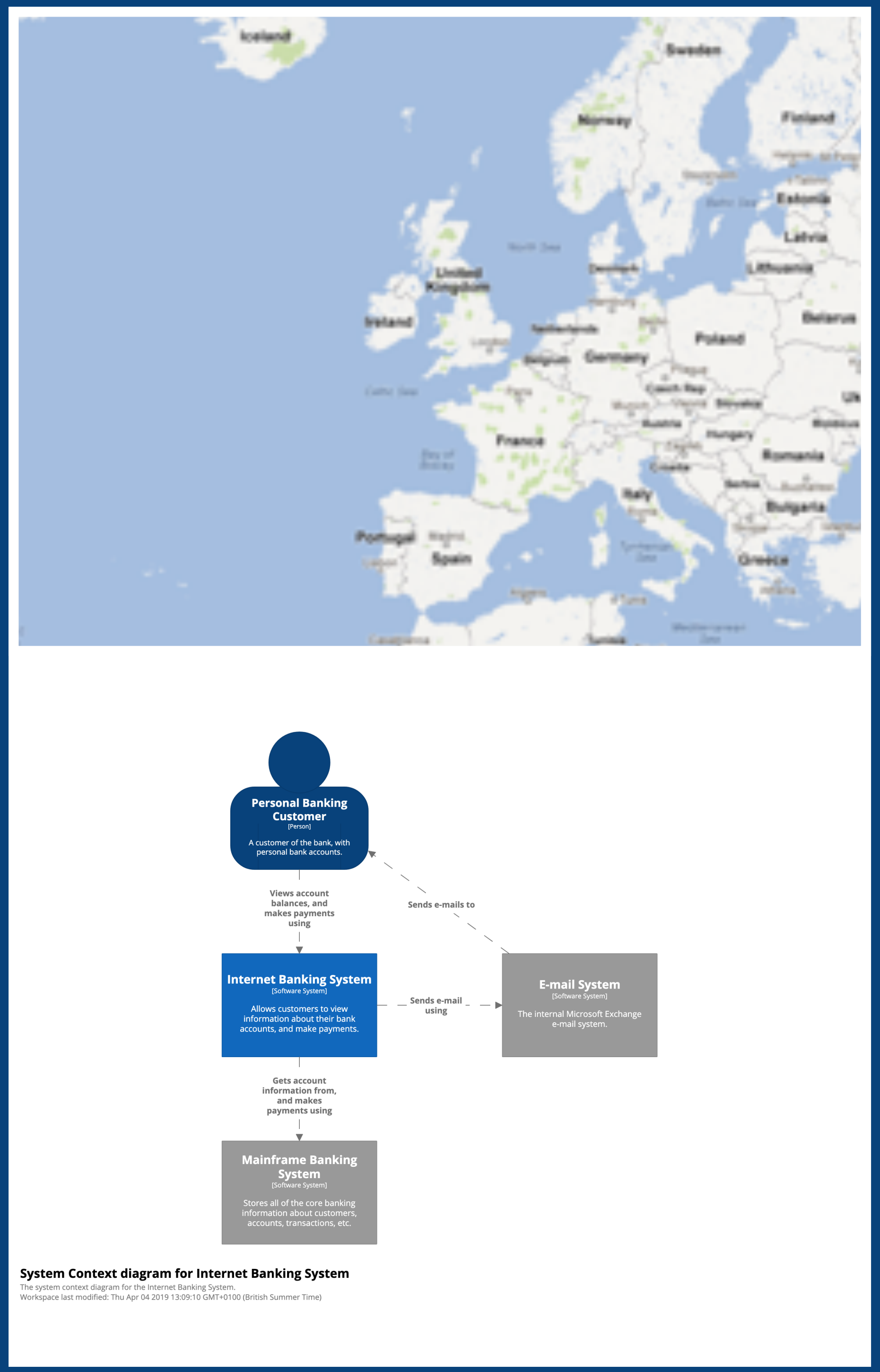

Personal Banking Customer
Internet Banking System
E-mail System
Mainframe Banking System
System Context diagram for Internet Banking System
Zoom in
Web Application
Single-Page Application
Mobile App
Database
API Application
Container diagram for Internet Banking System
Zoom in
Sign In Controller
Reset Password Controller
Accounts Summary Controller
Security Component
E-mail Component
Mainframe Banking System Facade
Component diagram for Internet Banking System - API Application
Zoom in
com.bigbankplc.internetbanking.component.mainframe
MainframeBankingSystemFacade
MainframeBankingSystemFacadeImpl
MainframeBankingSystemException
InternetBankingSystemException
GetBalanceRequest
GetBalanceResponse
BankingSystemConnection
AbstractRequest
AbstractResponse

