

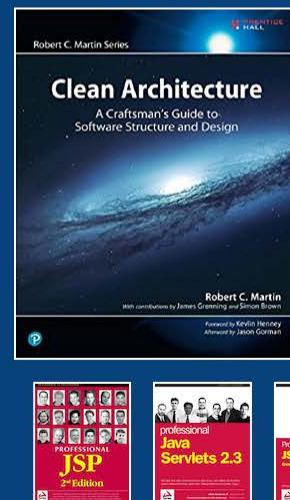
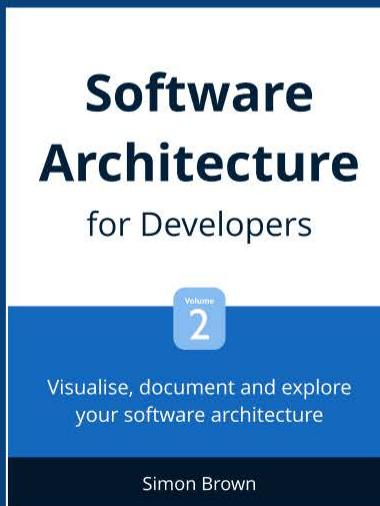
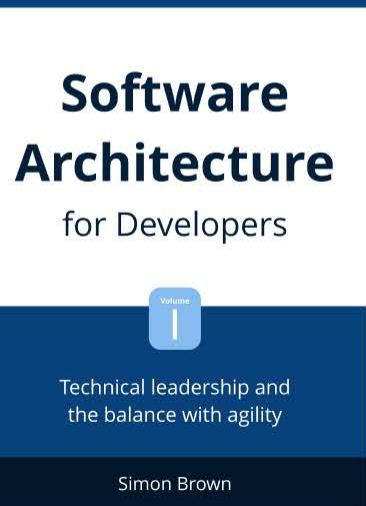
Software Architecture for Developers

@simonbrown

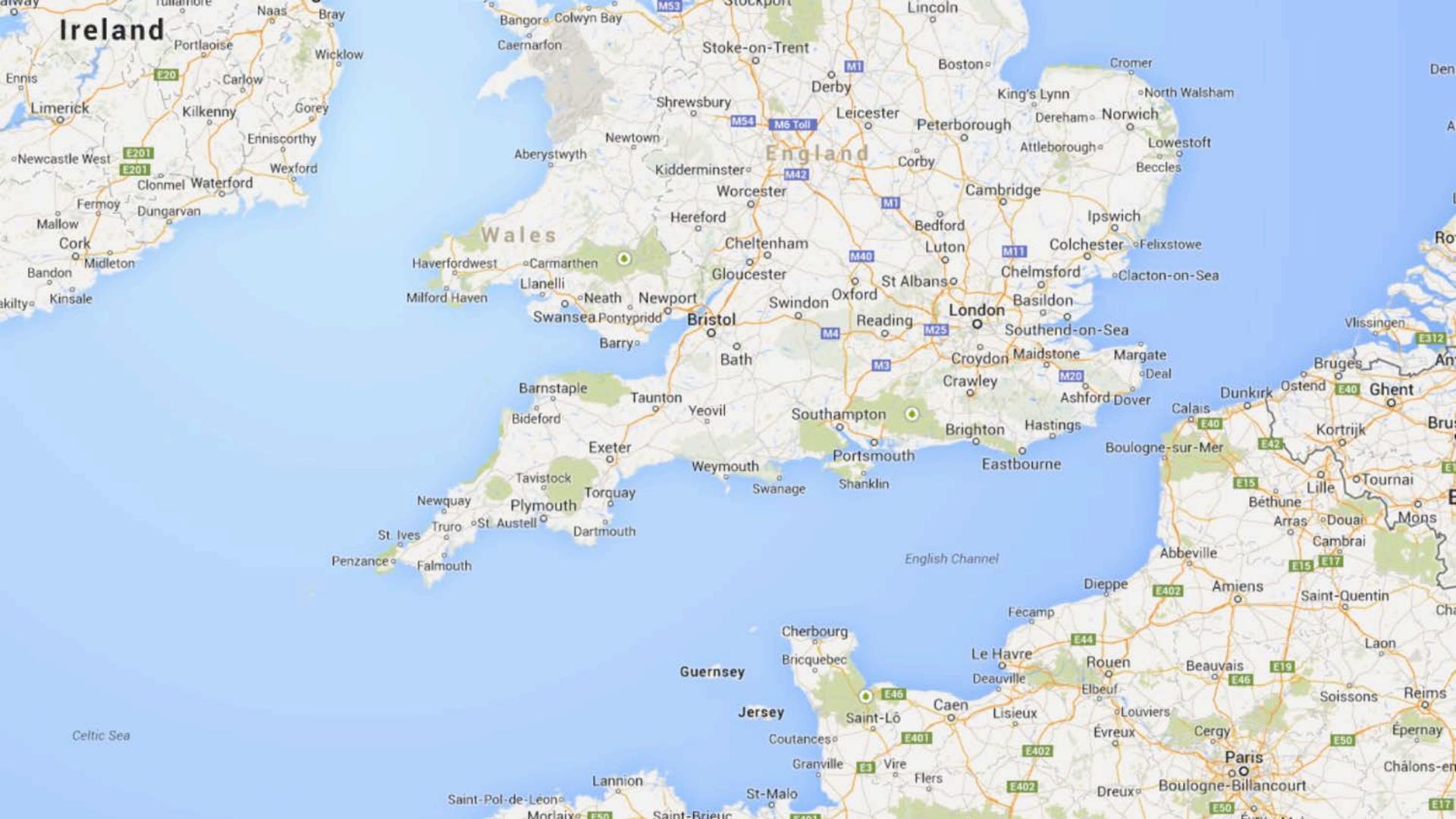
Simon Brown

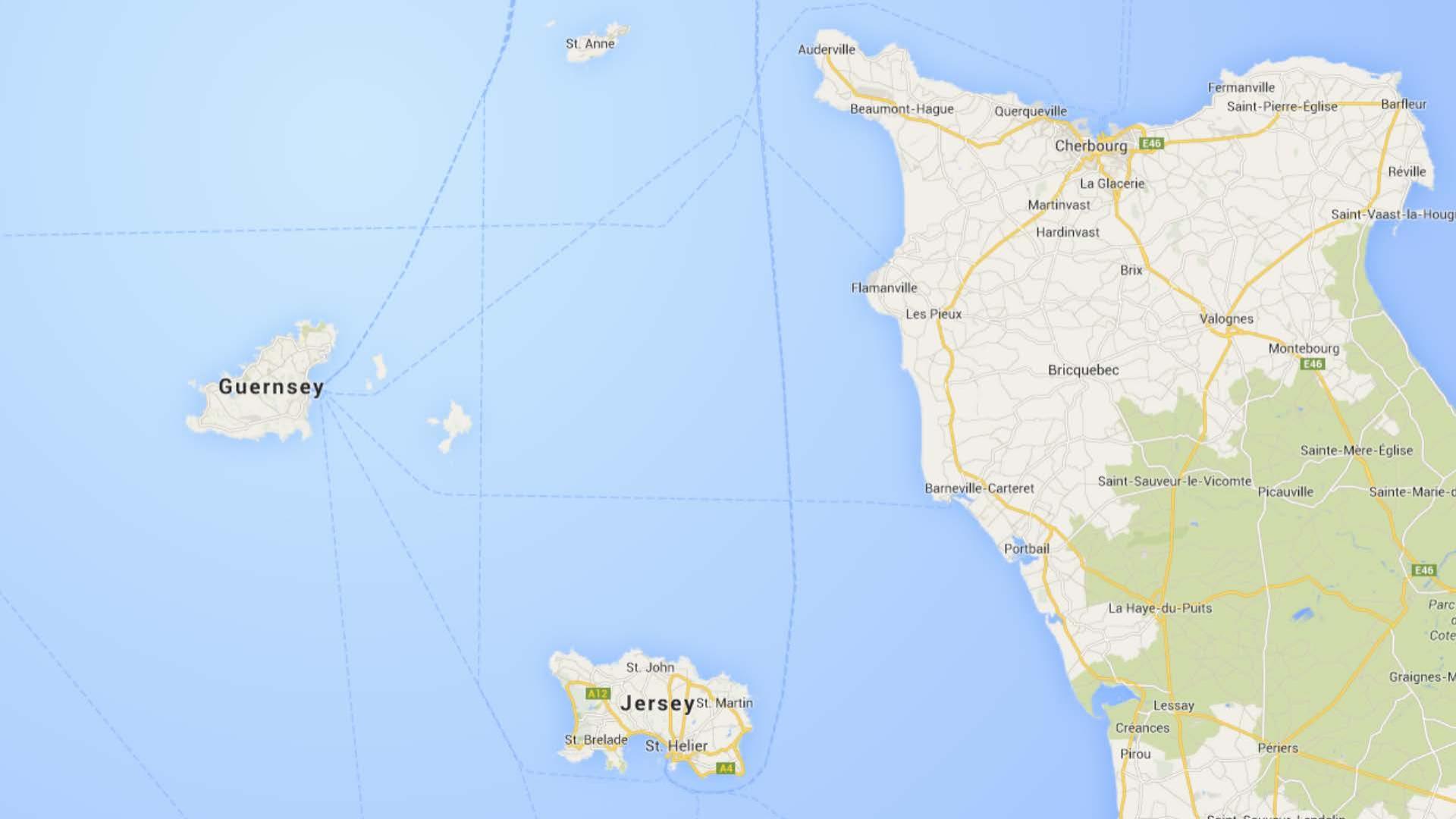
Independent consultant specialising in software architecture,
plus the creator of the C4 model and Structurizr

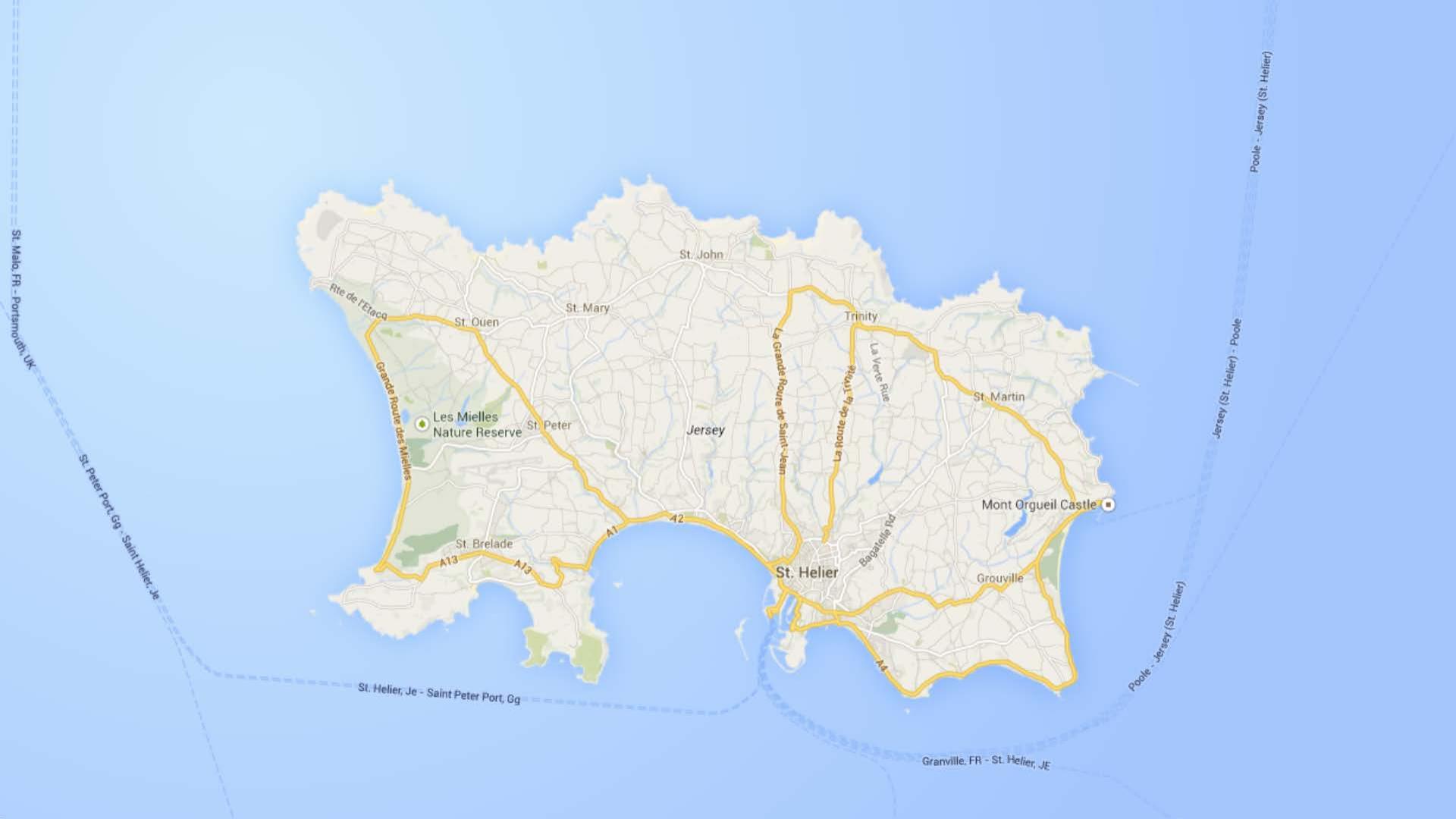
@simonbrown



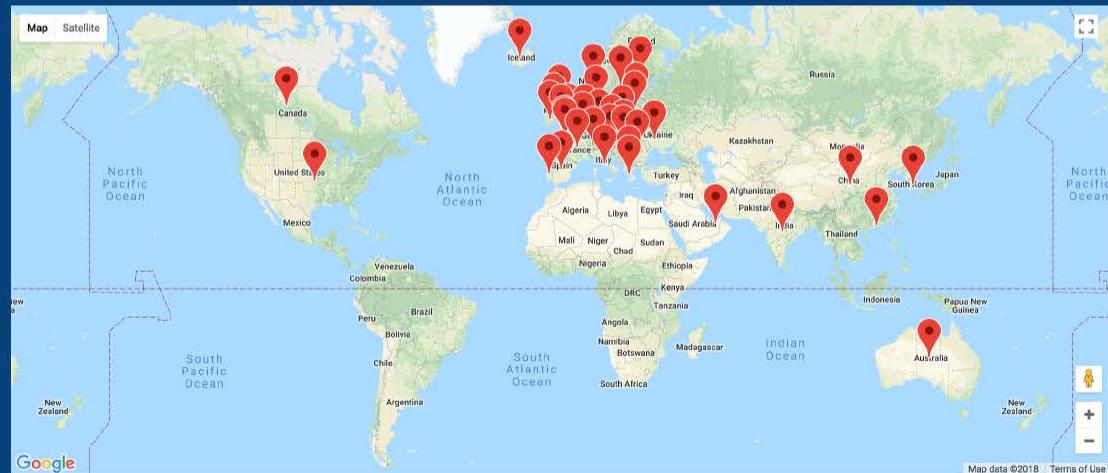








I've visited and spoken at
events/organisations in 30+ countries



What is software architecture?

Structure

The definition of software in terms
of its building blocks and their interactions

Vision

The process of architecting;
making decisions based upon business goals,
requirements and constraints,
plus being able to communicate this to a team

Application Architecture

The internal structure of an application

System Architecture

High-level structure of a software system
(software and infrastructure)

Application Architecture

The internal structure of an application

Enterprise Architecture

Structure and strategy across people, process and technology

System Architecture

High-level structure of a software system
(software and infrastructure)

Application Architecture

The internal structure of an application

Enterprise Architecture

Structure and strategy across people, process and technology

System Architecture

High-level structure of a software system
(software and infrastructure)

Application Architecture

The internal structure of an application

As a noun, design is the named structure or behaviour of a system whose presence resolves ... a force on that system. A design thus represents one point in a potential decision space.

Grady Booch

All architecture is design, but
not all design is architecture.

Grady Booch

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

Grady Booch

As architects, we define
the **significant decisions**

Architecture

Programming language
Monolith, microservices or hybrid approach

Design

Implementation

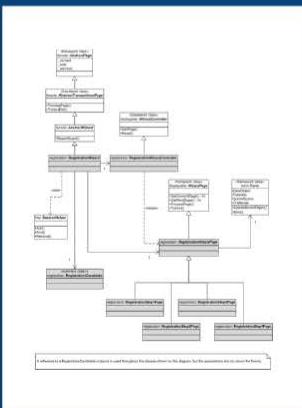
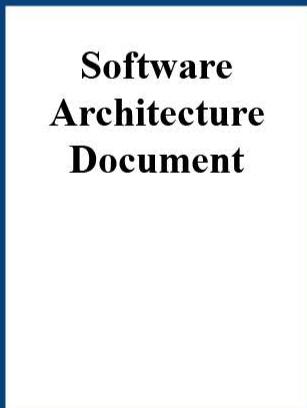
Curly braces on the same or next line
Whitespace vs tabs

What happens if a software development team **doesn't think about architecture?**

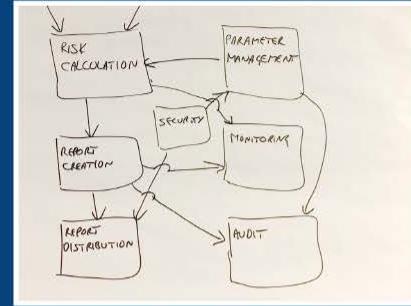
Chaos

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

Big design up front



VS



No design up front

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

Dave Thomas

Software architecture
helps us avoid chaos

Architectural drivers

Requirements drive architecture

(use cases, user stories, features, etc)

Quality attributes

(also known as non-functional requirements,
cross-cutting concerns, service-level agreements, etc)

- Performance
- Scalability
- Availability
- Security
- Disaster Recovery
- Accessibility
- Monitoring
- Management
- Audit
- Flexibility
- Extensibility
- Maintainability
- Interoperability
- Legal
- Regulatory
- Compliance
- i18n
- L10n

Create a **checklist** of quality attributes you regularly encounter

Understand how to **capture**, **refine**
and **challenge** quality attributes

Software lives in the real world,
and the real world has
constraints

Typical constraints include time and budget, technology, people and skills, politics, etc

Constraints can **sometimes**
be prioritised

Principles

are selected by the team

Development principles include
coding conventions, naming
guidelines, testing approaches,
review practices, etc

Architecture and design principles
typically relate to modularity
or crosscutting concerns

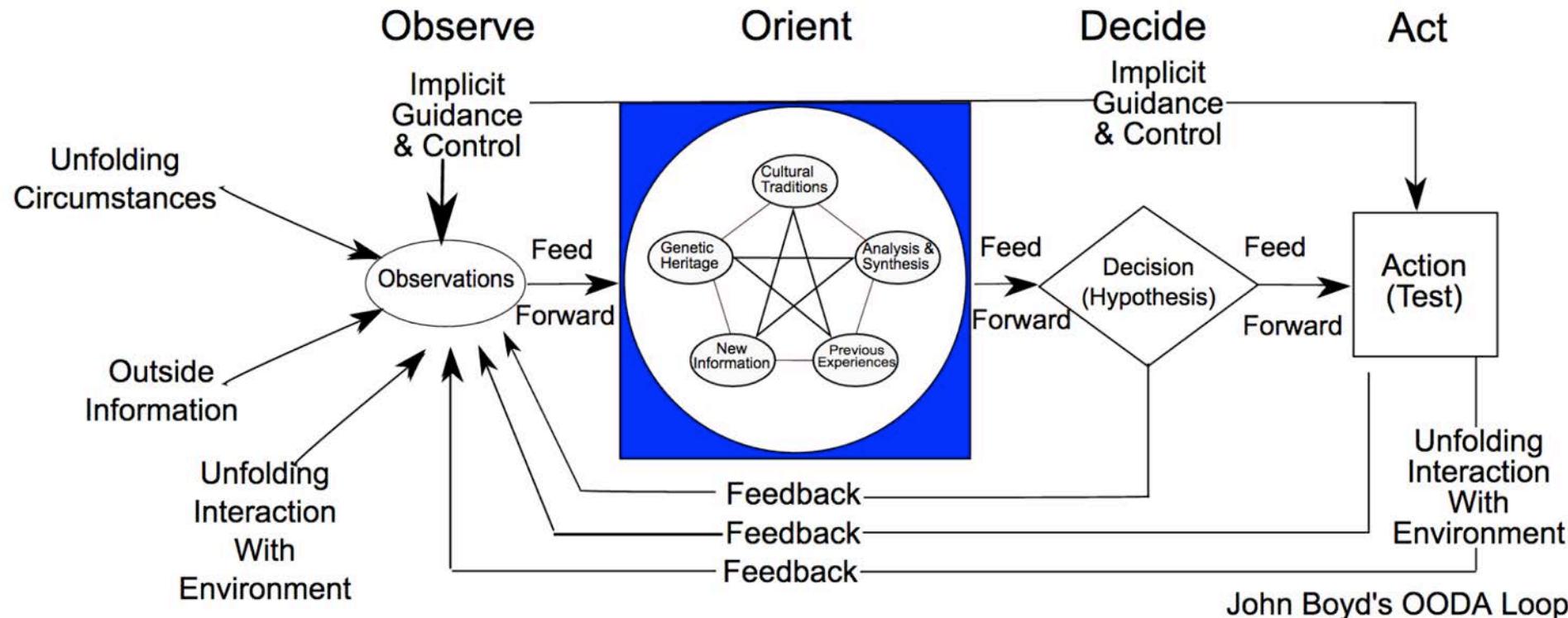
(architectural layering, separation of concerns,
stateless vs stateful, rich vs anaemic domain,
security, error handling, logging, etc)

Ensure you have a good
understanding of the requirements,
quality attributes, constraints
and principles to create
sufficient foundations

What about agile,
and agility?

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

Agile is about a mindset of
continuous improvement



John Boyd's OODA Loop

Inspect and adapt

Agility is relative and time-based

What Lessons Can the Agile Community Learn from A Maverick Fighter Pilot?

Steve Adolph

University of British Columbia

Vancouver, B.C.

Canada

steve@wsaconsulting.com

Abstract

For the agile software development community, agility is defined by the values expressed in the agile manifesto. But in concrete terms, what does it mean for a software project to be agile? US Air Force Colonel

based methodologies is it by definition not agile? Many RUP and UP experts claim their process is agile [5, 6, 7] and many practitioners (including this author) will enthusiastically claim that even though they were not following an agile methodology, their projects were agile. So what does it really mean for a software



Continuous attention to
technical excellence and
good design enhances agility.

