

The lost art of software design

Simon Brown

 @simonbrown

Over the past decade, many
teams have thrown away
big design up front

Unfortunately, architectural
thinking, documentation,
diagramming and modelling
were also often discarded

The Agile Manifesto
doesn't say
“don't do design” 🤔

You can't move fast
with  code

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

Dave Thomas



Big Design Up Front

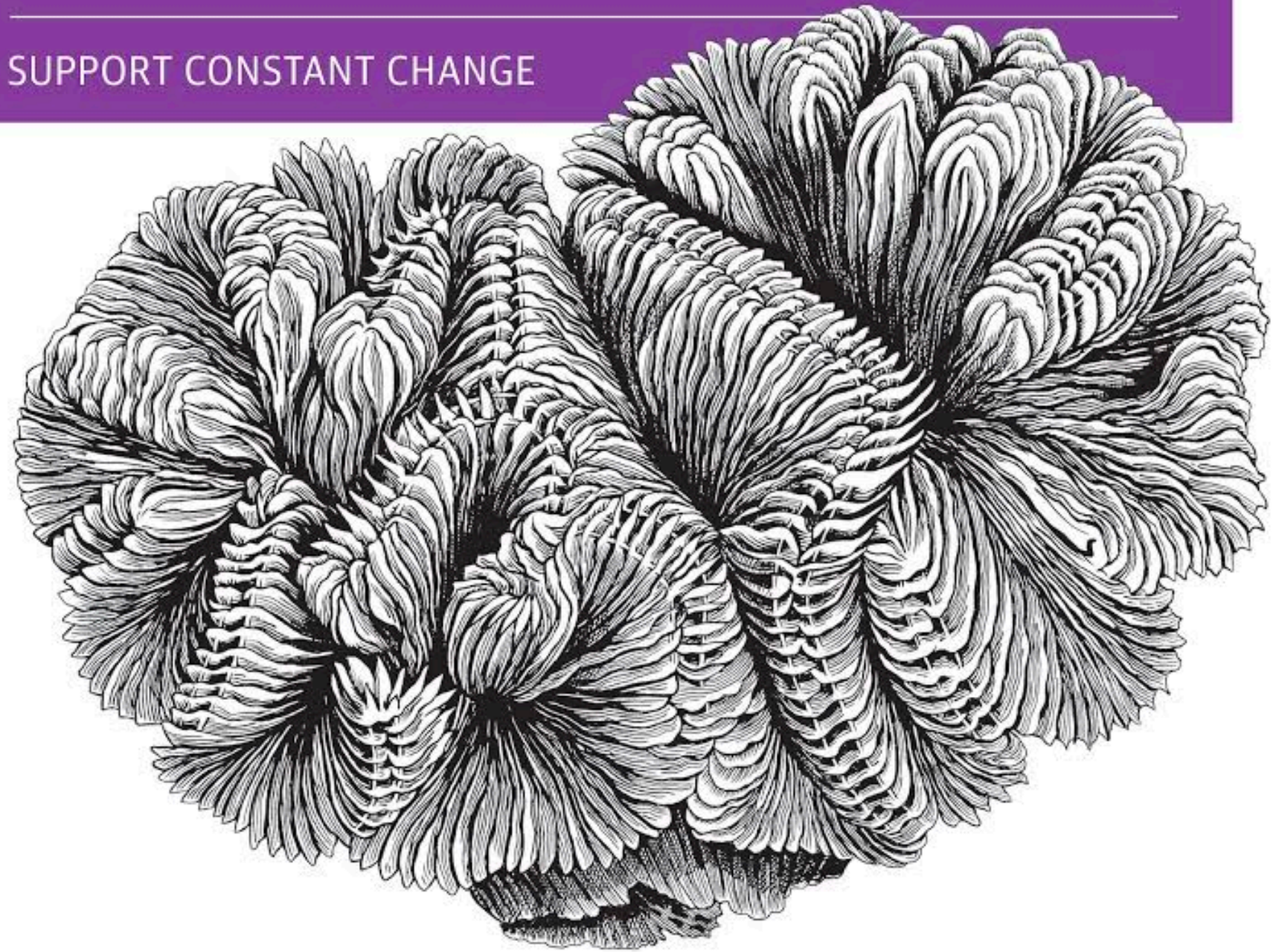
Evolutionary Design



O'REILLY®

Building Evolutionary Architectures

SUPPORT CONSTANT CHANGE



Neal Ford, Rebecca Parsons & Patrick Kua

Evolutionary architecture

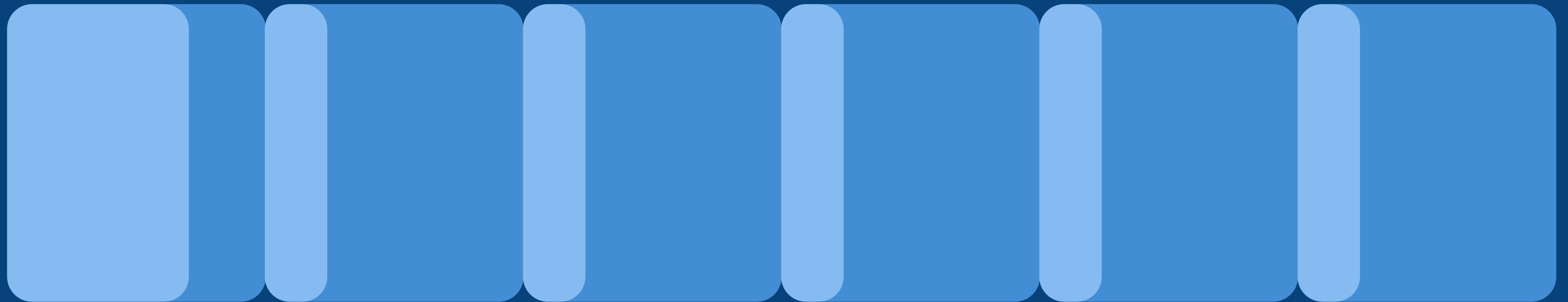
Evolutionary architecture

Architecting for change also results
in significant decisions being made!

Planned vs unplanned evolution
... both need to be guided

Goals

1. Explain why some up front design is useful
2. Provide some tips on how to do design better



Some Design Up Front + Evolutionary Design



A Venn diagram with two overlapping circles. The left circle is light blue and labeled 'Product Design'. The right circle is a darker blue and labeled 'Technical Design'. The overlapping area in the center is a medium blue. The background of the slide is a dark blue.

Product Design

Product vision,
UX, UI, A/B testing,
experimentation
business process,
etc

Technical Design

Technical vision,
technologies, modularity,
quality attributes,
environmental constraints,
etc

I'm referring to **technical design**
rather than product design

Simon Brown

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

@simonbrown

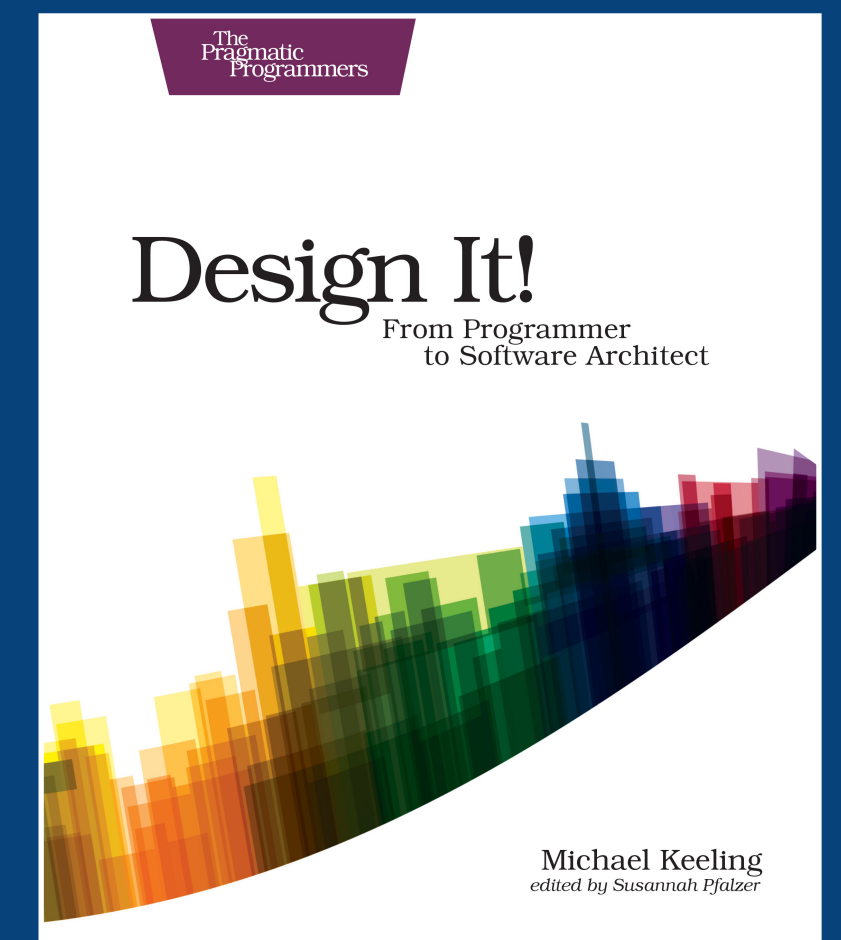
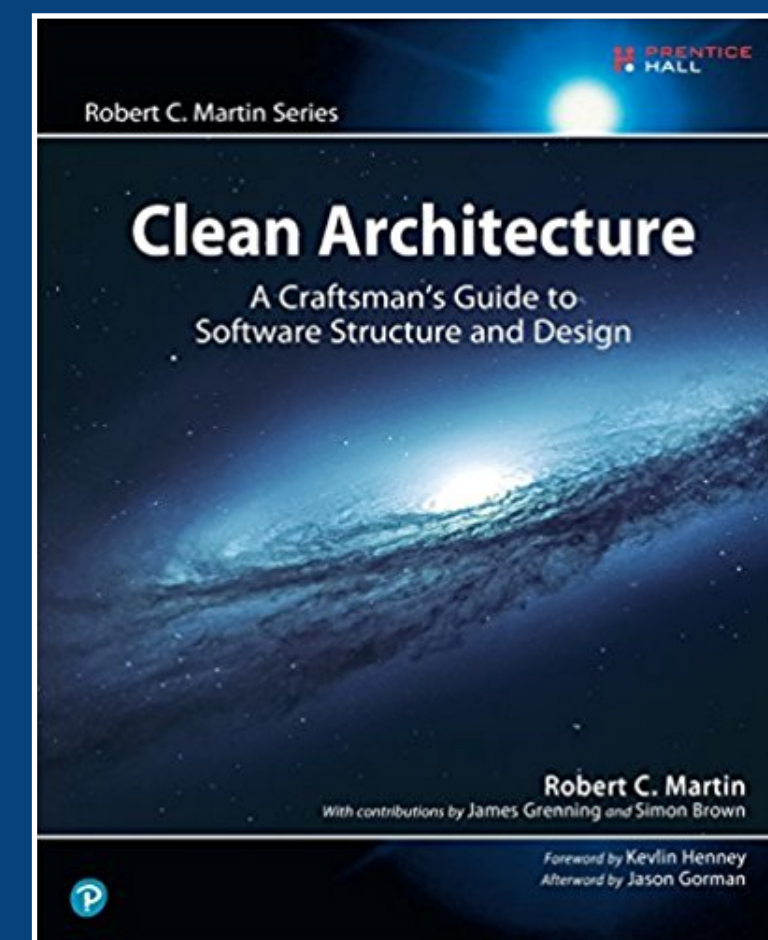
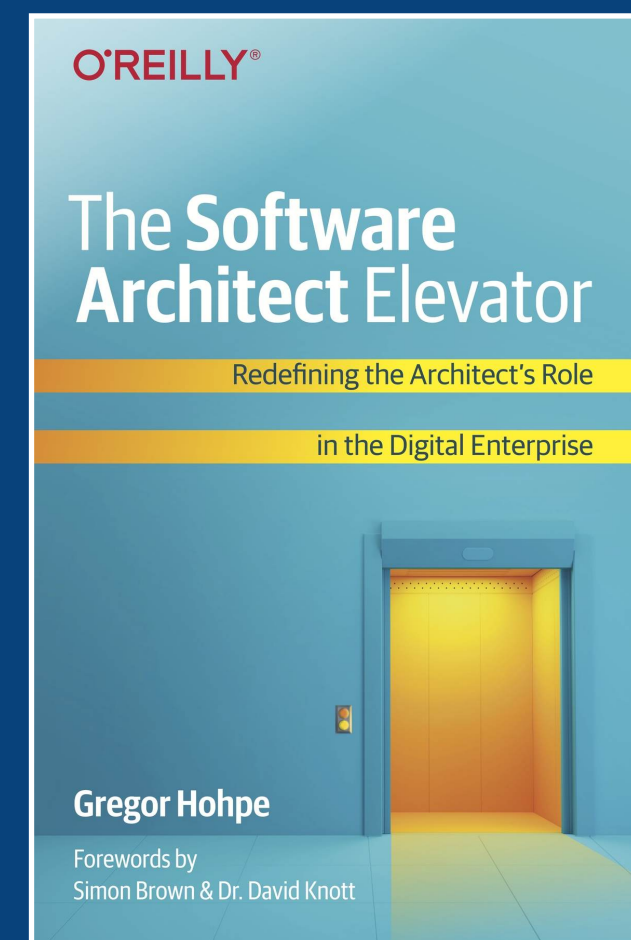
Software
architecture
for
developers

Simon Brown

The
C4
model

for visualising software architecture

Simon Brown



Architecture meets agile

“we’re about to start our agile transformation ... we need help making our architecture/design processes more agile”

vs

Agile meets architecture

“we’ve been on our agile journey for X years ... our software lacks structure, we have no documentation, etc”

Financial Risk System

1. Context

A global investment bank based in London, New York and Singapore trades (buys and sells) financial products with other banks (“counterparties”). When share prices on the stock markets move up or down, the bank either makes money or loses it. At the end of the working day, the bank needs to gain a view of how much risk of losing money they are exposed to, by running some calculations on the data held about their trades. The bank has an existing Trade Data System (TDS) and Reference Data System (RDS) but needs a new Risk System.

1.1. Trade Data System

The Trade Data System maintains a store of all trades made by the bank. It is already configured to generate a file-based XML export of trade data to a network share at the close of business at 5pm in New York. The export includes the following information for every trade made by the bank:

- Trade ID, Date, Current trade value in US dollars, Counterparty ID

1.2. Reference Data System

The Reference Data System stores all of the reference data needed by the bank. This includes information about counterparties (other banks). A file-based XML export is also generated to a network share at 5pm in New York, and it includes some basic information about each counterparty. A new reference data system is due for completion in the next 3 months, and the current system will eventually be decommissioned. The current data export includes:

- Counterparty ID, Name, Address, etc...

2. Functional Requirements

1. Import trade data from the Trade Data System.
2. Import counterparty data from the Reference Data System.
3. Join the two sets of data together, enriching the trade data with information about the counterparty.
4. For each counterparty, calculate the risk that the bank is exposed to.
5. Generate a report that can be imported into Microsoft Excel containing the risk figures for all counterparties known by the bank.
6. Distribute the report to the business users before the start of the next trading day (9am) in Singapore.
7. Provide a way for a subset of the business users to configure and maintain the external parameters used by the risk calculations.

3. Non-functional Requirements

a. Performance

- Risk reports must be generated before 9am the following business day in Singapore.

b. Scalability

- The system must be able to cope with trade volumes for the next 5 years.
 - The Trade Data System export includes approximately 5000 trades now and it is anticipated that there will be slow but steady growth of 10 additional trades per day.
 - The Reference Data System export includes approximately 20,000 counterparties and growth will be negligible.
- There are 40-50 business users around the world that need access to the report.

c. Availability

- Risk reports should be available to users 24x7, but a small amount of downtime (less than 30 minutes per day) can be tolerated.

d. Failover

- Manual failover is sufficient, provided that the availability targets can be met.

e. Security

- This system must follow bank policy that states system access is restricted to authenticated and authorised users only.
- Reports must only be distributed to authorised users.
- Only a subset of the authorised users are permitted to modify the parameters used in the risk calculations.
- Although desirable, there are no single sign-on requirements (e.g. integration with Active Directory, LDAP, etc).
- All access to the system and reports will be within the confines of the bank's global network.

f. Audit

- The following events must be recorded in the system audit logs:
 - Report generation.
 - Modification of risk calculation parameters.

g. Fault Tolerance and Resilience

- The system should take appropriate steps to recover from an error if possible, but all errors should be logged.
- Errors preventing a counterparty risk calculation being completed should be logged and the process should continue.

h. Internationalization and Localization

- All user interfaces will be presented in English only.
- All reports will be presented in English only.
- All trading values and risk figures will be presented in US dollars only.

i. Monitoring and Management

- A Simple Network Management Protocol (SNMP) trap should be sent to the bank's Central Monitoring Service in the following circumstances:
 - When there is a fatal error with the system.
 - When reports have not been generated before 9am Singapore time.

j. Data Retention and Archiving

- Input files used in the risk calculation process must be retained for 1 year.

k. Interoperability

- Interfaces with existing data systems should conform to and use existing data formats.