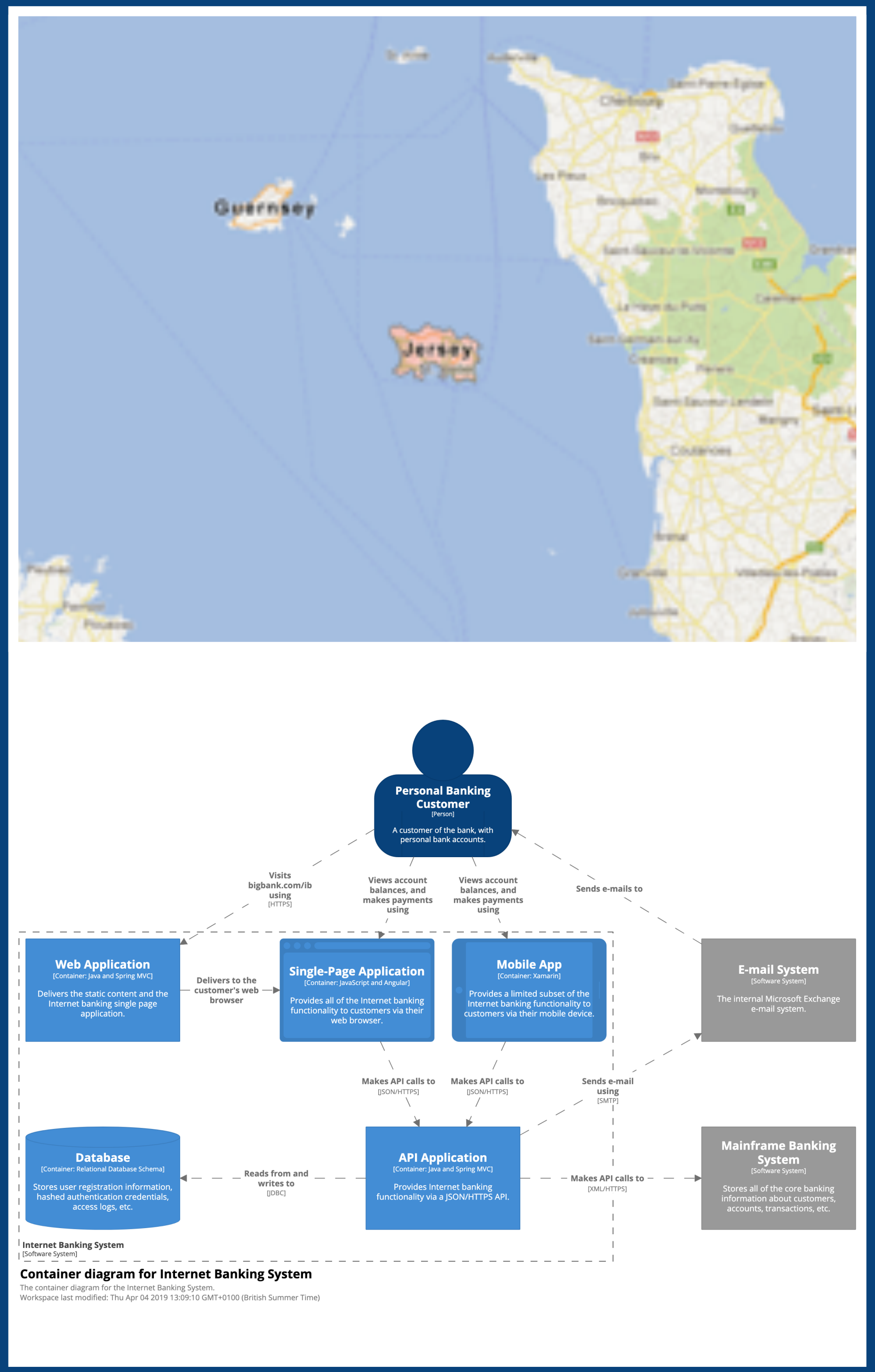
Level 1
Context

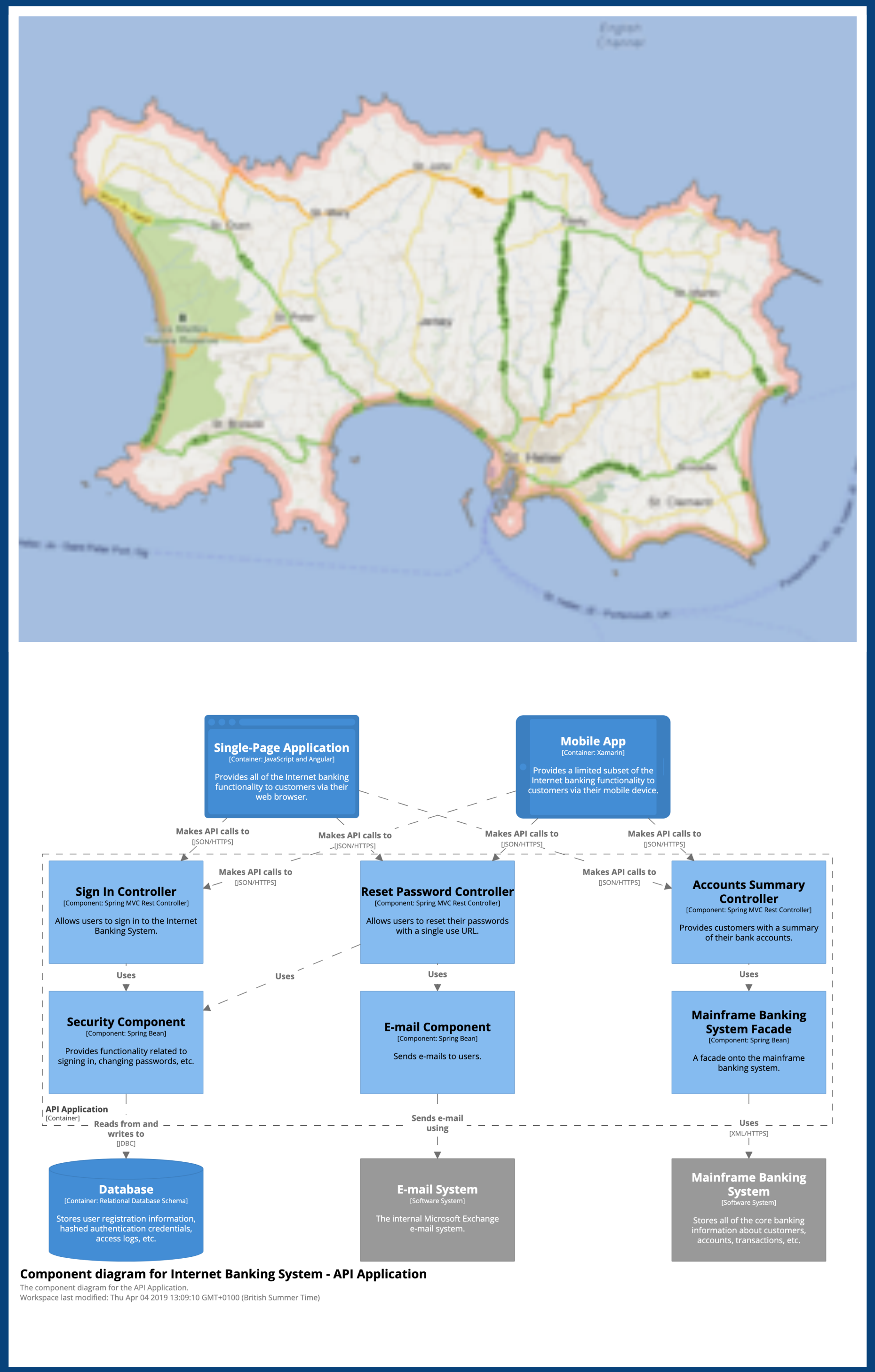
Level 2
Containers

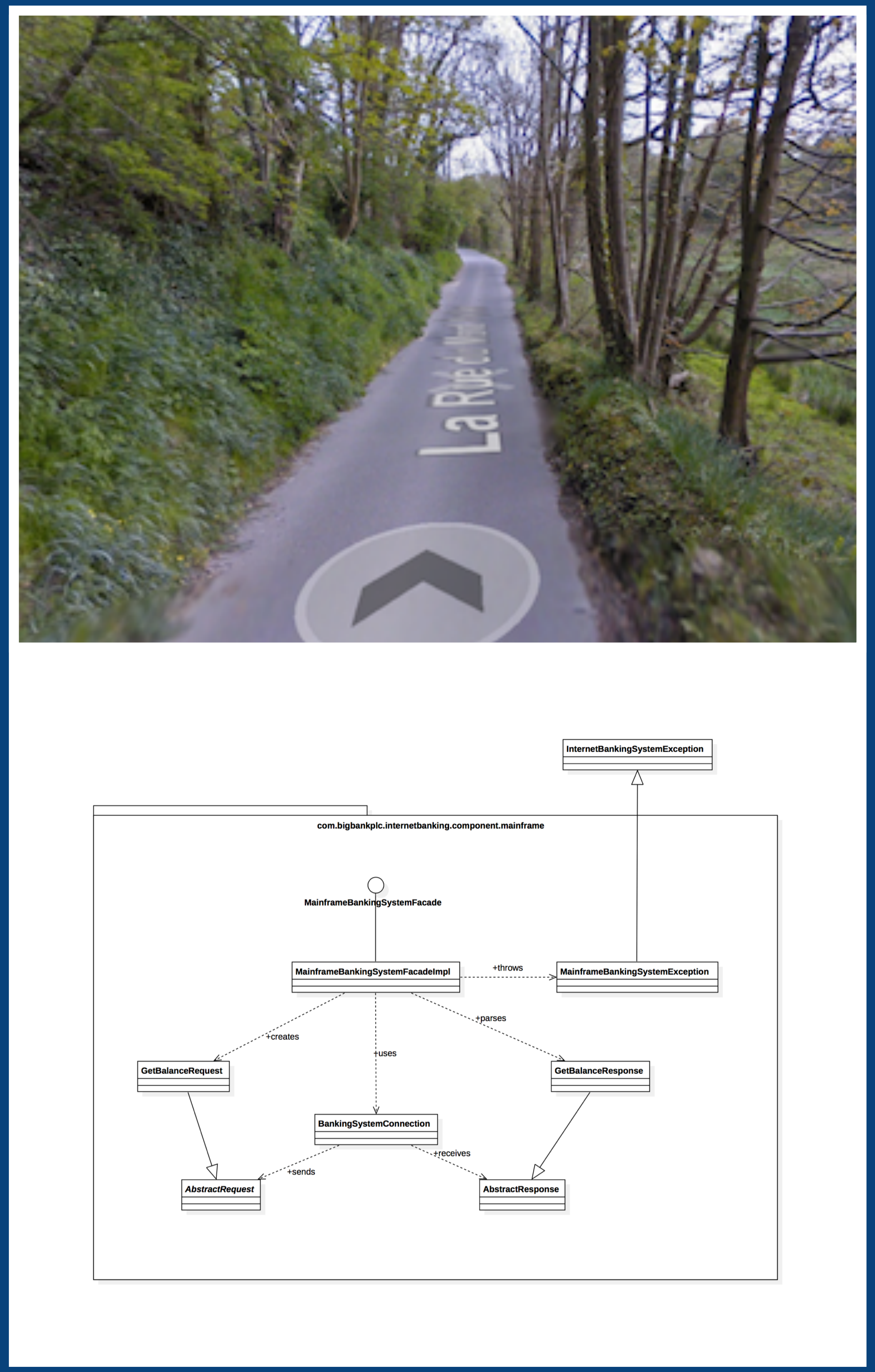
Level 3
Components

Level 4
Code


Iceland
Sweden
Norway
Finland
United Kingdom
Ireland
Germany
Poland
France
Spain
Portugal
Italy
Greece
Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Views account balances, and makes payments using
Sends e-mails to
Internet Banking System
[Software System]
Allows customers to view information about their bank accounts, and make payments.
Sends e-mail using
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Gets account information from, and makes payments using
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
System Context diagram for Internet Banking System
The system context diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)


Guernsey
Jersey
Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Visits bigbank.com/ib using
[HTTPS]
Views account balances, and makes payments using
Views account balances, and makes payments using
Sends e-mails to
Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.