Principle 9 of the Manifesto for Agile Software Development

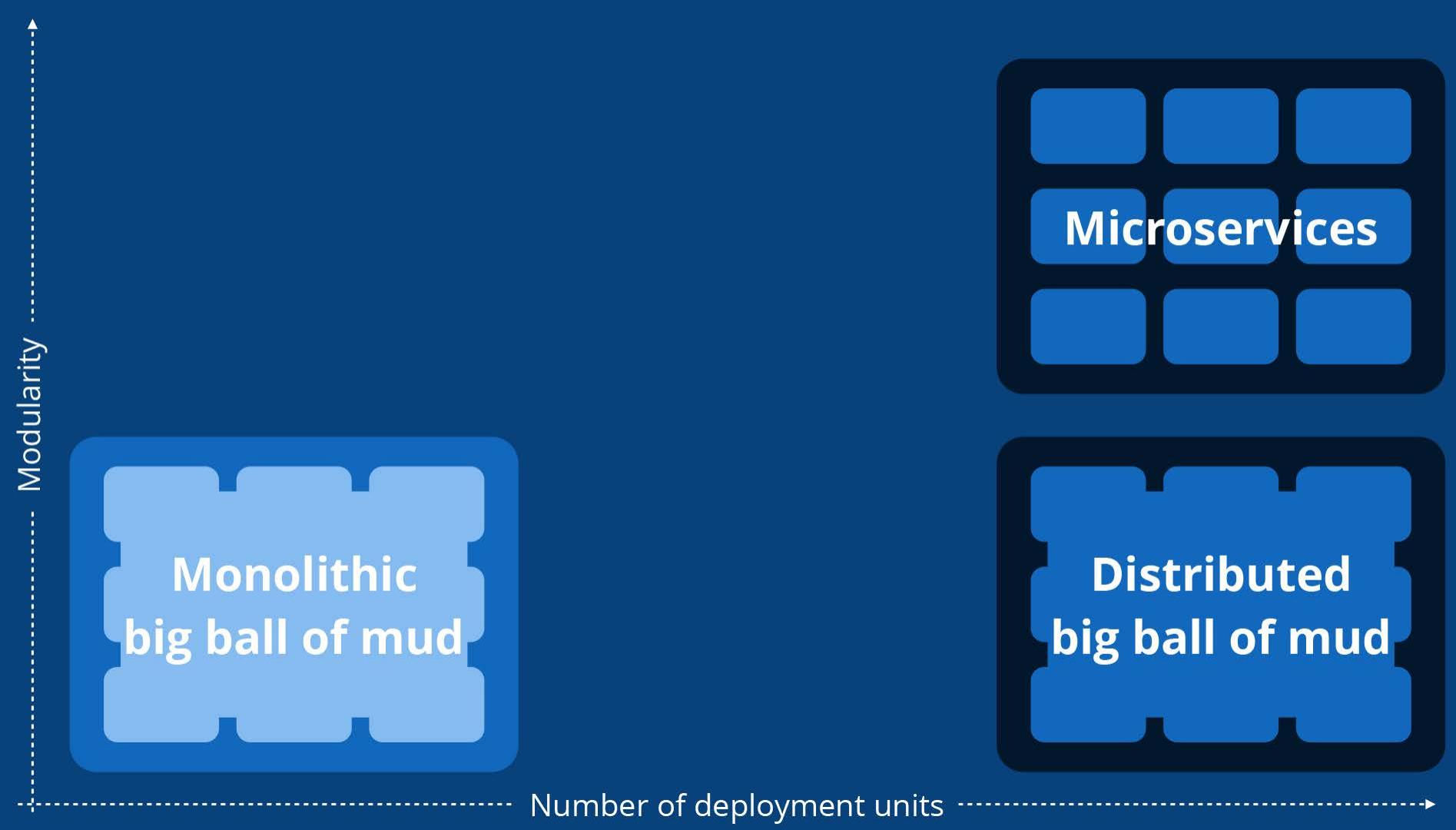
A good architecture
enables agility

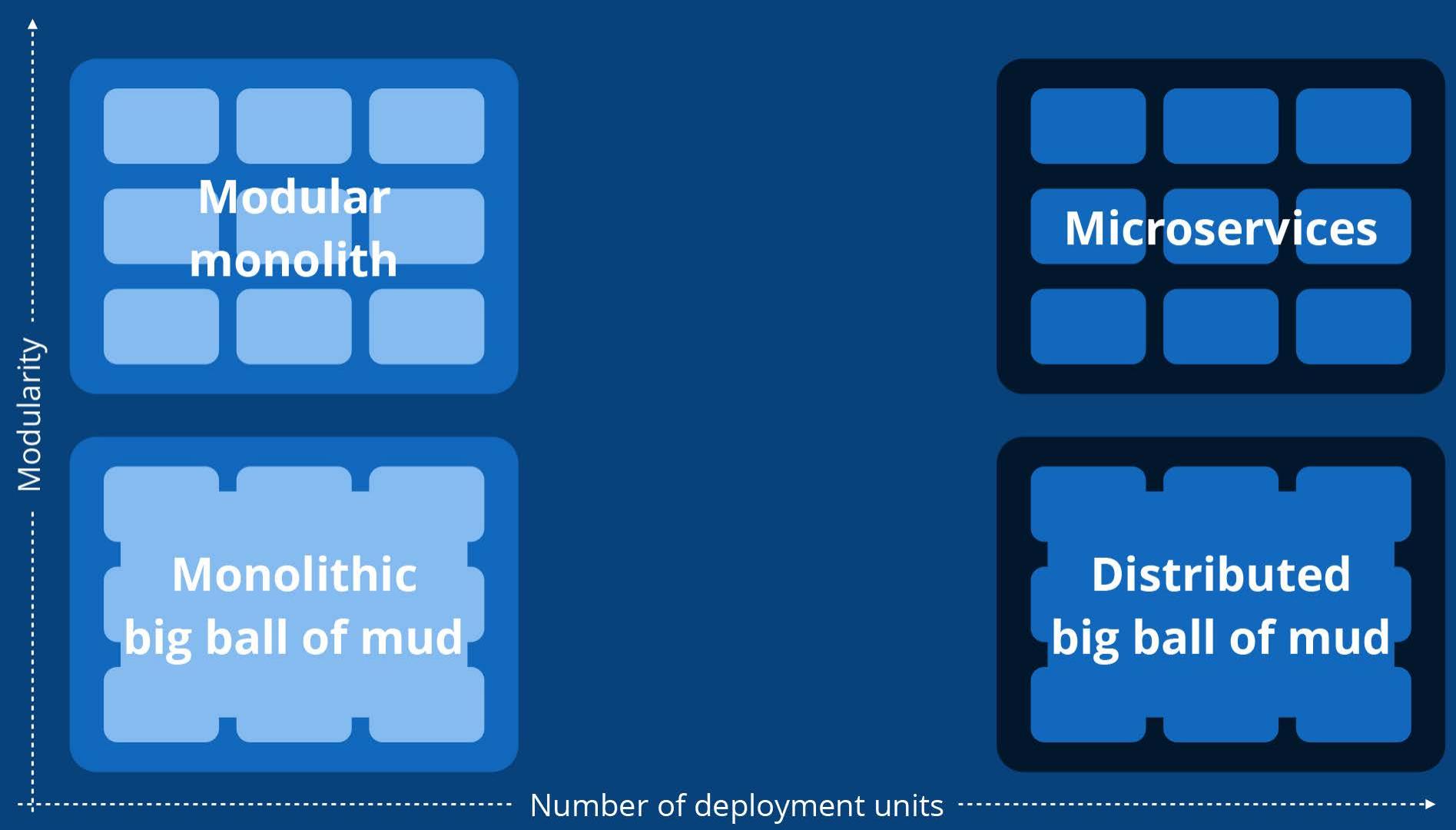
Modularity

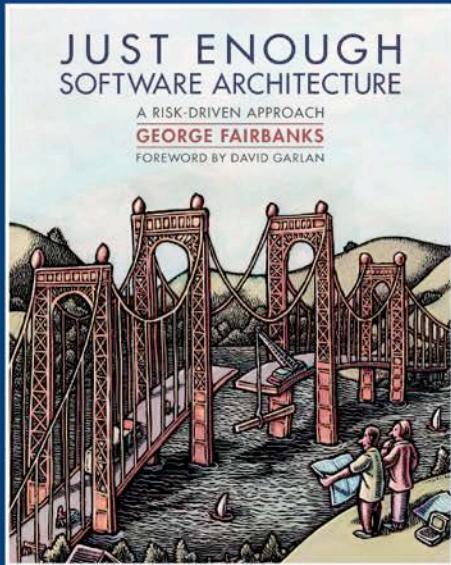


Number of deployment units









A good architecture rarely
happens through
architecture-indifferent design

Agility is a
quality attribute

Questions
and break

The software architecture role

The software architecture role
is about the “**big picture**”
and, sometimes, this means
stepping away from the code

The software architecture role

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

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.

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.

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”.

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
the software delivery.

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
the software delivery.

Quality assurance

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

I am a senior developer. Recently, I was promoted to the position as architect. Could anyone please let me know which tools/software an architect should master/be familiar with. Thank you

Experience is important ...
software architecture is not a rank!

Do you have your own definition
of the software architecture role?

Every team needs
technical leadership

Introducing control?
Avoiding chaos?

How much control do you need?



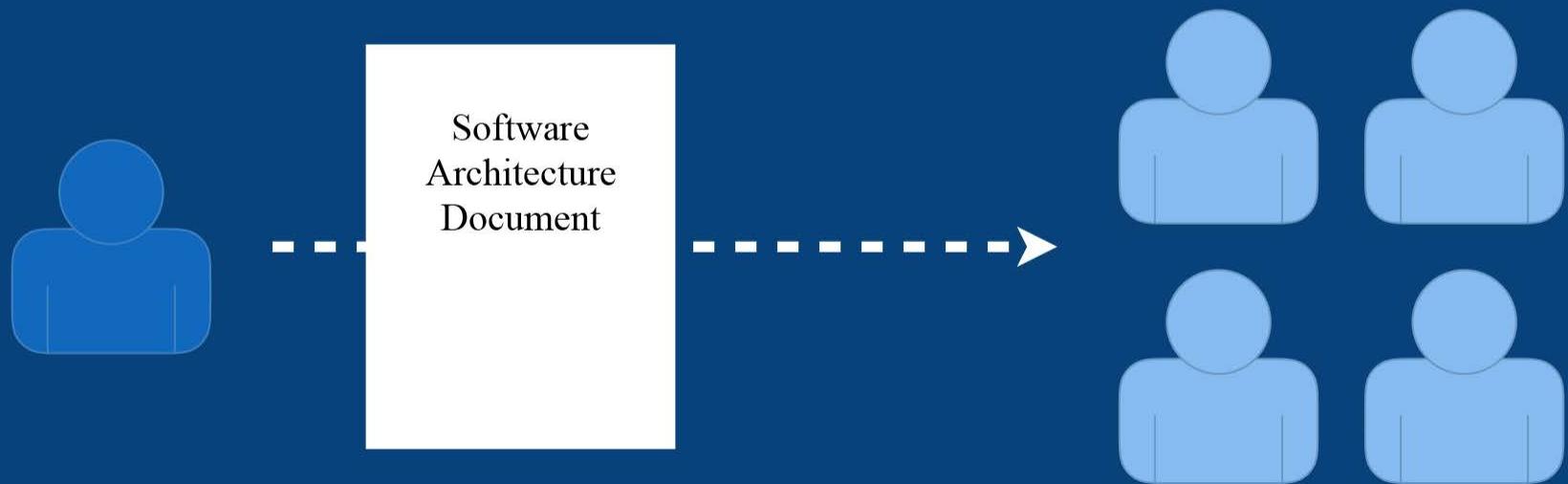
Different types of teams need
different leadership styles



Pair architecting

Collaborative technical leadership
is not easy

Software development is not a relay sport



AaaS

Architecture as a Service

Continuous technical leadership

(somebody needs to continuously steer the ship)

Should software architects
write code?

Poll

Should software architects write code?

Yes

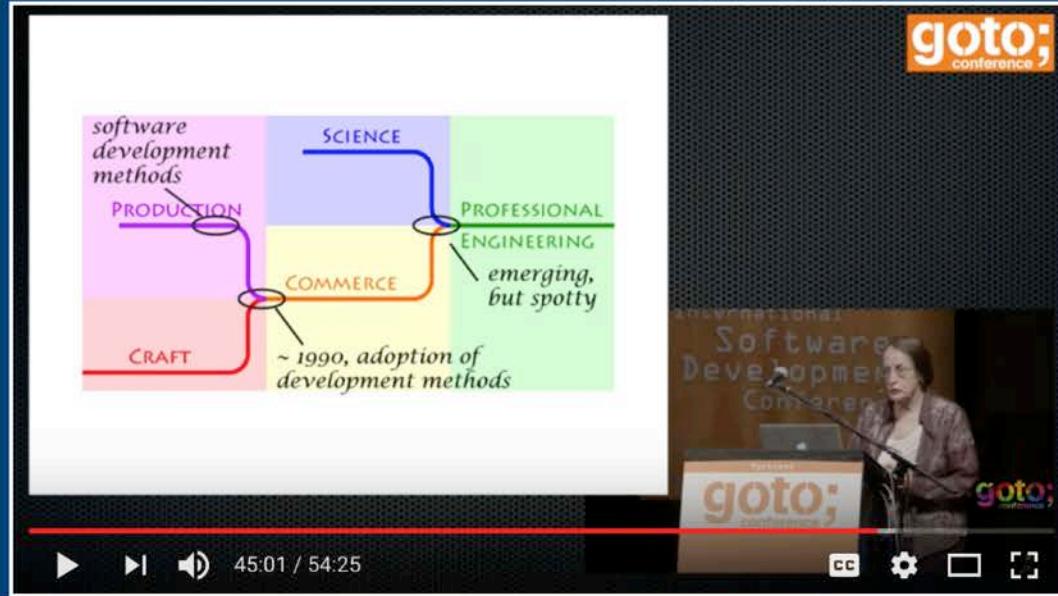
No

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

Don't code all of the time!

There is often a tension between
being “senior” and writing code...

Software architects
should be
master builders



Progress Toward an Engineering Discipline of Software

Mary Shaw

Software architecture is not a
“post-technical” career option!

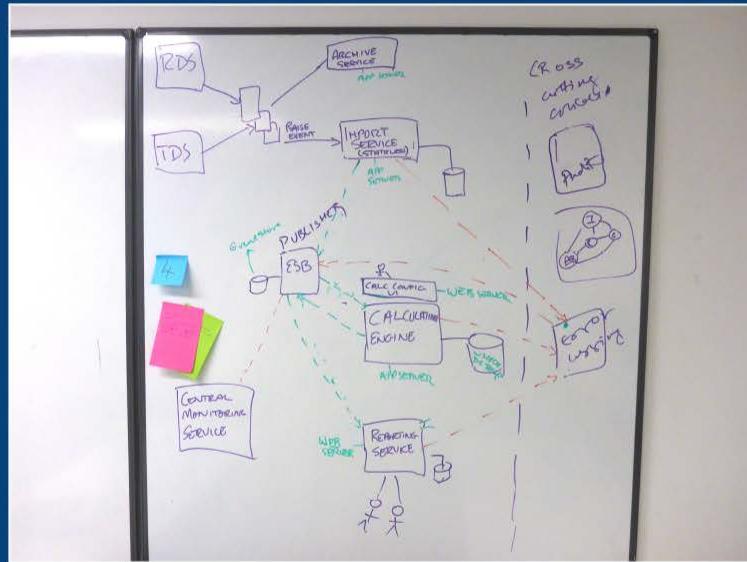
Technology skills

(depth and breadth)

Good software architects are
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?

Soft skills

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



Talking with Tech Leads

From Novices to Practitioners

Patrick Kua

Foreword by Jim Webber

Collaborate or fail

Domain knowledge

(or the ability to learn quickly)

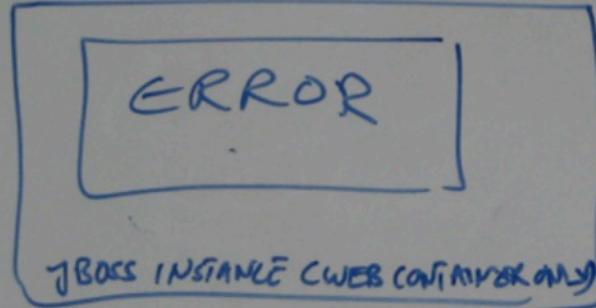
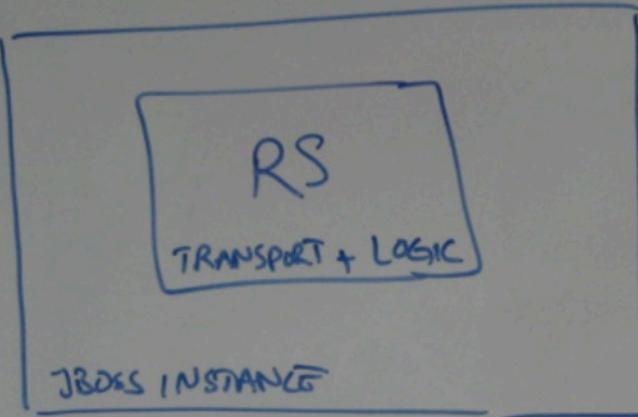
The software architecture role
is multi-faceted
(technology, soft skills, domain knowledge)

Questions
and break

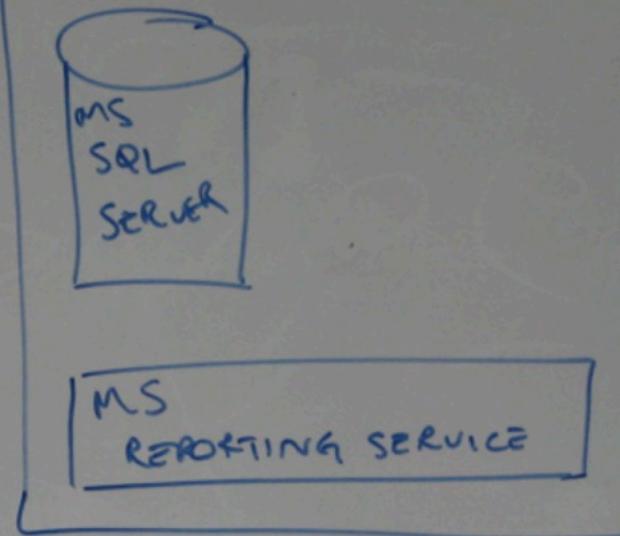
Visualising software architecture

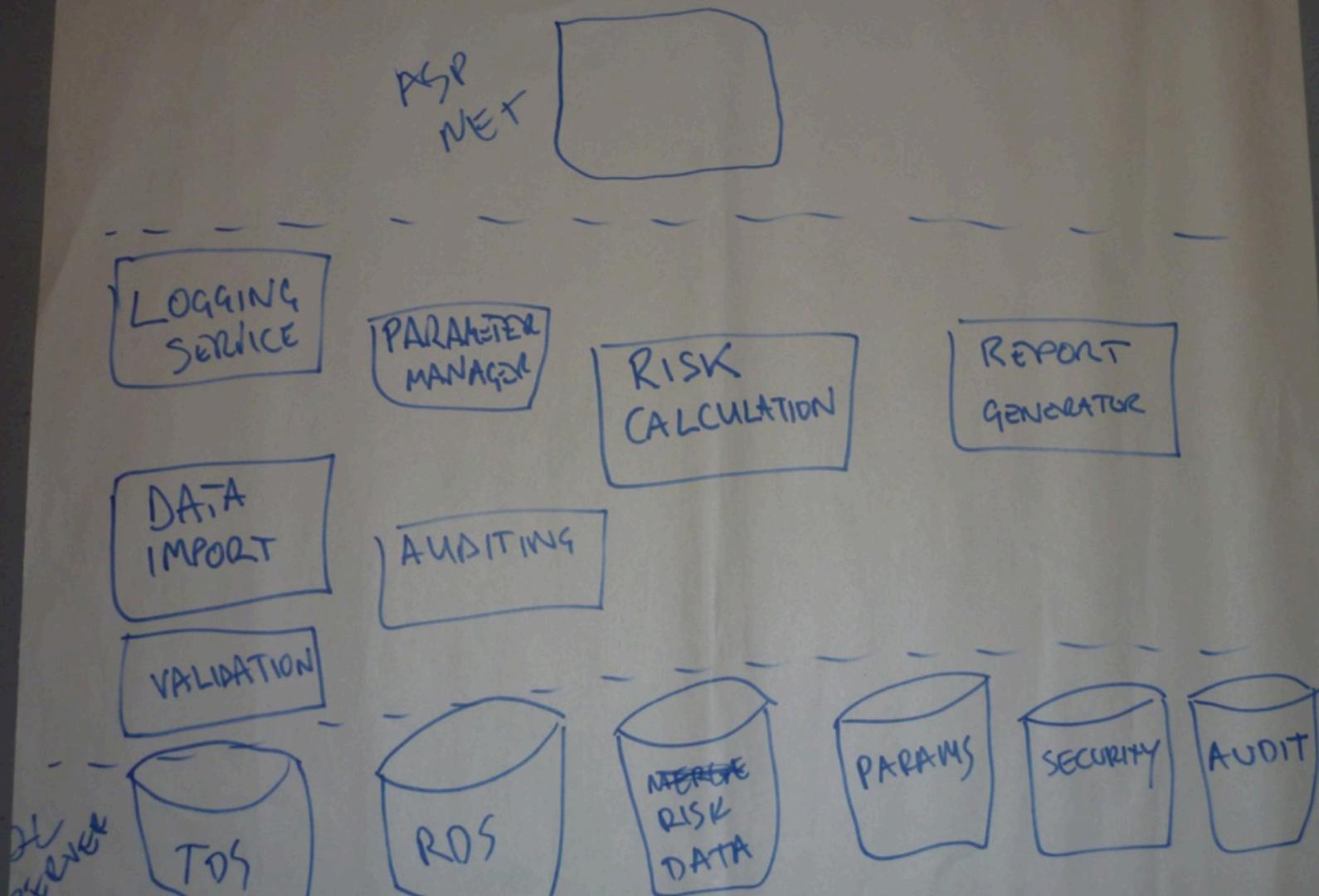
I work with teams around the world,
helping them to draw
software architecture diagrams