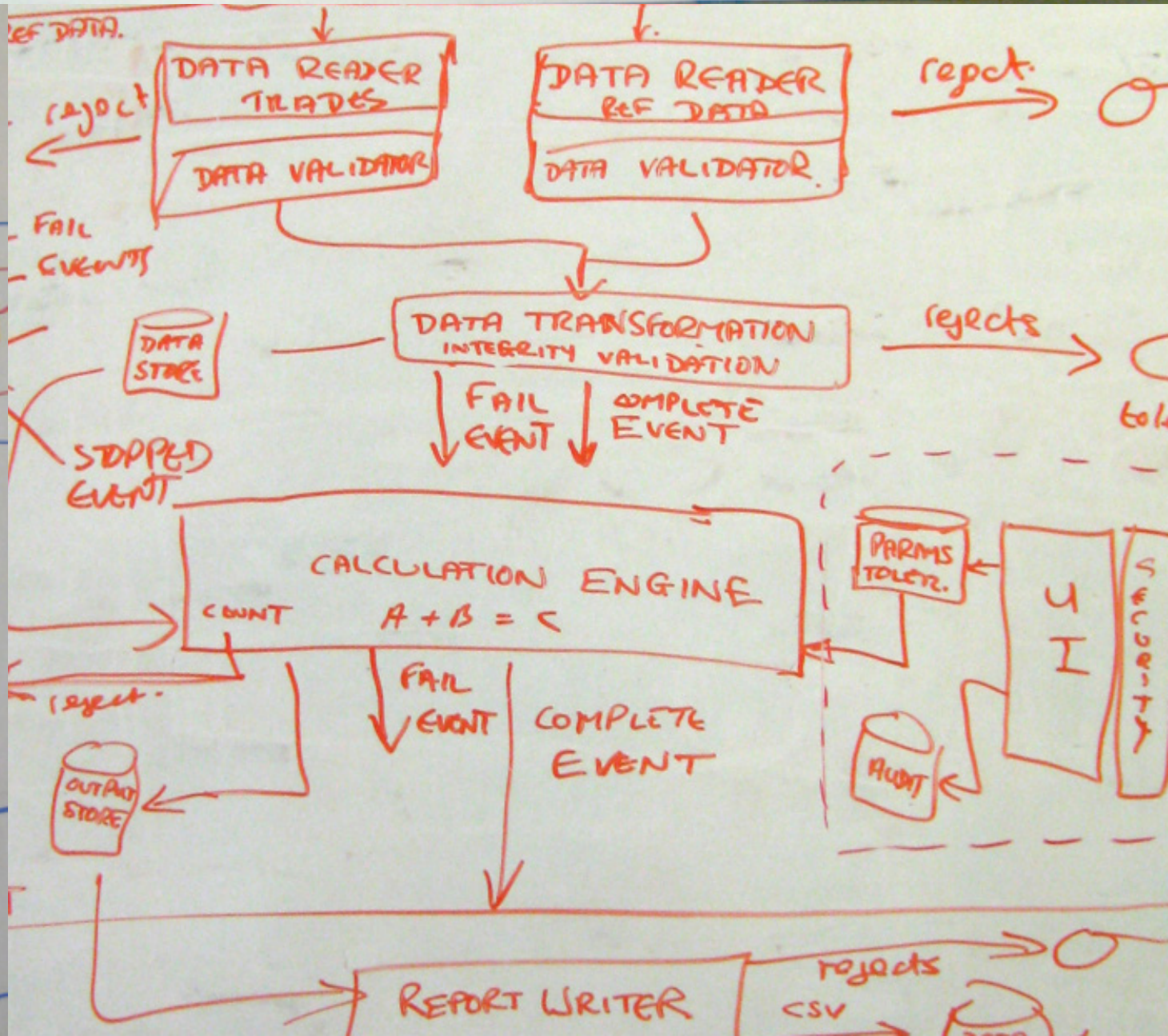
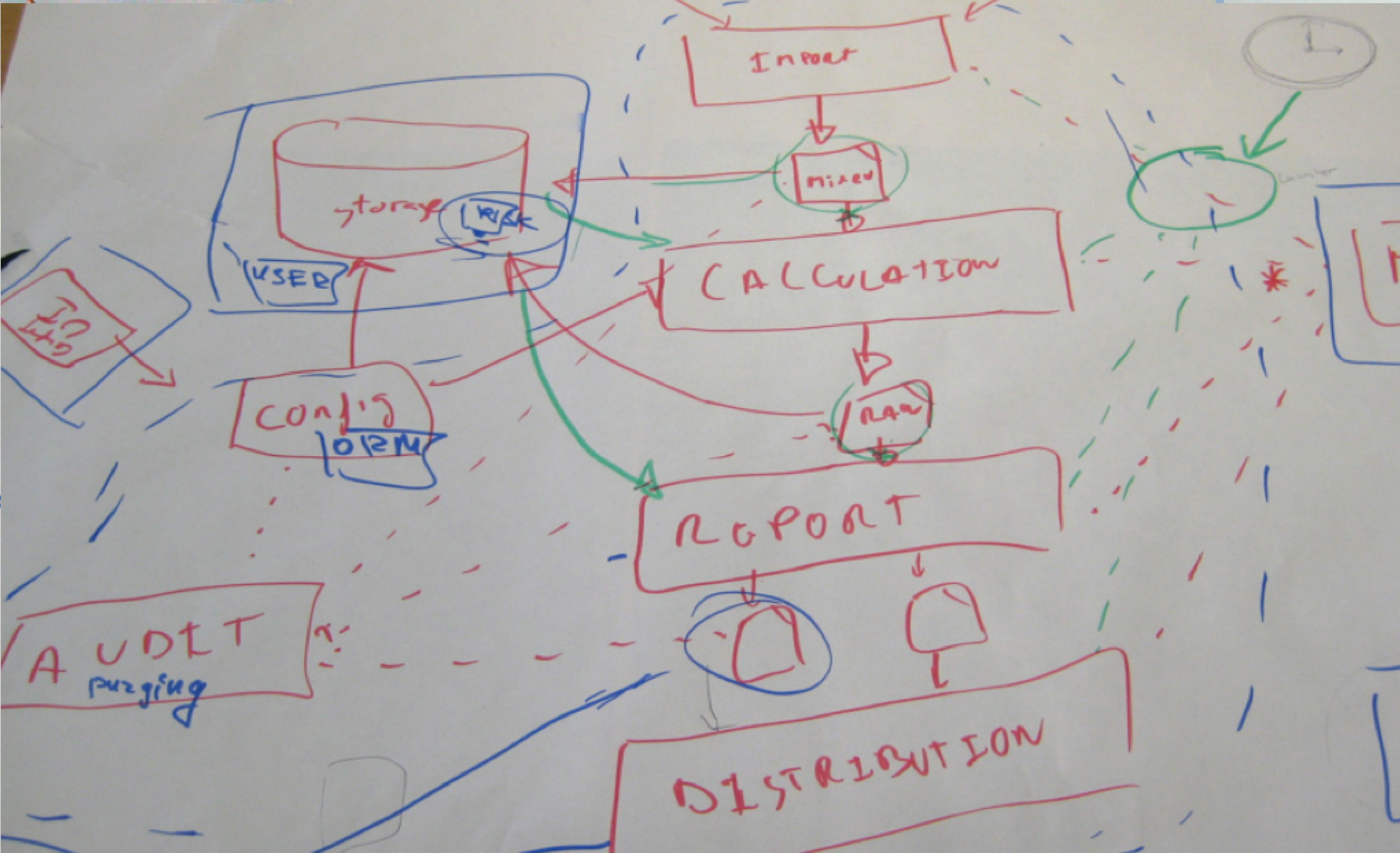
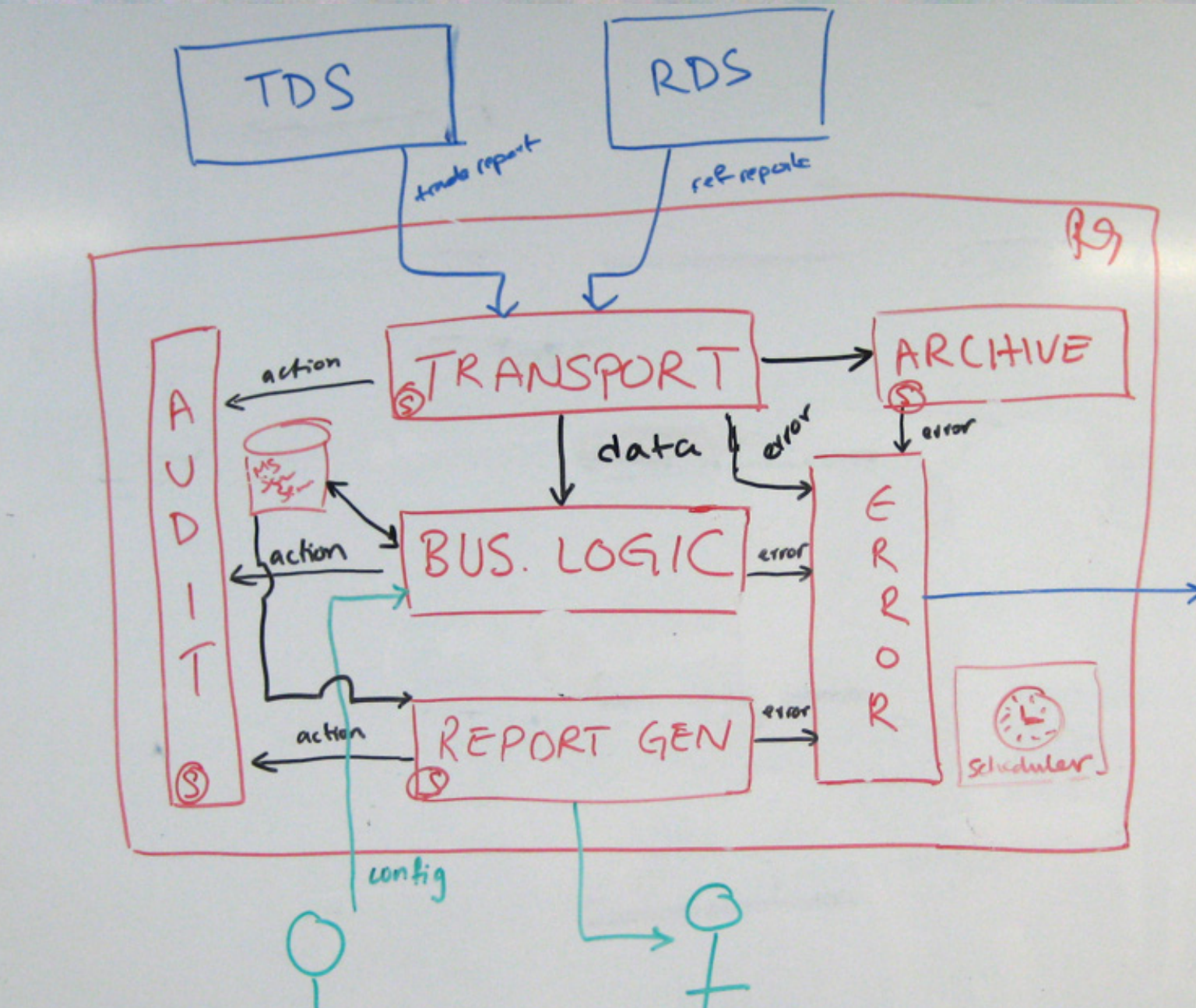
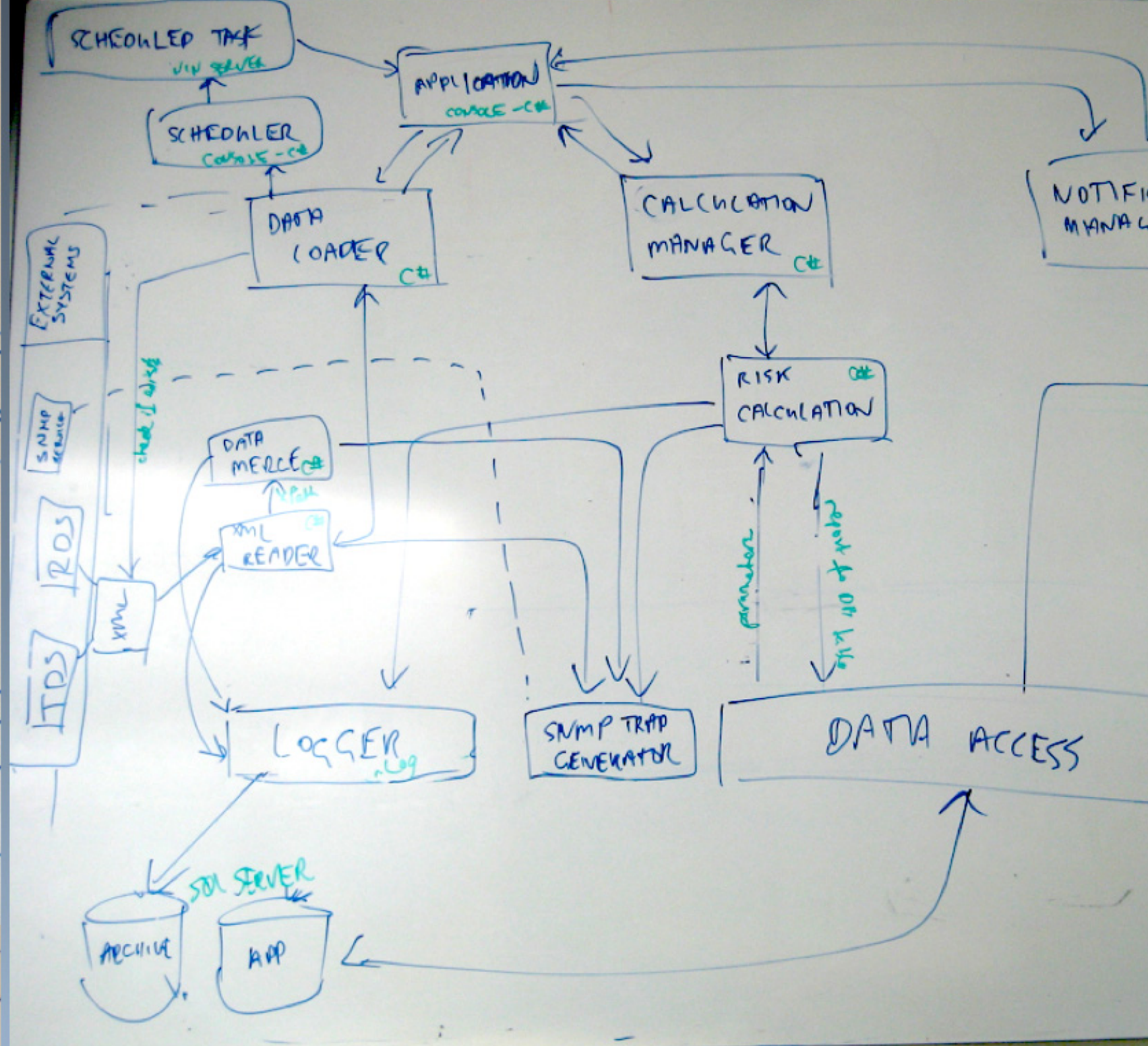
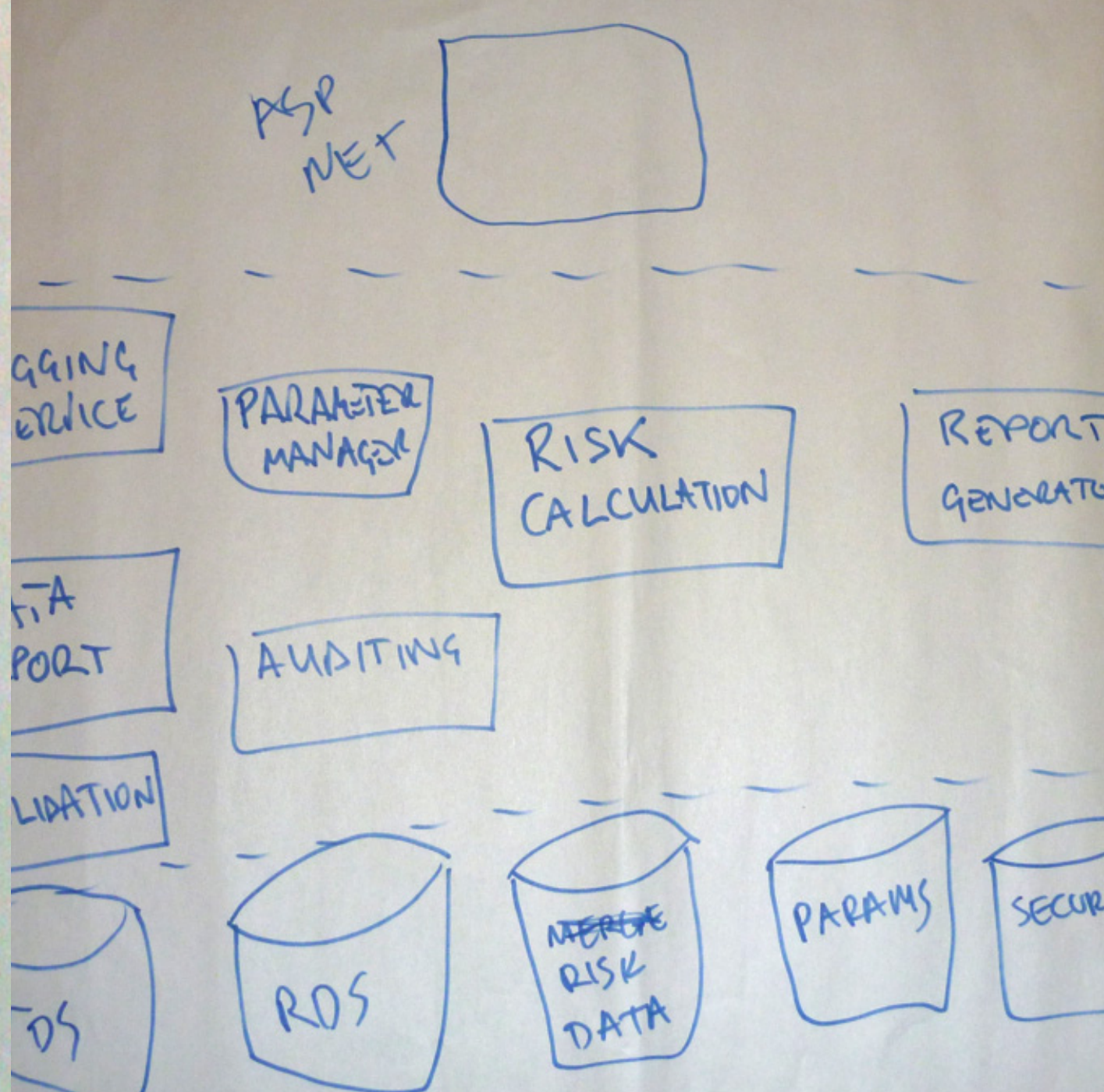
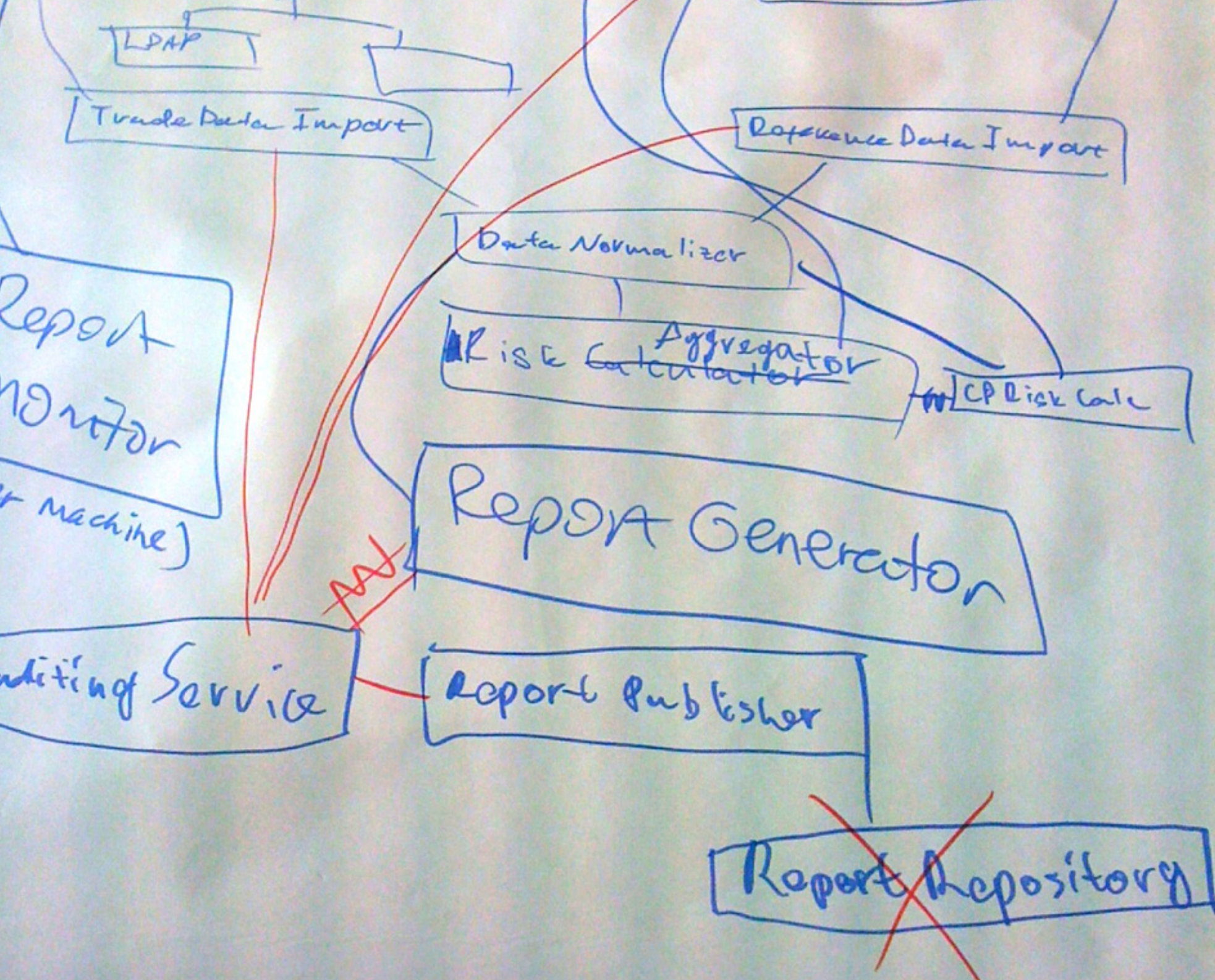


Design a software solution for the “Financial Risk System”, and **draw** one or more architecture diagrams to describe your solution



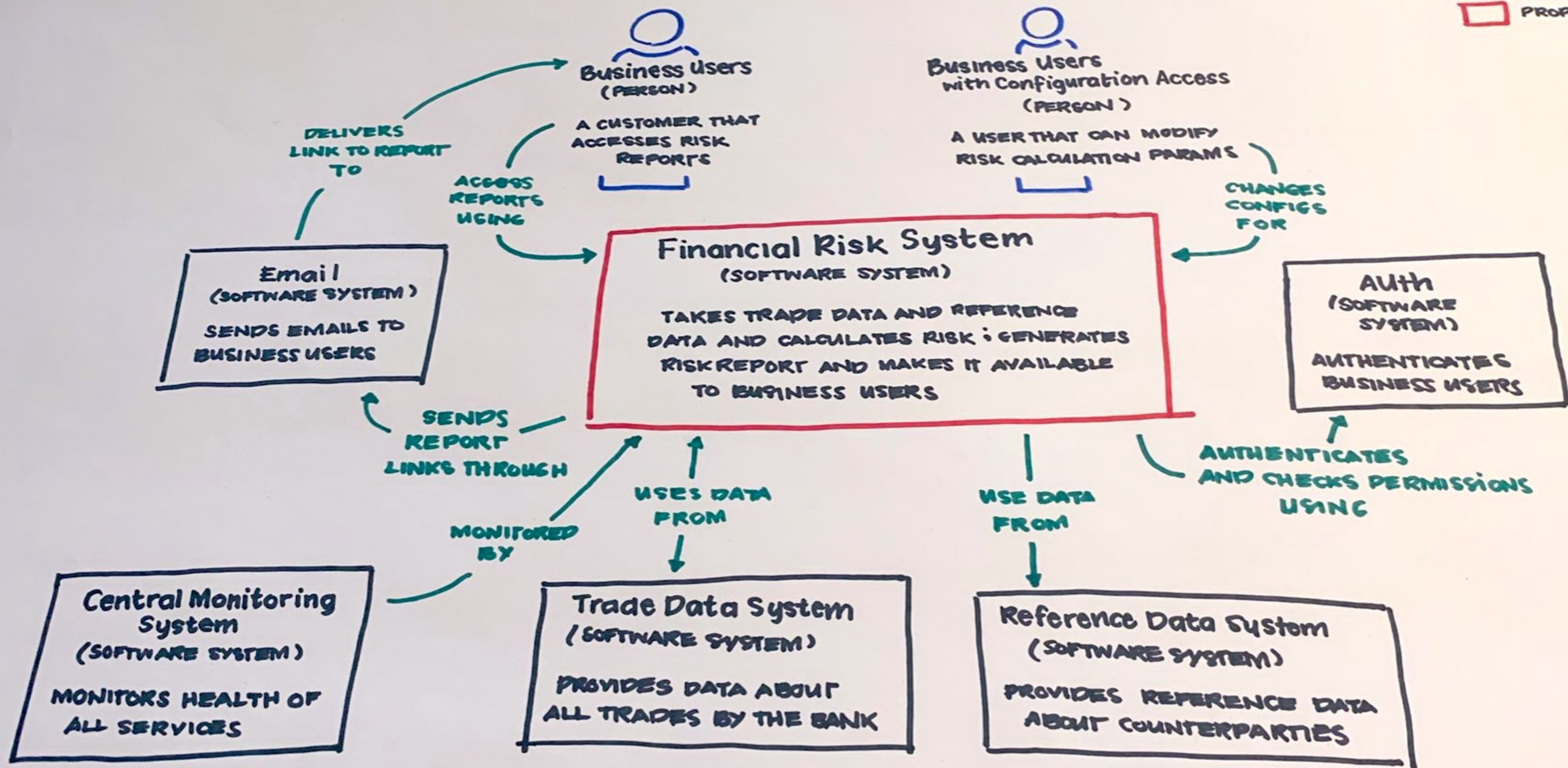
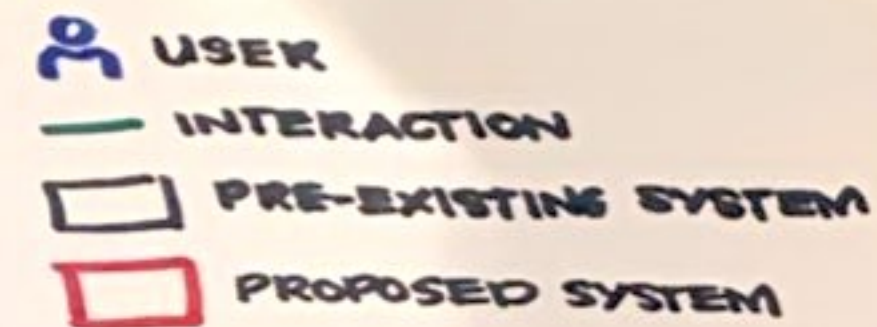
60-90 minutes

Iteration 1

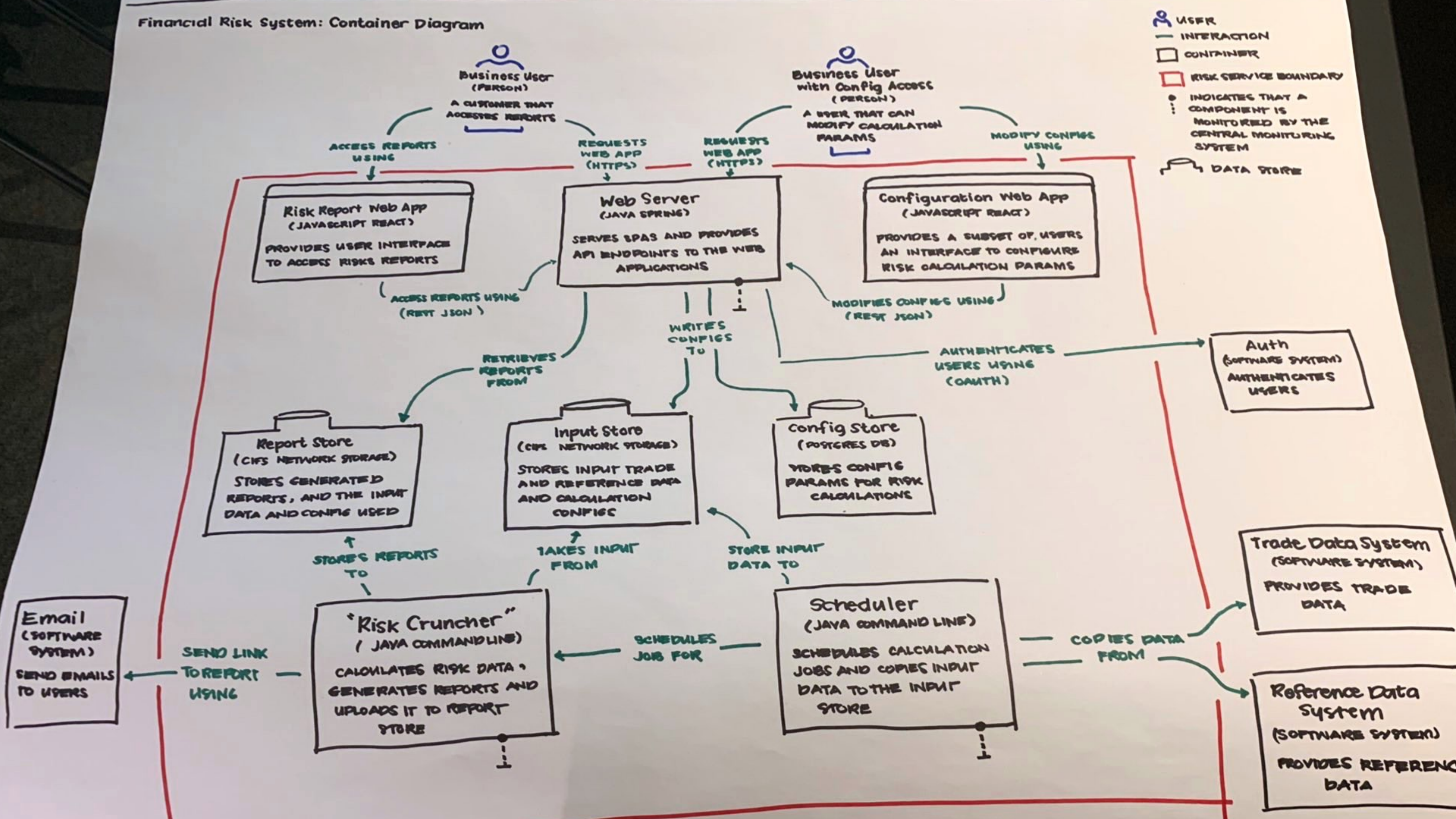


Iteration 2

Financial Risk System: Context Diagram



Financial Risk System: Container Diagram



So you're teaching teams
how to create nice diagrams?

Up Front Design



97 Strategies to Avoid
Up Front Design

O RLY?

Vera Gile

#1

“Are we allowed
to do
up front design?”



97 Strategies to Avoid Up Front Design

O RLY?

Vera Gile

#12

“We don't do up
front design
because we do
XP.”



97 Strategies to Avoid Up Front Design

O RLY?

Vera Gile

#17

“It’s not expected
in agile.”

There is no Big Design Up Front. Most of the design activity takes place **on the fly and incrementally**, starting with "the simplest thing that could possibly work" and adding complexity only when it's required by failing tests.

https://en.wikipedia.org/wiki/Extreme_programming

What role does an architecture play when you are using evolutionary design? Again XP's critics state that XP ignores architecture, that XP's route is to go to code fast and trust that refactoring that will solve all design issues. Interestingly they are right, and that may well be weakness. **Certainly the most aggressive XPers** - Kent Beck, Ron Jeffries, and Bob Martin - **are putting more and more energy into avoiding any up front architectural design.** Don't put in a database until you really know you'll need it. Work with files first and refactor the database in during a later iteration.