Delivers to the customer's web browser
Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Sends e-mail using
[SMTP]
Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Reads from and writes to
[JDBC]
API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
Internet Banking System
[Software System]
Container diagram for Internet Banking System
The container diagram for the Internet Banking System.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)


Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.
Mobile App
[Container: Xamarin]
Provides a limited subset of the Internet banking functionality to customers via their mobile device.
Makes API calls to
[JSON/HTTPS]
Sign In Controller
[Component: Spring MVC Rest Controller]
Allows users to sign in to the Internet Banking System.
Reset Password Controller
[Component: Spring MVC Rest Controller]
Allows users to reset their passwords with a single use URL.
Accounts Summary Controller
[Component: Spring MVC Rest Controller]
Provides customers with a summary of their bank accounts.
Uses
Security Component
[Component: Spring Bean]
Provides functionality related to signing in, changing passwords, etc.
E-mail Component
[Component: Spring Bean]
Sends e-mails to users.
Mainframe Banking System Facade
[Component: Spring Bean]
A facade onto the mainframe banking system.
API Application
[Container]
Reads from and writes to
[JDBC]
Sends e-mail using
Uses
[XML/HTTPS]
Database
[Container: Relational Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
Component diagram for Internet Banking System - API Application
The component diagram for the API Application.
Workspace last modified: Thu Apr 04 2019 13:09:10 GMT+0100 (British Summer Time)


La Rue ...
InternetBankingSystemException
com.bigbankplc.internetbanking.component.mainframe
MainframeBankingSystemFacade
MainframeBankingSystemFacadeImpl
+throws
MainframeBankingSystemException
+creates
+uses
+parses
GetBalanceRequest
GetBalanceResponse
BankingSystemConnection
+sends
+receives
AbstractRequest
AbstractResponse

Diagrams are maps

that help software developers navigate a large and/or complex codebase

The C4 model is...

A set of hierarchical abstractions

(software systems, containers, components, and code)