UNIX BOX



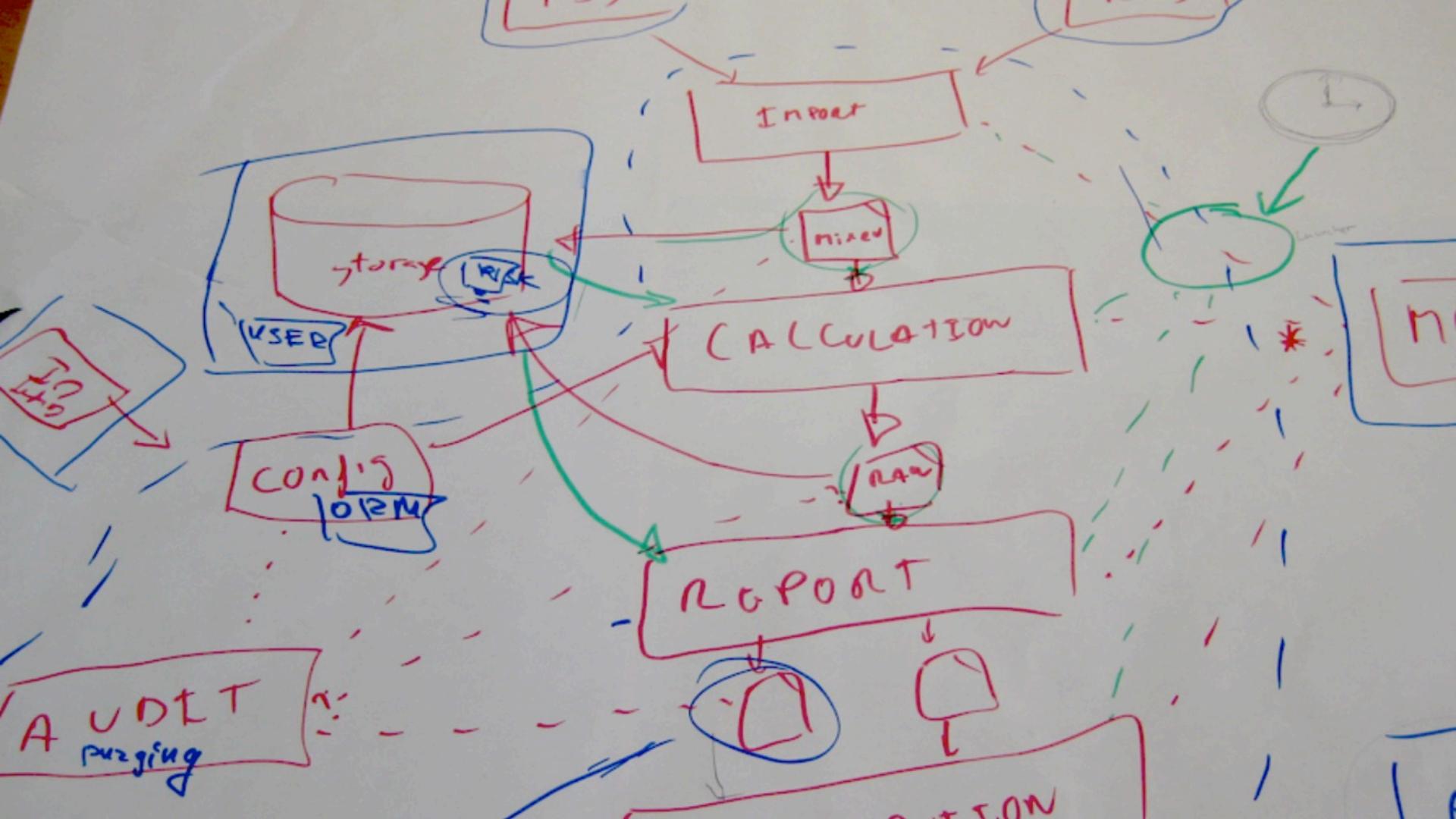
WINDOWS BOX

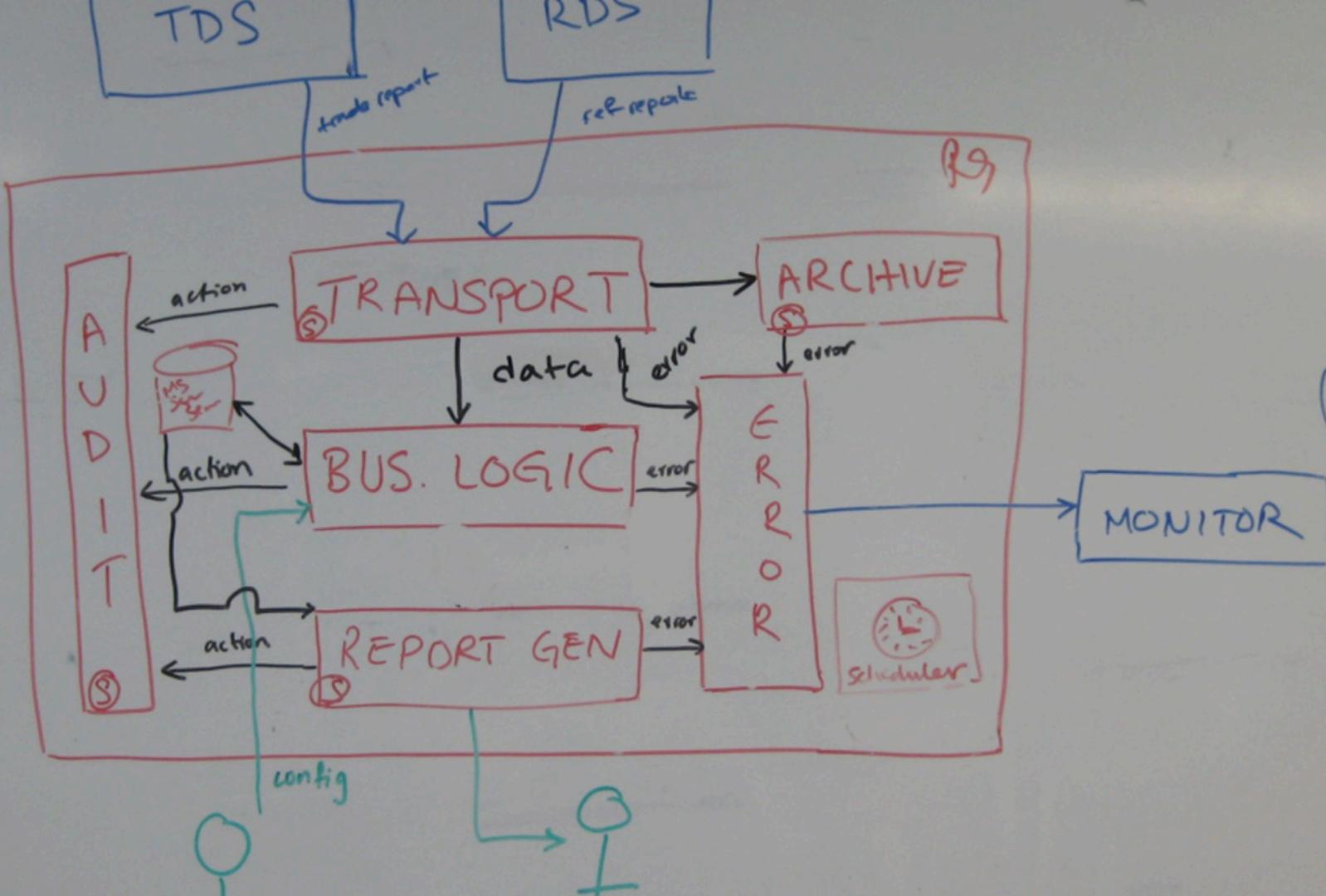


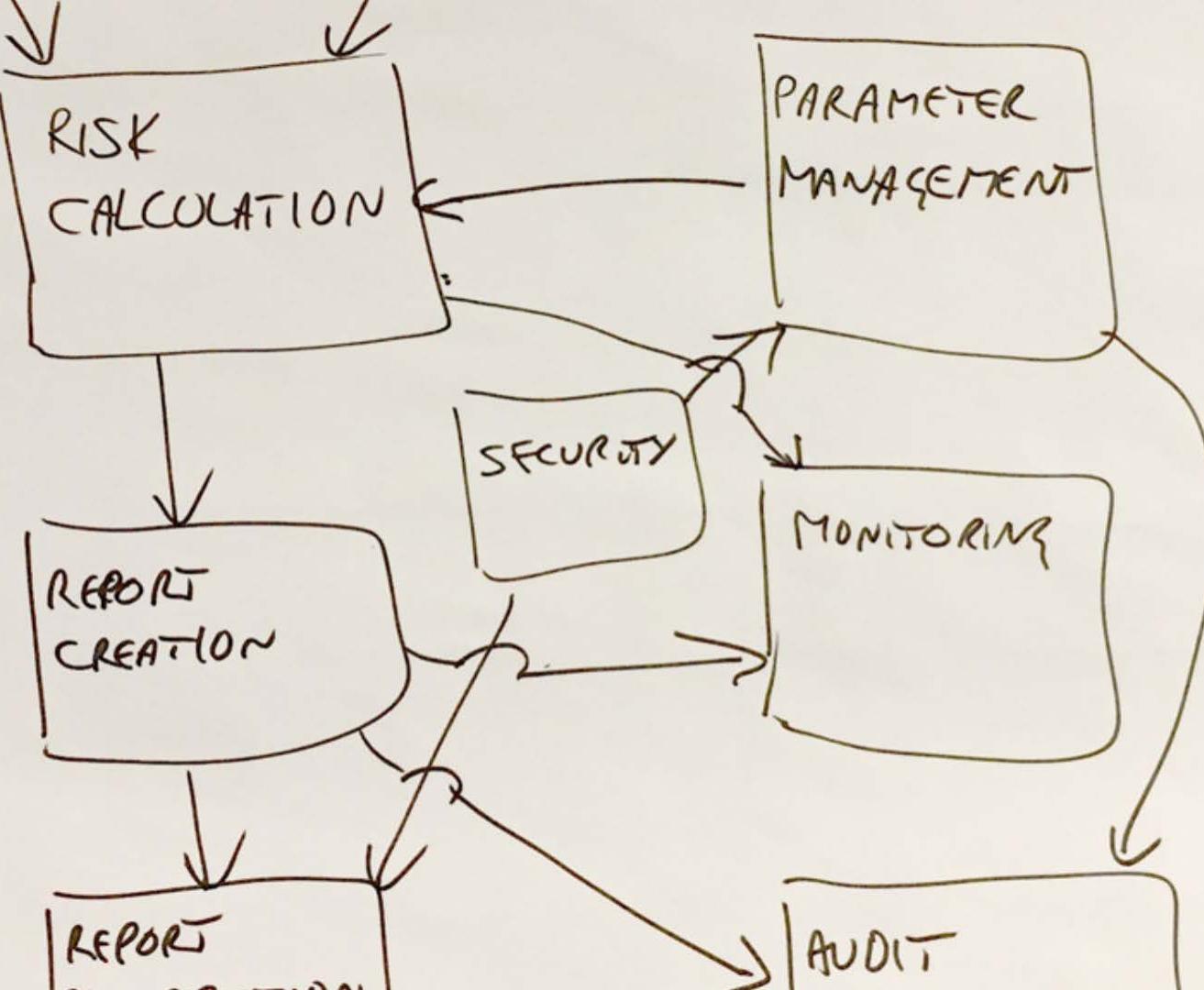


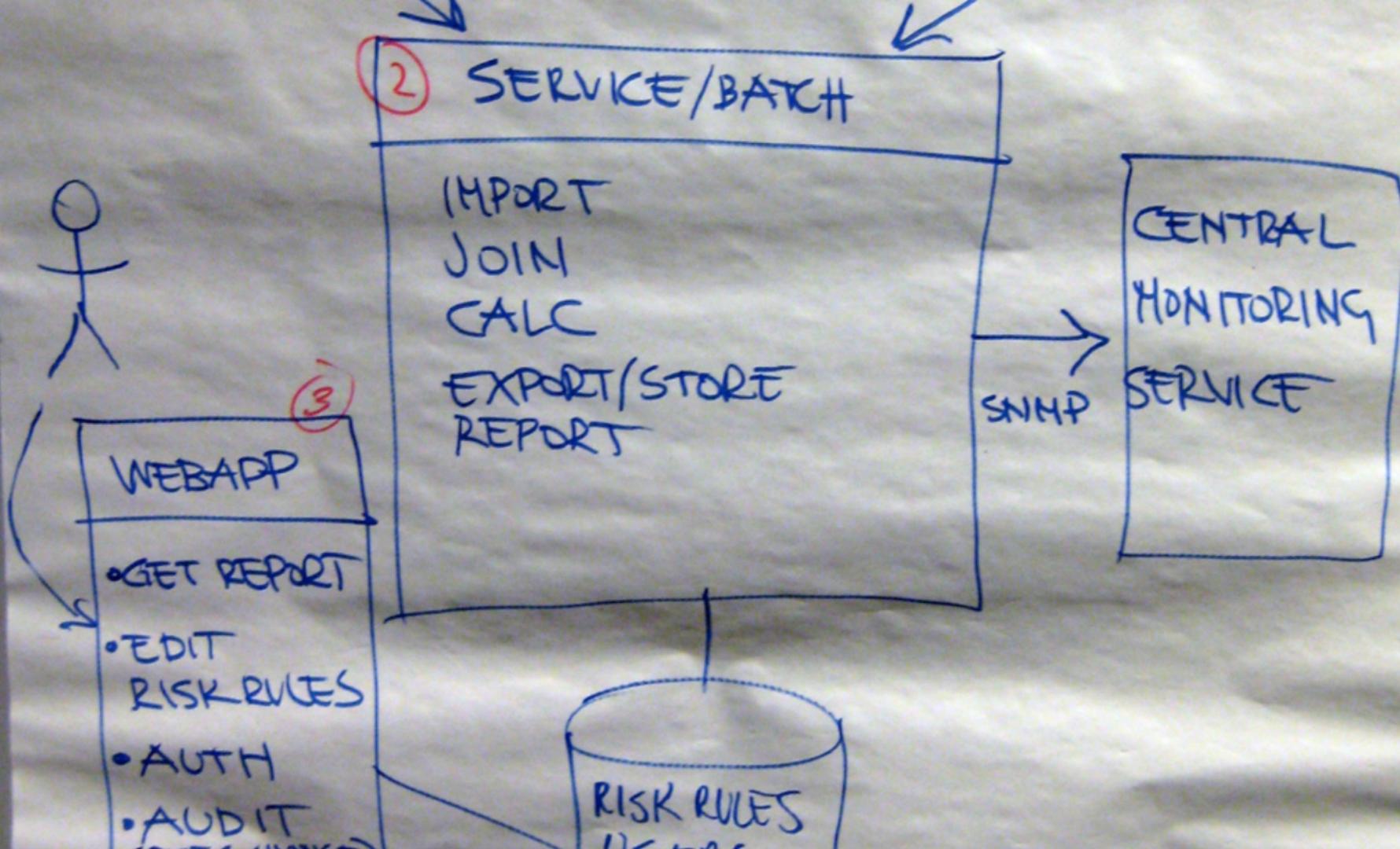
FUNCTIONAL VIEW

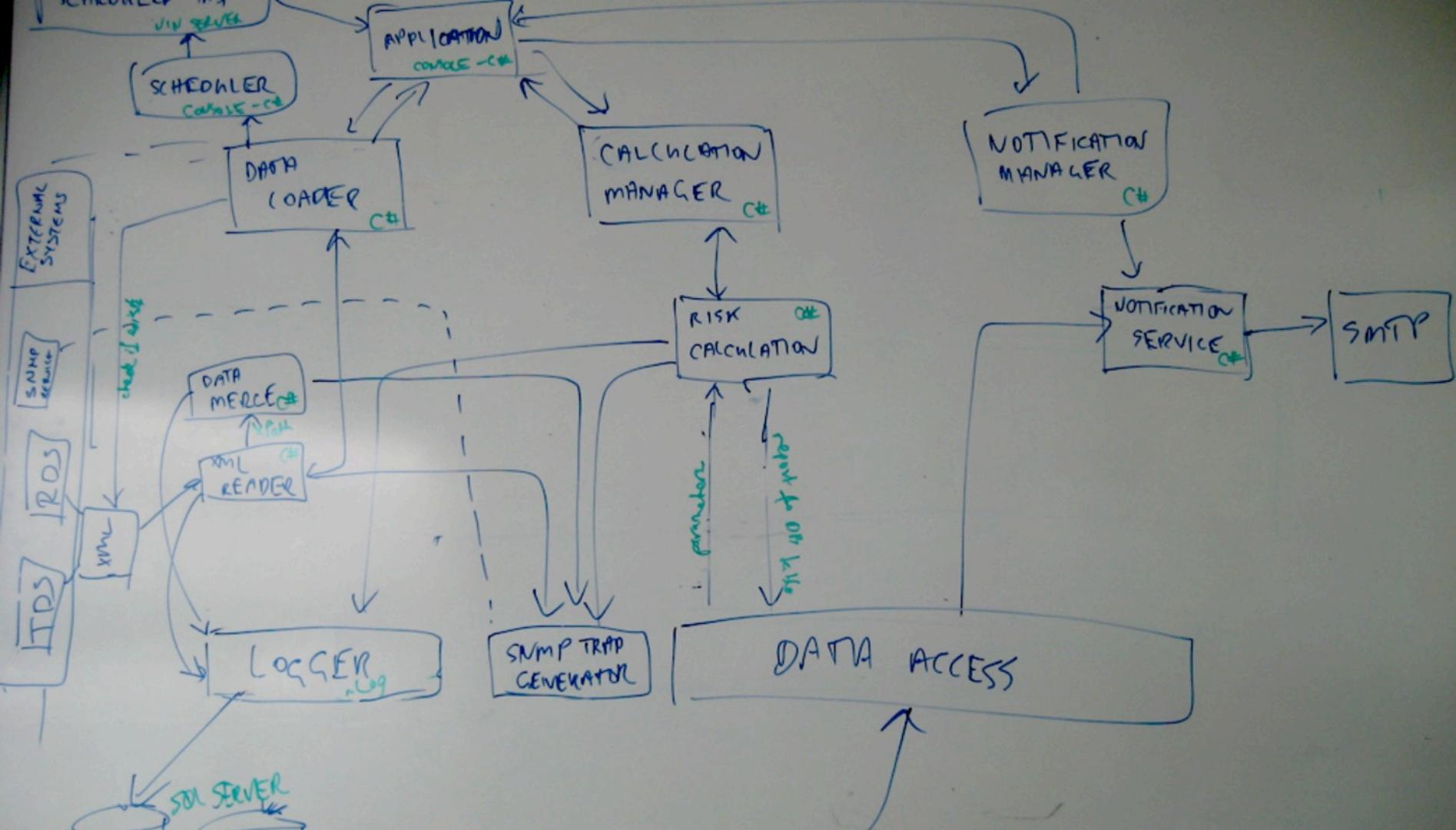


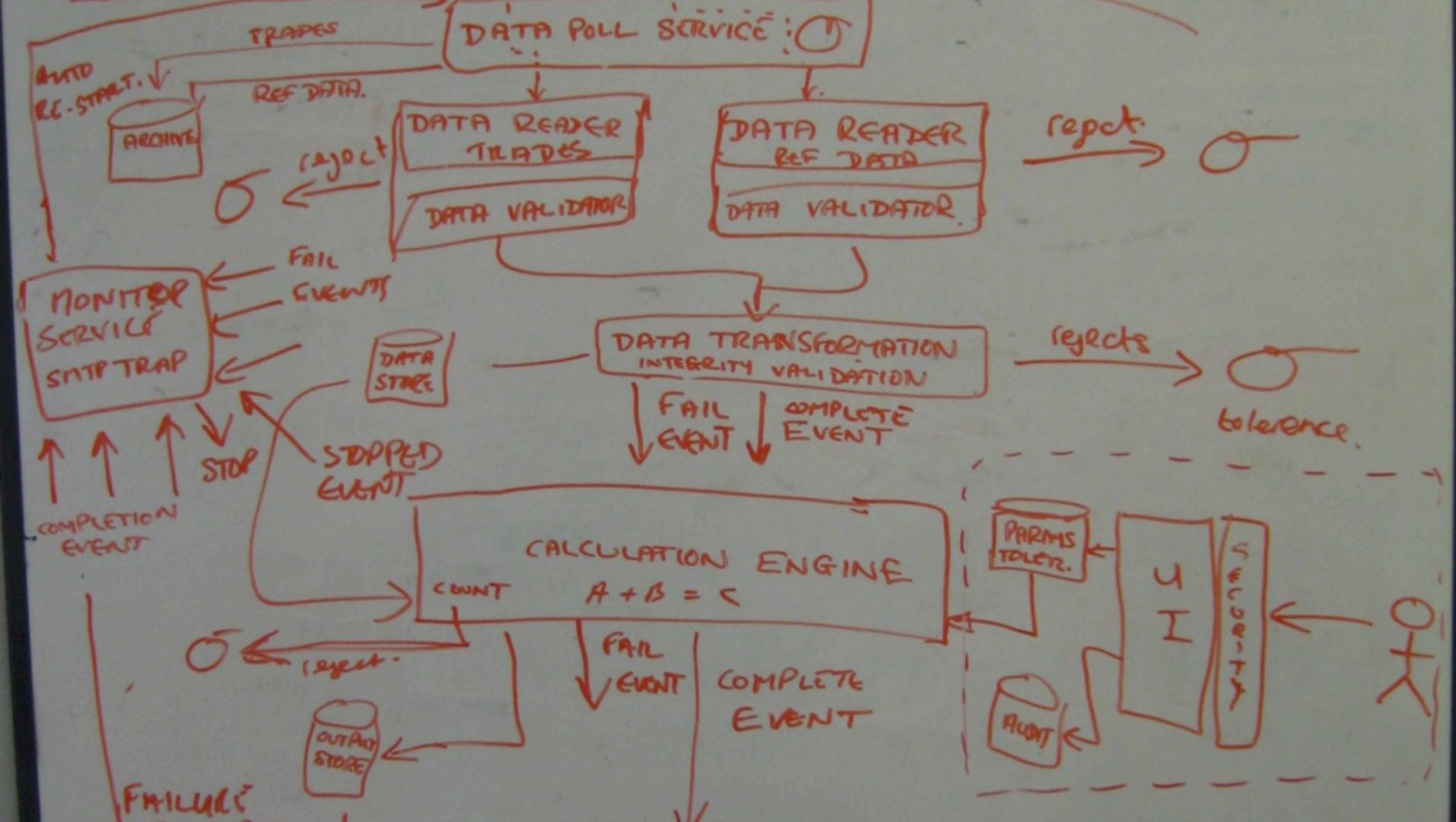


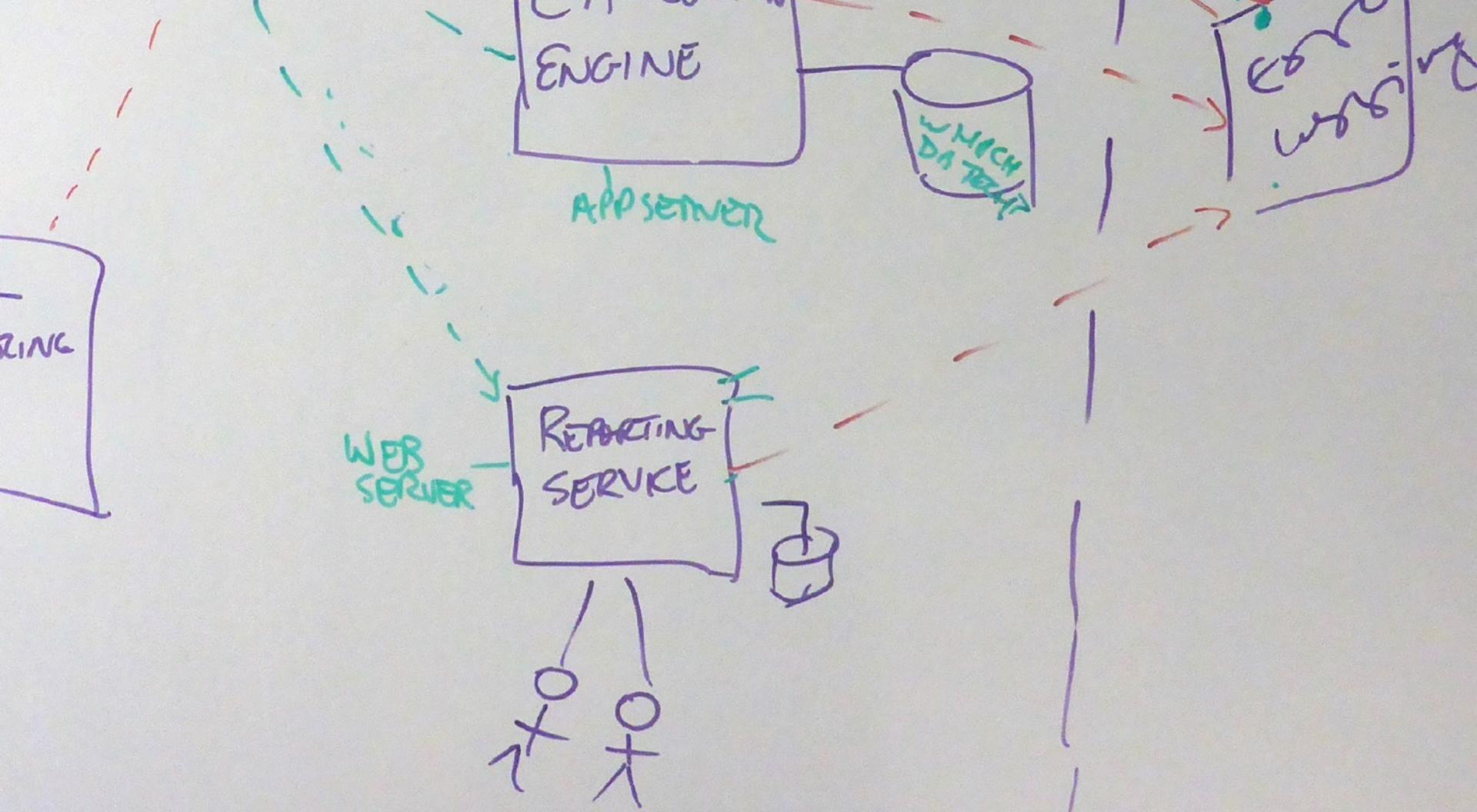




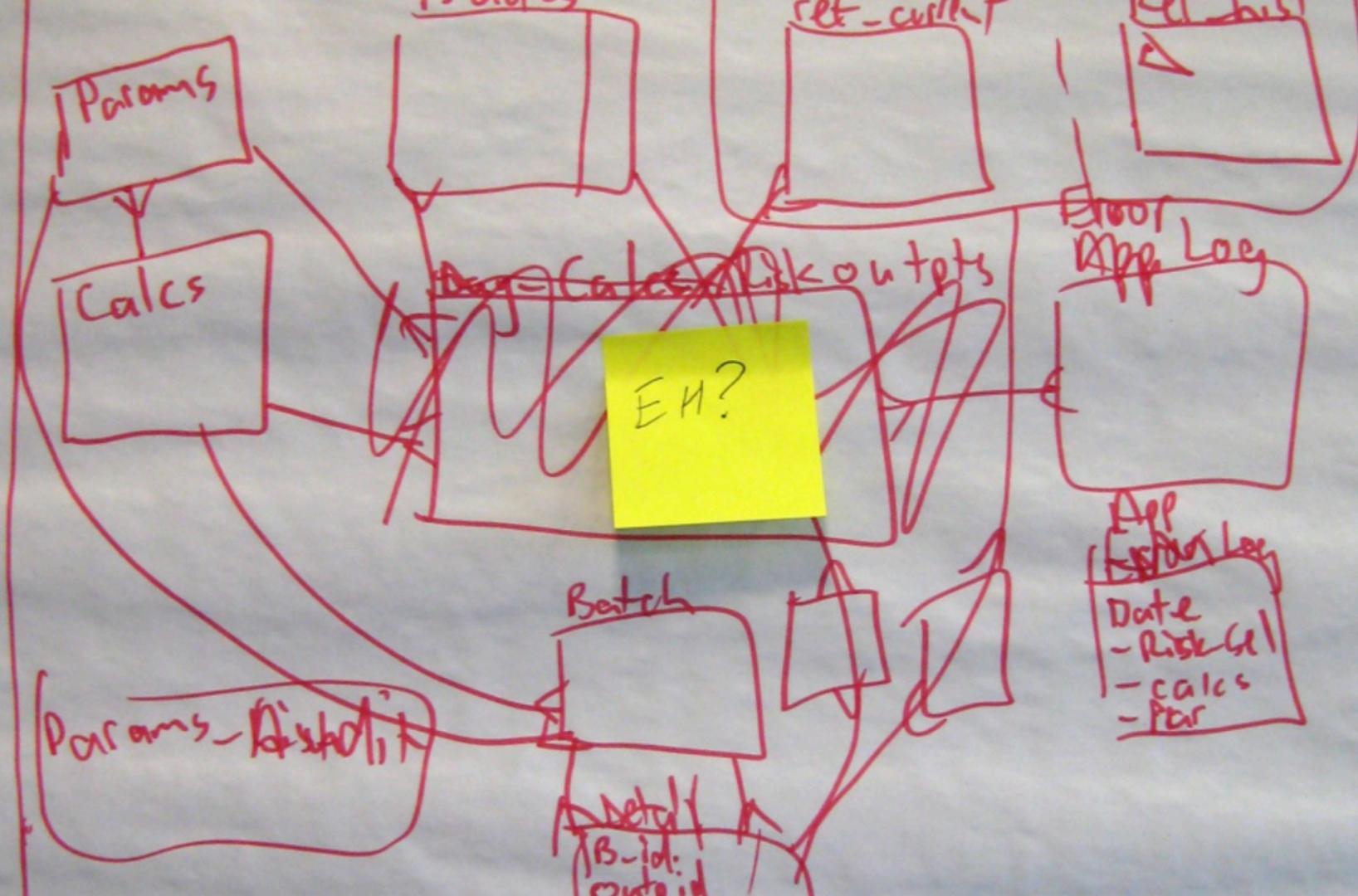










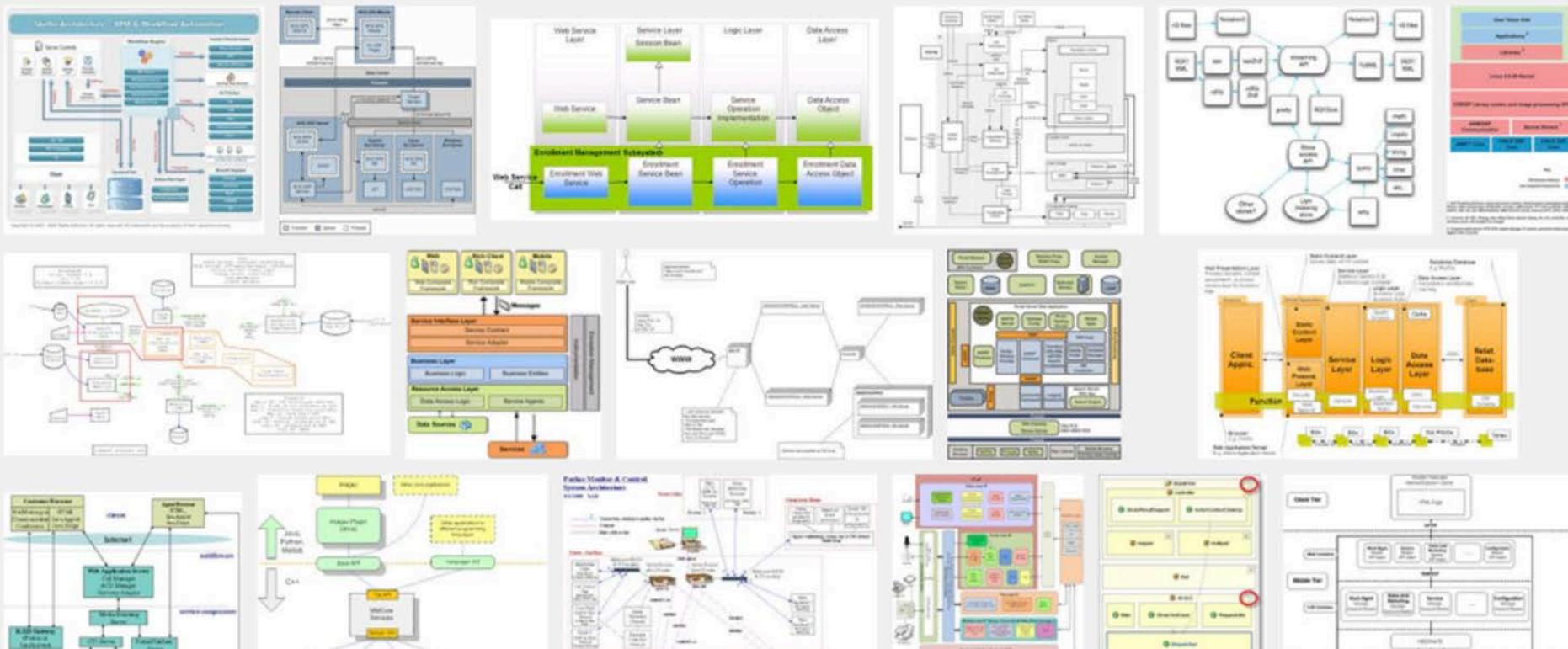


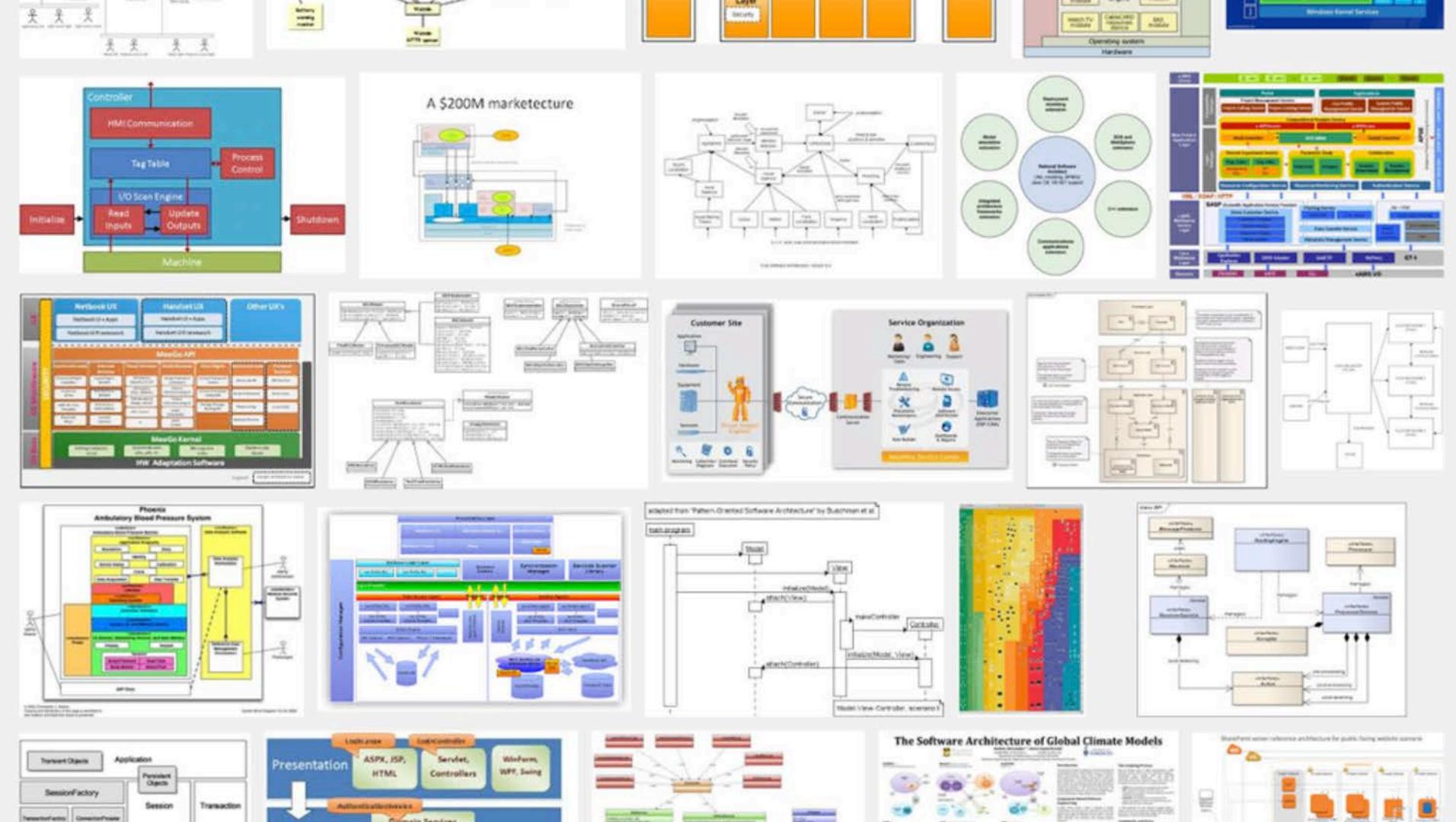


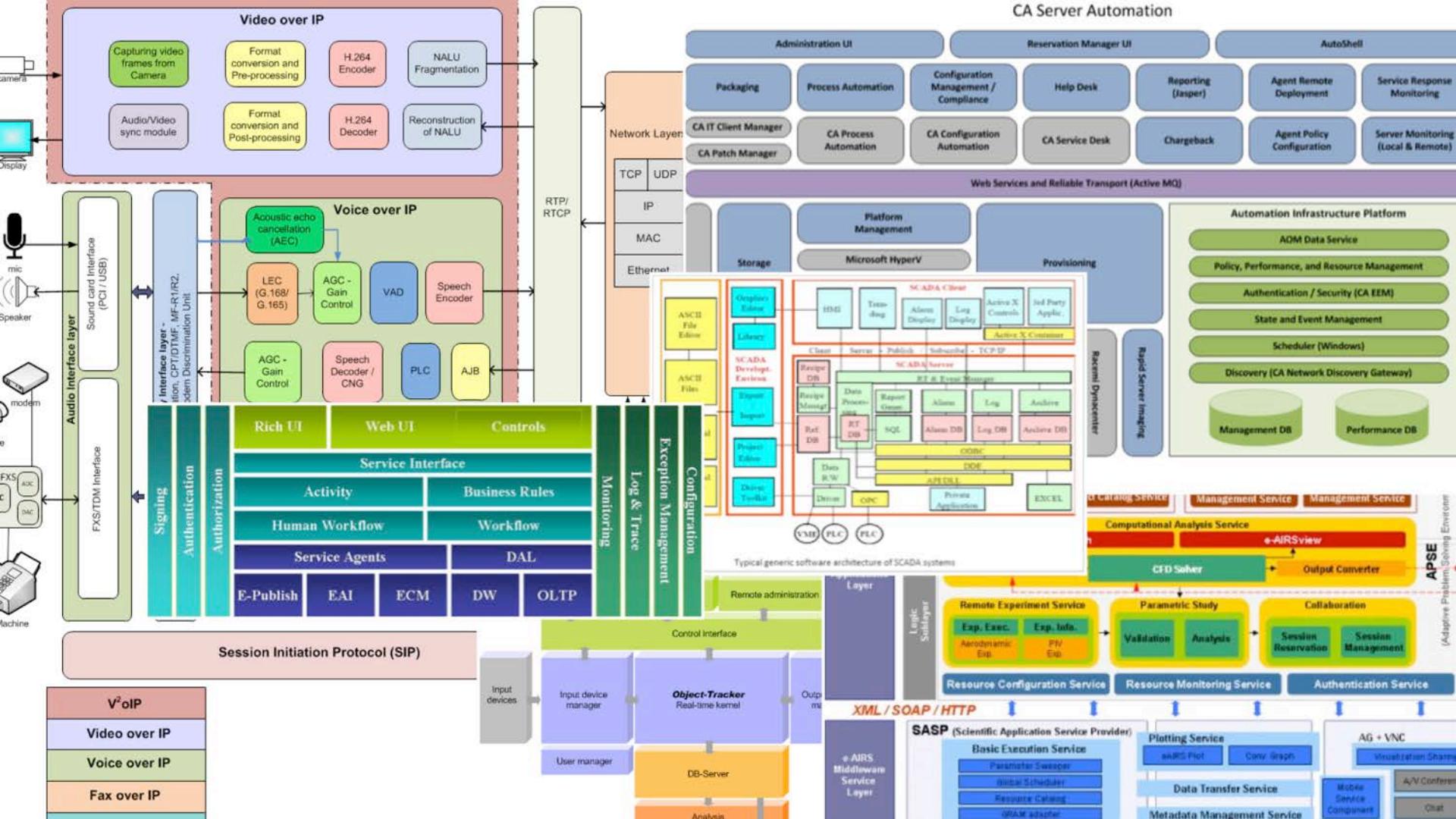
Cookies help us deliver our services. By using our services, you agree to our use of cookies.

[Learn more](#)

[Got it](#)







Moving fast in the same direction
as a team requires

good communication

In my experience, software architects struggle
to communicate software architecture

Do you use UML?

Poll

Do you use UML?

Yes

No



In my experience, optimistically,
1 out of 10 people use UML

Who are the **stakeholders** that
you need to communicate
software architecture to;
what **information** do they need?



There are many **different audiences** for diagrams
and documentation, all with **different interests**

(software architects, software developers, operations and support staff, testers,
Product Owners, project managers, Scrum Masters, users, management,
business sponsors, potential customers, potential investors, ...)

The primary use for
diagrams and documentation is
communication and learning

Software architecture diagrams,
when connected to the code,
are also a fantastic tool for
architectural improvement

To describe a software architecture,
we use a model composed of
multiple views or perspectives.

Architectural Blueprints - The “4+1” View Model of Software Architecture

Philippe Kruchten

The description of an architecture—the decisions made—can be organized around these four views, and then illustrated by a few selected *use cases*, or *scenarios* which become a fifth view. The architecture is in fact partially evolved from these scenarios as we will see later.

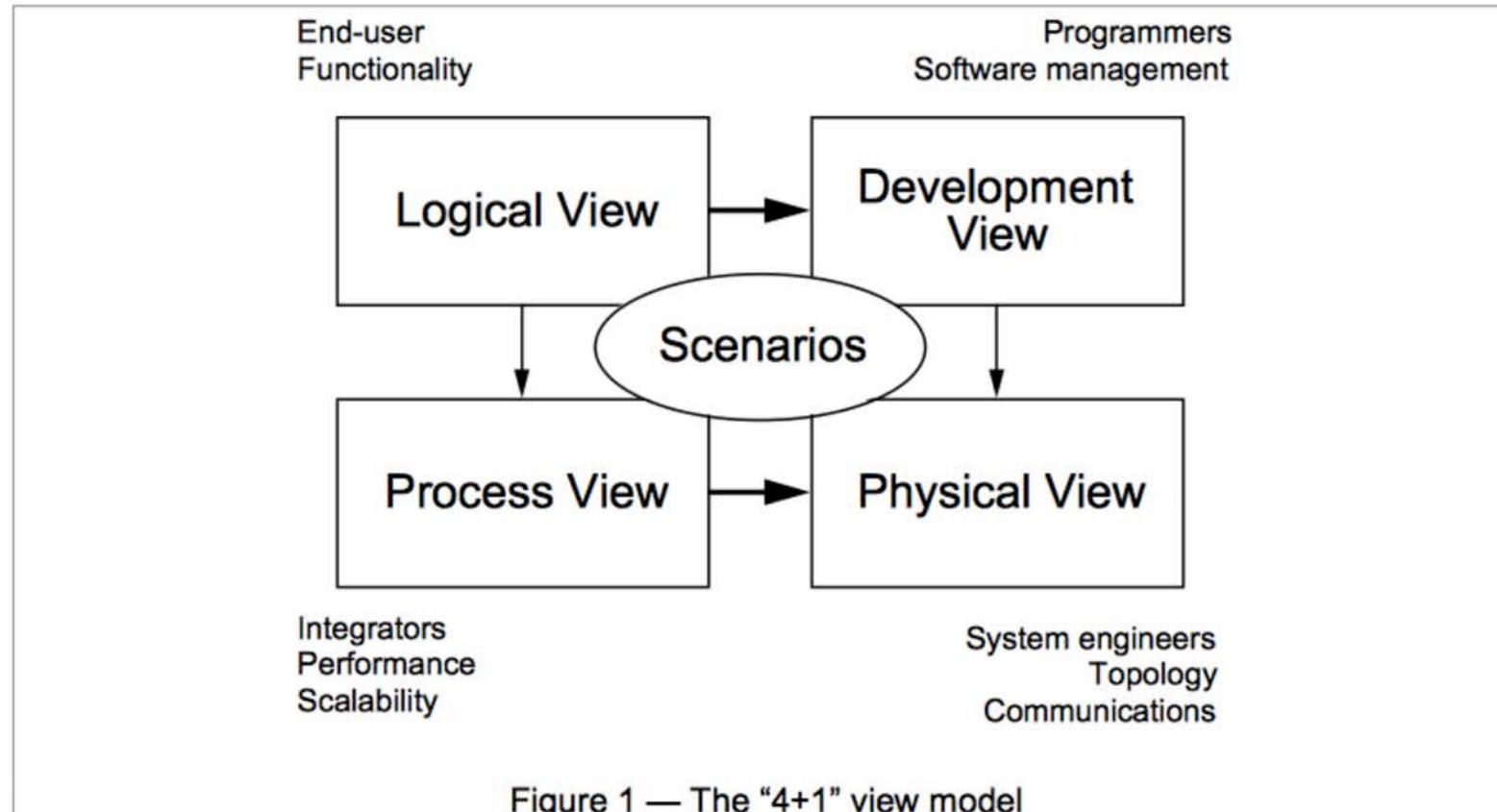
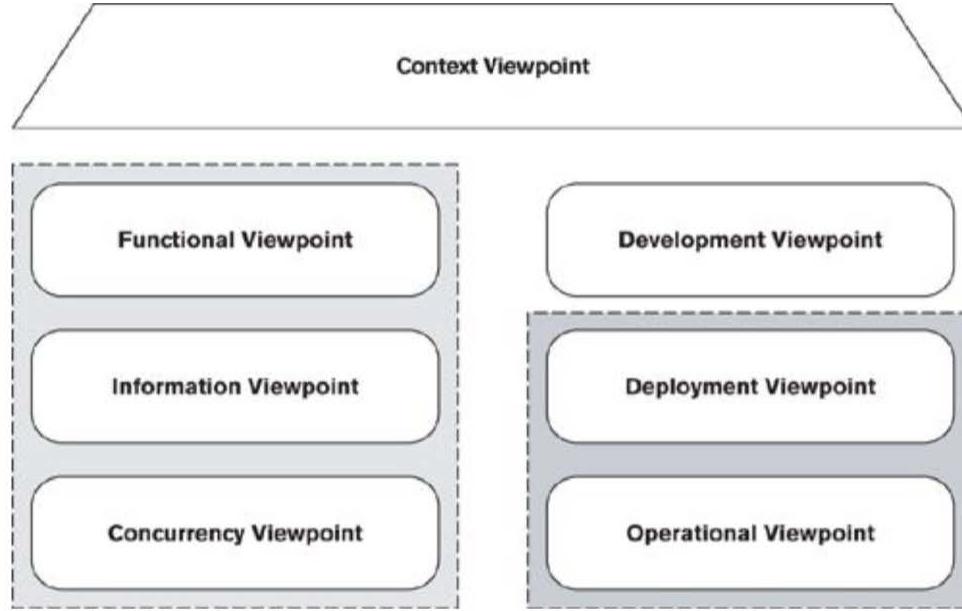
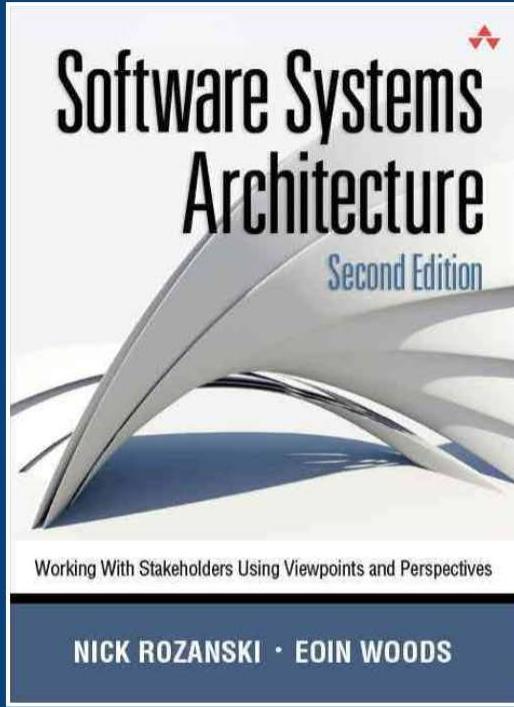


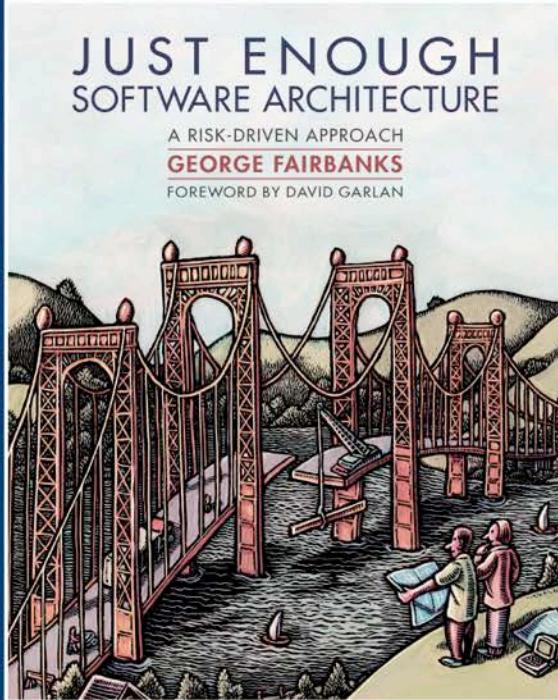
Figure 1 — The “4+1” view model



“Viewpoints and Perspectives”

Why is there a separation
between the **logical** and
development views?

Our architecture diagrams
don't match the code.



Model-code gap. Your architecture models and your source code will not show the same things. The difference between them is the *model-code gap*. Your architecture models include some abstract concepts, like components, that your programming language does not, but could. Beyond that, architecture models include intensional elements, like design decisions and constraints, that cannot be expressed in procedural source code at all.