Martin Fowler

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



Ron Jeffries

@RonJeffries

This is why I really don't believe that "luminaries" in Agile are telling people not to design. Too many supposedly agile team members, teams, and even their coaches really don't know what has been said, much less what it's about.

Stacy Cashmore @Stacy_Cash

Replying to @NativeWired @HelenLisowski

I think the phrase whilst we value those on the right, we value those on the left more is forgotten quite often...

5:05 PM - 18 Mar 2019

The “luminaries” in Agile are not telling people to **do** design either
(it's easy to make assumptions about what's **not** being said)

Remember that the folks
behind the agile manifesto
have a lot of experience.

**Most teams likely don't have
that same level of experience.**

Many people haven't been
exposed to the problems that
agile was trying to solve



@KentBeck

I unexpectedly have two weeks free before I start my new job (more on that later). I decided to time-box write the book on software design that's been ripening in my head for a decade or two. Here's the outline:

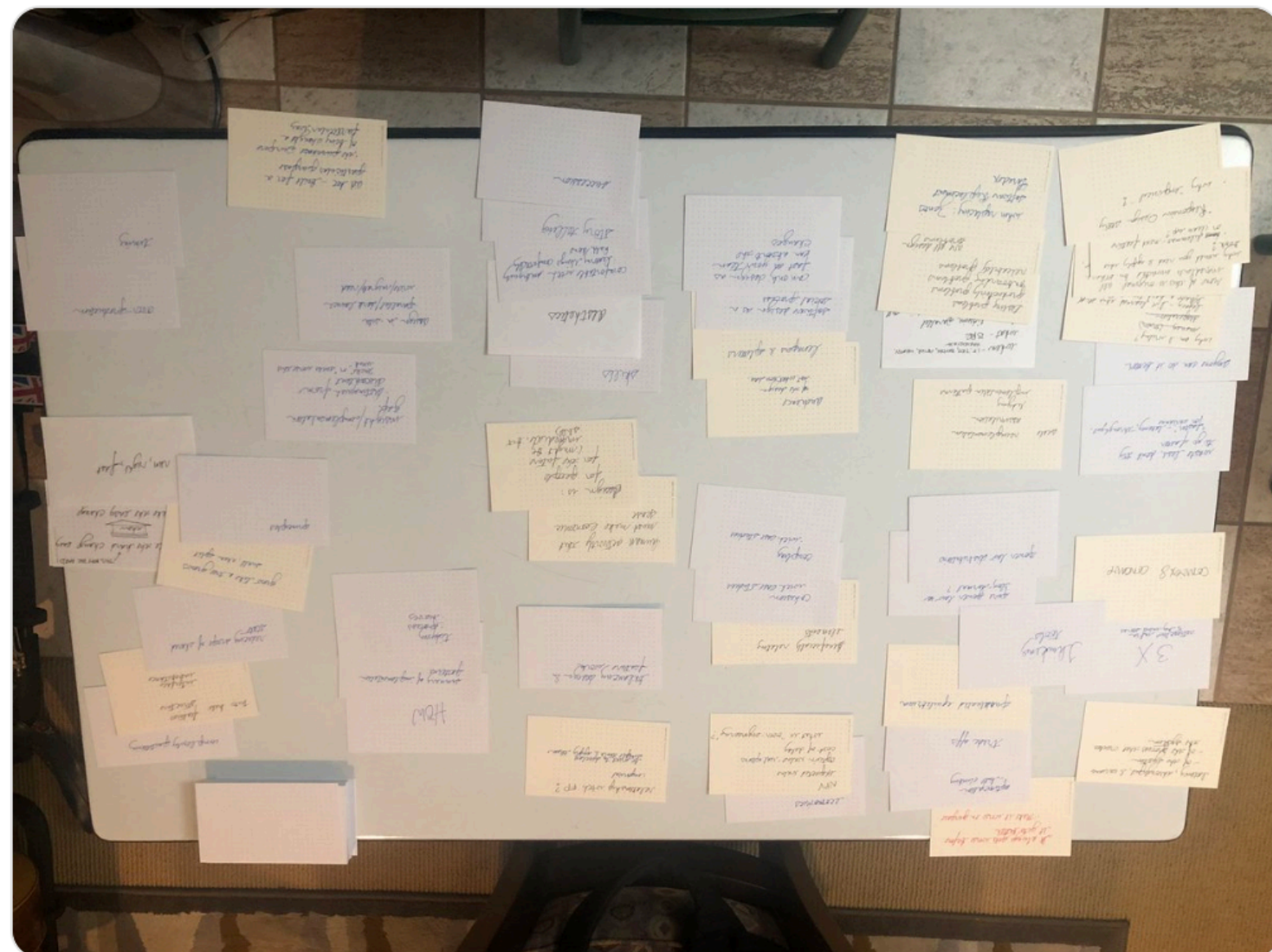




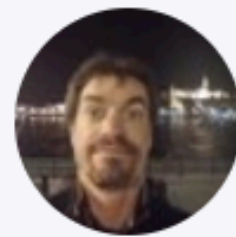
Kent Beck ✓

@KentBeck

I unexpectedly have two weeks free before I start my new job (more on that later). I decided to time-box write the book on software design that's been ripening in my head for a decade or two. Here's the outline:



1:24 AM - 18 Mar 2019



John Hearn @johnhearnbcn · Mar 18

Replying to @KentBeck

Interesting that you write an outline first. My daughter has discovered that writing with flow, keeping notes and refactoring as she goes gives her better results



2



5

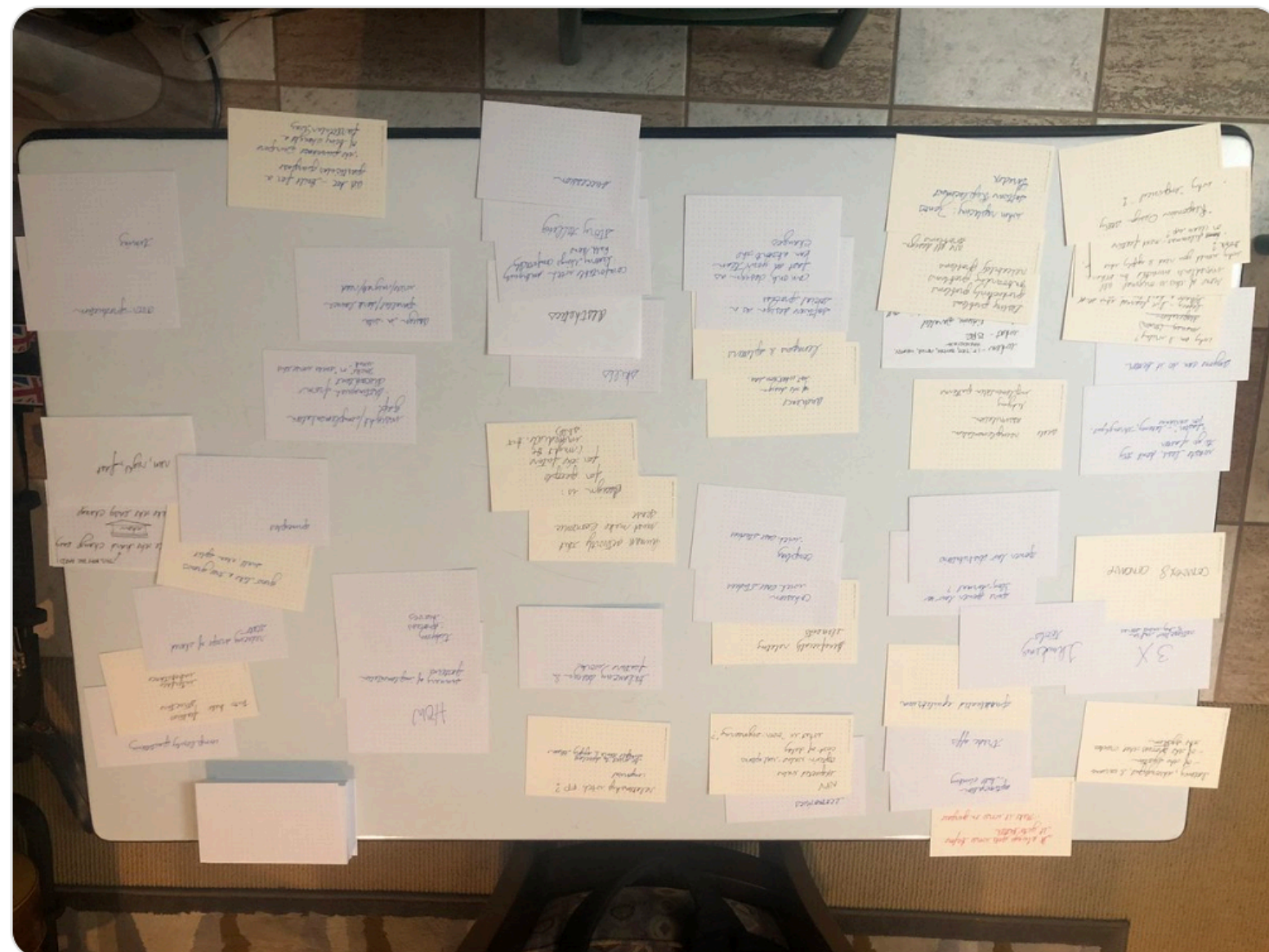




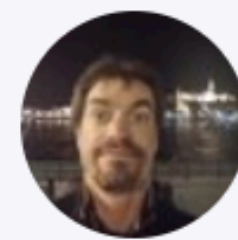
Kent Beck ✓

@KentBeck

I unexpectedly have two weeks free before I start my new job (more on that later). I decided to time-box write the book on software design that's been ripening in my head for a decade or two. Here's the outline:



1:24 AM - 18 Mar 2019



John Hearn @johnhearnbcn · Mar 18

Replying to @KentBeck

Interesting that you write an outline first. My daughter has discovered that writing with flow, keeping notes and refactoring as she goes gives her better results



2



5



Kent Beck ✓ @KentBeck · Mar 18

I've done ~20,000 words on software design (maybe more—not going to stop to check...not). To write a book I need to see a whole, in part just to reduce my anxiety.



1



17





Simon Brown @simonbrown · Mar 18

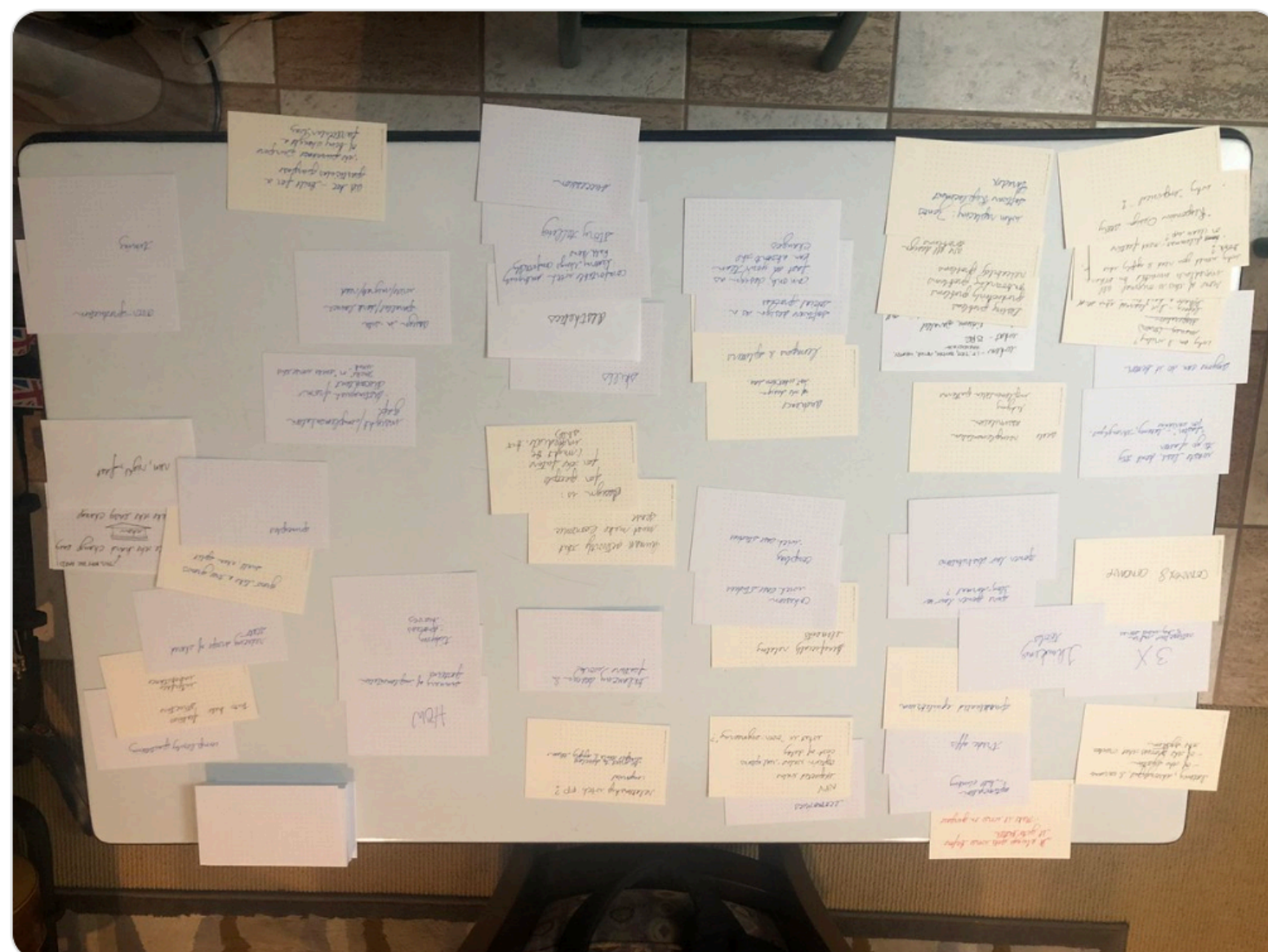
"To write a book I need to see a whole, in part just to reduce my anxiety." ... I'm the same when I'm building software. 😊 #SomeDesignUpFront



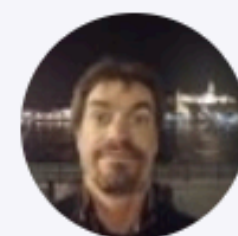
Kent Beck ✓

@KentBeck

I unexpectedly have two weeks free before I start my new job (more on that later). I decided to time-box write the book on software design that's been ripening in my head for a decade or two. Here's the outline:



1:24 AM - 18 Mar 2019



John Hearn @johnhearnbcn · Mar 18

Replying to @KentBeck

Interesting that you write an outline first. My daughter has discovered that writing with flow, keeping notes and refactoring as she goes gives her better results



2



5



Kent Beck ✓ @KentBeck · Mar 18

I've done ~20,000 words on software design (maybe more—not going to stop to check...not). To write a book I need to see a whole, in part just to reduce my anxiety.



1



17



Agility requires a toolbox of
techniques and practices
but many people don't have them,
and we've stopped teaching them

How do you design software?

The background is a solid dark blue. On the left and right sides, there are decorative elements consisting of several parallel, slanted bars in a lighter shade of blue, creating a sense of depth or a stylized 'W' shape.

we use a whiteboard

The background is a solid dark blue. There are four decorative elements consisting of parallel slanted bars in a lighter shade of blue. Two bars are on the left side, and two are on the right side, framing the central text.

we draw boxes and lines



the boxes
represent components



we use our
experience





WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Featured content](#)
[Current events](#)
[Random article](#)
[Donate to Wikipedia](#)
[Wikipedia store](#)

Interaction

[Help](#)
[About Wikipedia](#)
[Community portal](#)

[Article](#) [Talk](#)

[Read](#)

[Edit](#)

[View history](#)



Decomposition (computer science)

From Wikipedia, the free encyclopedia

Decomposition in [computer science](#), also known as **factoring**, is breaking a complex problem or system into parts that are easier to conceive, understand, program, and maintain.

Contents [\[hide\]](#)

- Overview
- Decomposition topics
 - Decomposition paradigm
 - Decomposition diagram
- See also
- References

Decomposition paradigm [\[edit \]](#)

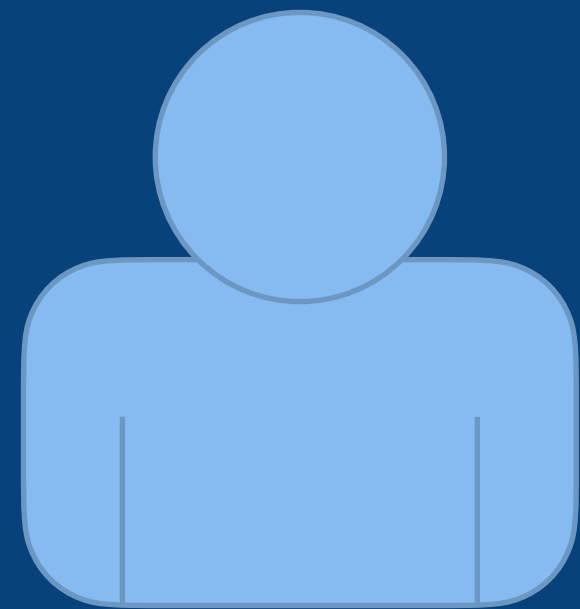
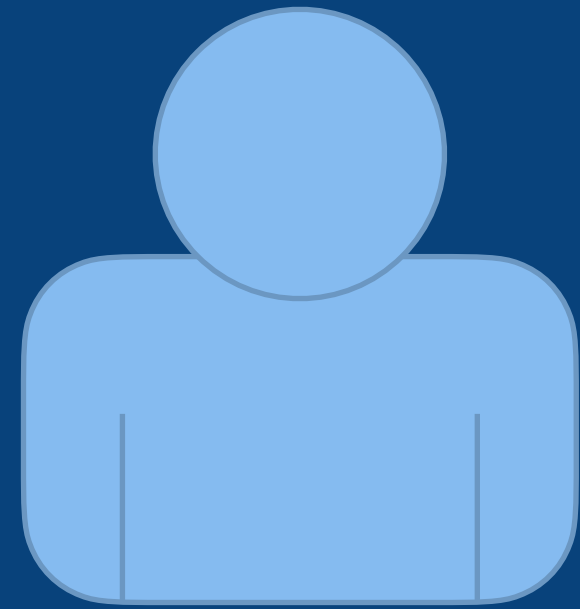
A decomposition paradigm in computer programming is a strategy for organizing a program as a number of parts, and it usually implies a specific way to organize a program text. Usually the aim of using a decomposition paradigm is to optimize some metric related to program complexity, for example the modularity of the program or its maintainability.

Most decomposition paradigms suggest breaking down a program into parts so as to minimize the static dependencies among those parts, and to maximize the [cohesiveness](#) of each part. Some popular decomposition paradigms are the procedural, modules, abstract data type and [object oriented](#) ones.

On the Criteria To Be Used in Decomposing Systems into Modules

Expected Benefits of Modular Programming

The benefits expected of modular programming are: (1) managerial—development time should be shortened because separate groups would work on each module with little need for communication; (2) product flexibility—it should be possible to make drastic changes to one module without a need to change others; (3) comprehensibility—it should be possible to study the system one module at a time. The whole system can therefore be better designed because it is better understood.



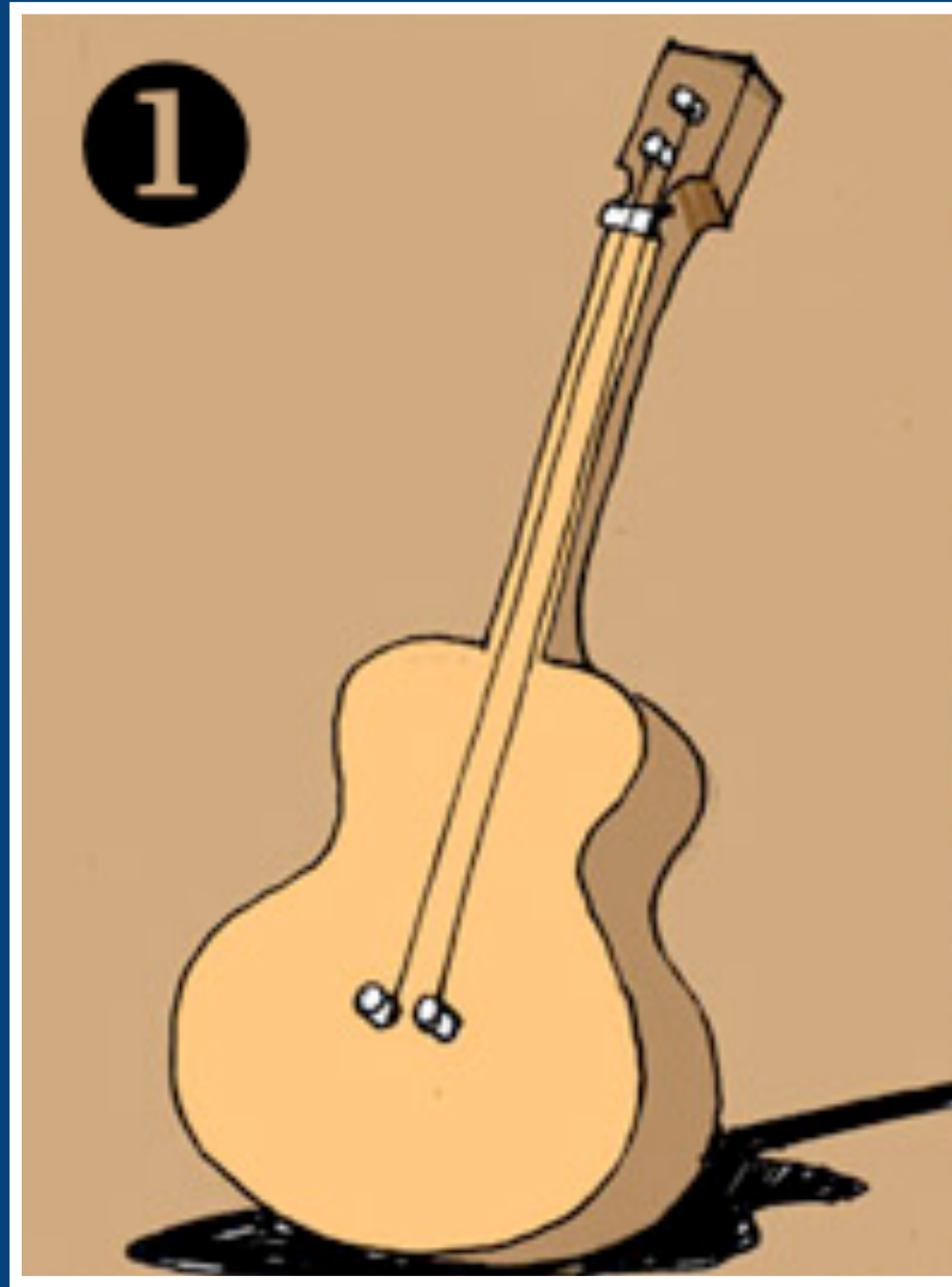
Class-Responsibility-Collaboration

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




Evolutionary Design

Beginning With A Primitive Whole



Evolutionary Design

Beginning With A Primitive Whole



Continuous attention to
technical excellence and
good design enhances agility.