A set of hierarchical diagrams

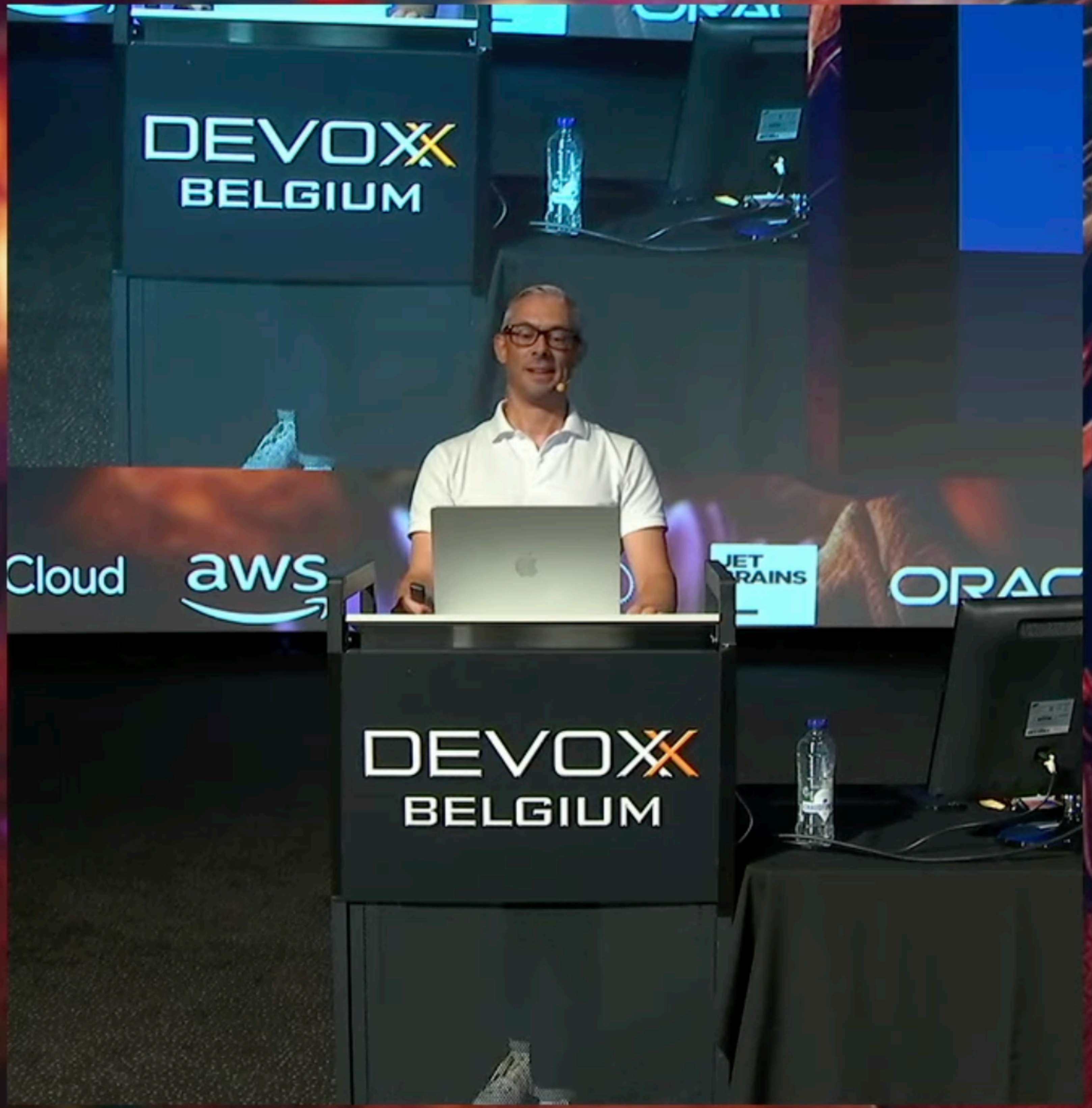
(system context, containers, components, and code)