Consequently, the relationship between the architecture model and source code is complicated. It is mostly a refinement relationship, where the extensional elements in the architecture model are refined into extensional elements in source code. This is shown in Figure 10.3. However, intensional elements are not refined into corresponding elements in source code.

Upon learning about the model-code gap, your first instinct may be to avoid it. But reflecting on the origins of the gap gives little hope of a general solution in the short term: architecture models help you reason about complexity and scale because they are abstract and intensional; source code executes on machines because it is concrete and extensional.

“model-code gap”

Software Reflexion Models:
Bridging the Gap between Source and High-Level Models*

Gail C. Murphy and David Notkin

Dept. of Computer Science & Engineering
University of Washington
Box 352350
Seattle WA, USA 98195-2350
{gmurphy, notkin}@cs.washington.edu

Kevin Sullivan

Dept. of Computer Science
University of Virginia
Charlottesville VA, USA 22903
sullivan@cs.virginia.edu

Abstract

Software engineers often use high-level models (for instance, box and arrow sketches) to reason and communicate about an existing software system. One problem with high-level models is that they are almost always inaccurate with respect to the system's source code. We have developed an approach that helps an engineer use a high-level model of the structure of an existing software system as a lens through which to see a model of that system's source code. In particular, an engineer defines a high-level model and specifies how the model maps to the source. A tool then computes a software reflexion model that shows where the engineer's high-level model agrees with and where it differs from a model of the source.

The paper provides a formal characterization of reflexion models, discusses practical aspects of the approach, and relates experiences of applying the approach and tools to a number of different systems. The illustrative example used in the paper describes the application of reflexion models to NetBSD, an implementation of Unix comprised of 250,000 lines of C code. In only a few hours, an engineer computed several reflexion models that provided him with a useful, global overview of the structure of the NetBSD virtual memory subsystem. The approach has also been applied to aid in the understanding and experimental reengineering of the Microsoft Excel spreadsheet product.

*This research was funded in part by the NSF grant CCR-8858804 and a Canadian NSERC post-graduate scholarship.

¹Permitting to make digital/hard copies of all or part of this material without fee is granted provided that the copies are not made or distributed for profit or commercial advantage, the copyright/Server notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the Association for Computing Machinery, Inc. (ACM). To copy otherwise, or to republish, requires prior specific permission and/or written permission to redistribute to lists, requires prior specific permission and/or written permission.