Principle 9 of the Manifesto for Agile Software Development

A good architecture
enables agility

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

A *starting point*
adds value

Every team needs **technical leadership**

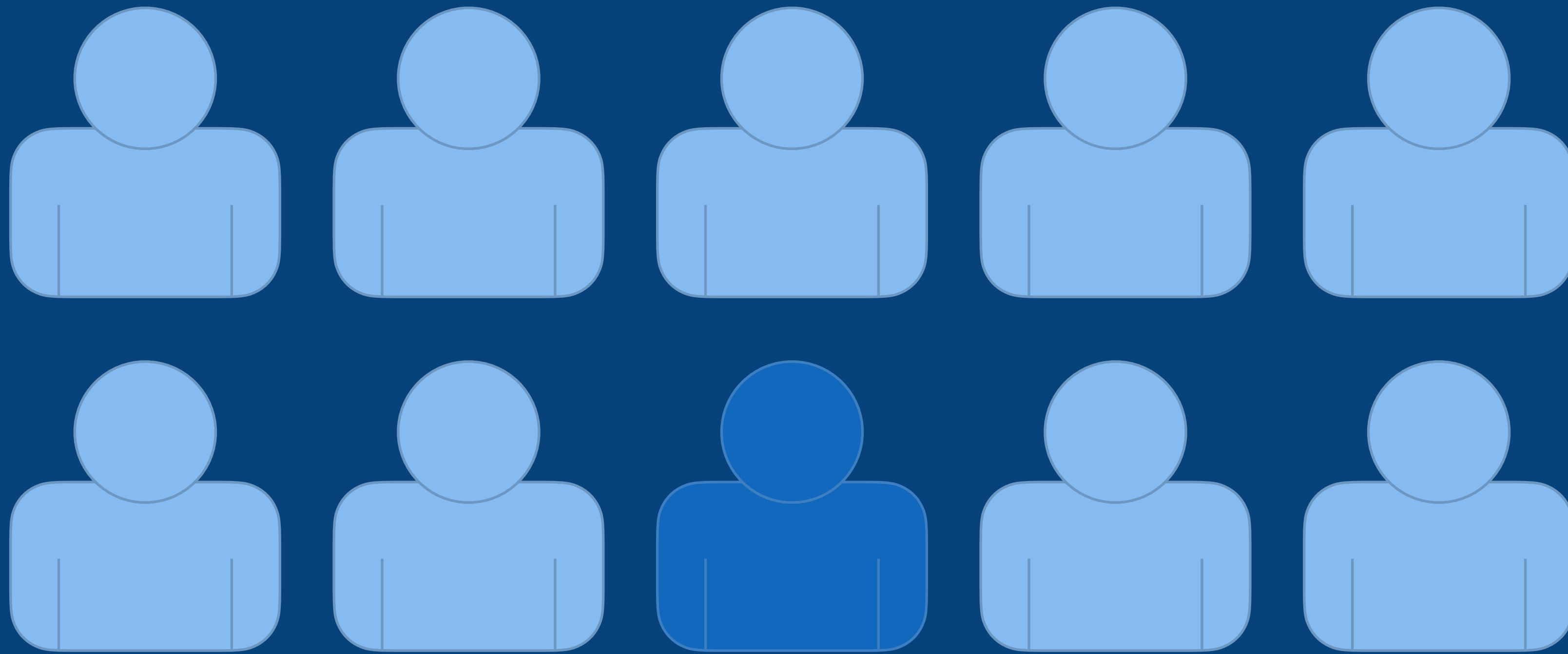
(irrespective of team size)

Technical leadership exists at **multiple levels and dimensions** within most organisations

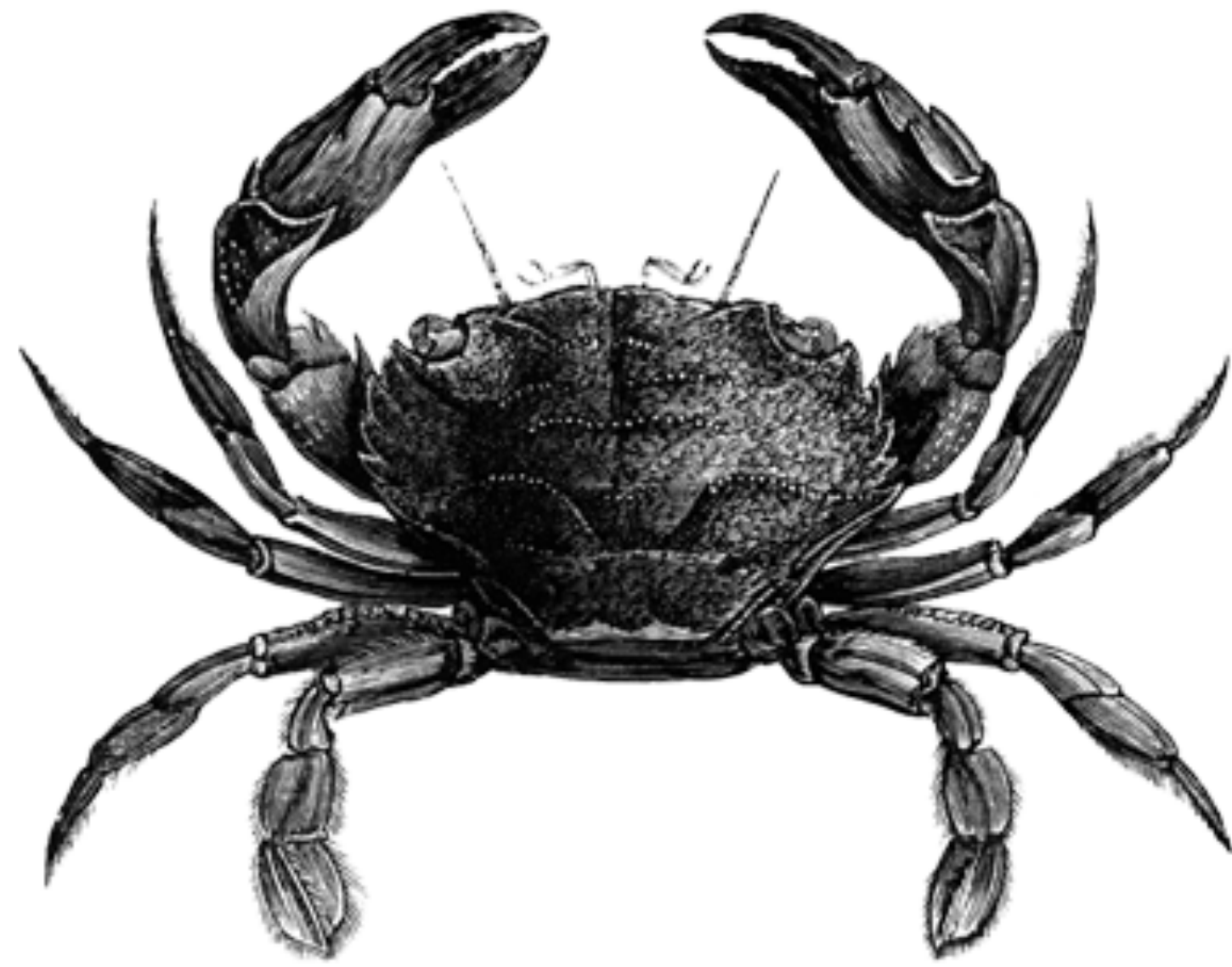
(from the enterprise perspective and platform teams; through to individual delivery teams,
irrespective of whether they have a system or service/capability focus)

**Incomprehensible software
architecture diagrams**

UML?



UML usage is low



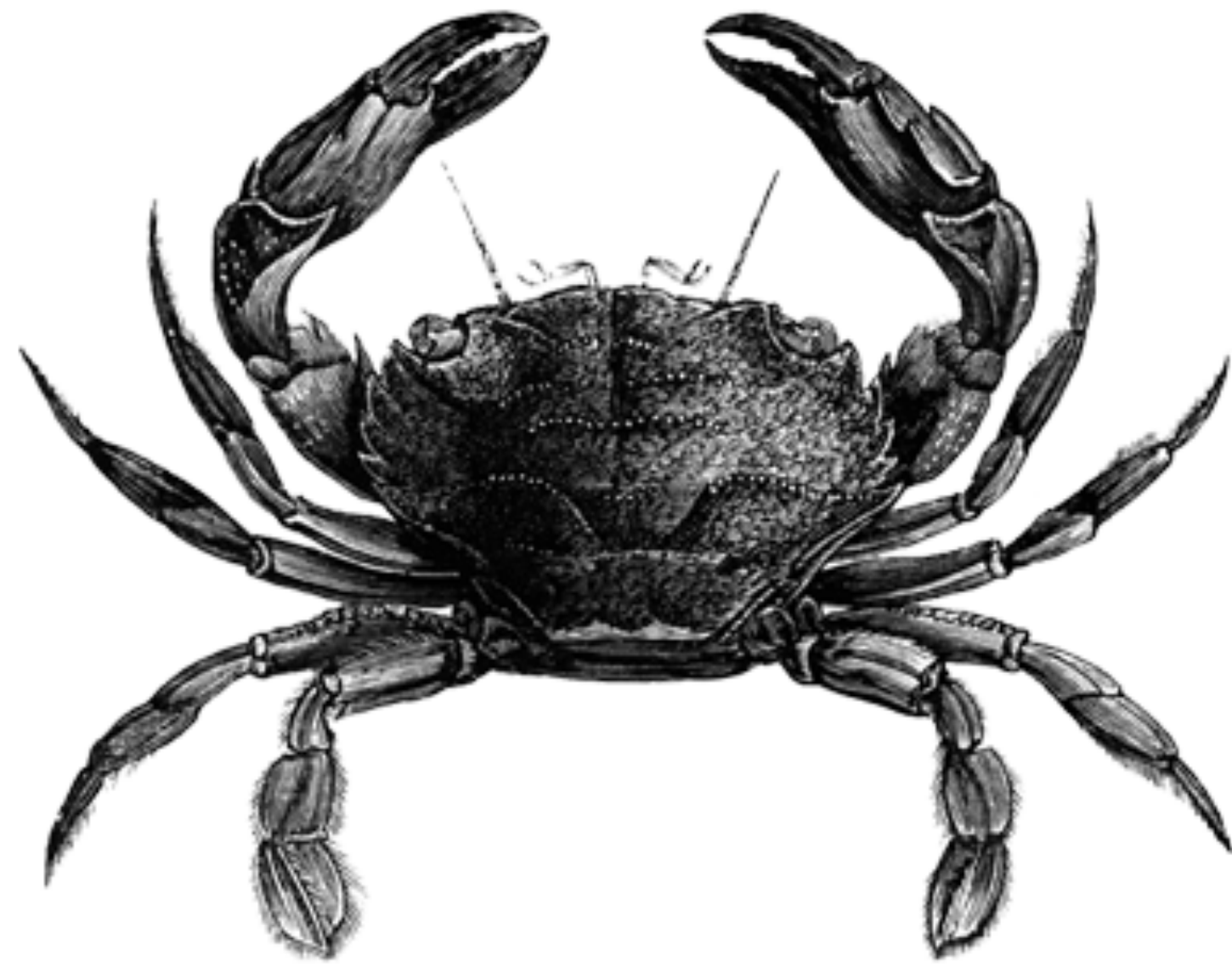
97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#1

"I don't know it."



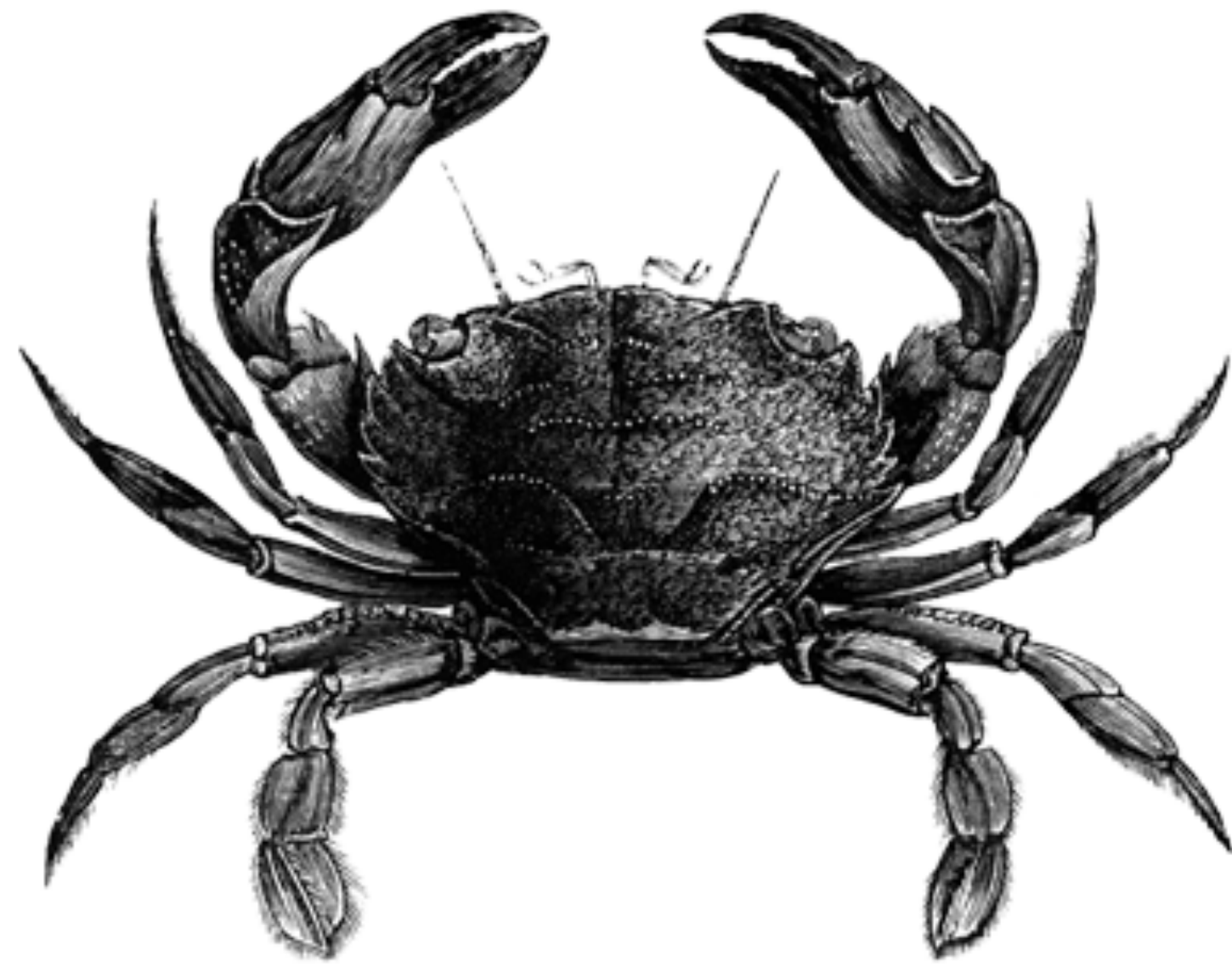
97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#36

“You’ll be seen as
old.”



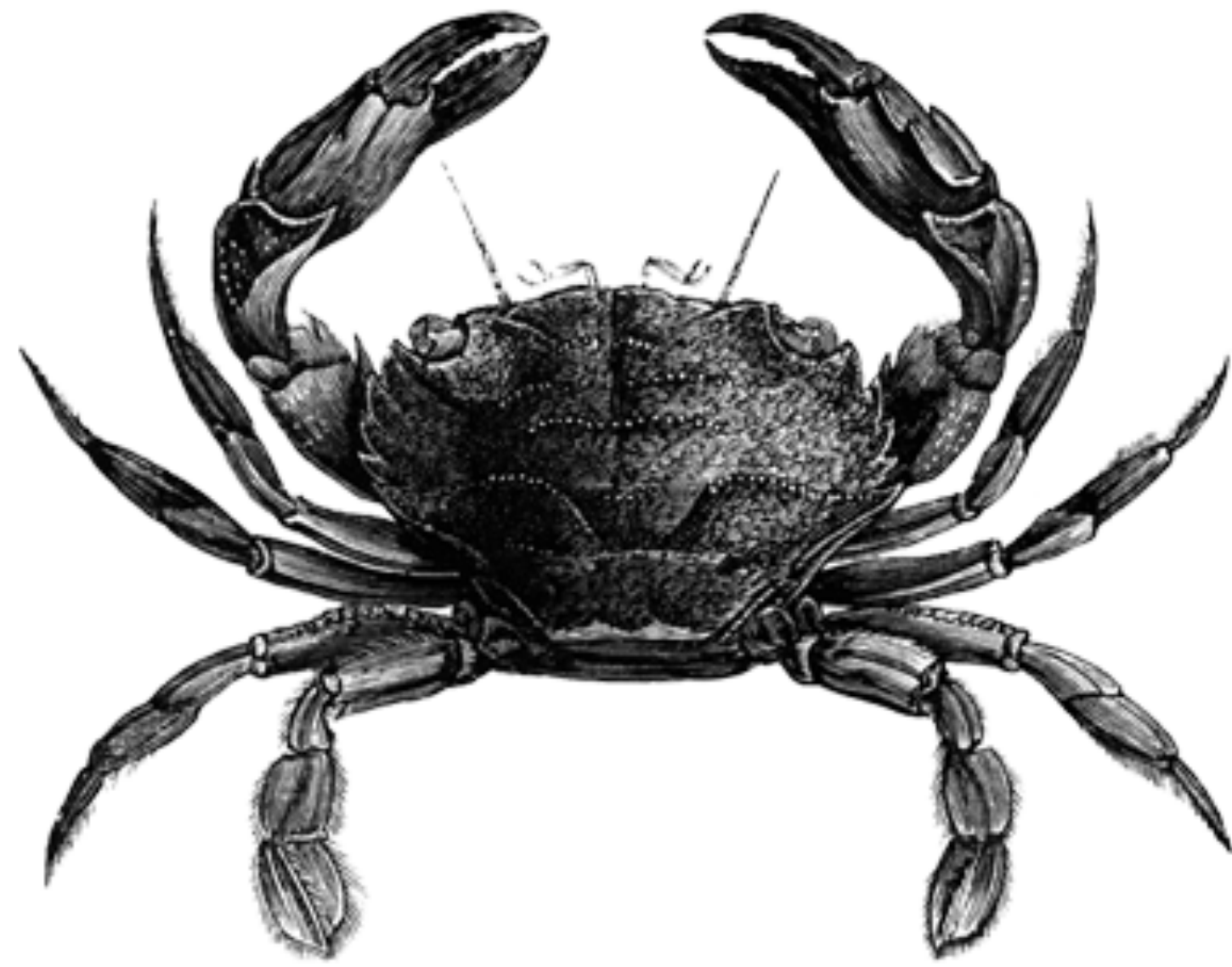
97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#37

“You’ll be seen as
old-fashioned.”



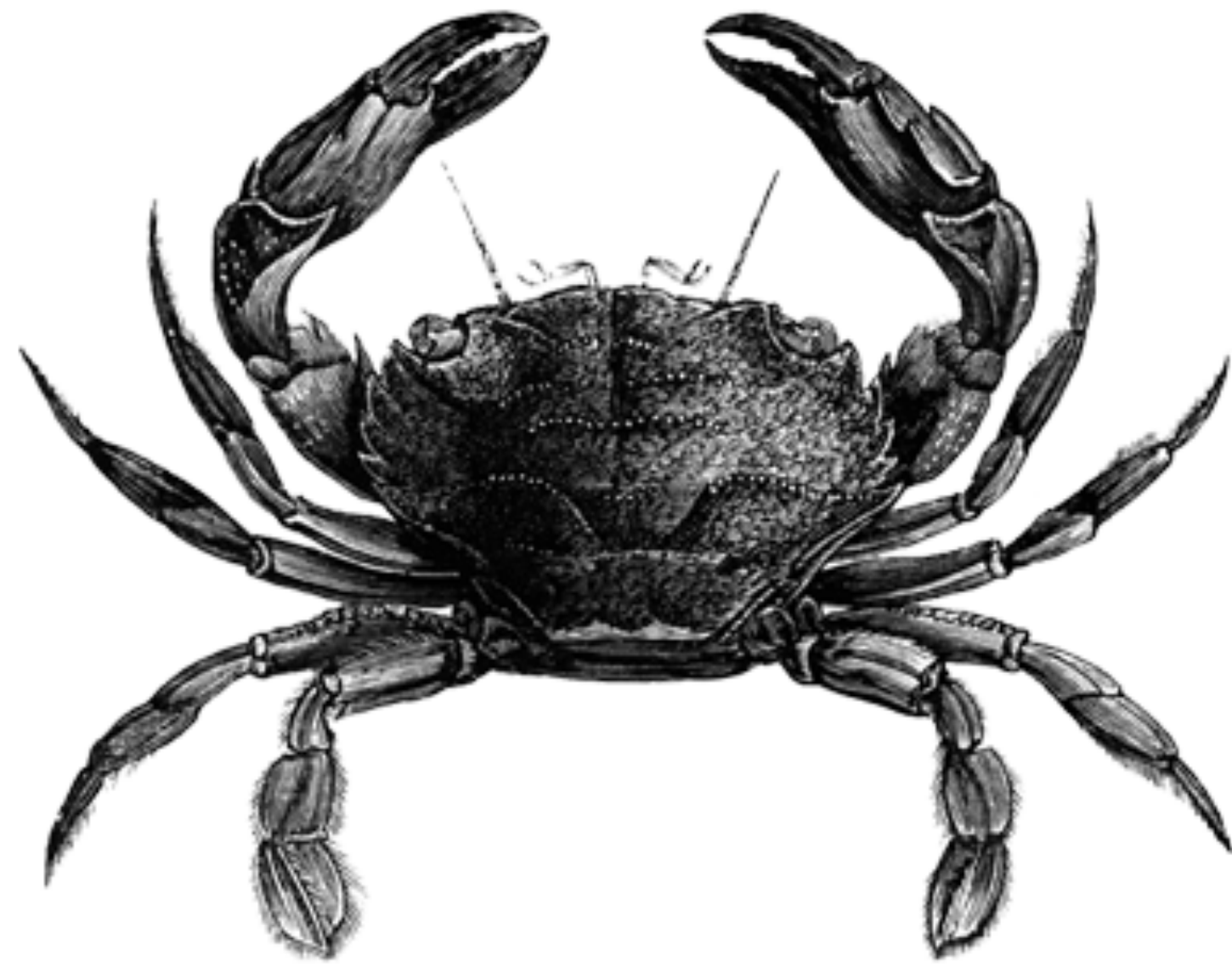
97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#80

“It’s too detailed.”



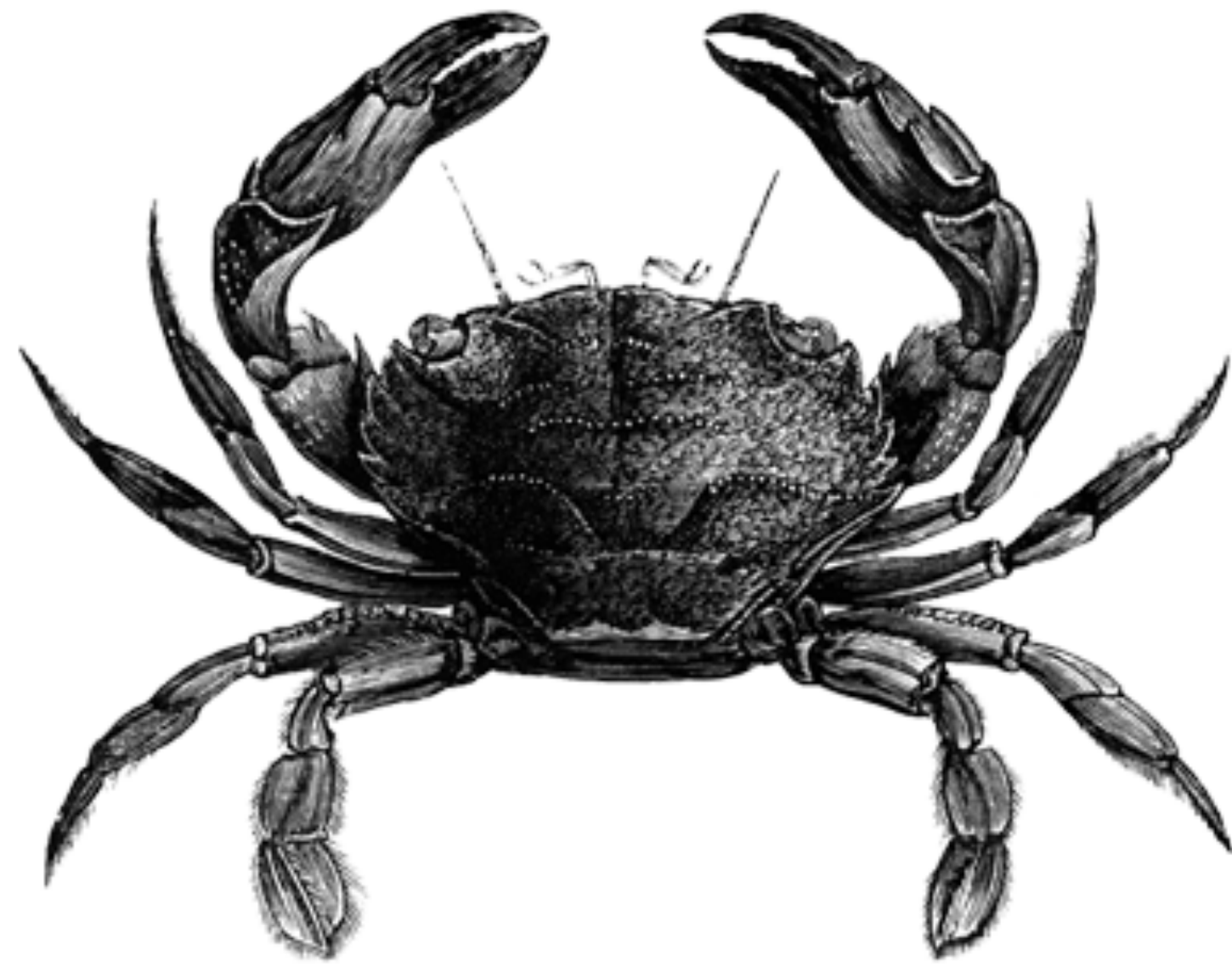
97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#46

“We don’t want to
tell developers
what to do.”