Notation independent

Tooling independent

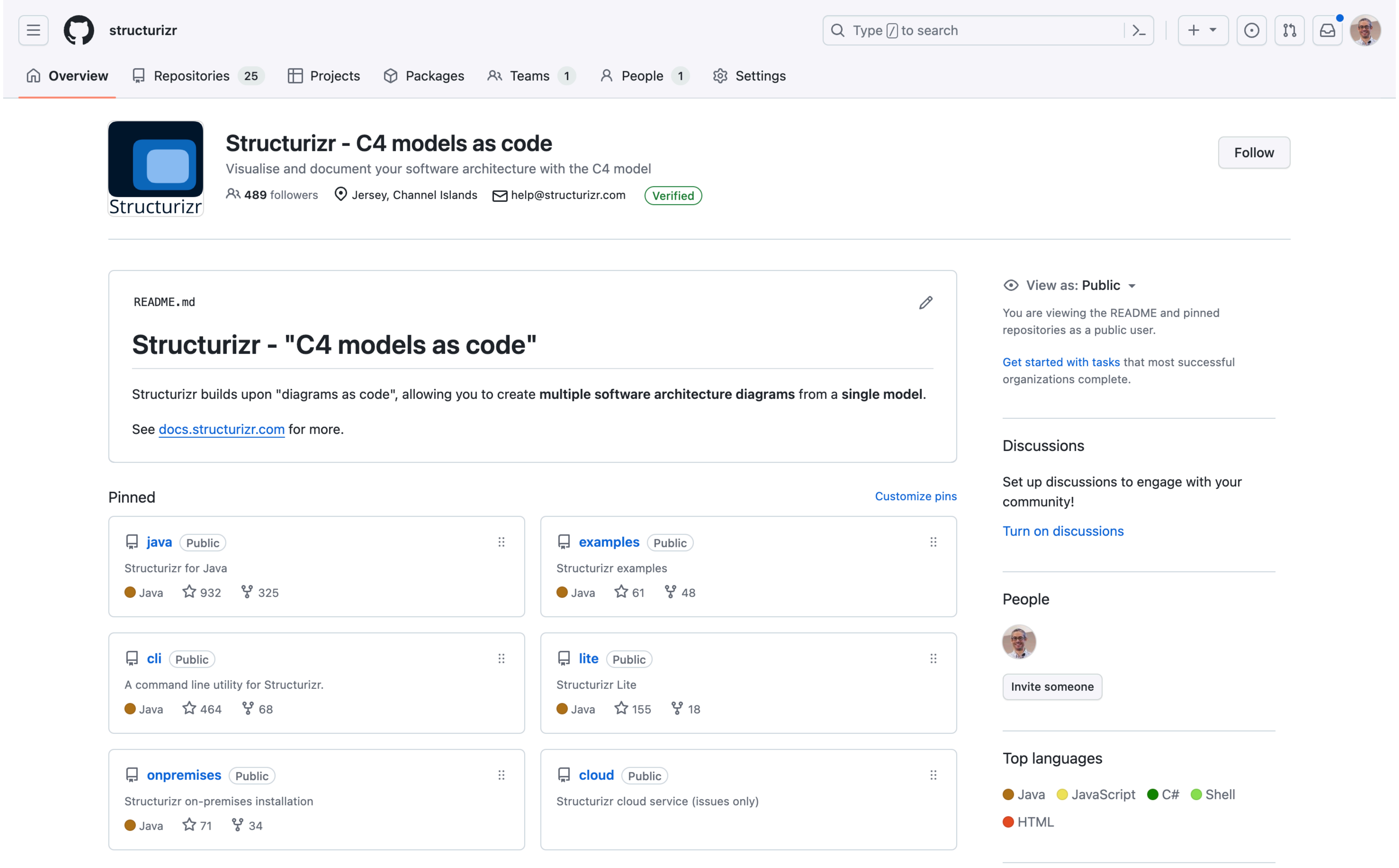
c4model.com

(for more information about software architecture diagrams)



C4 models as code

Simon Brown
@simonbrown



structurizr

Type / to search

Overview | Repositories 25 | Projects | Packages | Teams 1 | People 1 | Settings

Structurizr

Structurizr - C4 models as code

Visualise and document your software architecture with the C4 model

489 followers · Jersey, Channel Islands · help@structurizr.com · Verified

Follow

README.md

Structurizr - "C4 models as code"

Structurizr builds upon "diagrams as code", allowing you to create **multiple software architecture diagrams** from a **single model**.

See docs.structurizr.com for more.

Pinned

Customize pins

java Public
Structurizr for Java
● Java ☆ 932 325

examples Public
Structurizr examples
● Java ☆ 61 48

cli Public
A command line utility for Structurizr.
● Java ☆ 464 68

lite Public
Structurizr Lite
● Java ☆ 155 18

onpremises Public
Structurizr on-premises installation
● Java ☆ 71 34

cloud Public
Structurizr cloud service (issues only)

View as: Public

You are viewing the README and pinned repositories as a public user.

Get started with tasks that most successful organizations complete.

Discussions

Set up discussions to engage with your community!