SIGSOFT '95 Washington, D.C., USA
©1995 ACM 0-89791-716-2/95/0010-13\$0.50

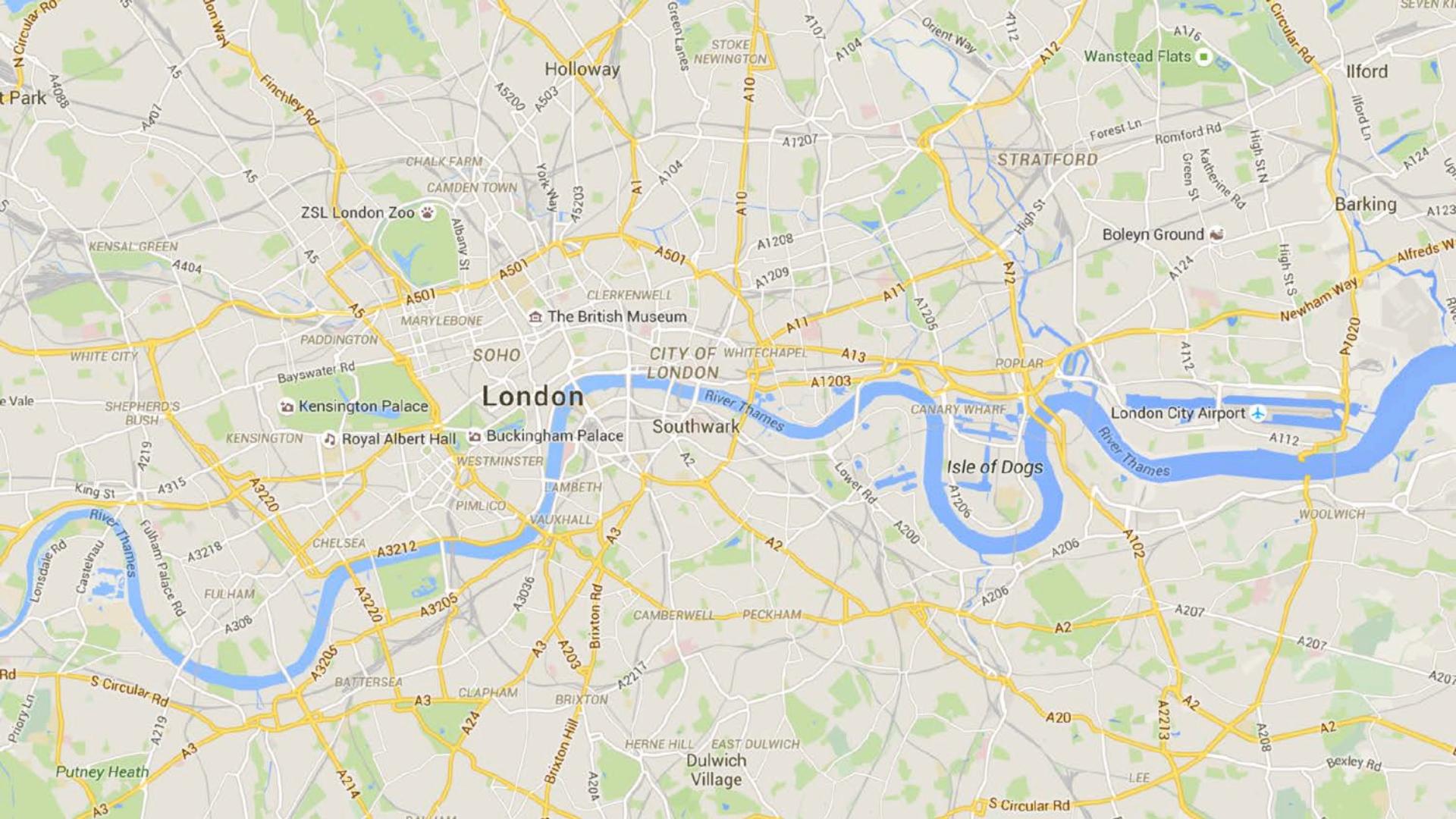
1 Introduction

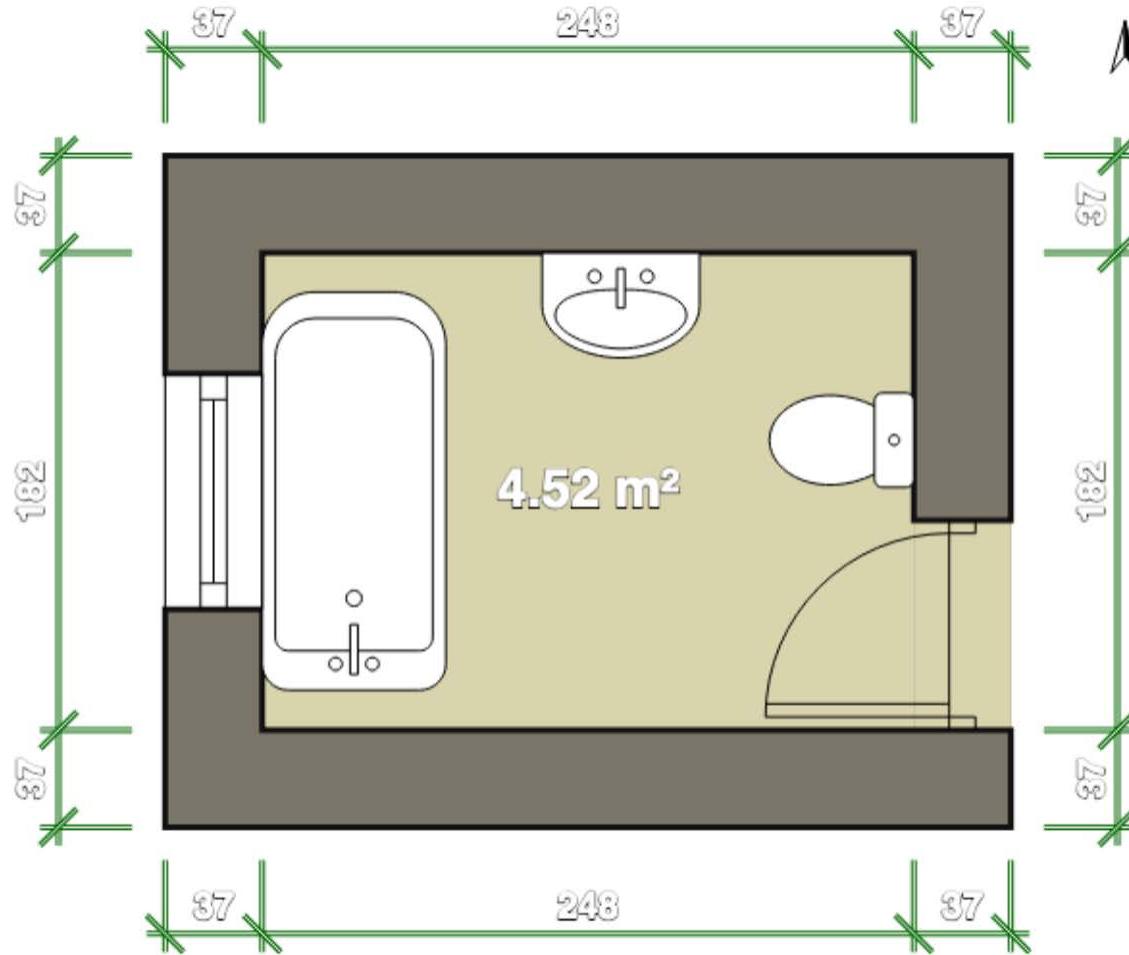
Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers' whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because the models are almost always inaccurate with respect to the system's source.

Current reverse engineering systems derive high-level models from the source code. These derived models are useful because they are, by their very nature, accurate representations of the source. Although accurate, the models created by these reverse engineering systems may differ from the models sketched by engineers; an example of this is reported by Wong et al. [WTMS95]. We have developed an approach, illustrated in Figure 1, that enables an engineer to produce sufficiently accurate high-level models in a different way. The engineer defines a high-level model of interest, extracts a source model (such as a call graph or an inheritance hierarchy) from the source code, and defines a declarative mapping between the two models. A *software reflexion model* is then computed to determine where the engineer's high-level model does and does not agree with the source model.¹ An engineer interprets the reflexion model and, as necessary, modifies the input to iteratively compute additional reflexion models.

¹The old English spelling differentiates our use of "reflexion" from the field of reflective computing [Smi84].

We lack a **common vocabulary**
to describe software architecture





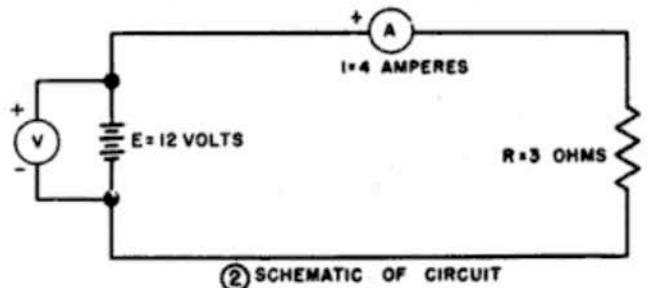
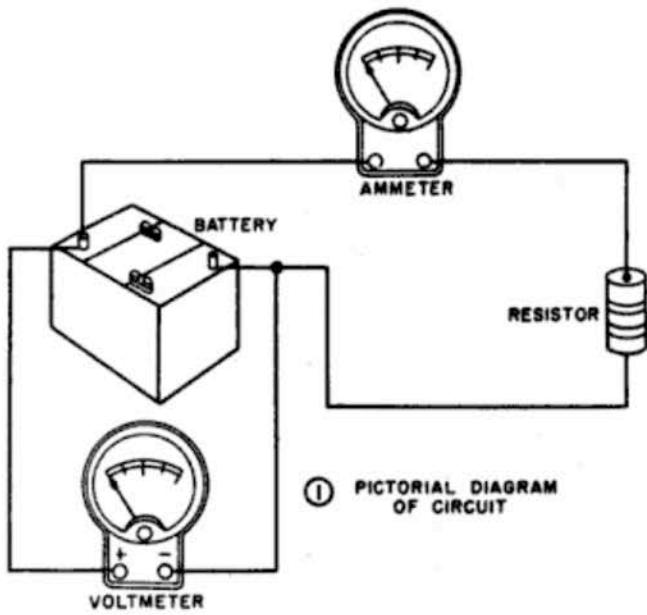
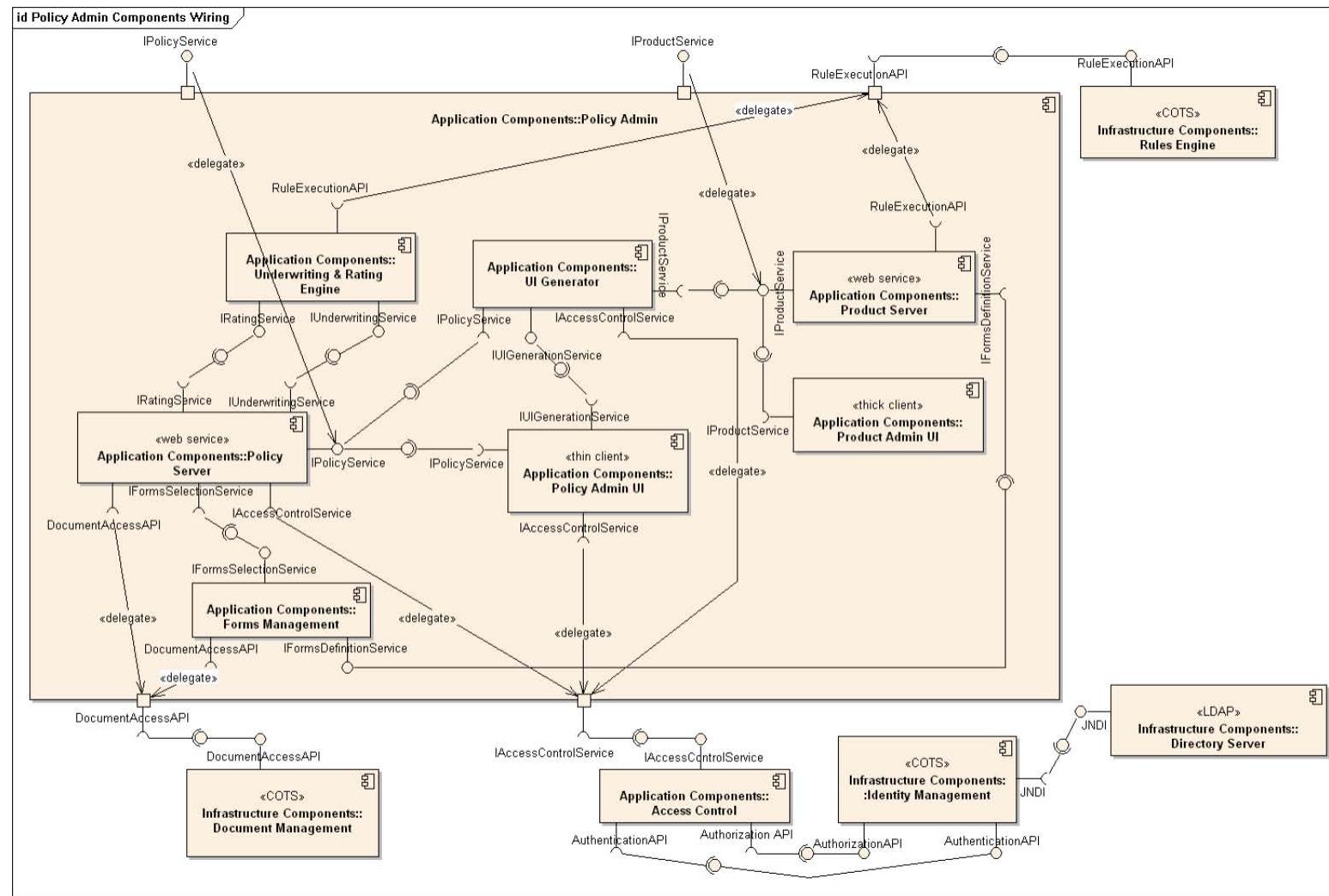


Figure 48. Diagram of a basic circuit.



Software System

Web
Application

Logging
Component



Relational
Database

¹component

noun | com·po·nent | \kəm-'pō-nənt, 'käm-, käm-'\

Simple Definition of COMPONENT

Popularity: Top 30% of words

: one of the parts of something (such as a system or mixture) : an important piece of something

Source: Merriam-Webster's Learner's Dictionary

Ubiquitous
language

Abstractions

Would you code it that way?

(ensure that your diagrams reflect
your implementation intent)

When drawing software
architecture diagrams,
think like a software developer

A **common set of abstractions**
is more important
than a common notation

Software System

A **software system** is made up of one or more **containers**,
each of which contains one or more **components**,
which in turn are implemented by one or more **code elements**.

Software System

Container

(e.g. client-side web app, server-side web app, console application, mobile app, microservice, database schema, file system, etc)

Container

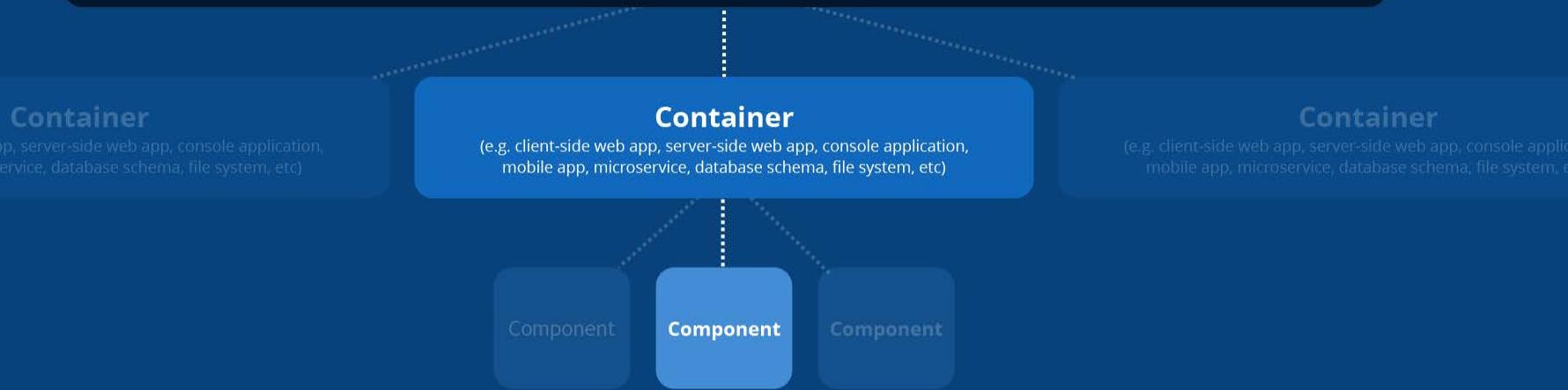
(e.g. client-side web app, server-side web app, console application, mobile app, microservice, database schema, file system, etc)

Container

(e.g. client-side web app, server-side web app, console application, mobile app, microservice, database schema, file system, etc)

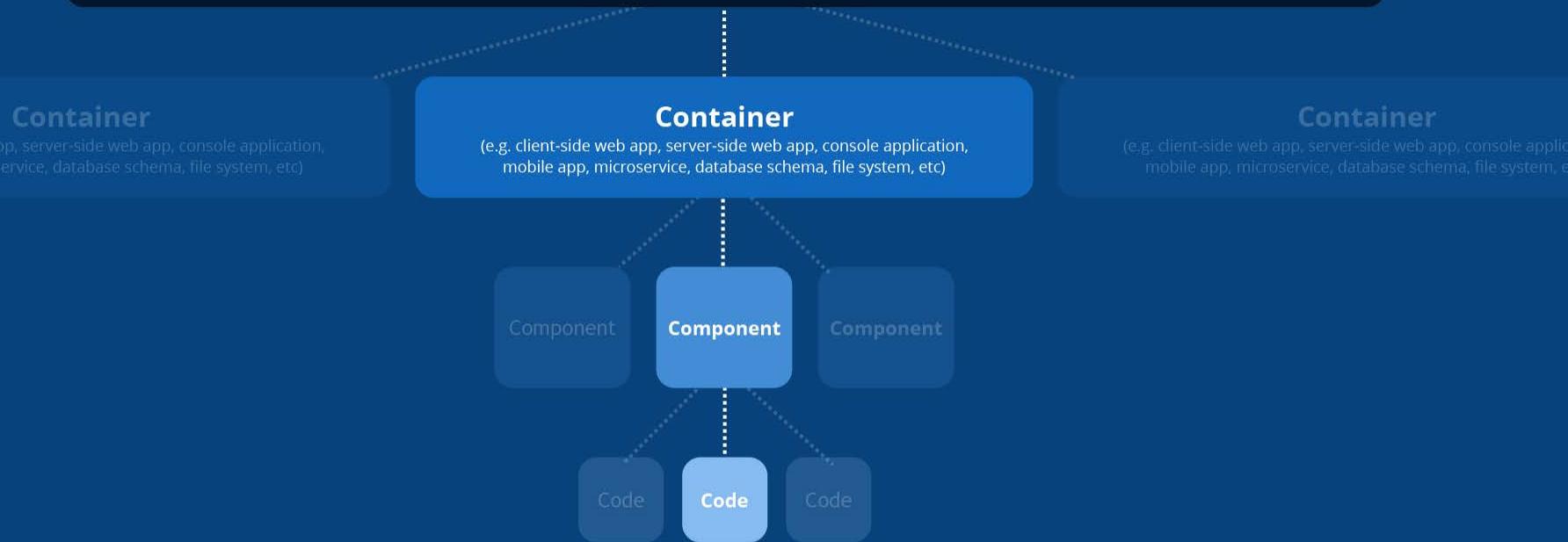
A **software system** is made up of one or more **containers**,
each of which contains one or more **components**,
which in turn are implemented by one or more **code elements**.

Software System



A **software system** is made up of one or more **containers**,
each of which contains one or more **components**,
which in turn are implemented by one or more **code elements**.

Software System



A **software system** is made up of one or more **containers**,
each of which contains one or more **components**,
which in turn are implemented by one or more **code elements**.

C4

c4model.com

1. System Context

The system plus users and system dependencies.

Overview first

2. Containers

The overall shape of the architecture and technology choices.

Zoom & filter

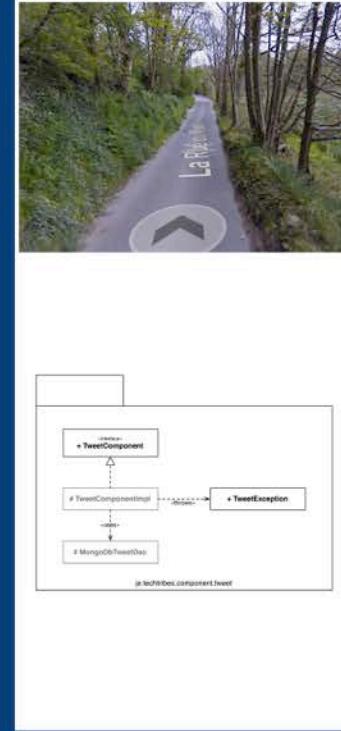
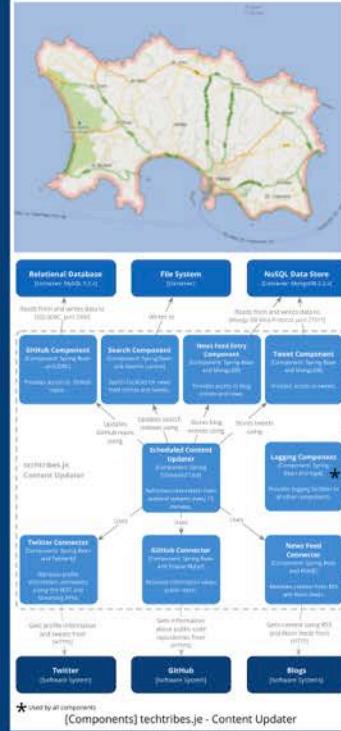
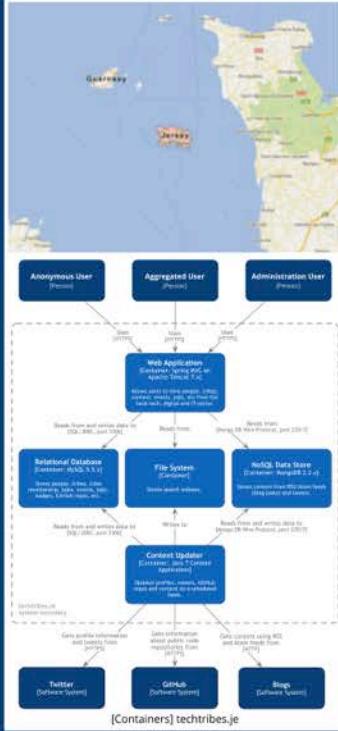
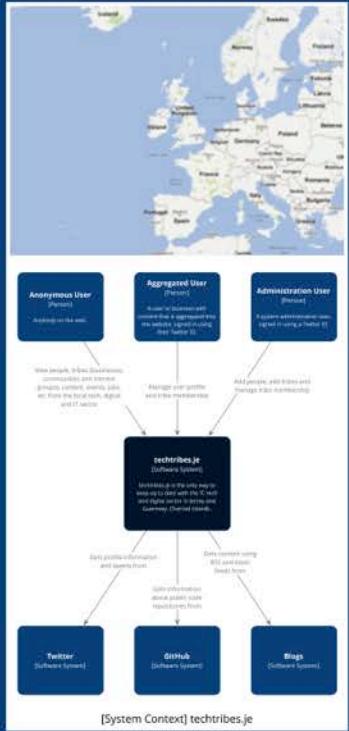
3. Components

Logical components and their interactions within a container.

Details on demand

4. Code (e.g. classes)

Component implementation details.