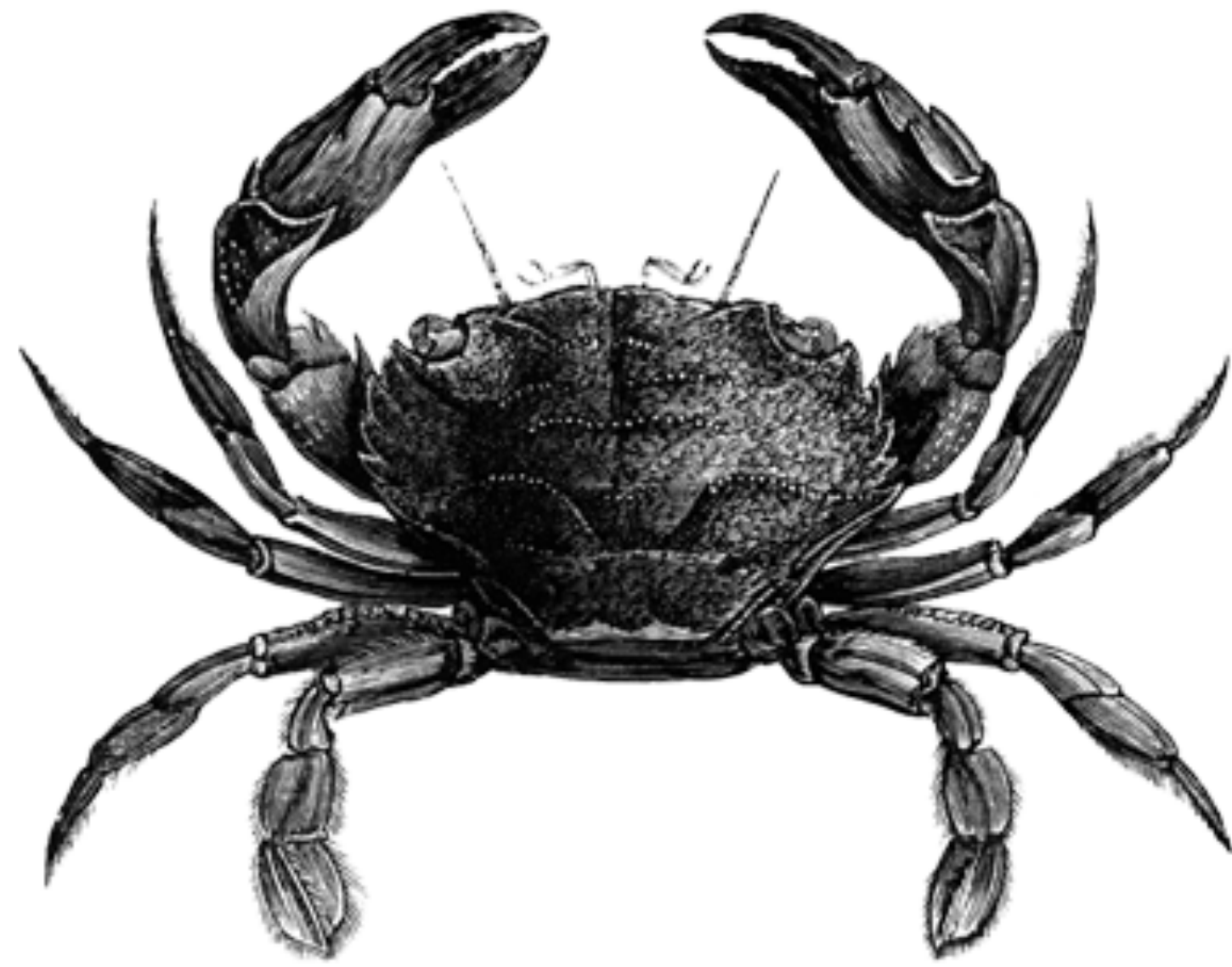
97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#66

“The tooling
sucks.”



97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#92

“It’s not expected
in agile.”

Would it be better if we used a CASE tool to lay out the design?

No, it wouldn't. The design is more readily expressed, changed, and understood when done less formally, with CRC or on the whiteboard or a bar napkin.

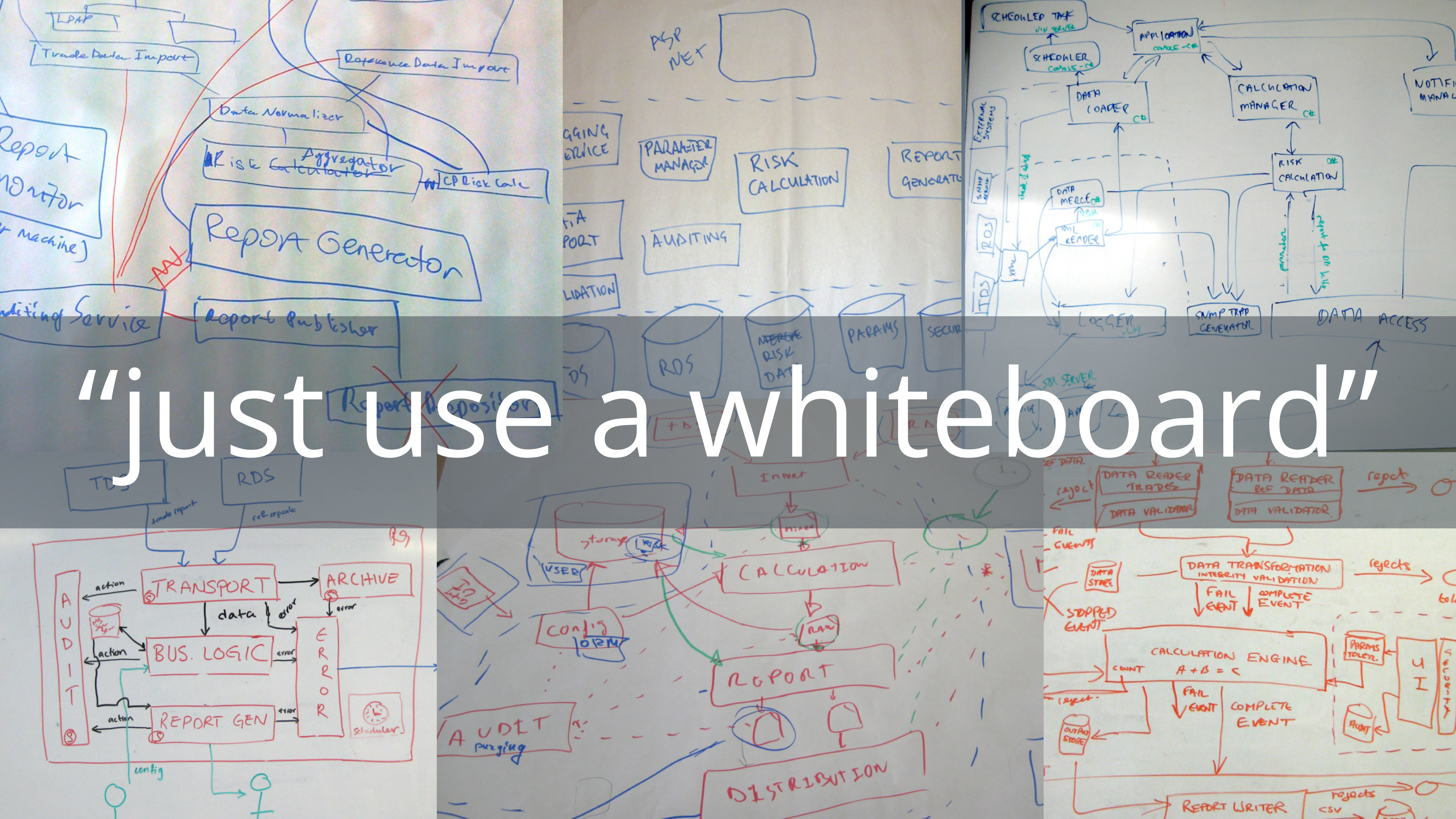
Ron Jeffries

<https://ronjeffries.com/xprog/articles/fussaboutdocumentation/>





Just use a whiteboard!



“just use a whiteboard”

What's wrong these diagrams?

The perfection game

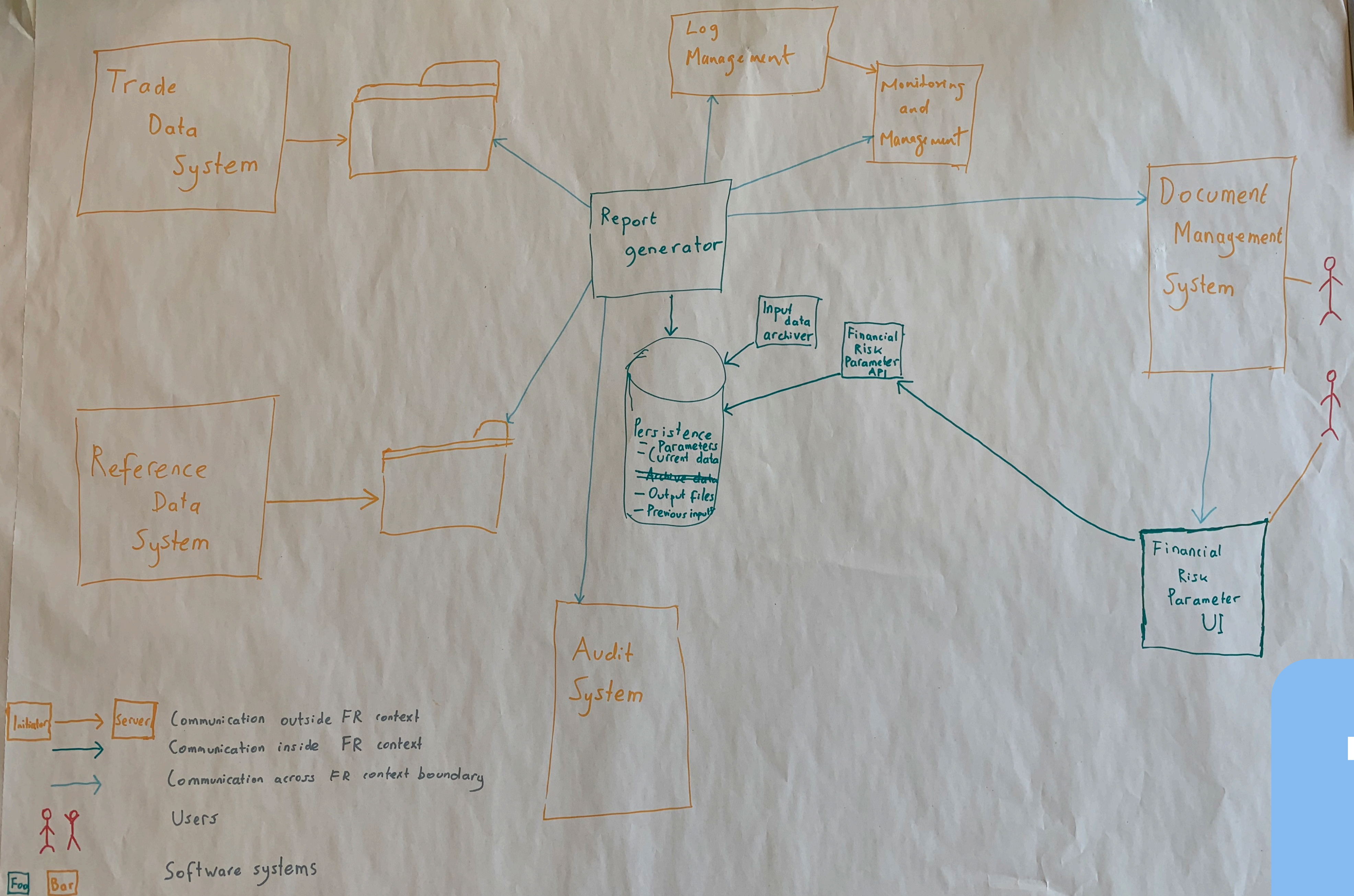
We rate the diagrams... (1-10)

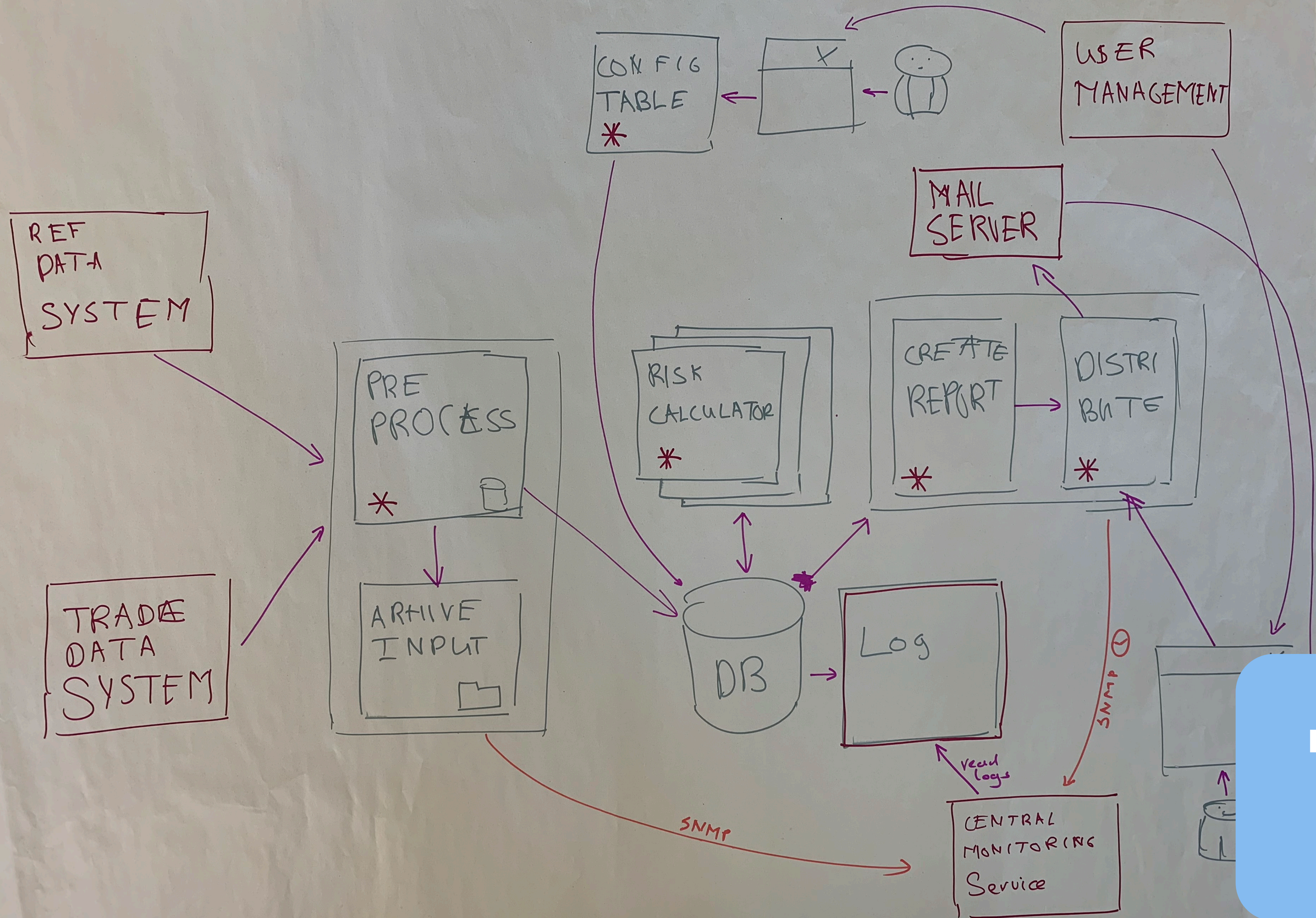
We liked...

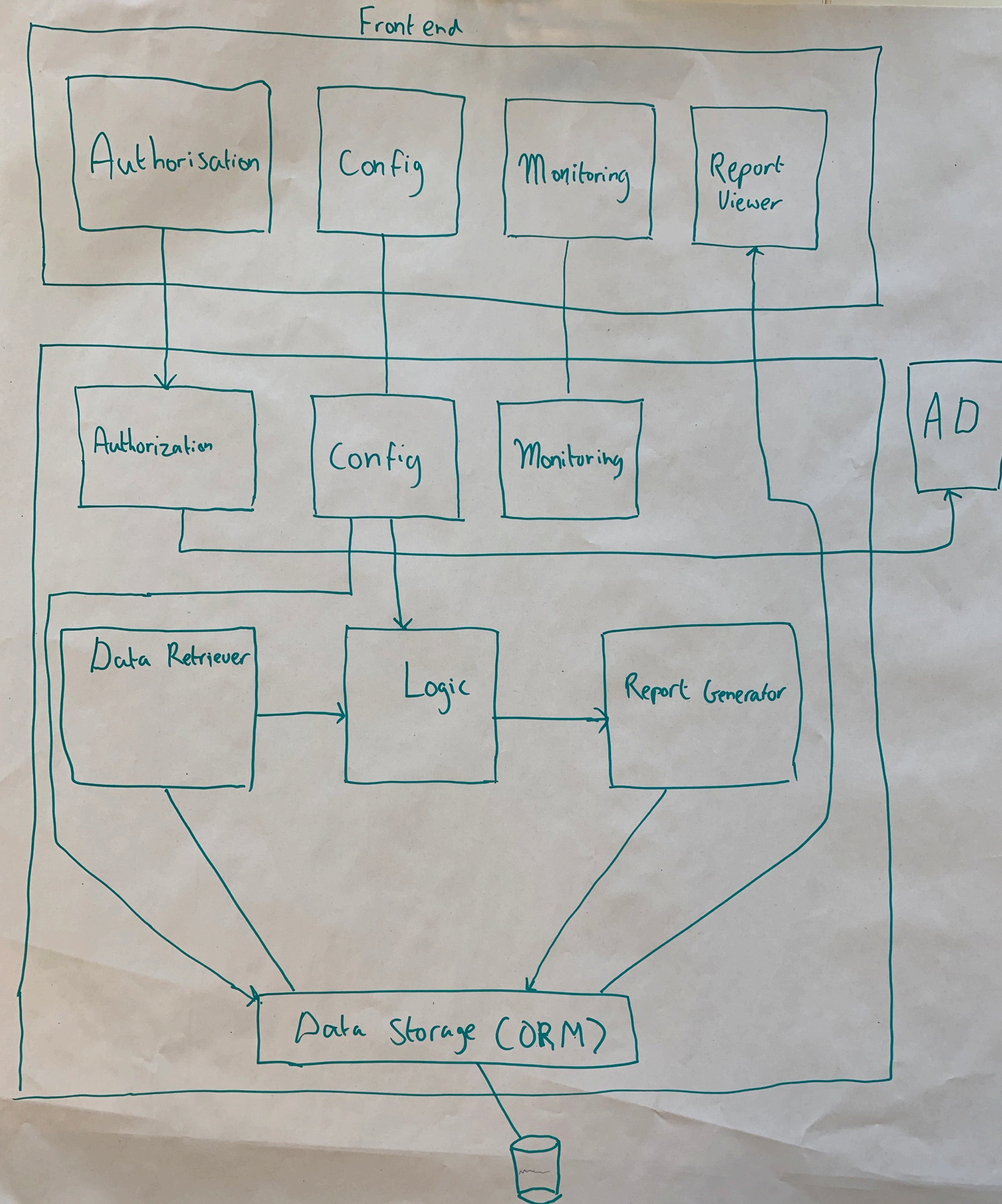
To make the diagrams perfect...



15 minutes





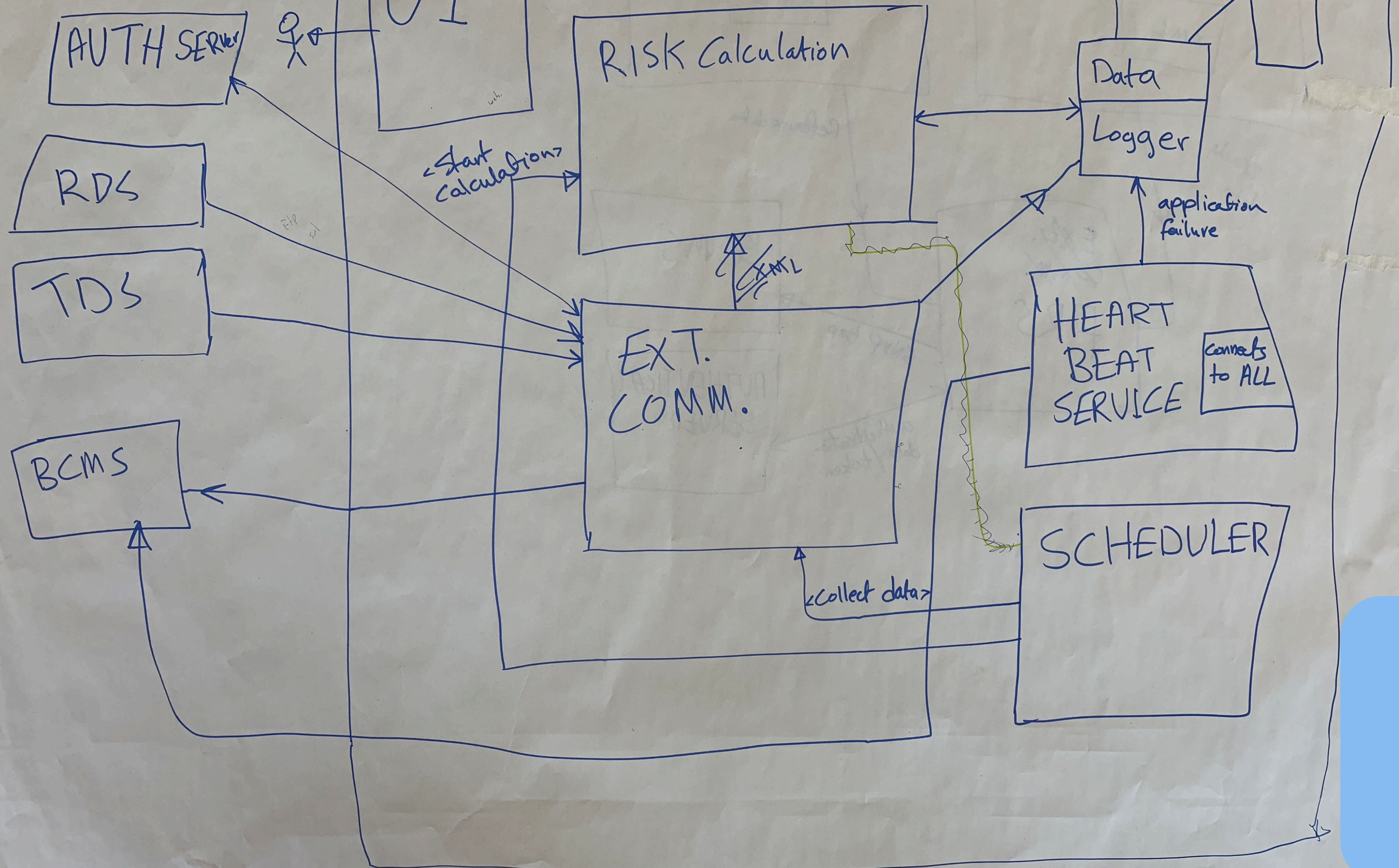


Significant decisions

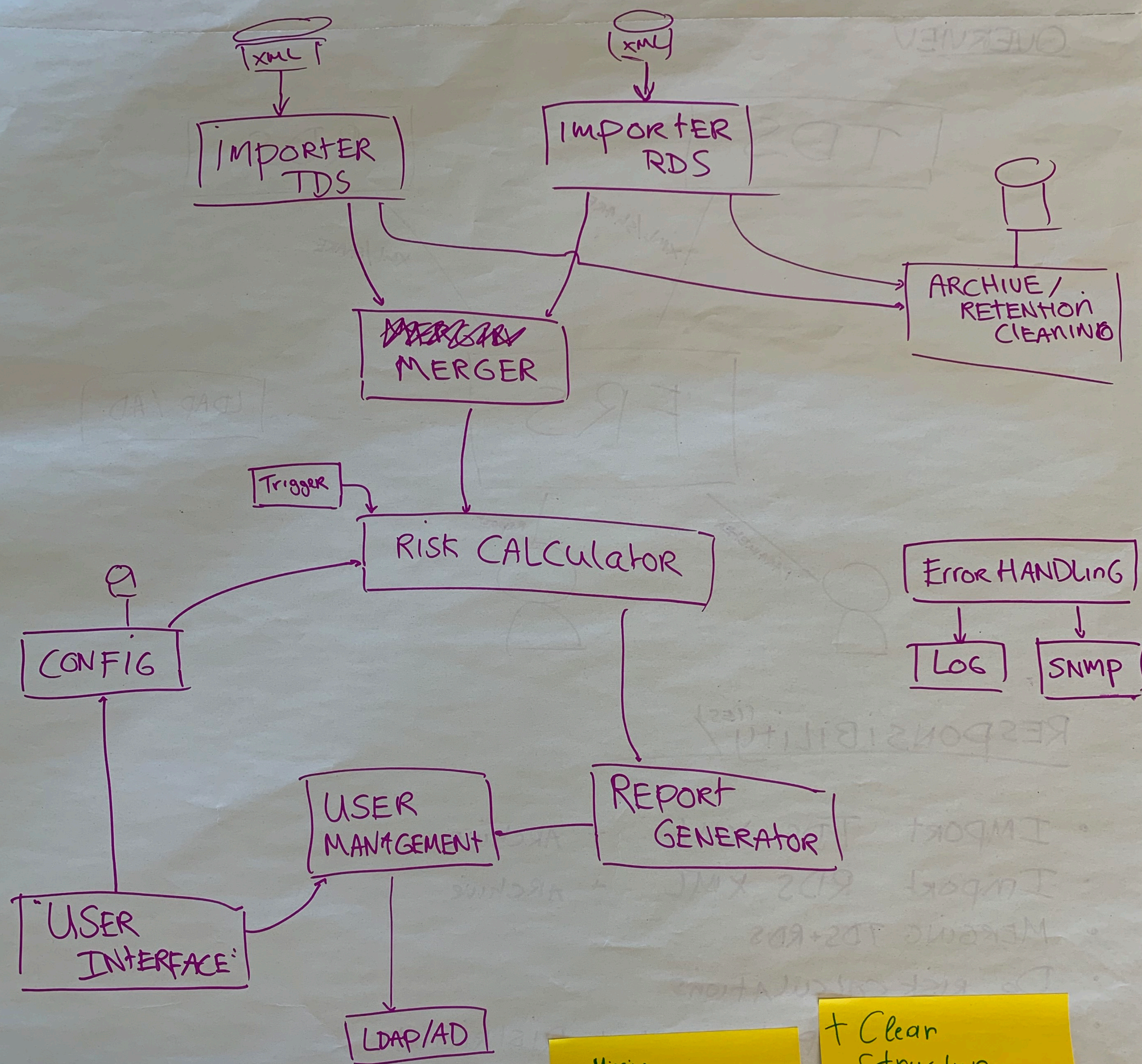
- F/E < > B/E
- Make use of OS' watchdog mechanism
- Data storage ORM-framework: Entity
- ASP.NET B/E
- Angular F/E