Turn on discussions

People

Invite someone

Top languages

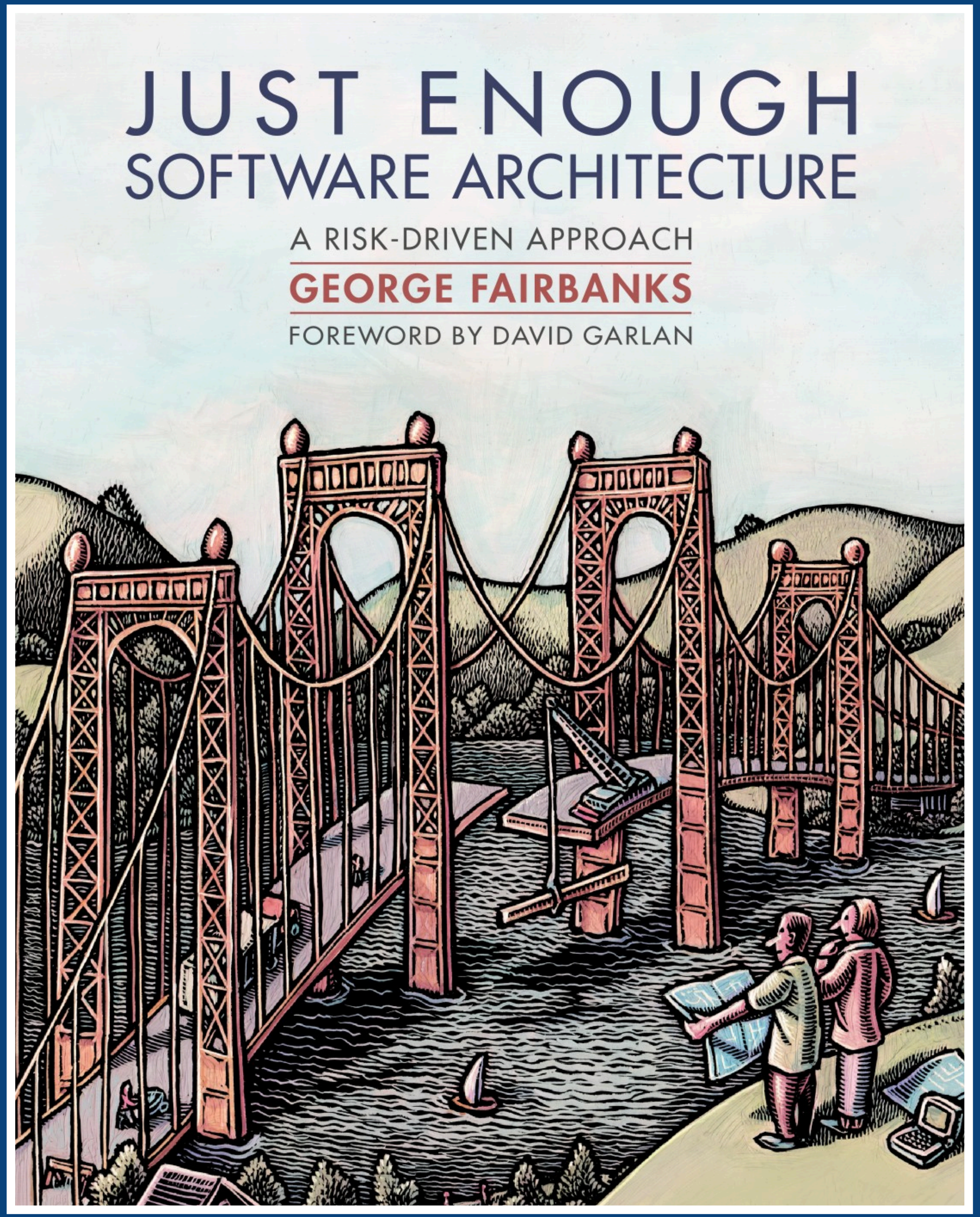
● Java ● JavaScript ● C# ● Shell ● HTML

5. A good software architecture enables agility

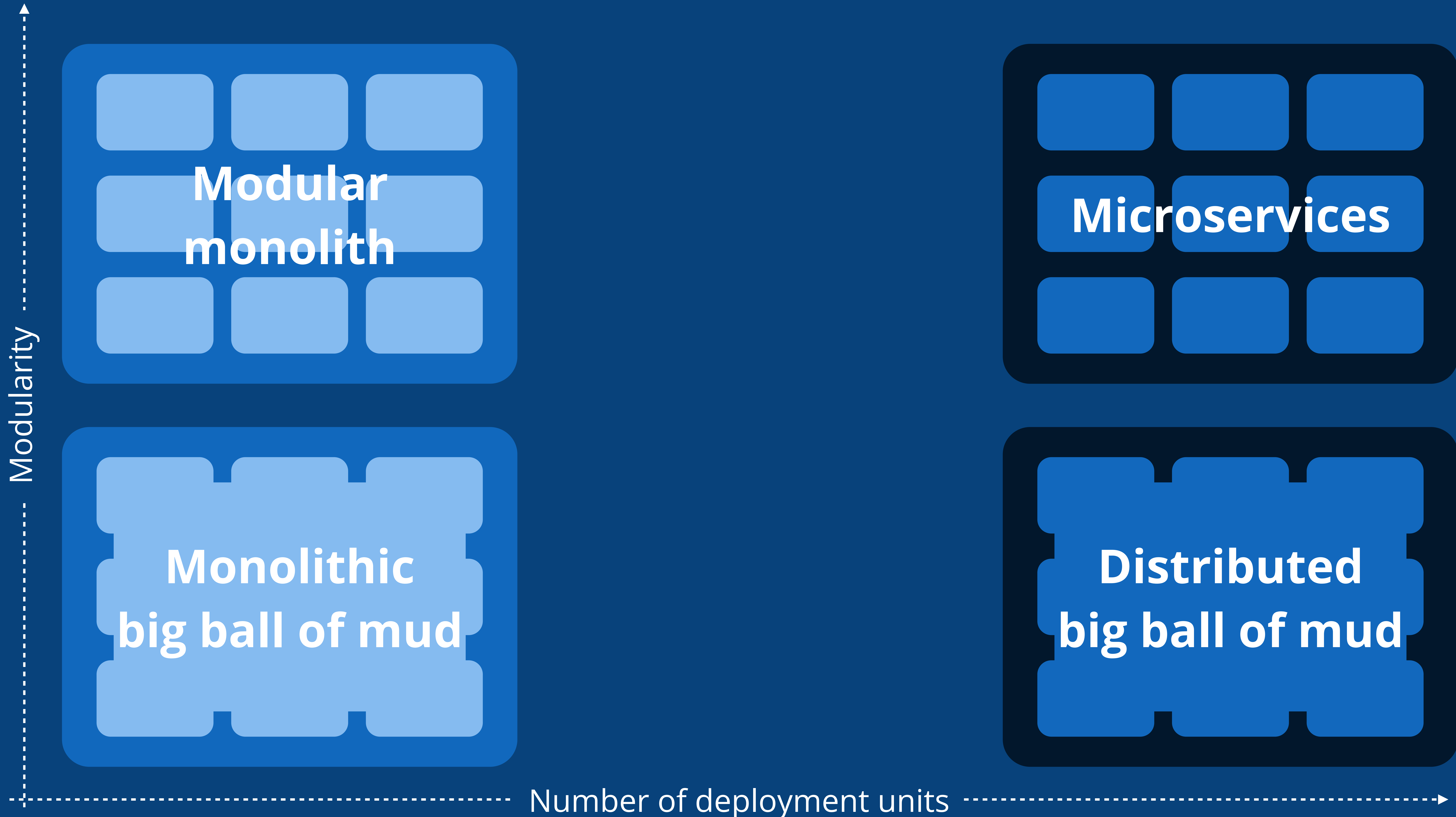
Agile is about moving fast, embracing change, releasing often, getting feedback, …