Diagrams are maps

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

Example

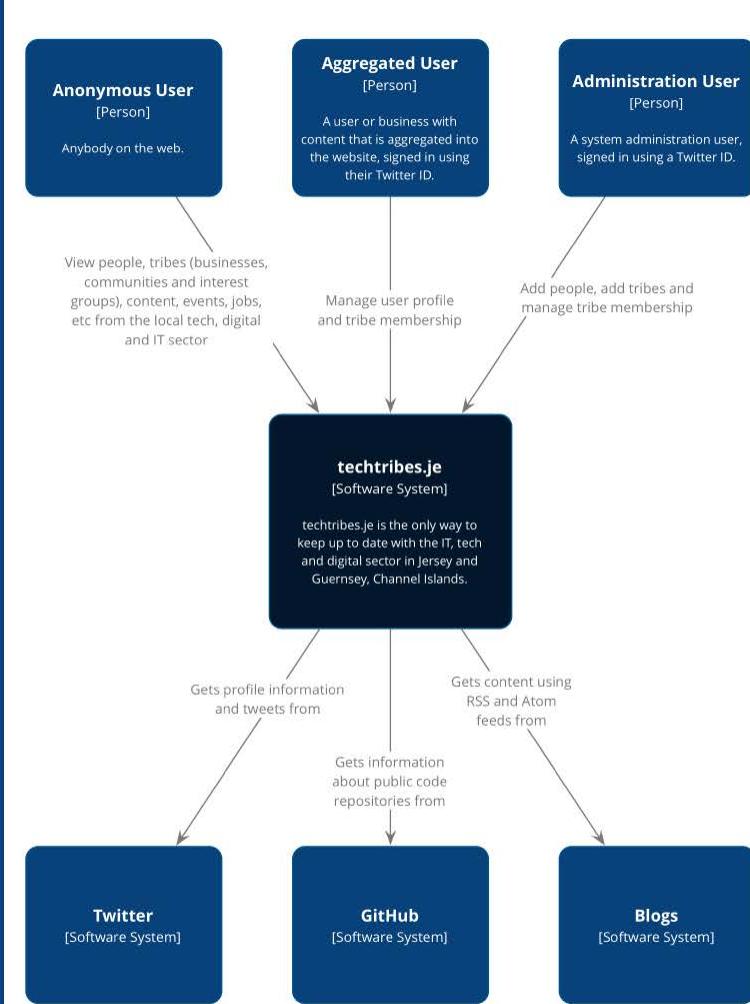
(techtribes.je)

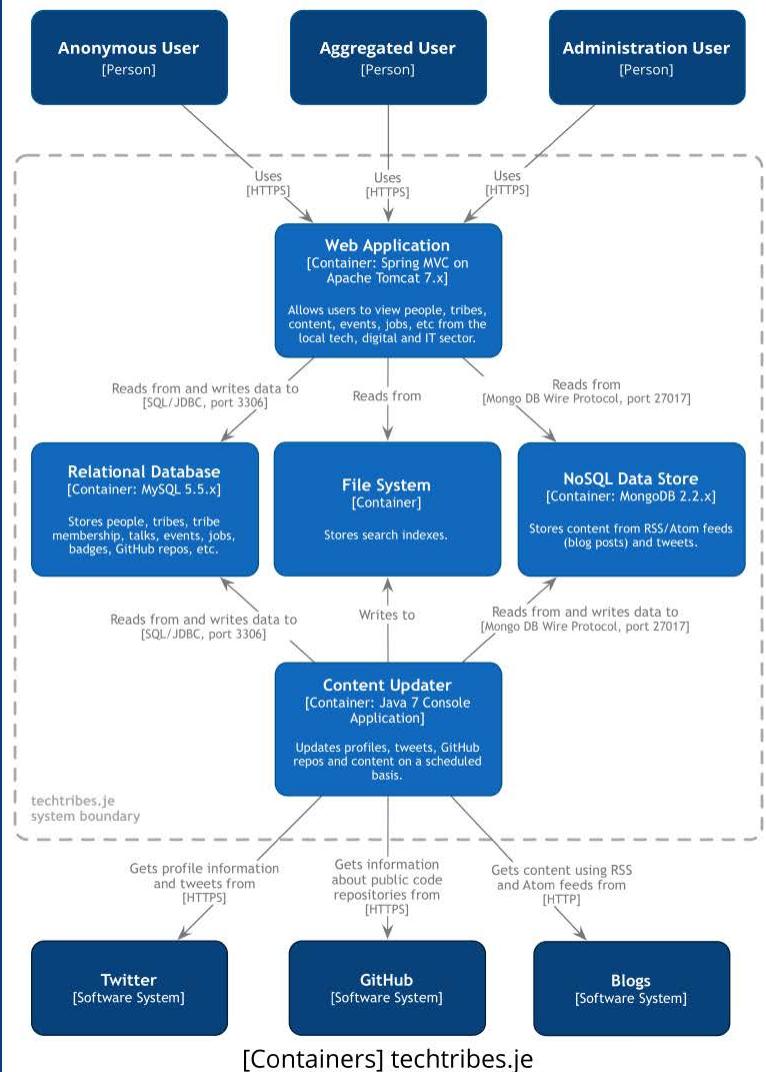
The screenshot shows the homepage of techtribes.je, a content aggregator for the local tech and digital industry. The layout is organized into several sections:

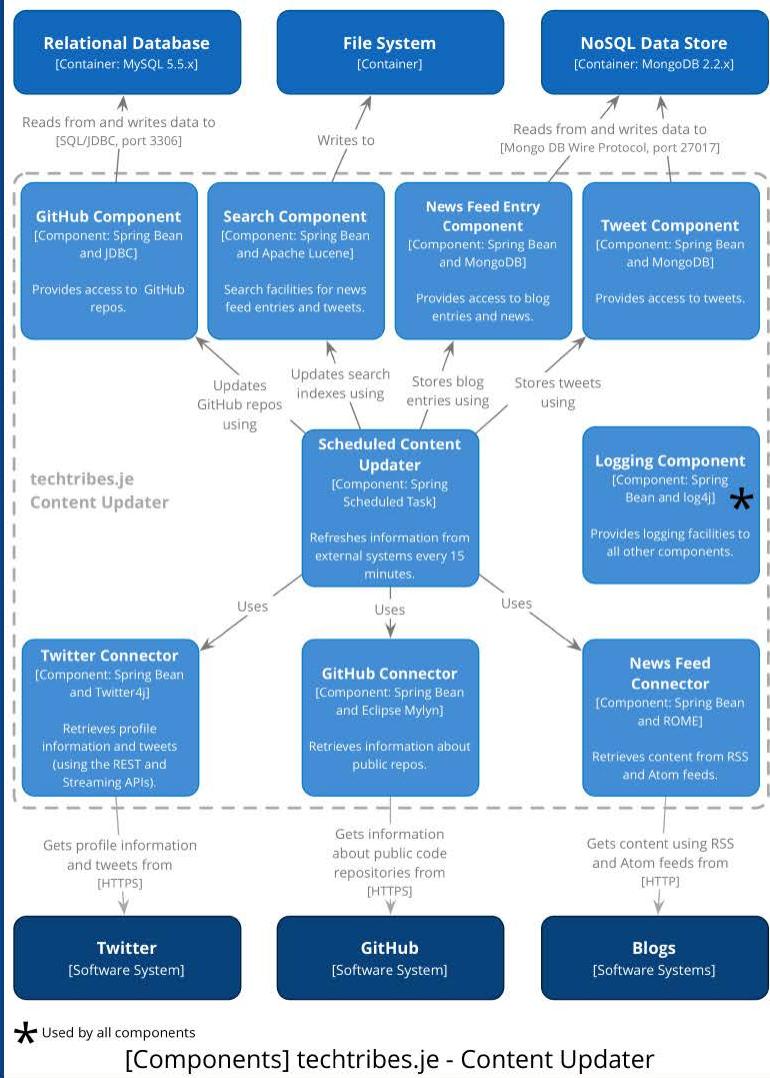
- News:** Headlines include "B C Alliance plans Microsoft events in Channel Islands" and "B Jersey residents set to have choice in fibre broadband".
- Local events:** Listings for "Tech Week Talk", "The Internet of Everything and digital Jersey", and "Agile software architecture sketches and Matrix".
- Talks by local speakers:** Presentations by "Ed Trellich", "B C Alliance", "Software architecture and the balance with agility", and "Book on the Future - it's déjà vu".
- Blog posts, etc.:** Posts like "Increase Your Productivity Power - Pin Using Visually", "Book on the Future - it's déjà vu Call Previews", and "B Jersey residents set to have choice in fibre broadband".
- Tweets:** A section featuring various tweets from users.

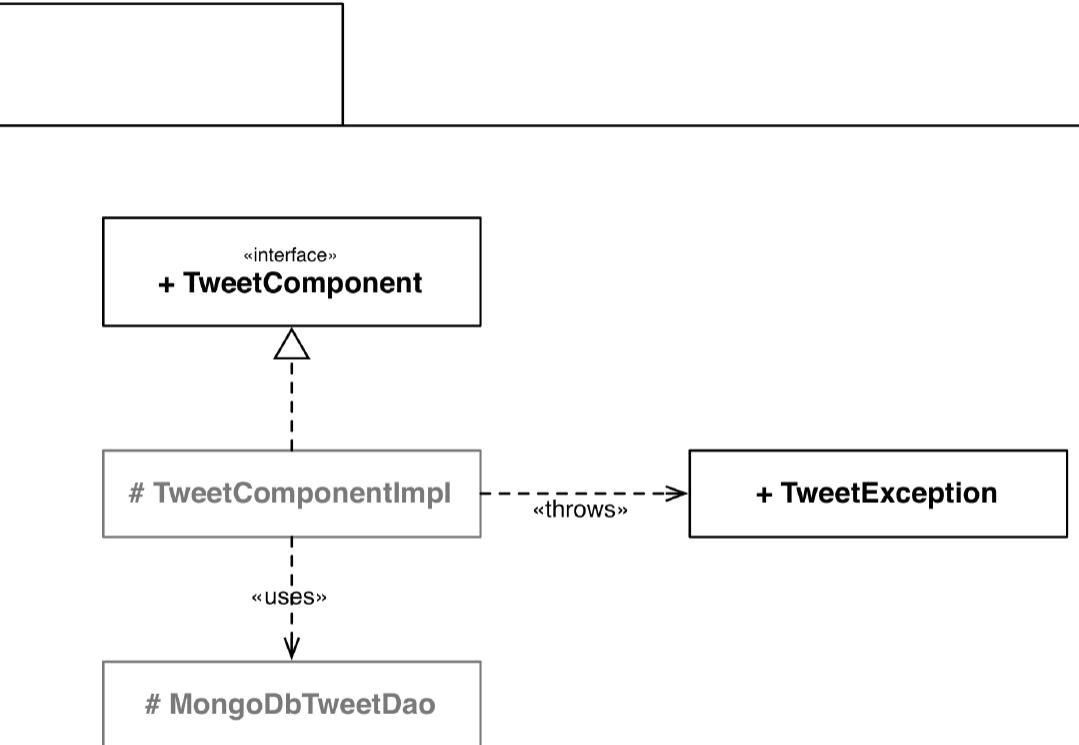
techtribes.je

A simple content aggregator for
the local tech and digital industry









je.techtribes.component.tweet

Notation

Elements

Start with simple boxes containing the element name, type, technology (if appropriate) and a description/responsibilities

Anonymous User

[Person]

Anybody on the web.

techtribes.je

[Software System]

techtribes.je is the only way to keep up to date with the IT, tech and digital sector in Jersey and Guernsey, Channel Islands.

Web Application

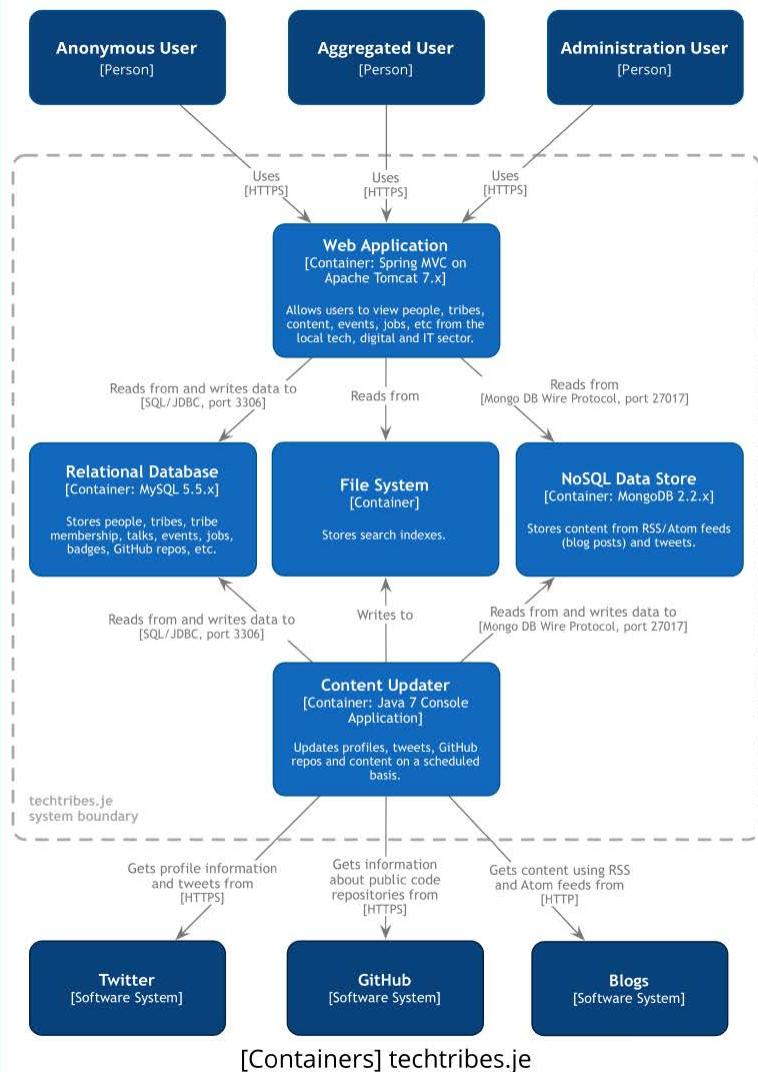
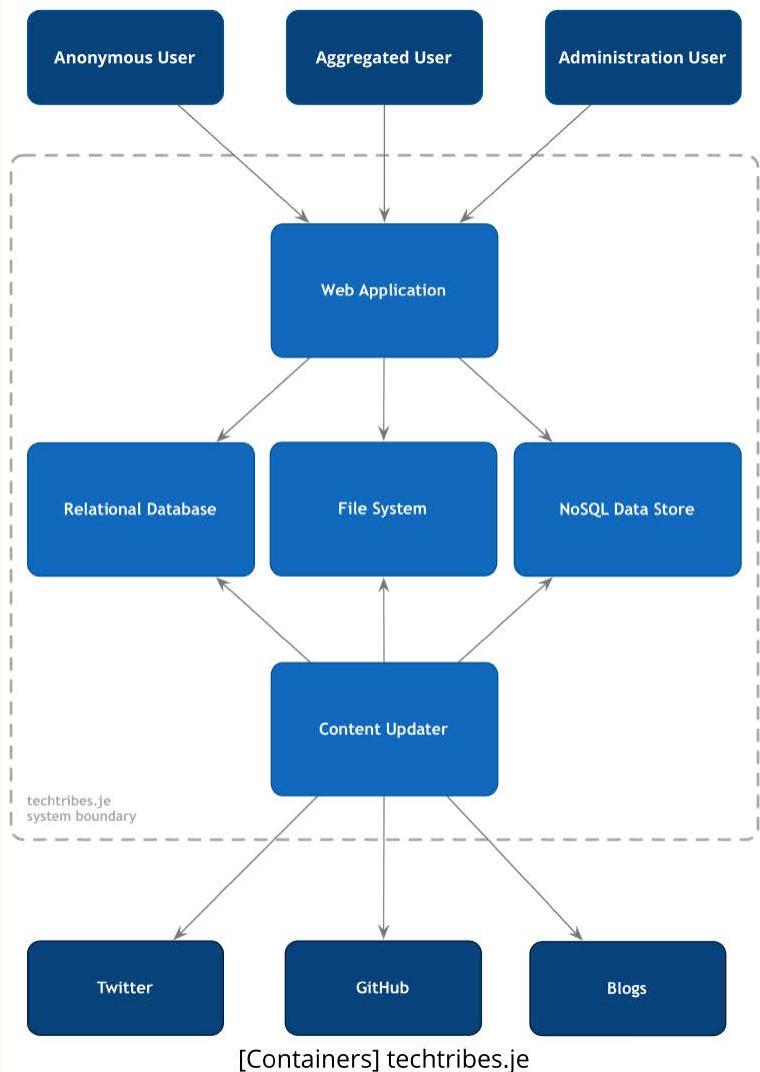
[Container: Java + Spring MVC]

Allows users to view people, tribes, content, events, jobs, etc from the local tech, digital and IT sector.

Twitter Connector

[Component: Spring Bean + Twitter4j]

Retrieves profile information and tweets (using the REST and Streaming APIs).



Lines

Favour uni-directional lines showing the most important dependencies or data flow, with an annotation to be explicit about the purpose of the line and direction



Dependency vs data flow

Key/legend

Explain shapes, line styles, colours, borders, acronyms, etc
... even if your notation seems obvious!

Notation, notation, notation

A software architecture diagram review checklist

General

Does the diagram have a title?	Yes	No
Do you understand what the diagram type is?	Yes	No
Do you understand what the diagram scope is?	Yes	No
Does the diagram have a key/legend?	Yes	No

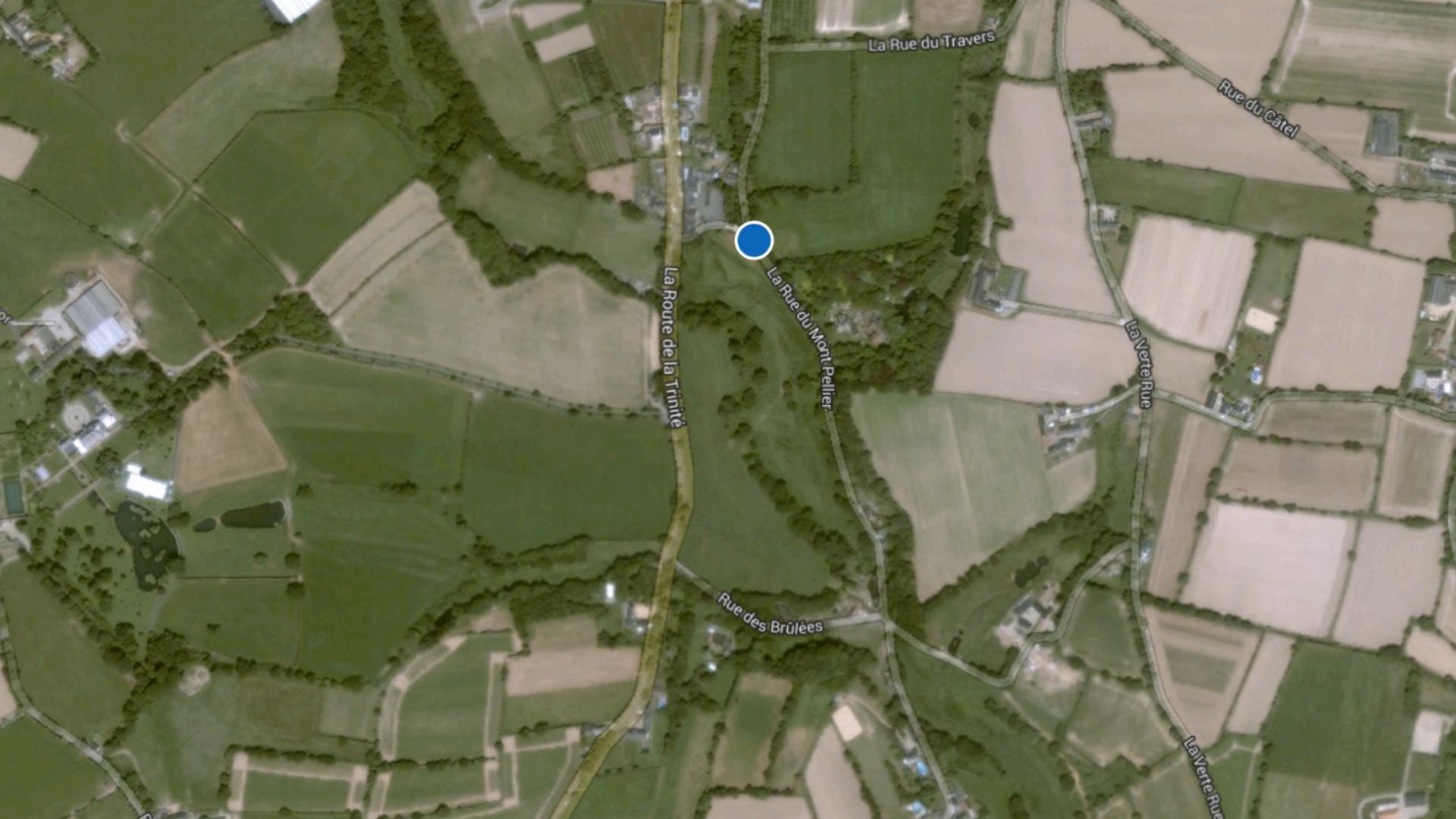
Elements

Does every element have a name?	Yes	No
Do you understand the type of every element? (i.e. the level of abstraction; e.g. software system, container, etc)	Yes	No
Do you understand what every element does?	Yes	No
Where applicable, do you understand the technology choices associated with every element?	Yes	No
Do you understand the meaning of all acronyms and abbreviations used?	Yes	No
Do you understand the meaning of all colours used?	Yes	No