7

FRS



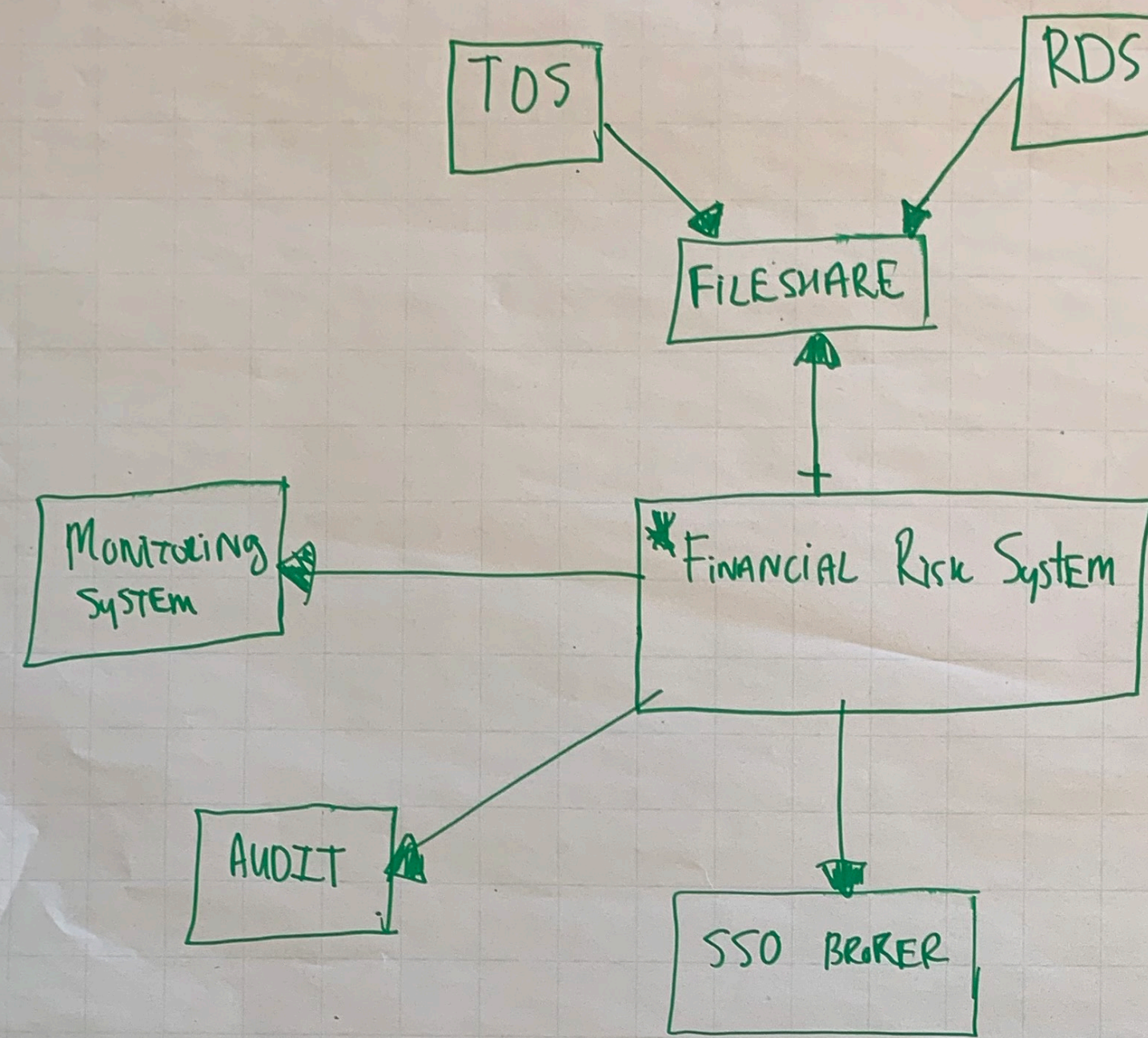
7



— Missing

+ Clear Structure



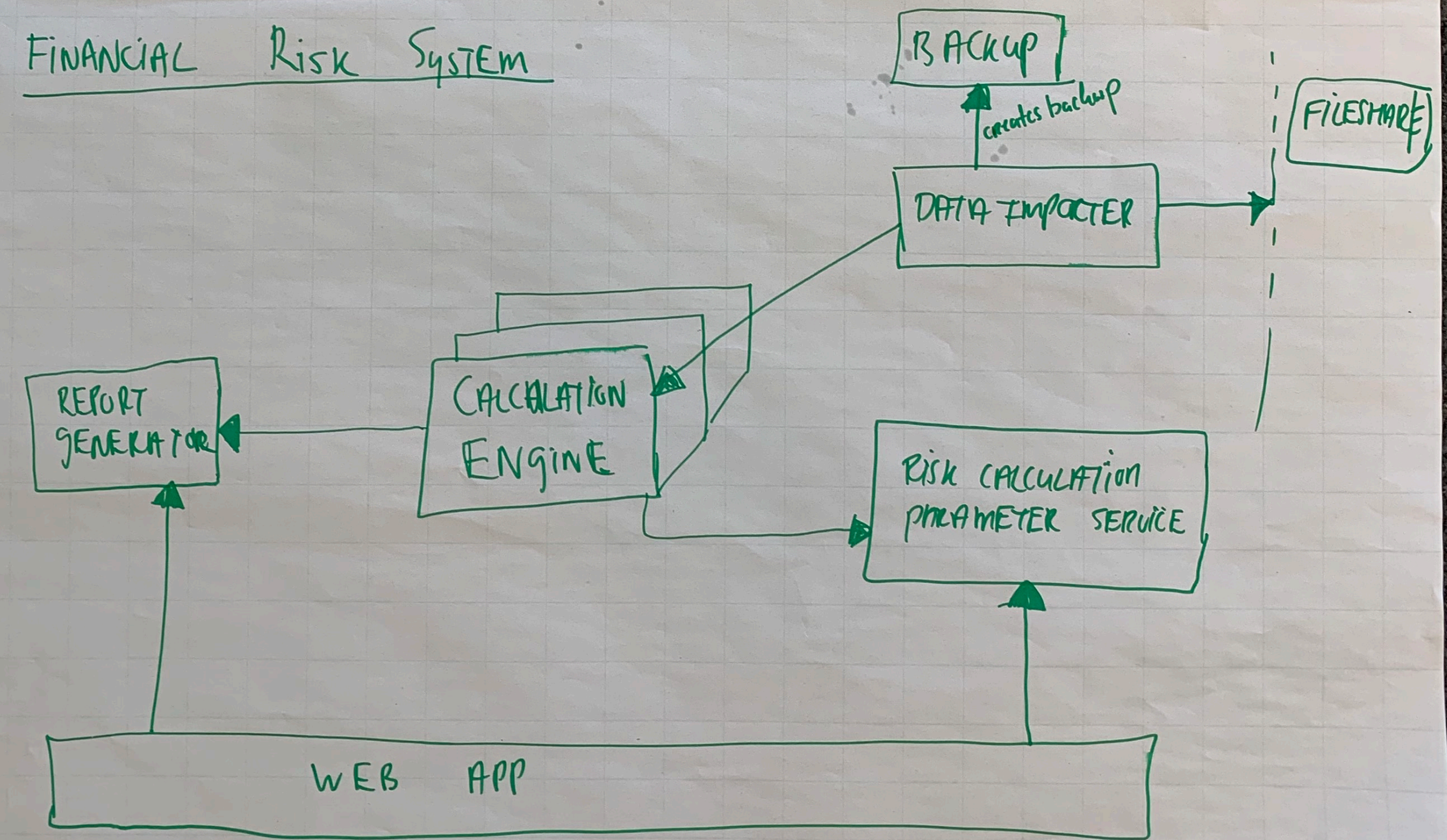


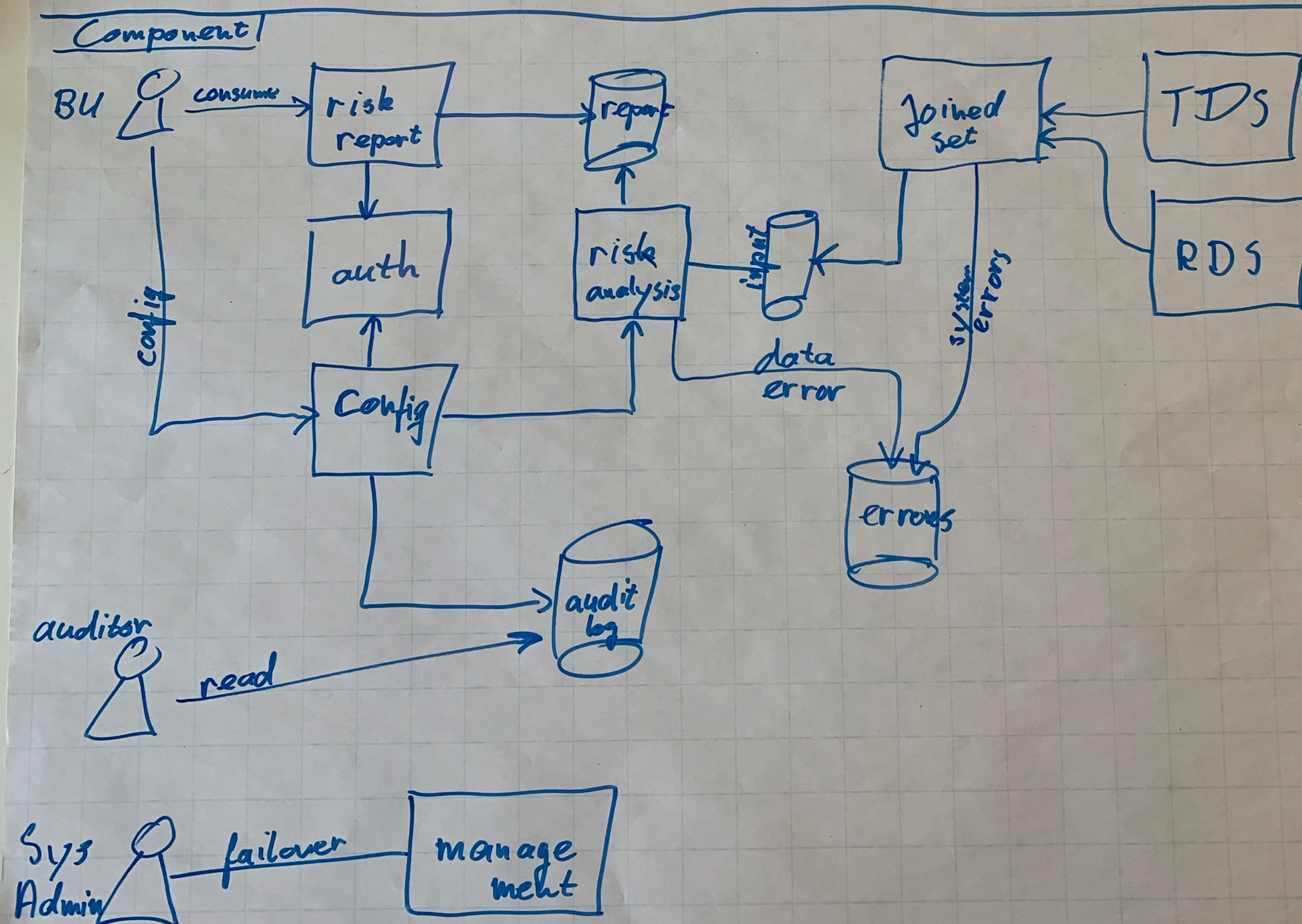
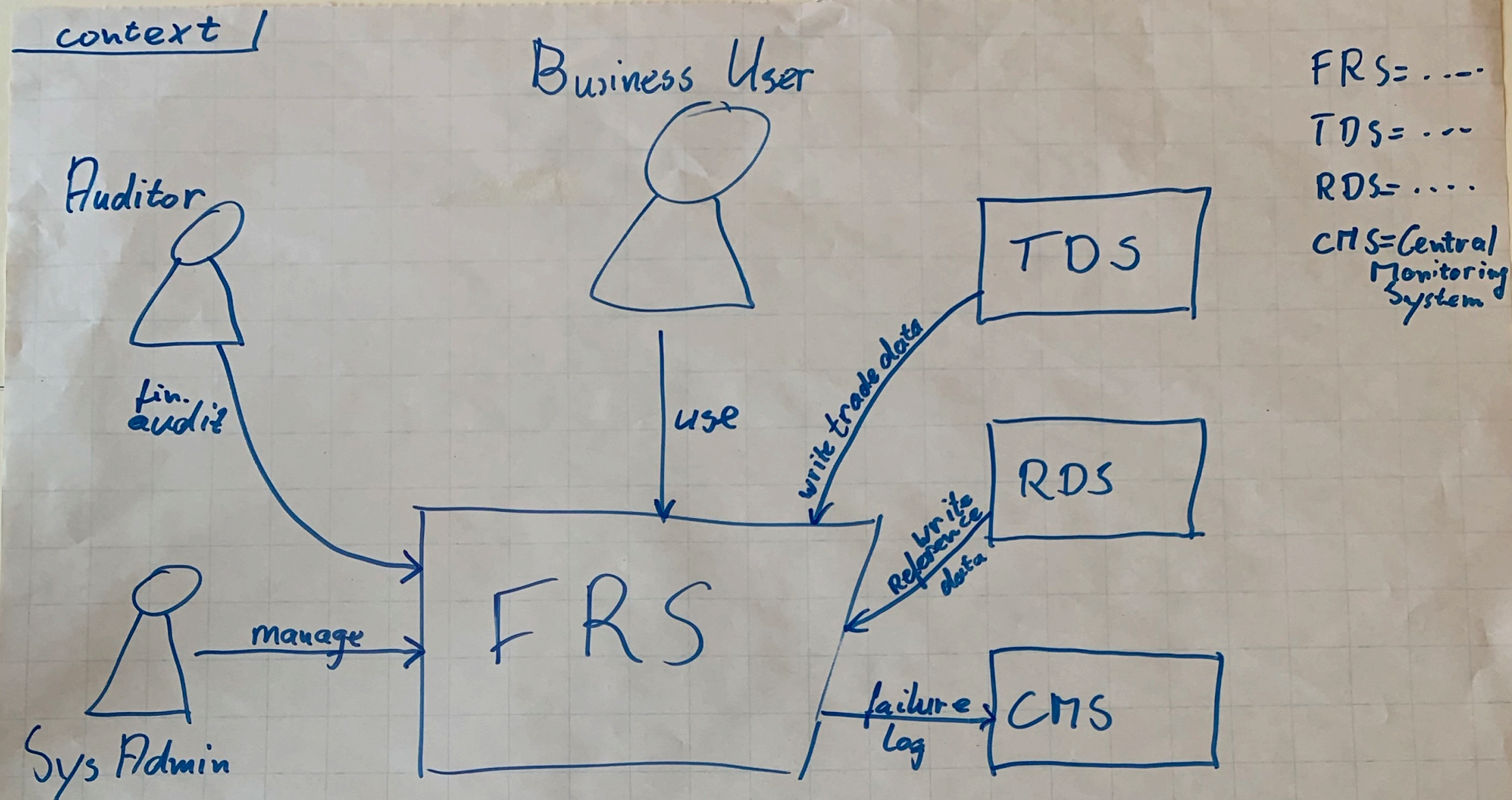
~~BACKUP~~

DECISIONS:

- * CALCULATIONS ARE A JOB TRIGGERED ON SCHEDULE
- * EXECUTE CALCULATIONS IN PARALLEL FOR EACH COUNTERPARTY
- * WEBUI FOR VIEWING REPORTS AND MODIFYING RISK PARAMETERS.
- * AUTHENTICATE AND AUTHORIZE USERS BASED ON SSO
- * SINGLE POINT OF ENTRY

* FINANCIAL Risk SYSTEM







7/10

Swap and review your diagrams

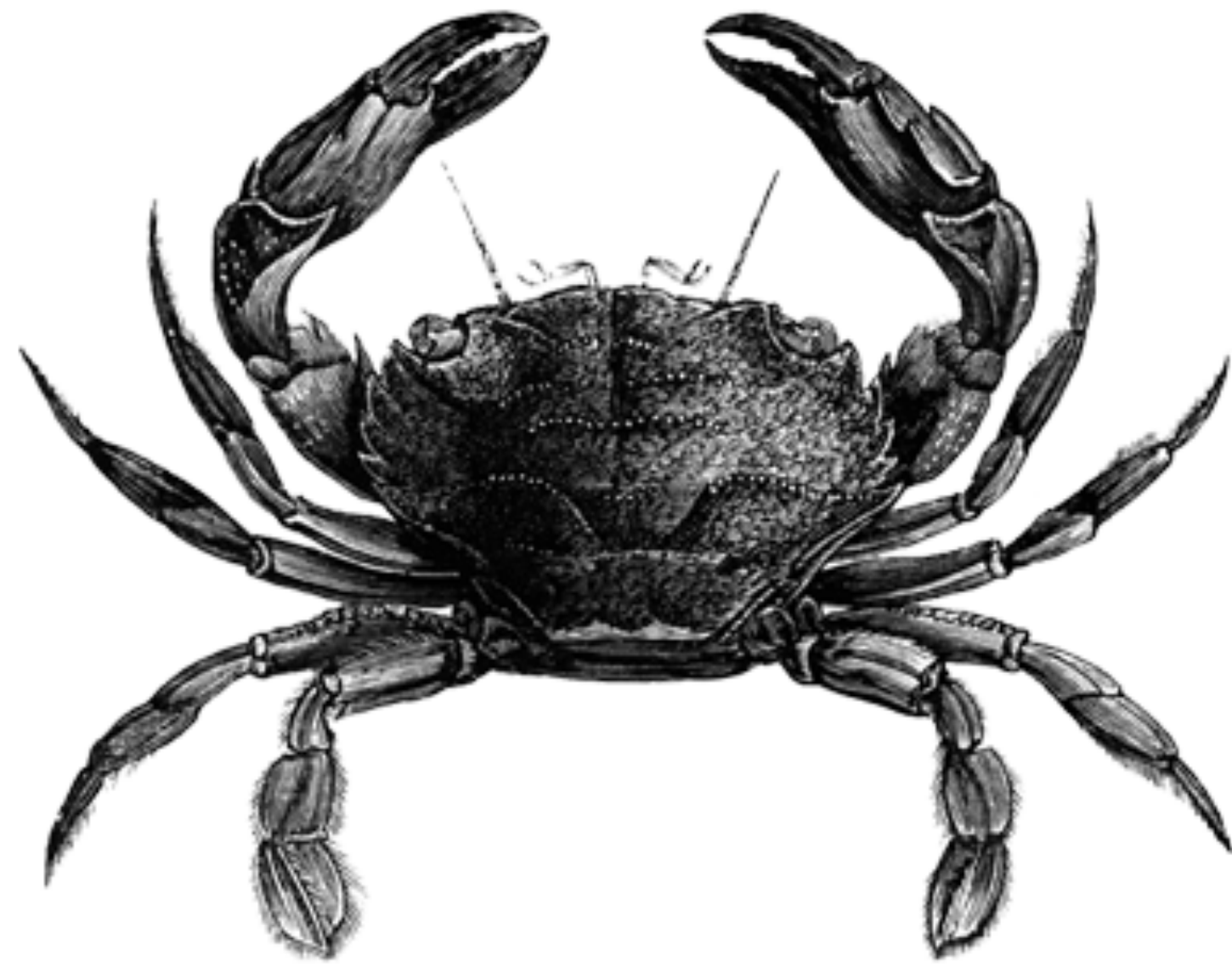
1. Do the solutions satisfy the architectural drivers?
2. If you were the bank, would you buy this solution?





It's impossible to
answer those questions

If you can't see and understand
a solution, you can't evaluate it



97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#97

“The value is
in the
conversation.”

Now don't get me wrong (again). **You may well need some nicely formatted UML for your project**, or you may need to print out Javadoc when you distribute your code to other users, or you may need to document the requirements for management or as part of a contract. If and when you really need these things, then by all means you should do them. **But inside your collocated Whole Team, you most probably will not need them, because the information you need will be communicated through the more effective medium of conversation.**

Ron Jeffries

<https://ronjeffries.com/xprog/articles/fussaboutdocumentation/>

They are all excellent, as long as there is a conversation about their meaning and intent. It's the accompanying conversation that matters.

“the value is in the conversation”
only works if you’re having
the right conversations

Superficial up front design



97 Questions To
Ask Before Coding

O RLY?

Assam Shun

#4

“Is this a
microservices
architecture?”