Agile is about a mindset of **continuous improvement**

A good architecture enables agility

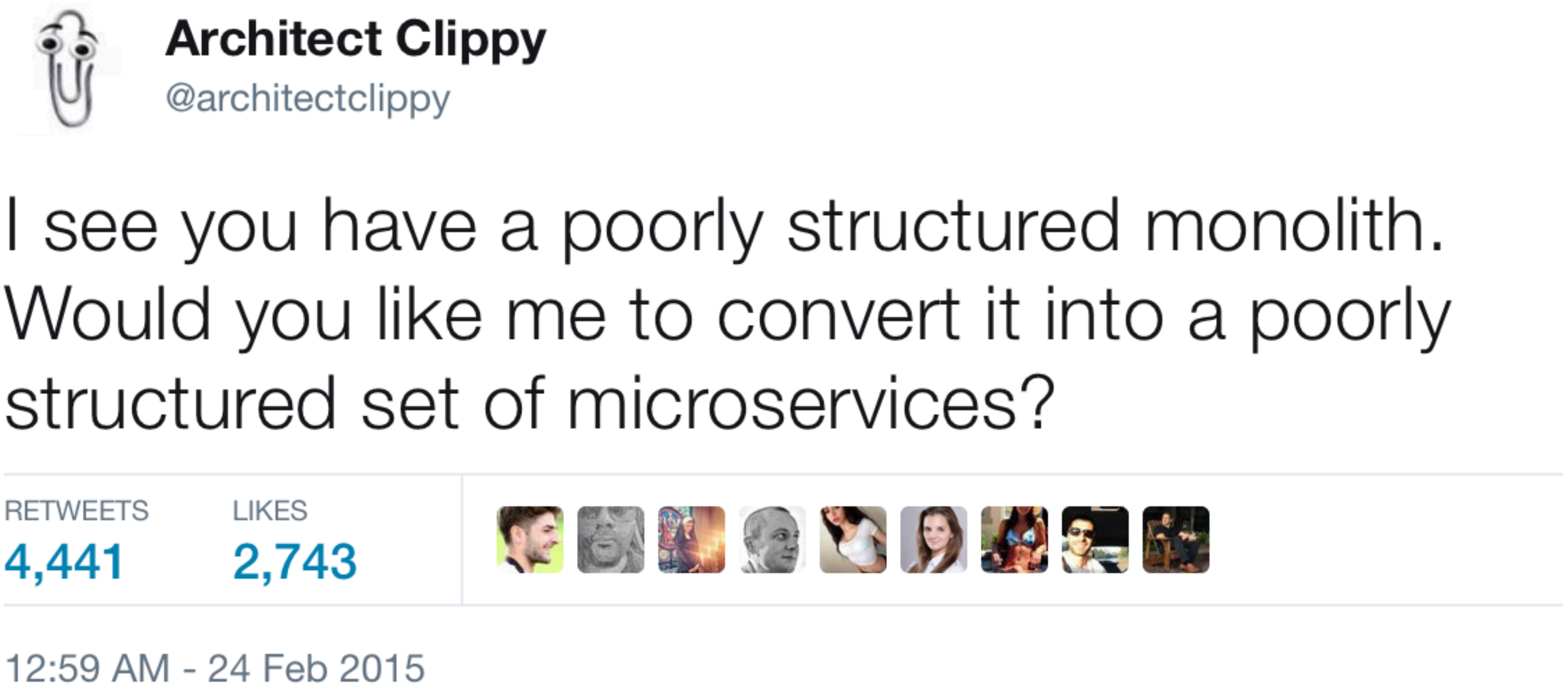


A good architecture rarely happens through **architecture-indifferent design**


Modular monolith
Microservices
Monolithic big ball of mud
Distributed big ball of mud
Modularity
Number of deployment units

Agility is a

quality attribute



Architect Clippy
@architectclippy

I see you have a poorly structured monolith. Would you like me to convert it into a poorly structured set of microservices?

RETWEETS 4,441 LIKES 2,743

12:59 AM - 24 Feb 2015

Five things every developer should know about **software architecture**

1. Software architecture isn't about big design up front
2. Every team needs technical leadership
3. The software architecture role is about coding, coaching and collaboration
4. You don't need to use UML
5. A good software architecture enables agility

Simon Brown