Documenting software architecture

La Rue du Mont





La Rue du Mont Pelier

La Rue du Travers

Rue du Câtel

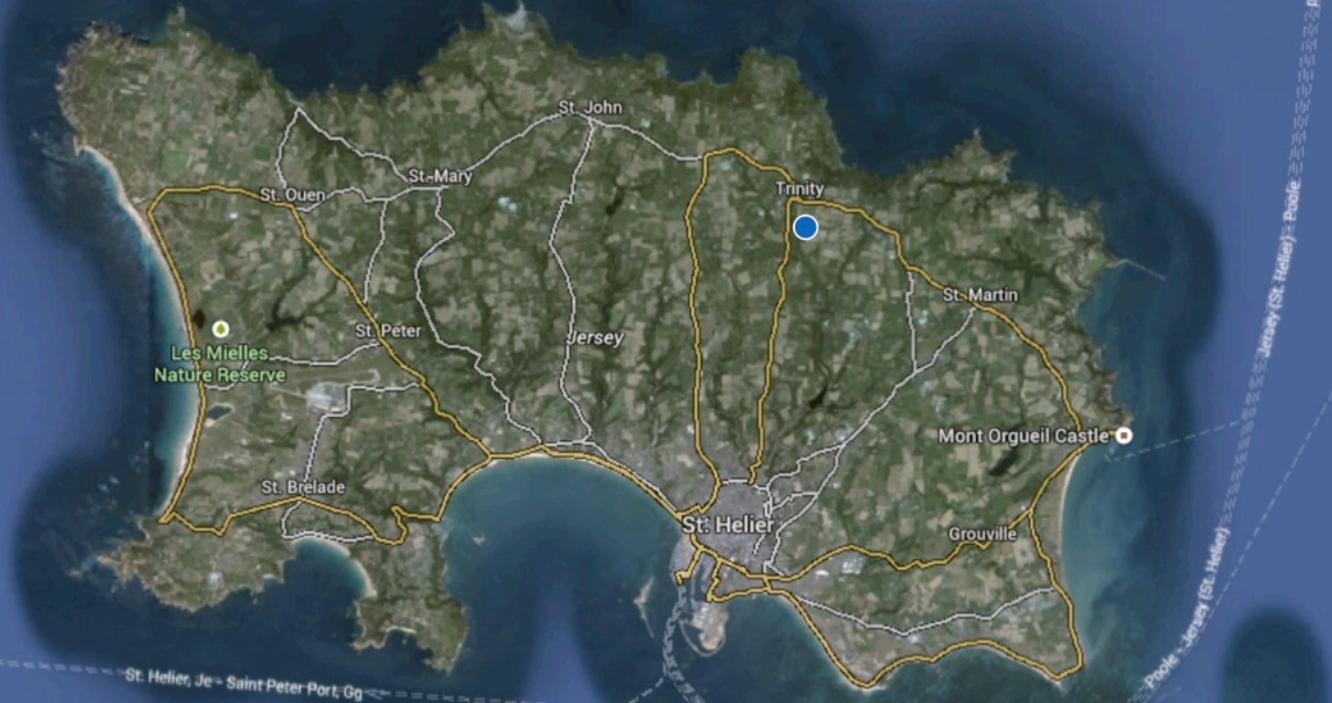
La Verte Rue

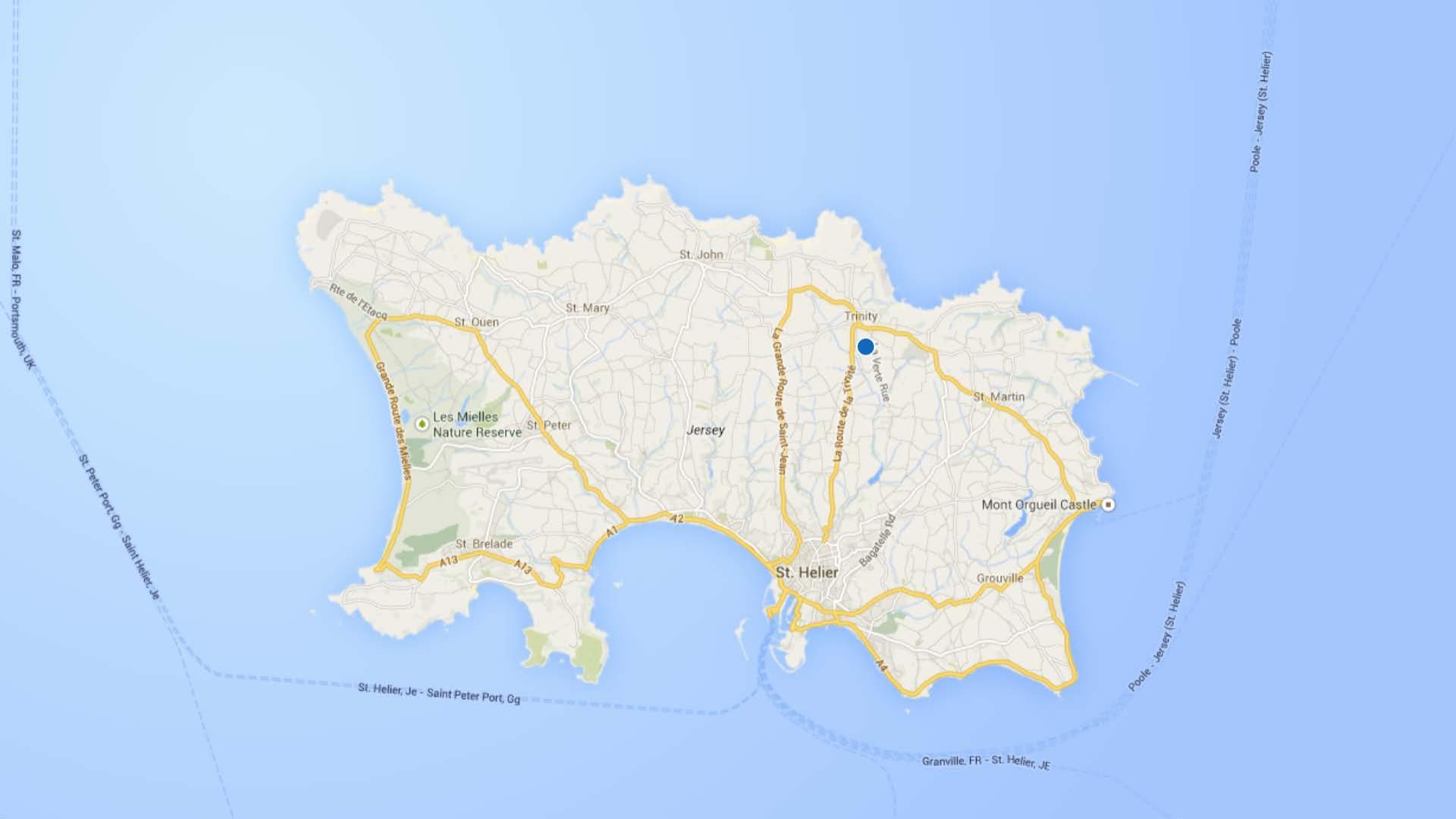
Rue des Brûlées

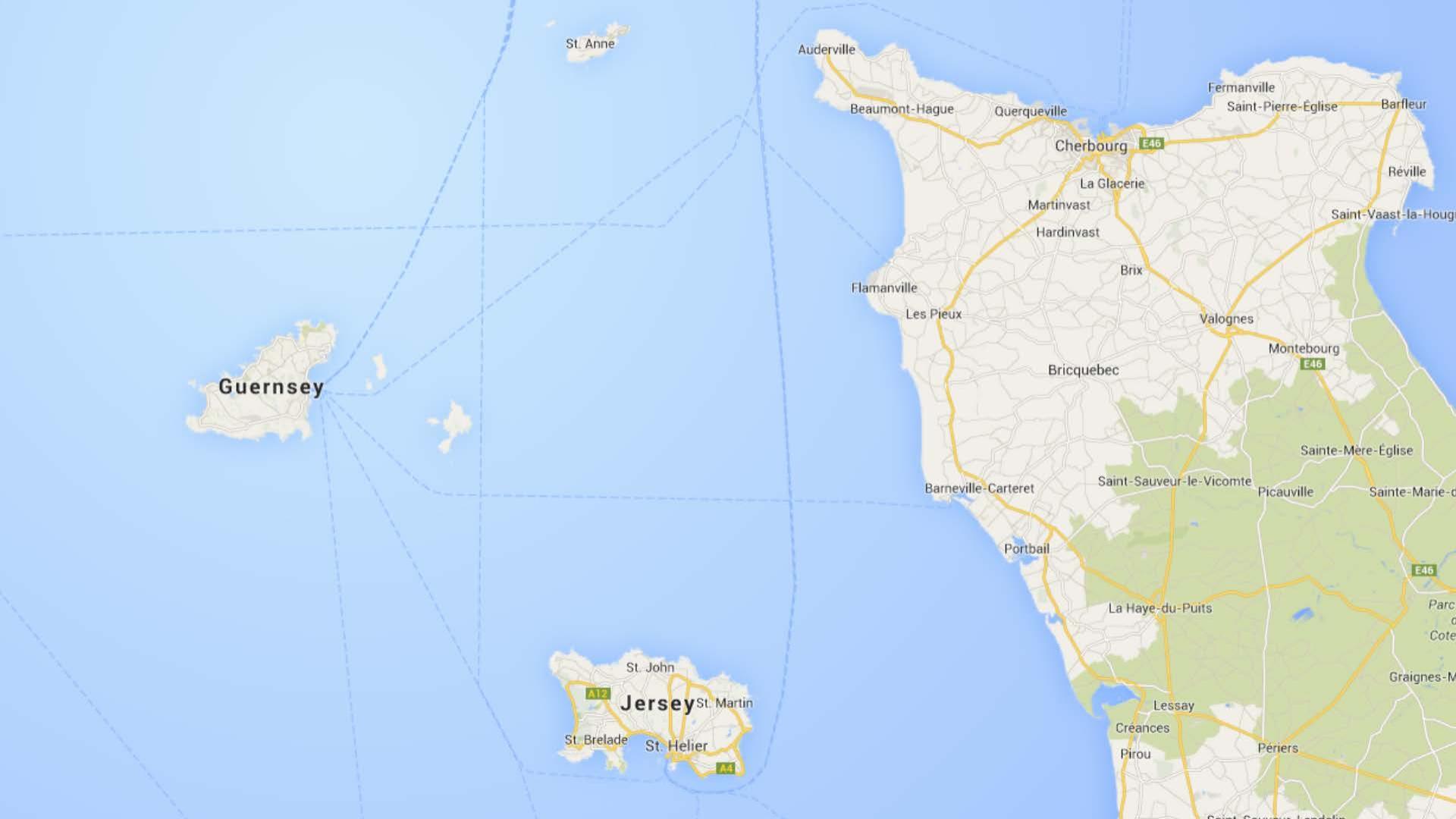
La Verte Rue

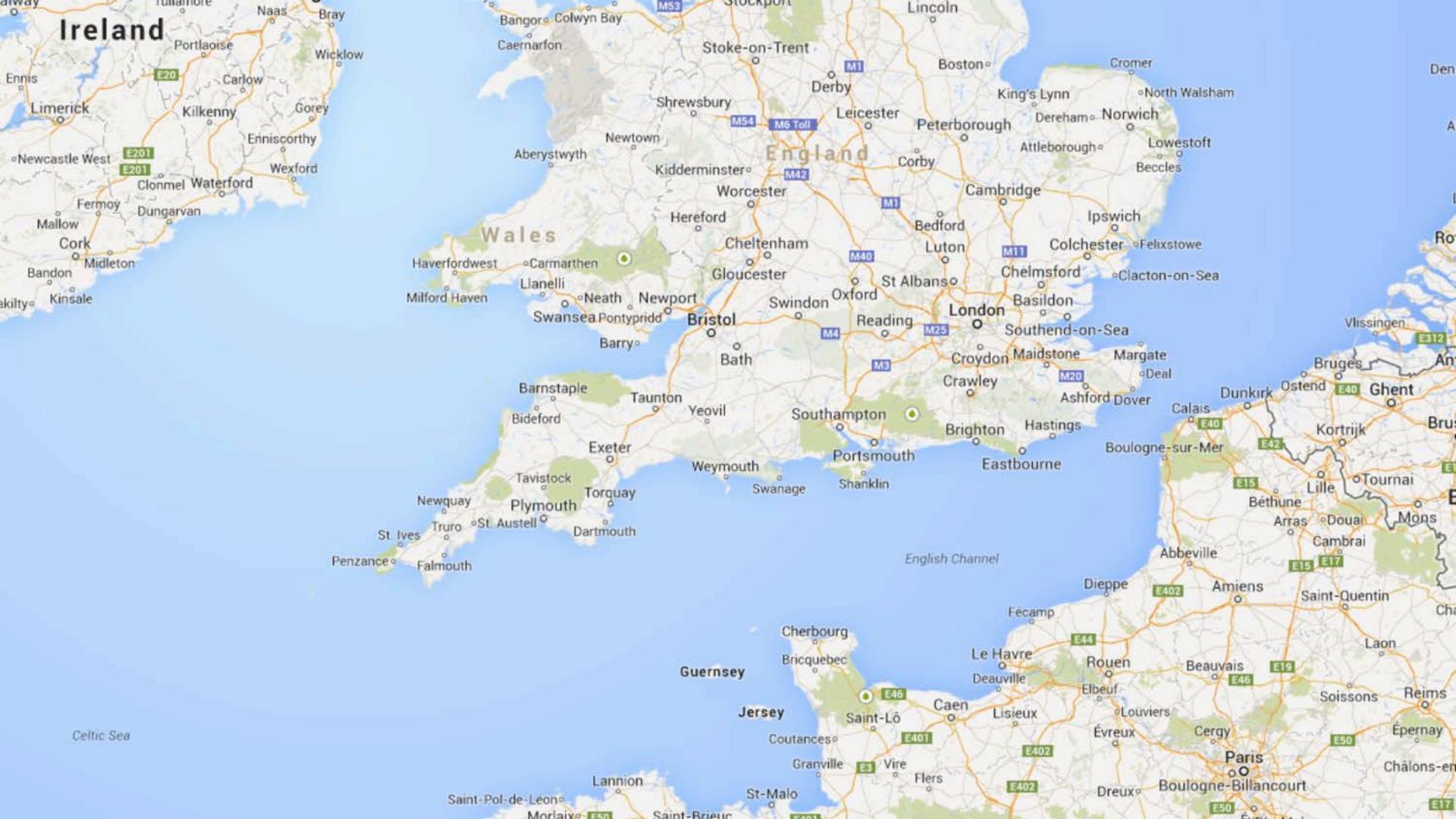
or

or

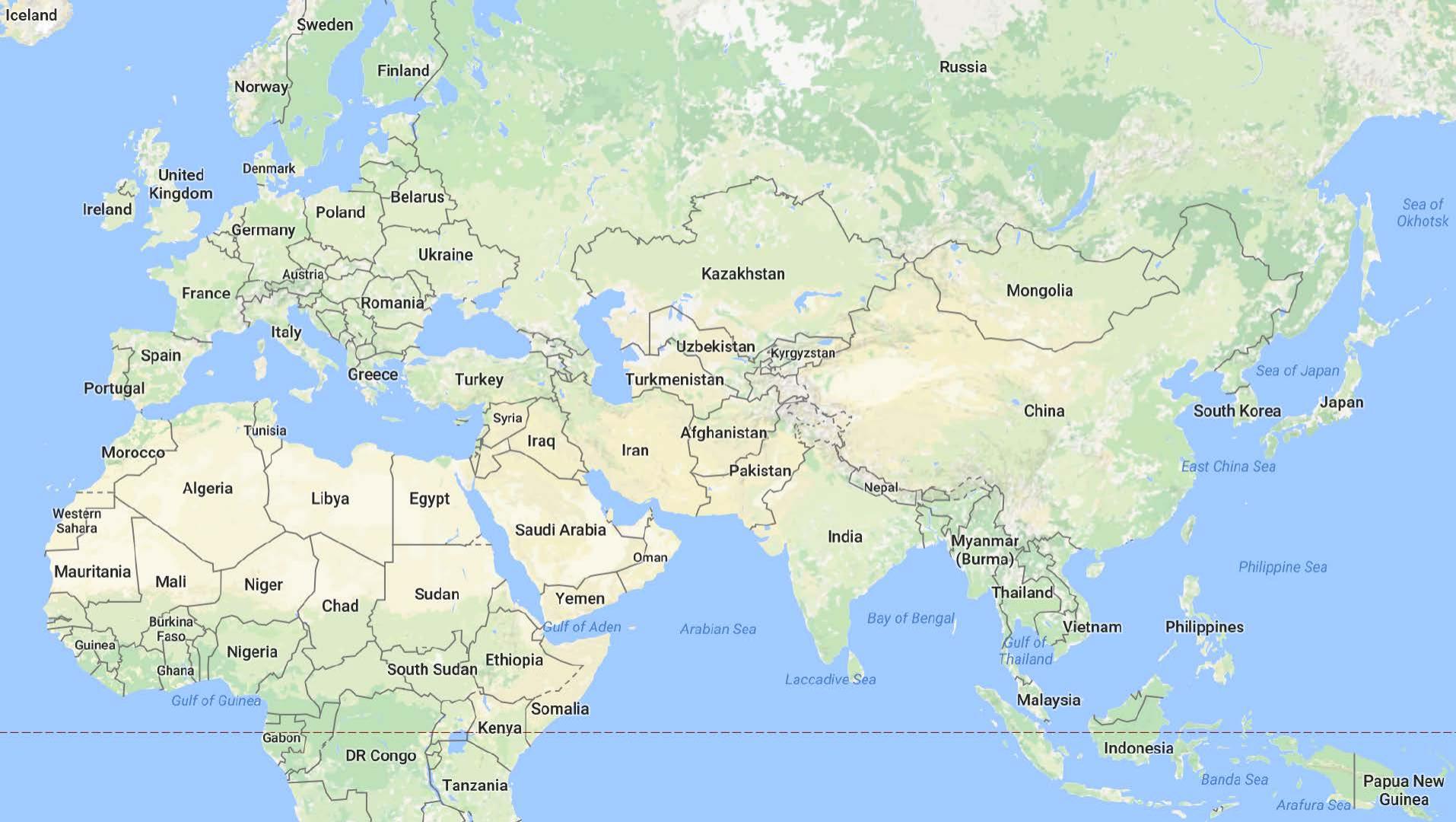


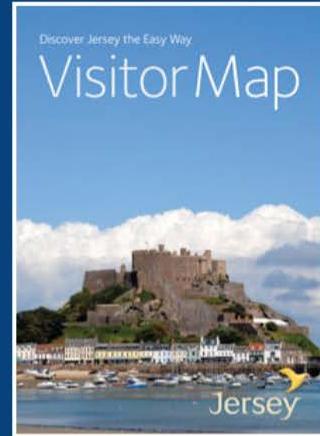






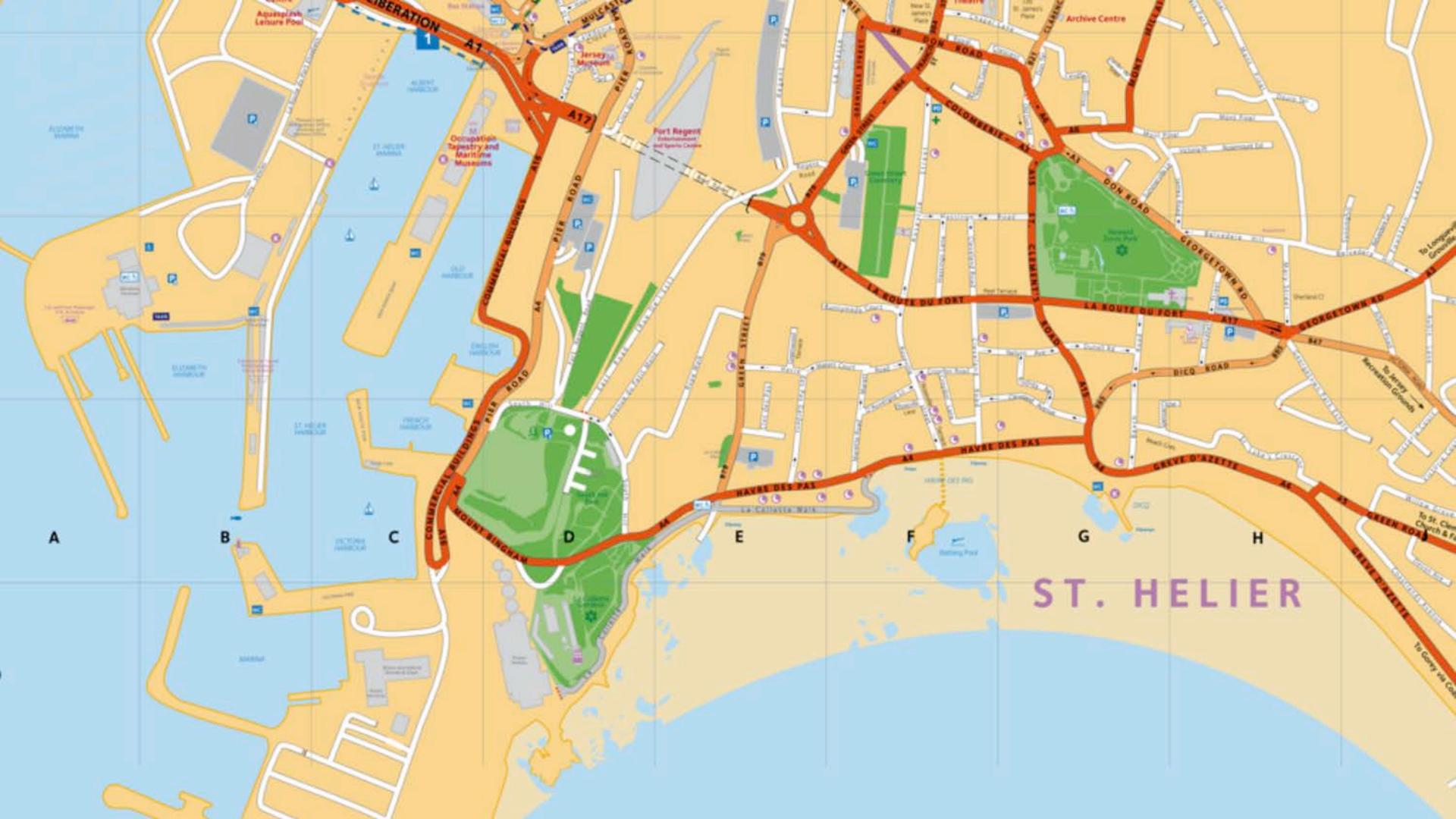


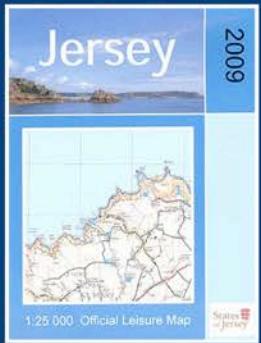




Enough detail to
start **exploring**







Very detailed and precise (terrain, buildings, etc)

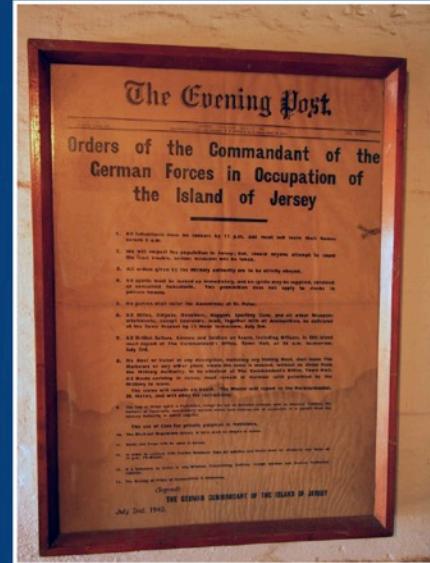
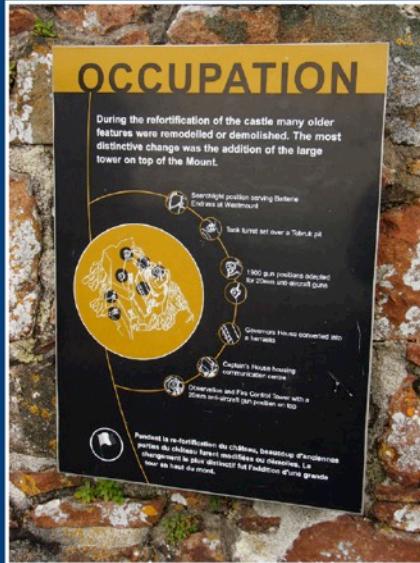
Points of
interest

Elizabeth Castle





Granite and concrete?!



History





Travel Guidebook

(maps, points of interest, sights, itineraries,
history, culture, practical information, etc)

Working software over comprehensive documentation

Manifesto for Agile Software Development

The code doesn't tell
the whole story

Software Architecture Document

Useful information
spread across
hundreds of pages;
rarely read or updated



Software Guidebook

(maps, points of interest, sights, itineraries,
history, culture, practical information, etc)

The scope is a single
software system

Describe what you
can't get from the code

Documentation should
be **constantly evolving**

Context

A system context diagram, plus some narrative text to "set the scene".

Functional Overview

An overview of the software system; perhaps including wireframes, UI mockups, screenshots, workflow diagrams, business process diagrams, etc.

Quality Attributes

A list of the quality attributes (non-functional requirements; e.g. performance, scalability, security, etc).

Constraints

A list of the environmental constraints (e.g. timescales, budget, technology, team size/skills, etc).

Principles

A list of the development and architecture principles (e.g. coding conventions, separation of concerns, patterns, etc).

Software Architecture

A description of the software architecture, including static structure (e.g. containers and components) and dynamic/runtime behaviour.

Code

A description of important or complicated component implementation details, patterns, frameworks, etc.

Data

Data models, entity relationship diagrams, security, data volumes, archiving strategies, backup strategies, etc.

Infrastructure Architecture

A description of the infrastructure available to run the software system.

Deployment

The mapping of software (e.g. containers) to infrastructure.

Development Environment

A description of how a new developer gets started.

Operation and Support

An overview of how the software system is operated, supported, monitored, etc.

Decision Log

A log of the major decisions made; e.g. as free format text or a collection of "Architecture Decision Records".

This is a **starting point**; add and remove sections as necessary.

Questions
and break

Software architecture in the delivery process

Structured Programming

(Software Craftsmanship v1.0)

Historically there's been
a tendency towards
big design up front

“Waterfall”

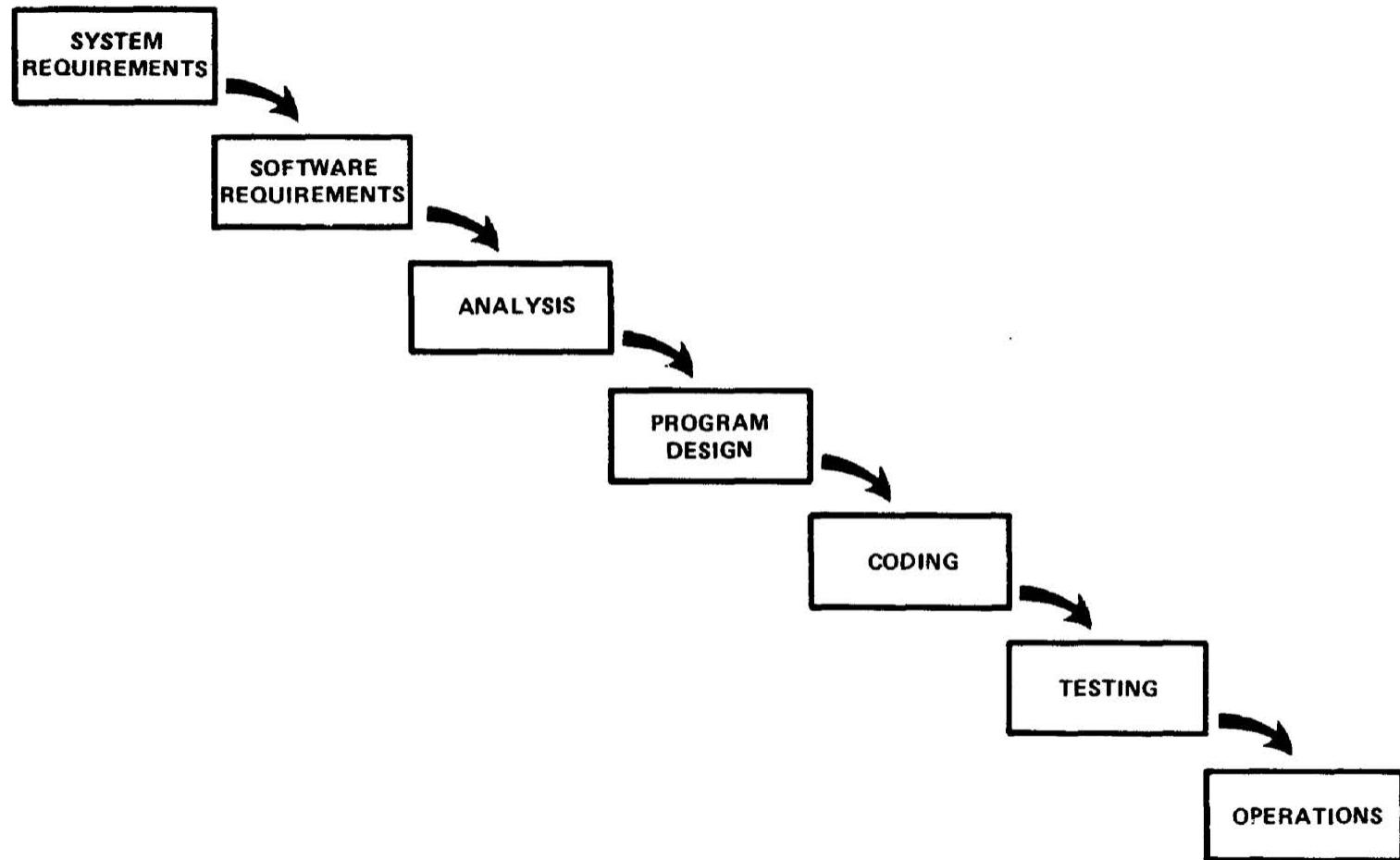


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.



time spent early in the software production cycle can reduce costs at later stages