#73

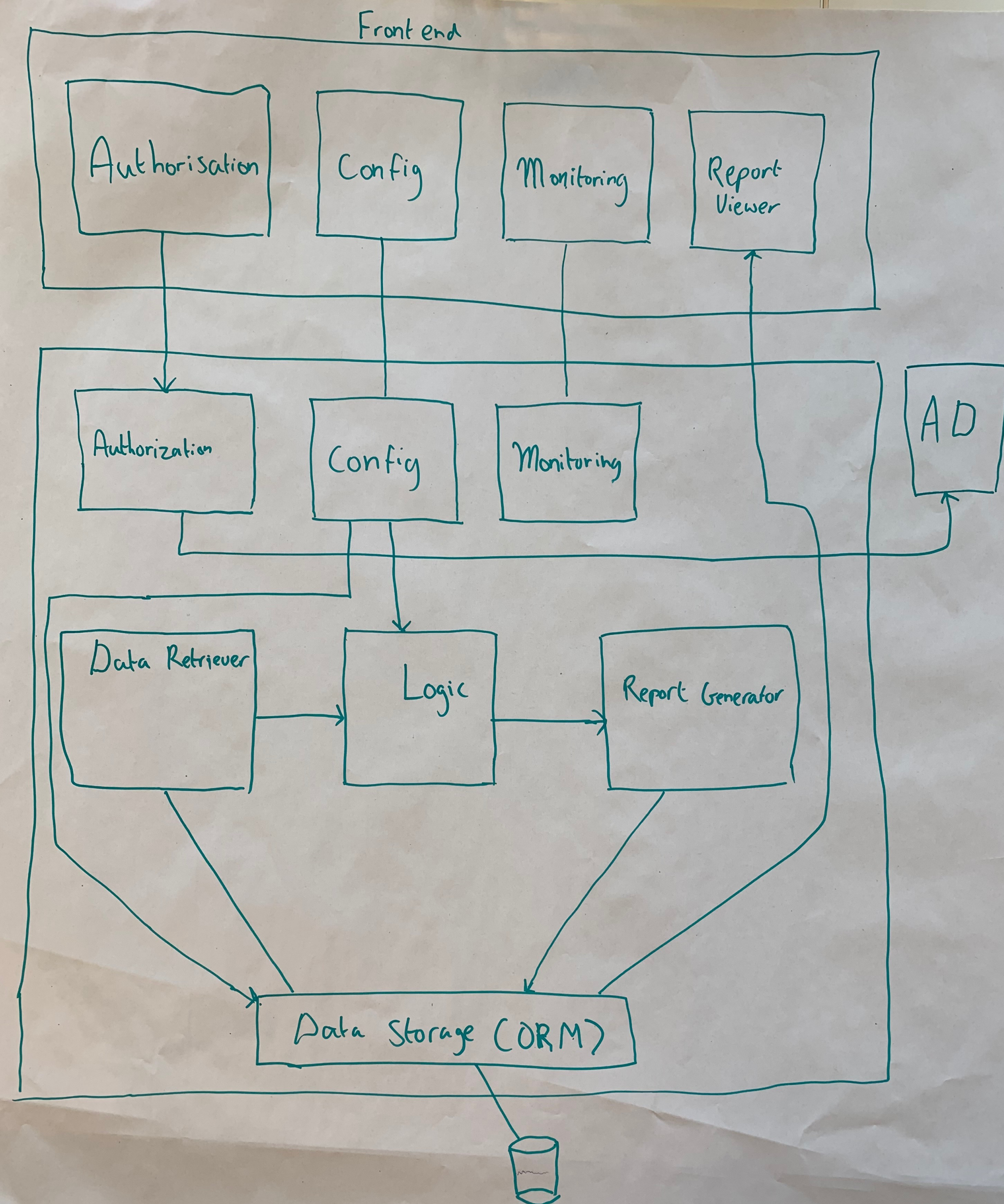
“Why is the ORM
directly connected
to the Angular
front-end?”



97 Questions To
Ask Before Coding

O RLY?

Assam Shun



Significant decisions

- F/E < > B/E
- Make use of OS' watchdog mechanism
- Data storage ORM framework: Entity
- ASP .NET B/E
- Angular F/E

Why is the ORM directly connected to the Angular front-end?



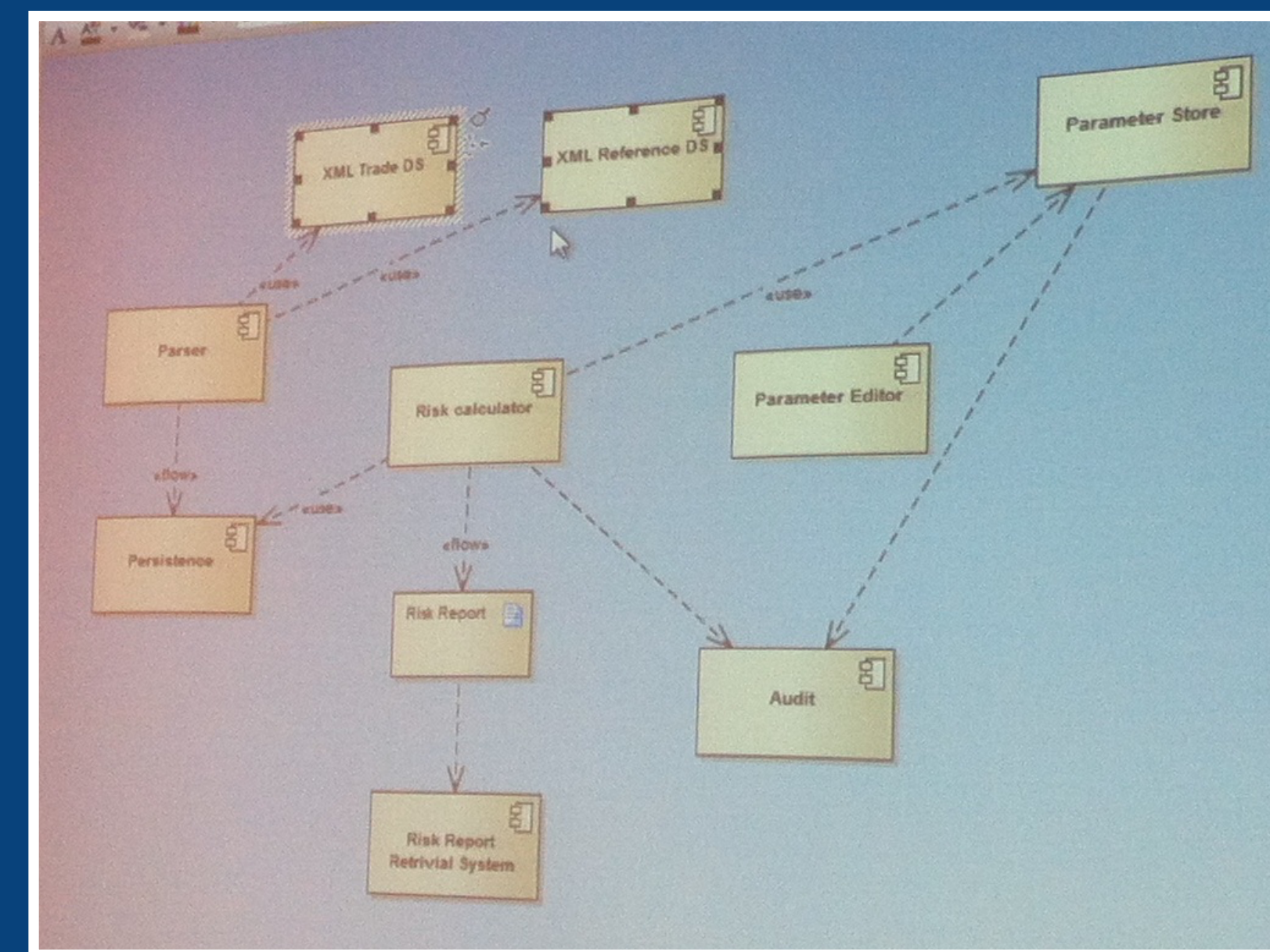
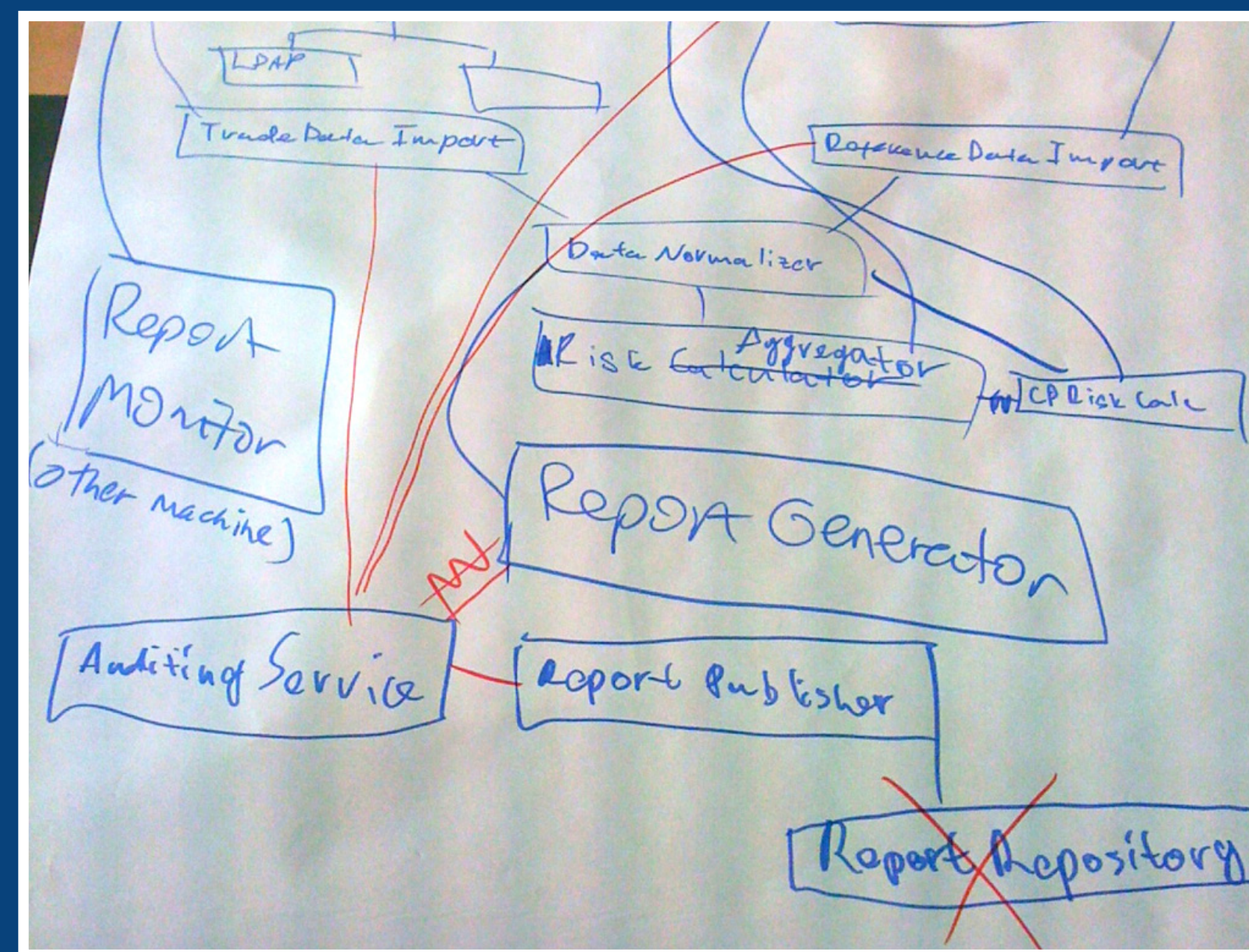
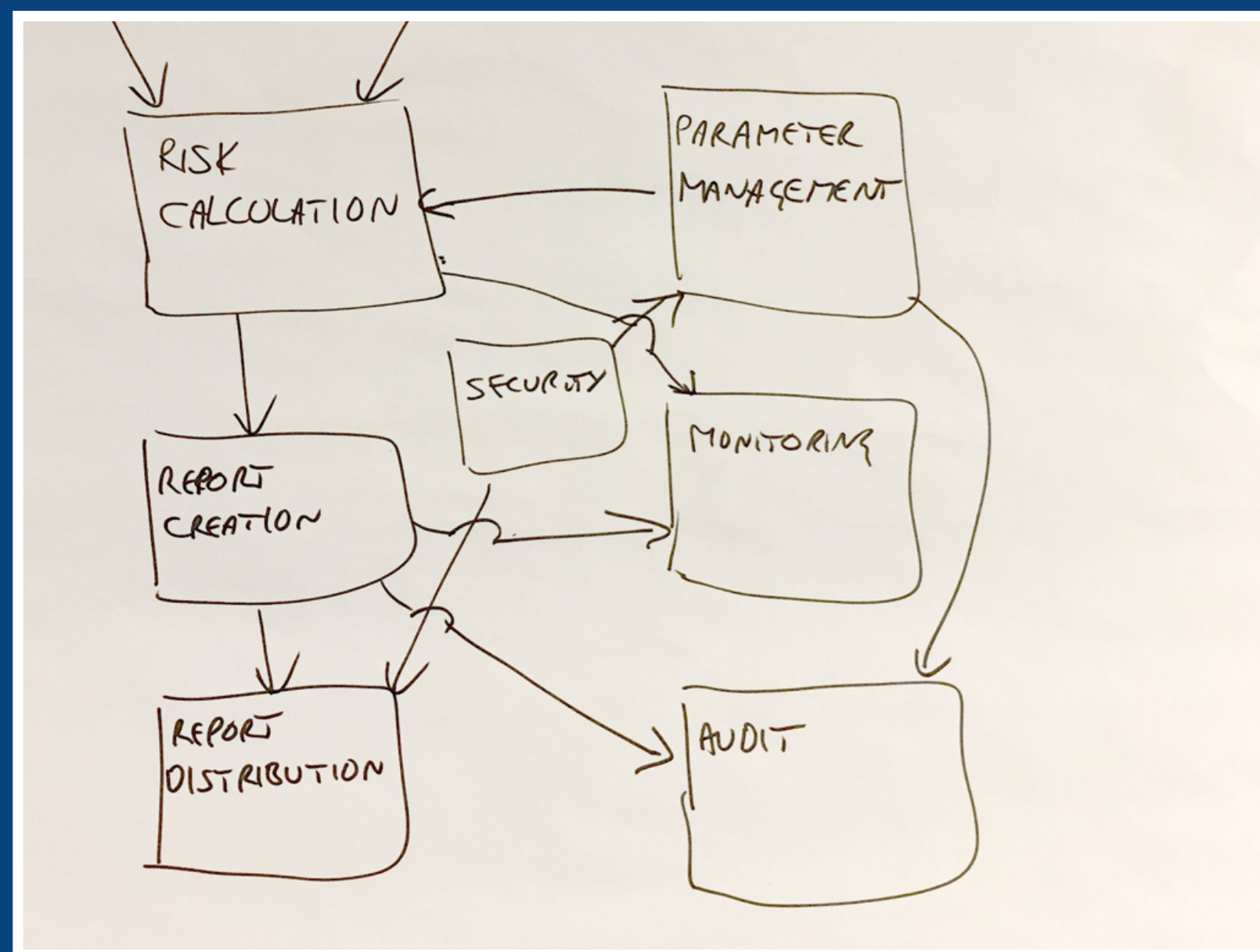
97 Questions To
Ask Before Coding

O RLY?

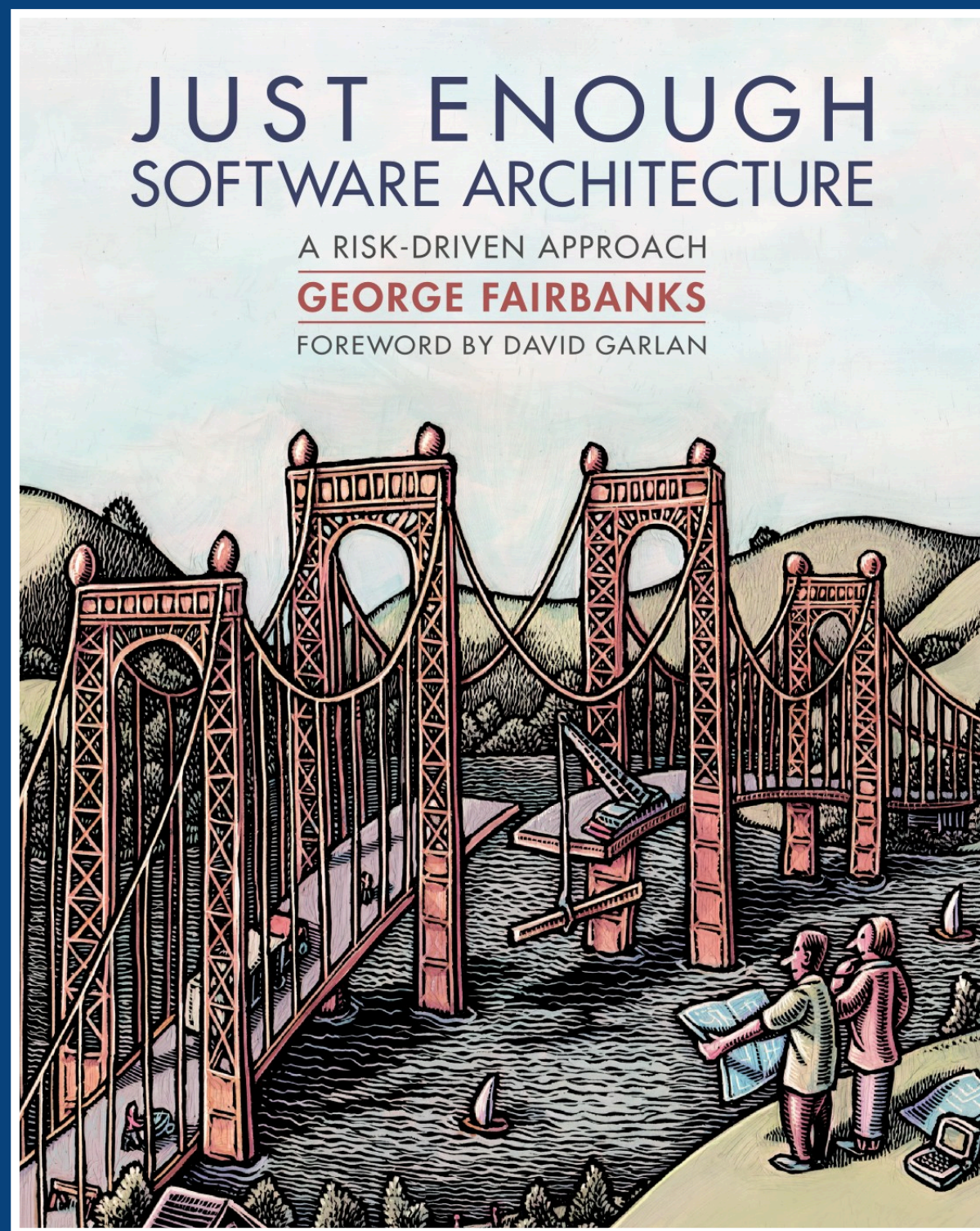
Assam Shun

#76

“Is the web UI
getting data from
Amazon S3?”

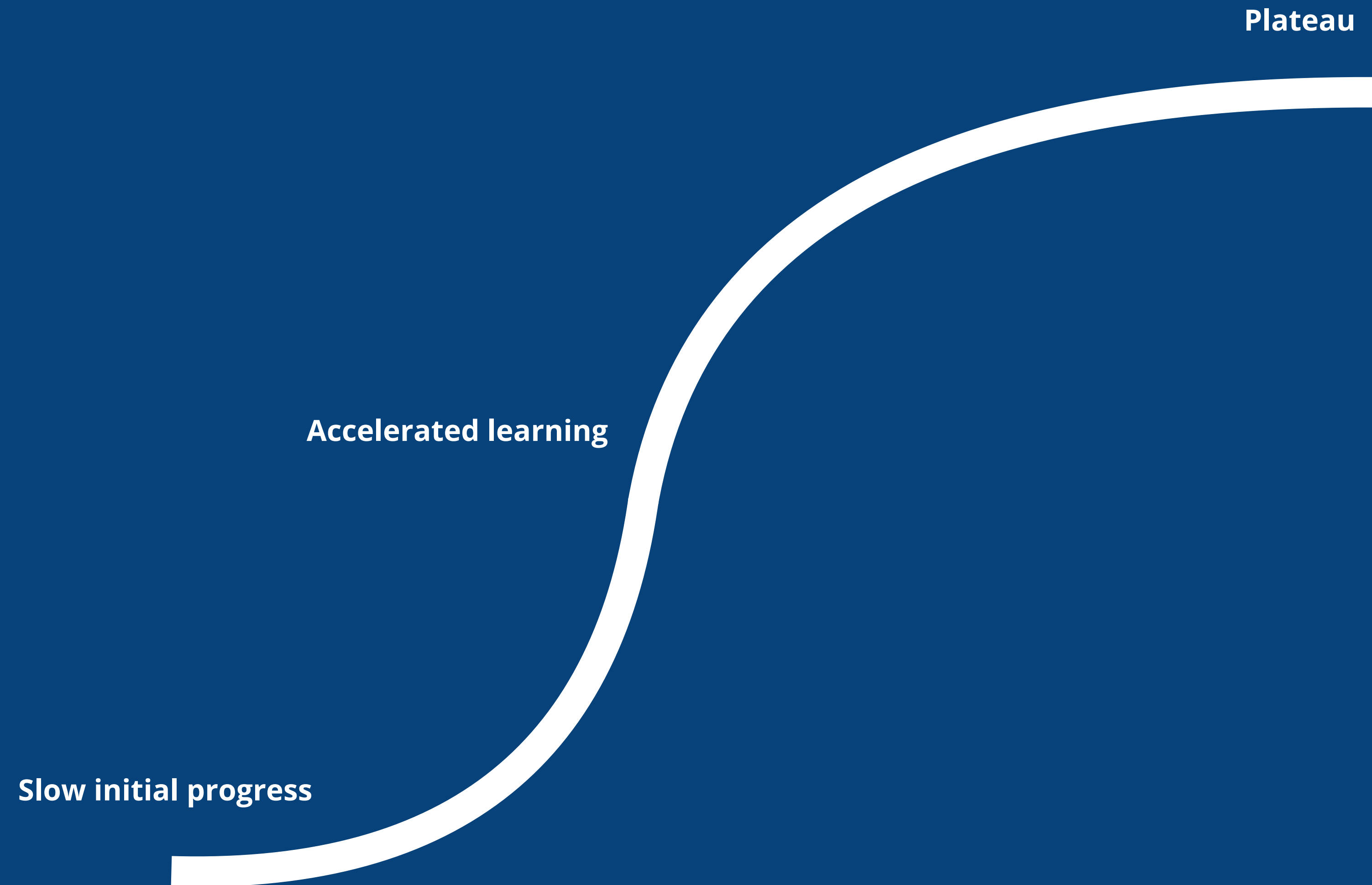


If you don't **engage** in the problem, you end up with a very simplified and superficial view of the solution



A good architecture rarely
happens through
architecture-indifferent design

Part of the design activity is about
discovering “unknown unknowns”

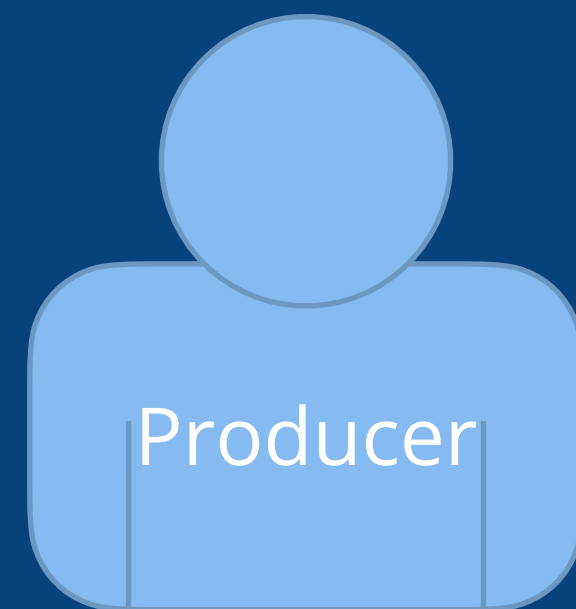


The typical s-curve of learning

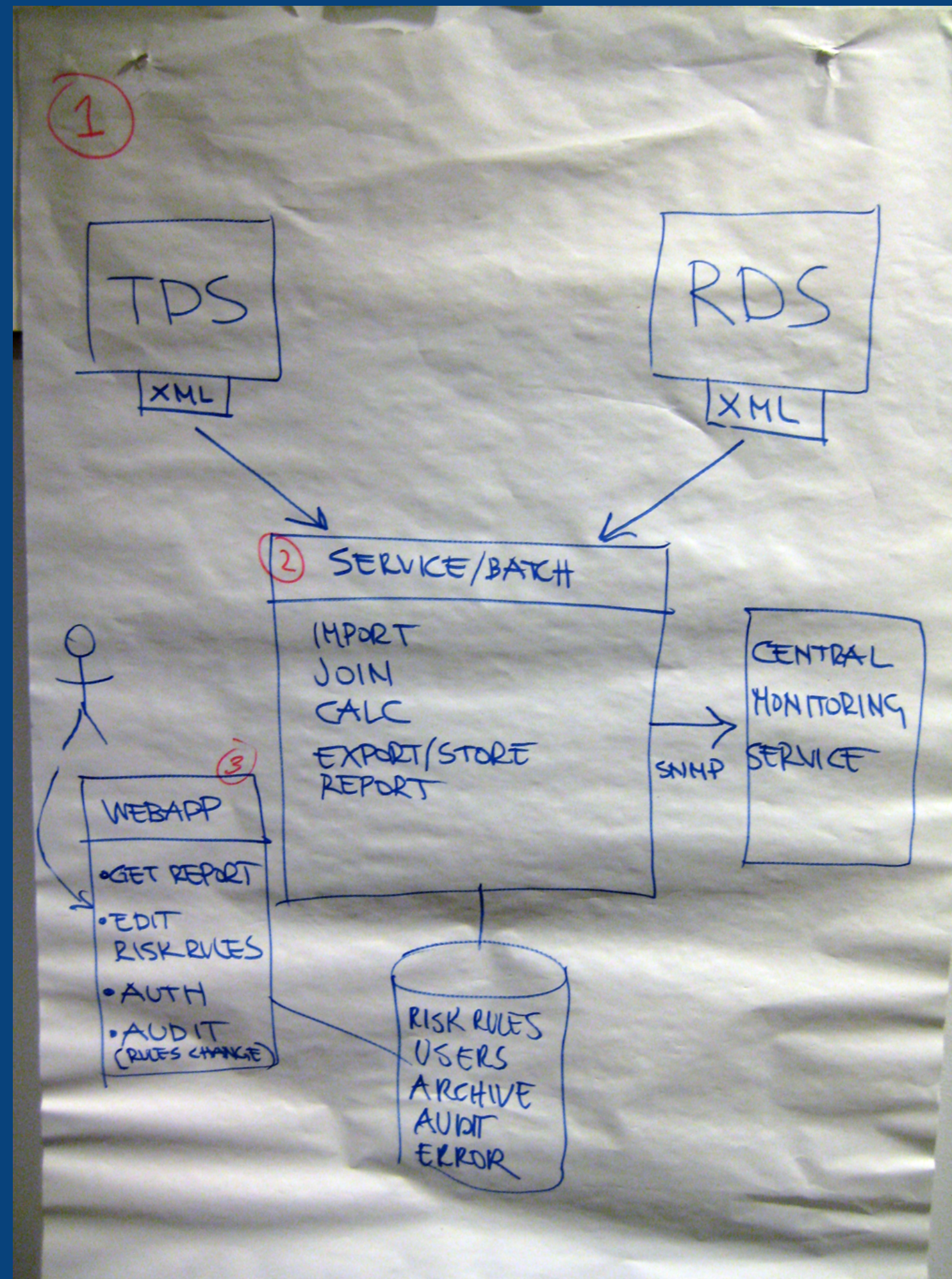
Technology Decisions

The producer-consumer conflict of software architecture diagrams

I don't want to put technology choices on the diagrams...