Wikipedia

A “structured approach”
with an emphasis on
comprehensive documentation

SSADM

Structured Systems Analysis
and Design Method

a systems approach to the analysis and design of information systems

Wikipedia

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

Managing the development of large software systems

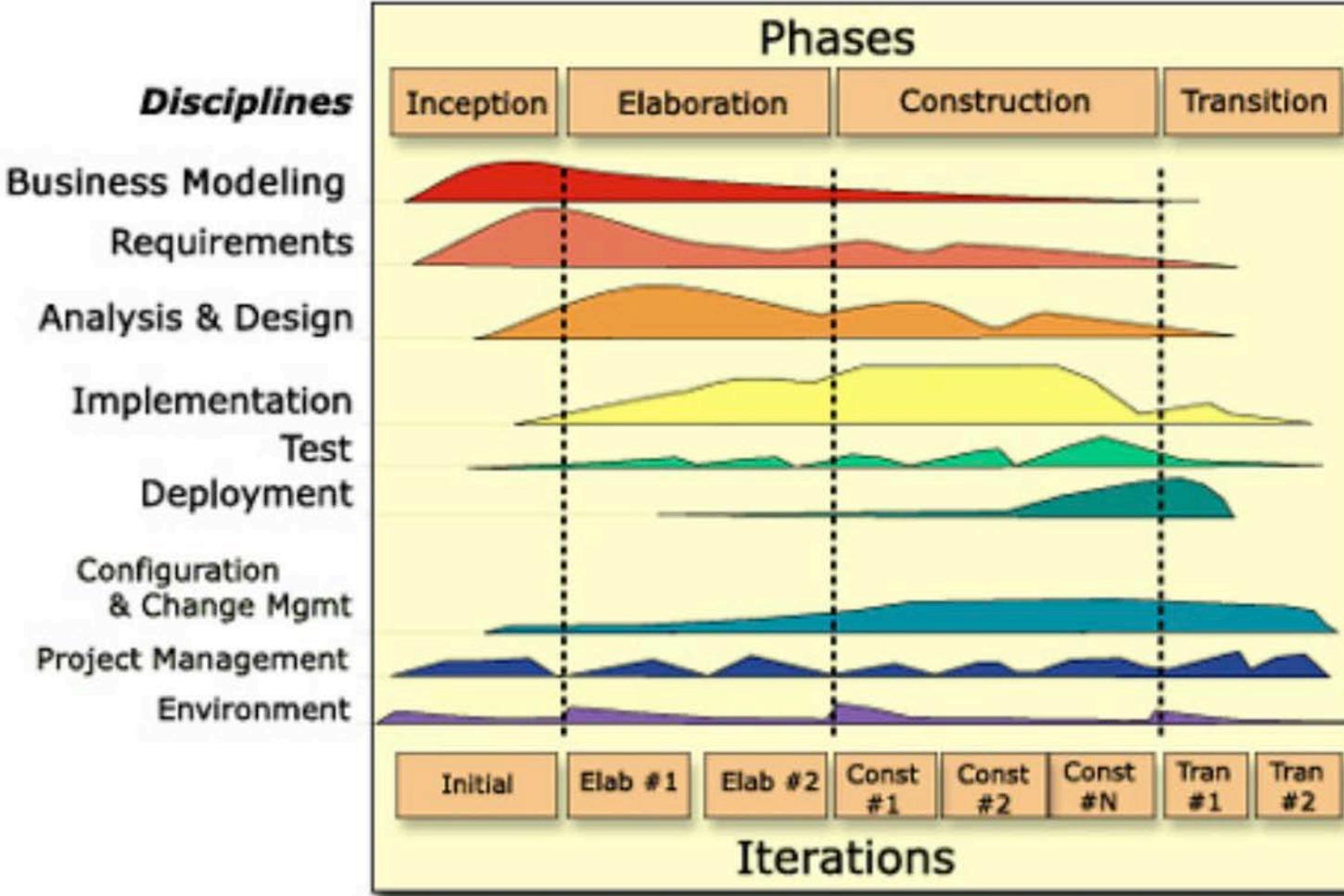
Dr Winston W. Royce

Long feedback loops
often resulted in the
wrong thing being delivered

Iterative and
incremental
development

RUP

Rational Unified Process



Roles and artifacts

Rapid Application Development ("prototype-driven design")

Agile

Manifesto for Agile Software Development

A photograph of a diverse group of people of various ages and ethnicities, all wearing dark-colored clothing. They are gathered closely together in a huddle, looking towards the center. The background is slightly blurred, creating a sense of depth and focus on the group.

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

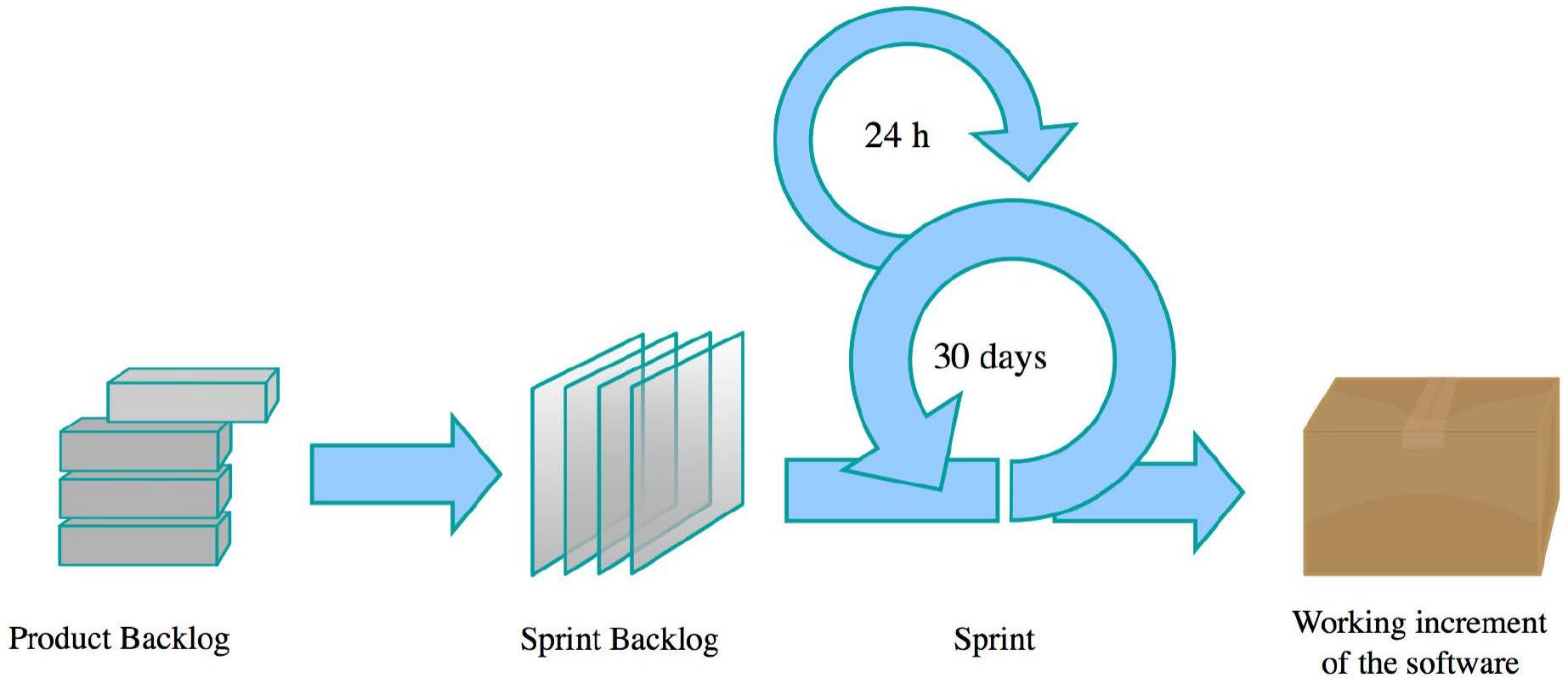
Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.





We do Scrum
(as described in the book)

Is there a conflict between
agile and architecture?

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

Principle 11 of the Manifesto for Agile Software Development

Architects? We don't need
no stinkin' architects!

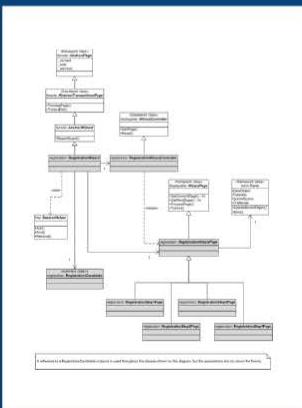
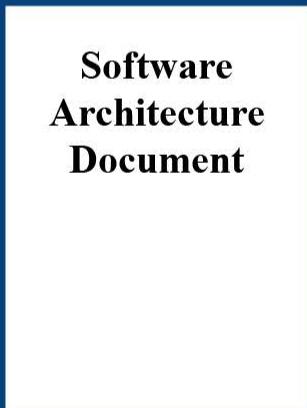


Small, self-organising teams
of generalising specialists,
often where everybody
is jointly responsible

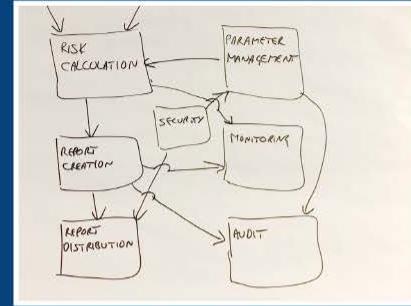


Responding to change
over
following a plan

Big design up front



VS



No design up front

Evolutionary architecture



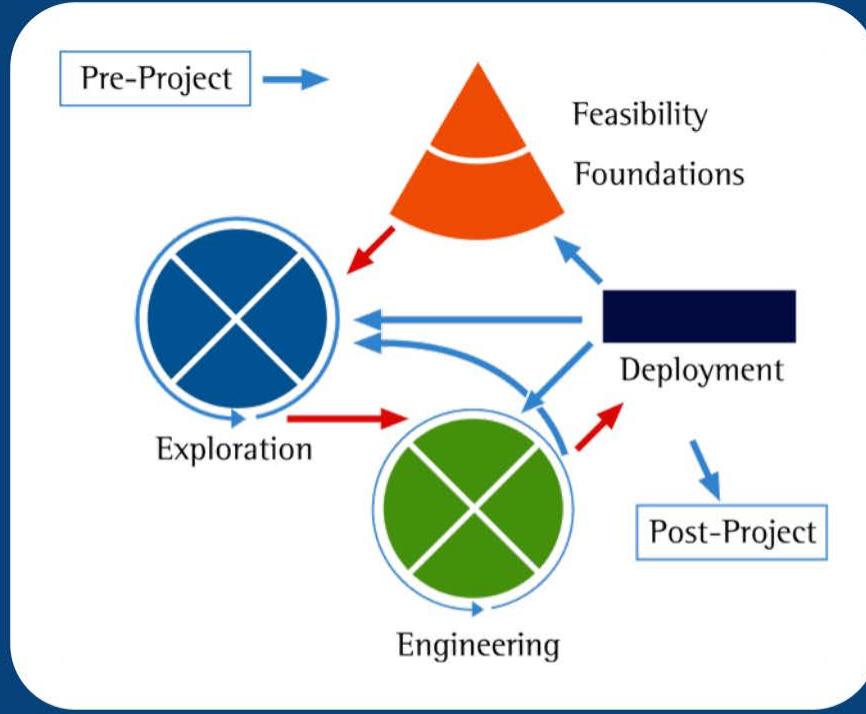
Are we allowed to do
up front design if we're agile?

Beyond Scrum

DSDM Atern

Disciplined Agile Delivery (DAD)

Scaled Agile Framework (SAFe)



“The Atern lifecycle process”

<https://www.dsdm.org/content/lifecycle>

The Foundations phase
is aimed at establishing
firm and enduring foundations
for the project.

It's not about creating a
perfect end-state or
complete architecture

You need a
starting point

Detailed blueprints

vs

setting a direction

Continuous
technical
leadership

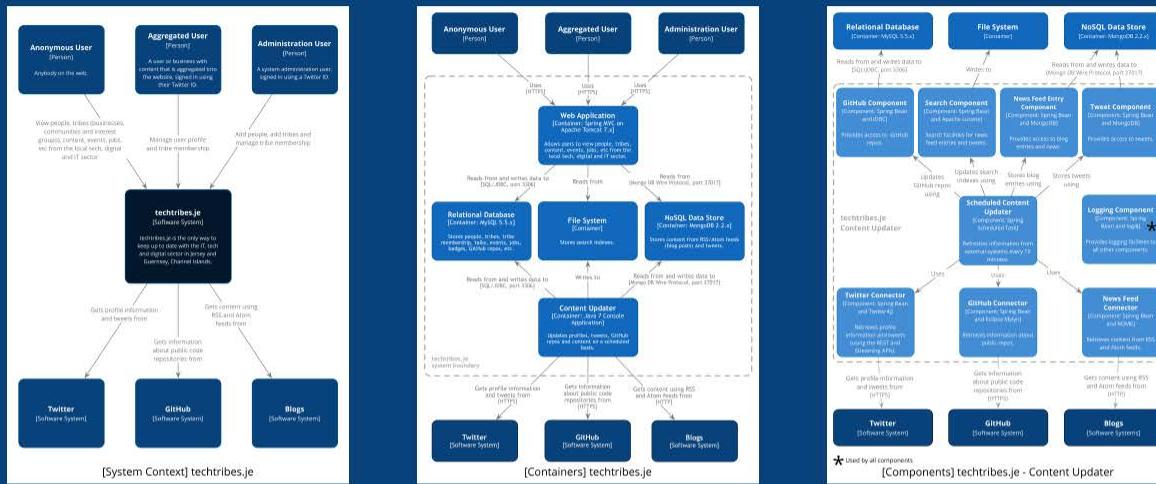
Add value,
eliminate waste

How much up front design should you do?



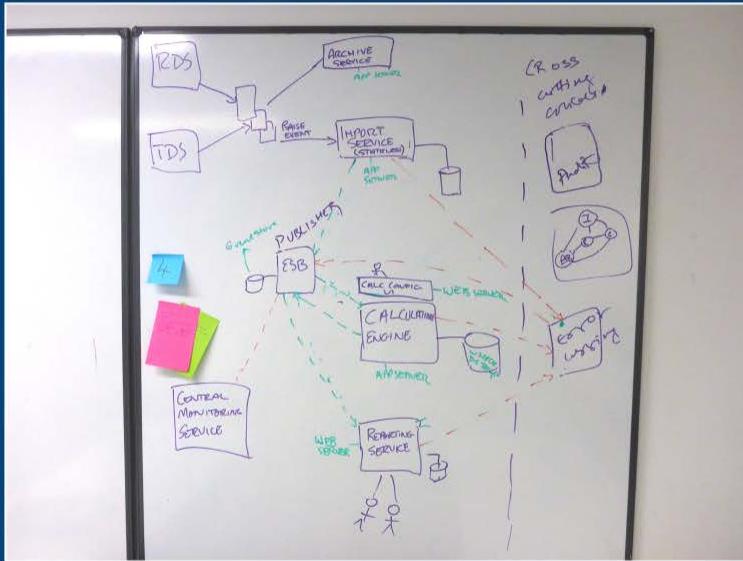
just enough

Not too much,
not too little!



Understand the structure and create a shared vision

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



2. Is it going to work?

Teams need to explicitly
manage technical risk



An example timeline from “Beyond Retrospectives”
Linda Rising, GOTO Aarhus 2011



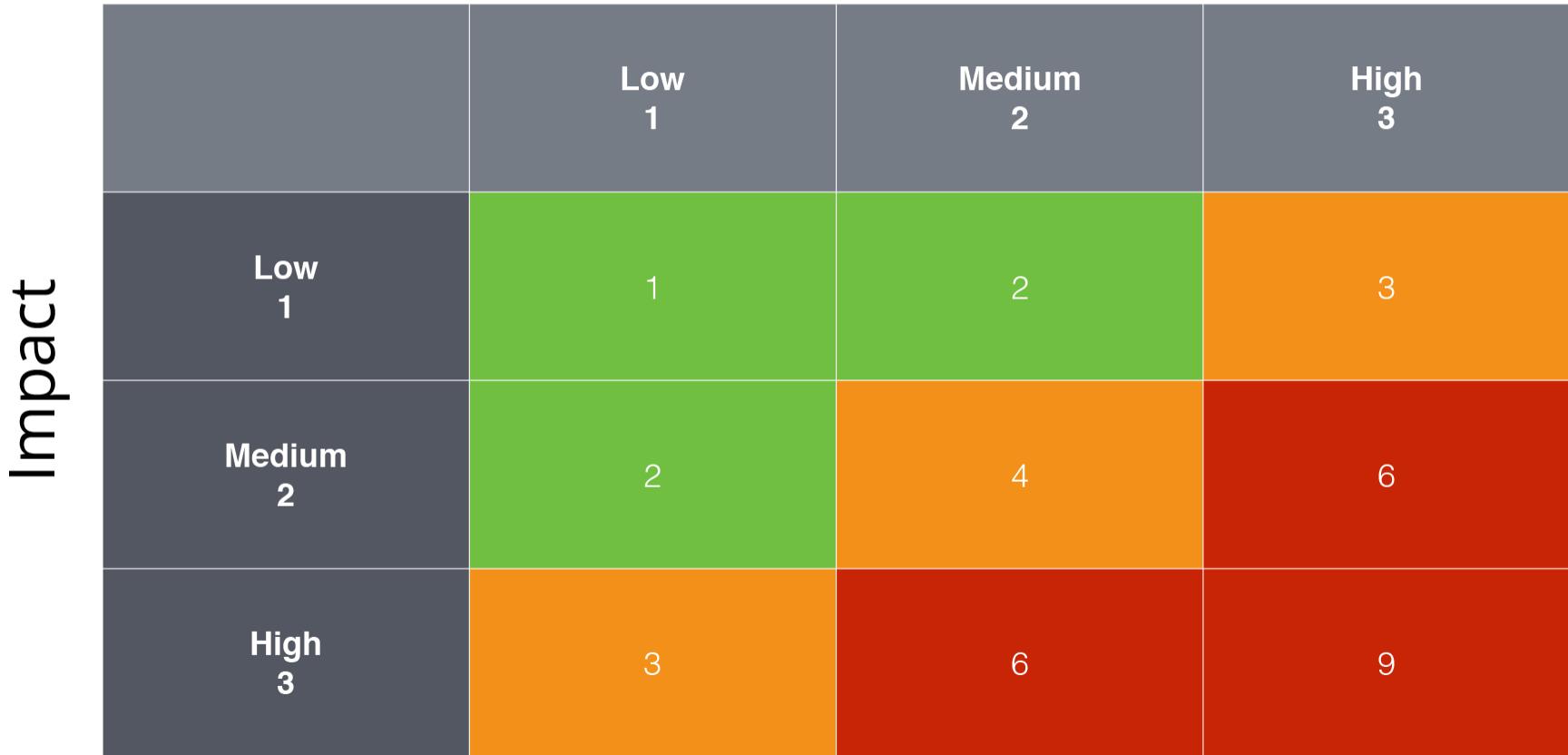
Problems with New Technology

Problems with new technology

An example timeline from “Beyond Retrospectives”
Linda Rising, GOTO Aarhus 2011

**Identify and mitigate
your highest priority risks**

Probability

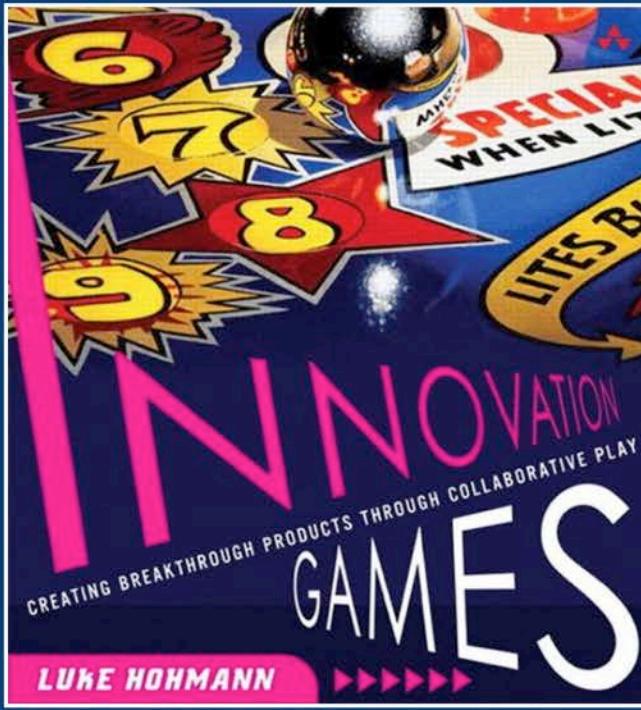
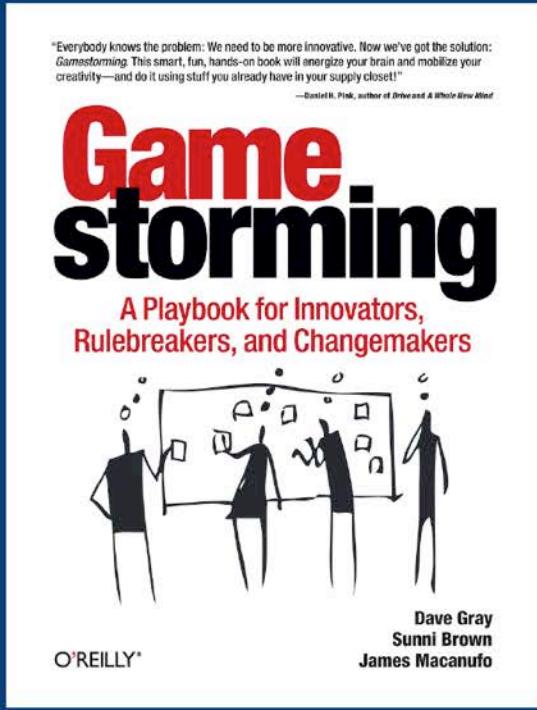


The software architecture role
should own the technical risks

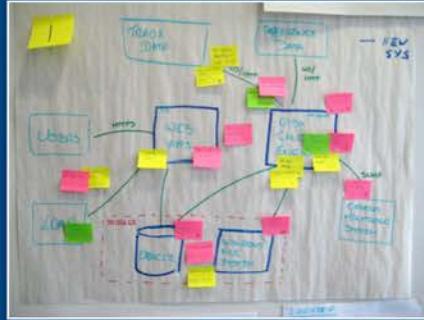
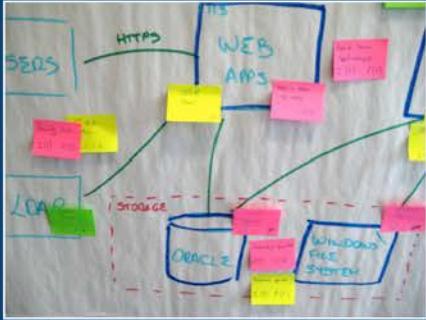
Architecturally significant?

Costly to change, complex or new

Like estimates,
risks are subjective



Visual and collaborative “games”



Risk-storming

A visual and collaborative technique for identifying risk

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, ...

Just enough up front design to create
firm and sufficient foundations

Explore

Architectural drivers

(requirements, quality attributes, constraints and principles)

System context diagram,
event storming, domain
modelling, interviews,
workshops, etc

Context and scope

(users, usage scenarios, other software system dependencies in the environment)

The goal is a quick (hours or days rather than months or years), visual and collaborative approach to up front design to provide a starting point and set a direction for the team

Explore

Design

Architectural drivers

(requirements, quality attributes, constraints and principles)

System context diagram,
event storming, domain
modelling, interviews,
workshops, etc

Context and scope

(users, usage scenarios, other software system dependencies in the environment)

Structure

(understand the significant structural elements and how they interact)

Design (OOAD, CRC, DDD, etc) with container and component diagrams as key outputs

Vision

(be able to effectively and efficiently share your vision with other people)

The goal is a quick (hours or days rather than months or years), visual and collaborative approach to up front design to provide a starting point and set a direction for the team

Explore

Architectural drivers
(requirements, quality attributes, constraints and principles)

System context diagram, event storming, domain modelling, interviews, workshops, etc

Context and scope
(users, usage scenarios, other software system dependencies in the environment)

Design

Structure
(understand the significant structural elements and how they interact)

Design (OOAD, CRC, DDD, etc) with container and component diagrams as key outputs

Vision

(be able to effectively and efficiently share your vision with other people)

Assess

Review and test
(check the design satisfies the architectural drivers)

Architecture reviews, dry runs, risk-storming and concrete experiments

Risks

(identify and mitigate the highest priority risks)

The goal is a quick (hours or days rather than months or years), visual and collaborative approach to up front design to provide a starting point and set a direction for the team

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

You understand the context and scope of what you're designing.

You understand the significant design decisions (i.e. technology, modularity, etc).

You are confident that your design satisfies the architectural drivers.

You have identified and are comfortable with the risks associated with building the software.

You have a way to communicate your vision to other people.

How long?

Hours, days or weeks ... not months or years

Estimates?

Features vs components

we used to do stuff like this,
it worked but we stopped doing
it when we became agile

We all came away impressed with
the practical, agile approach to
software architecture. I'm going to
push that we adopt most of these
practices as a baseline...

Adopt an agile mindset

Choose a starting point and continuously improve
to discover what works for you

Questions

Thank you!

simon.brown@codingthearchitecture.com
@simonbrown