Producer



I wish these diagrams included technology choices...



Consumer



97 Ways to Defer Technology Decisions

O RLY?

Ima Duncoaden

#8

“We don’t solutionize.”

#10

“Our architects
are not allowed
to do
solutioneering.”



97 Ways to Defer
Technology Decisions

O RLY?

Ima Duncoaden

#39

“We don’t want to
impose a solution
upon the
development
team.”



97 Ways to Defer
Technology Decisions

O RLY?

Ima Duncoaden



97 Ways to Defer
Technology Decisions

O RLY?

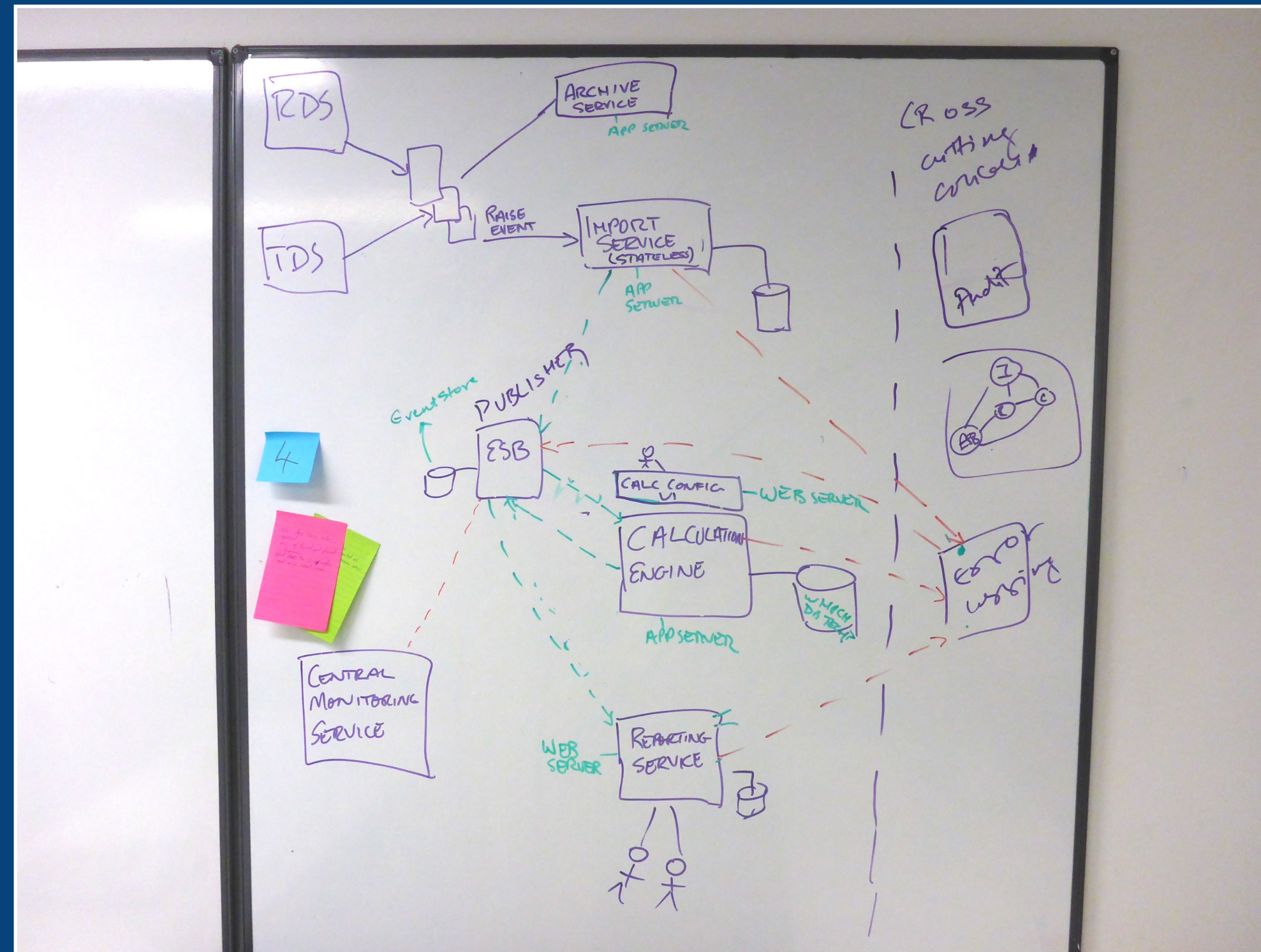
Ima Duncoaden

#42

“We’re a Java team,
so it’s obviously
a Java solution.”

How much up front design?

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



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

We're not trying to
make every decision

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

Grady Booch

Architecture

Programming languages
Technologies and platforms
Monolith, microservices or hybrid approach

Design

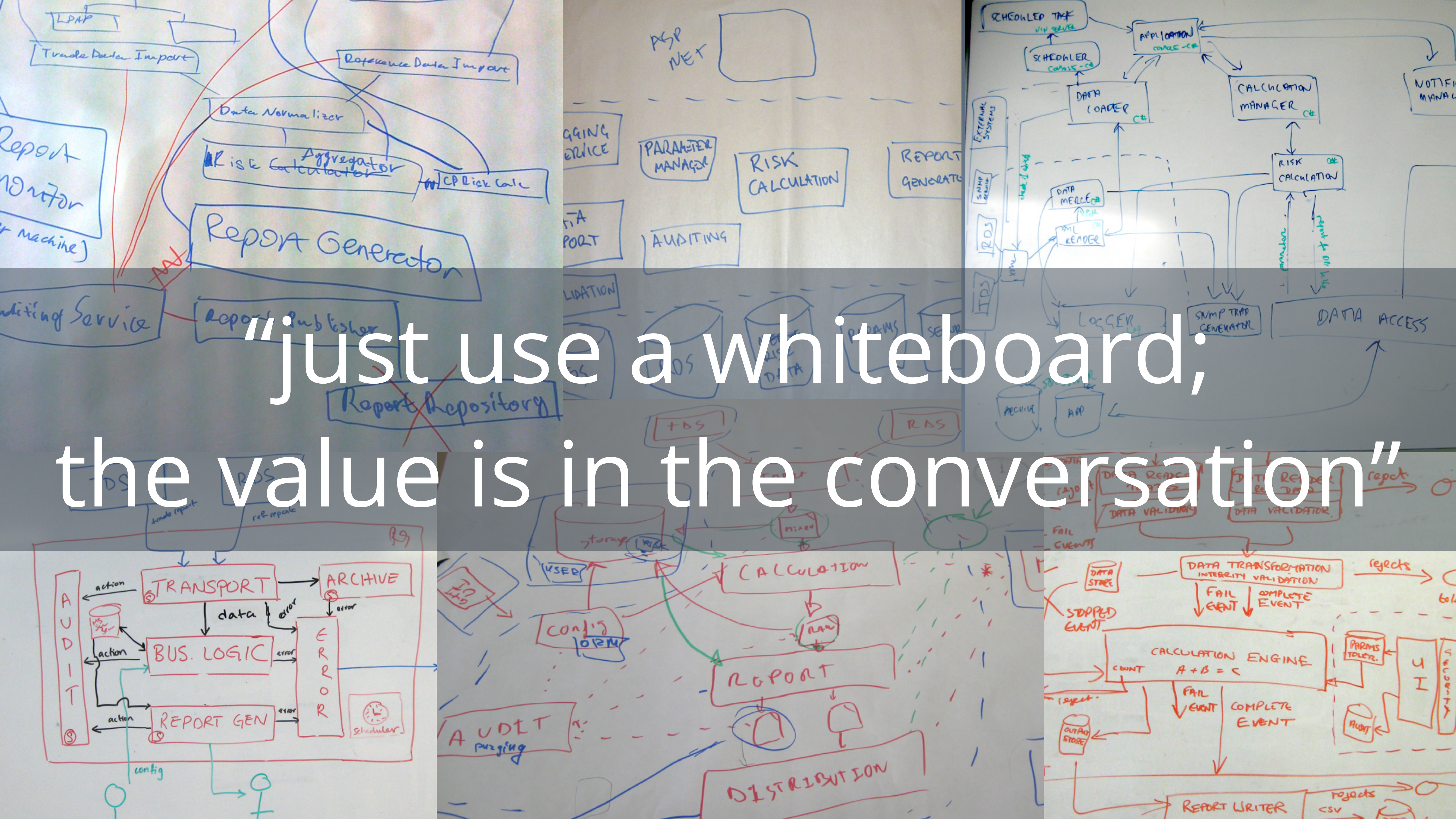
Implementation

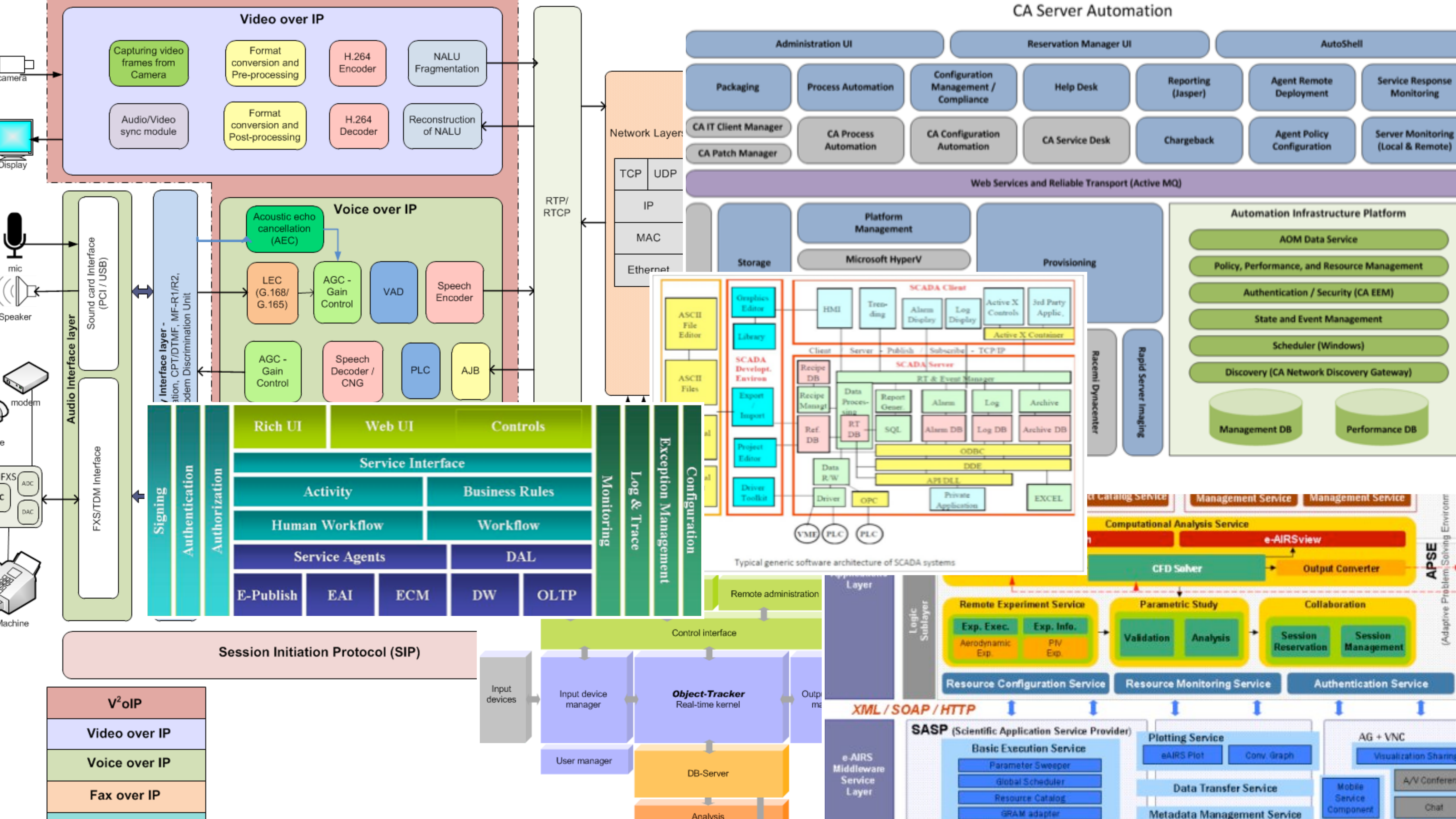
Curly braces on the same or next line
Whitespace vs tabs

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.

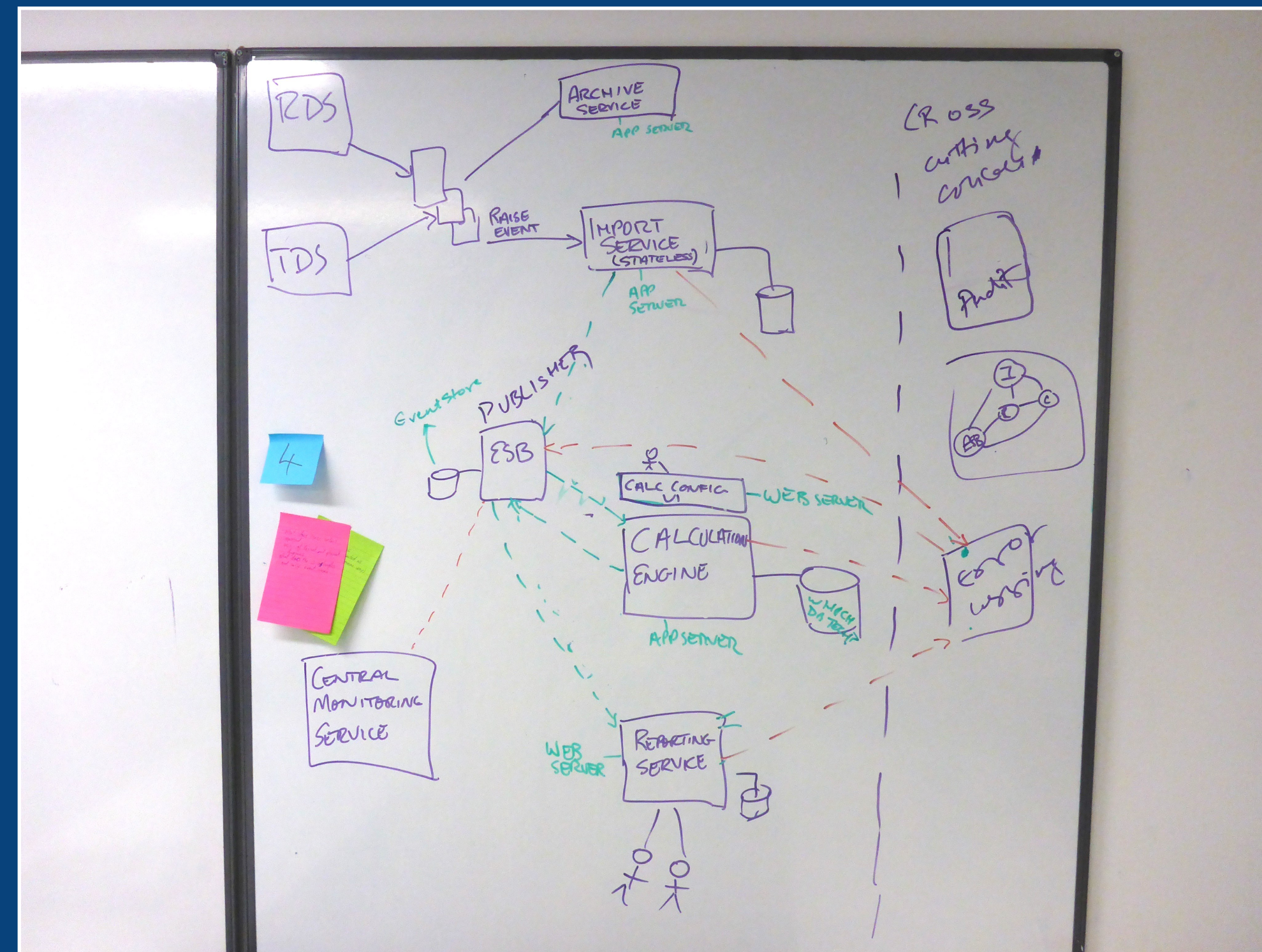
Martin Fowler

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





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



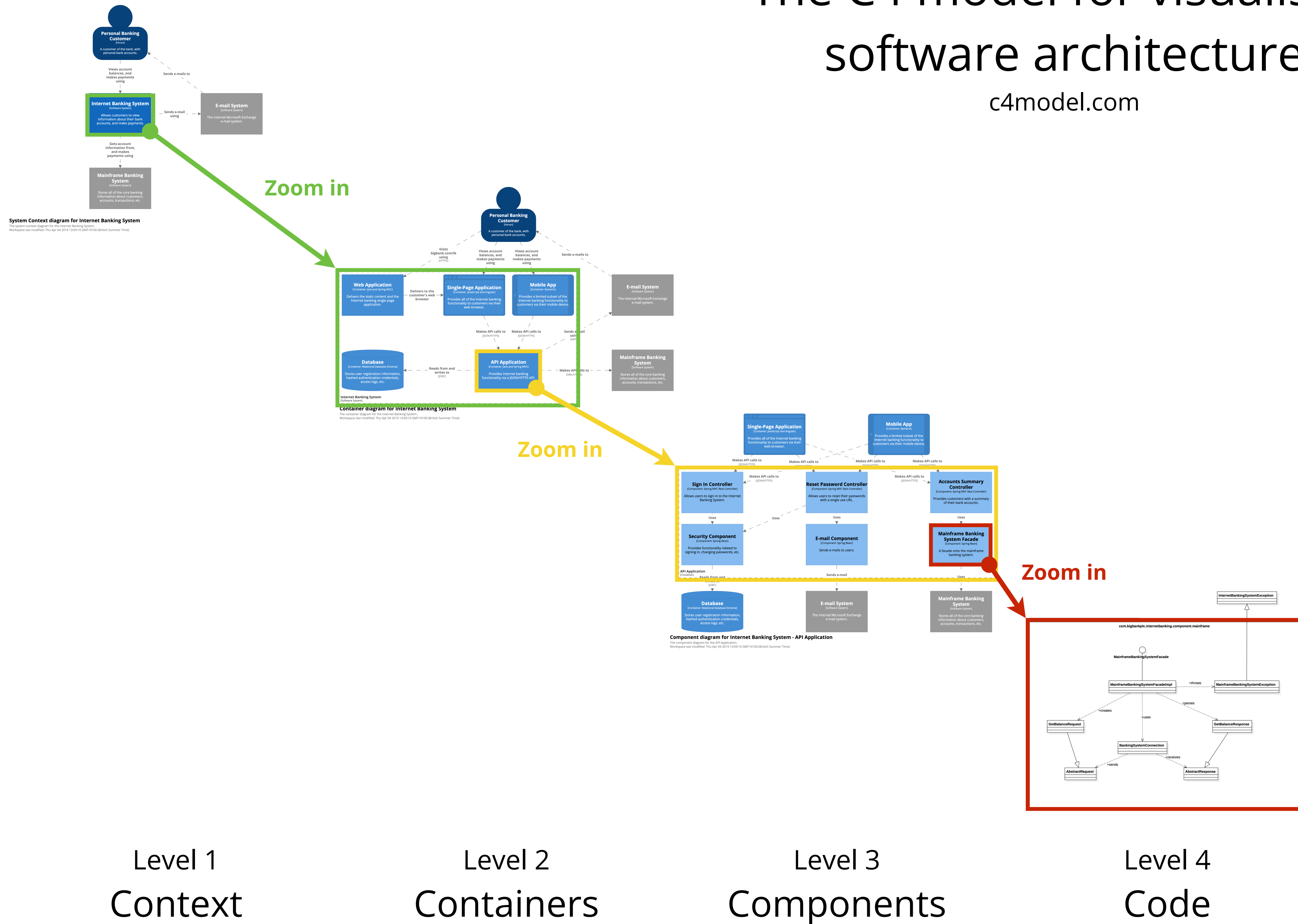
2. Is it going to **work**?

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

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

The C4 model for visualising software architecture

c4model.com

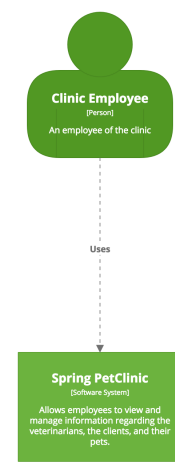
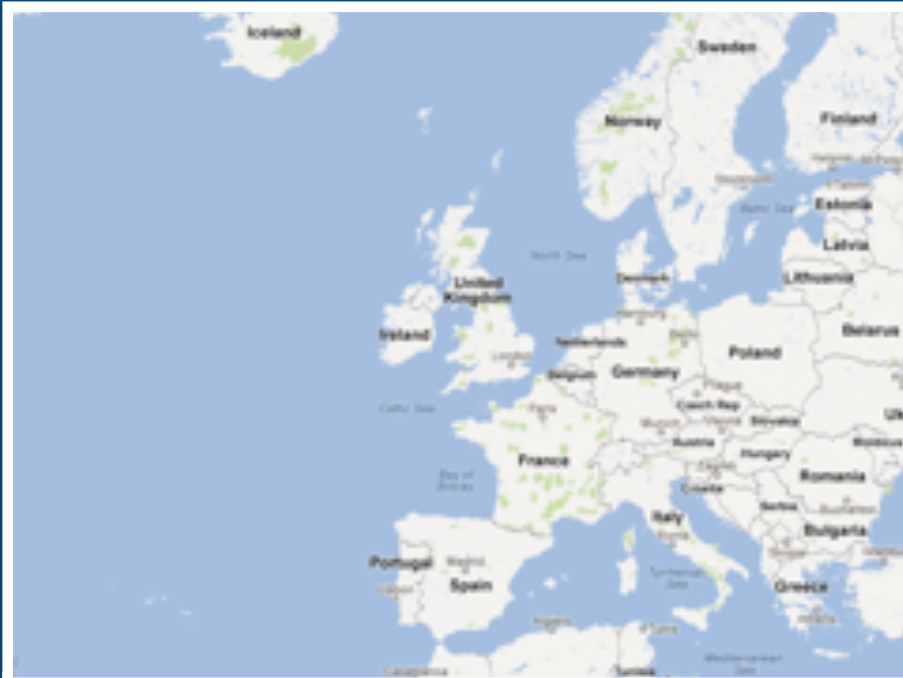


Level 1
Context

Level 2
Containers

Level 3
Components

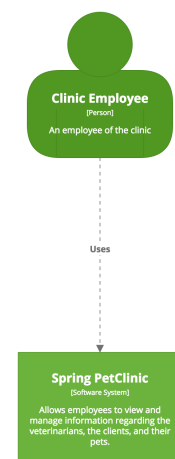
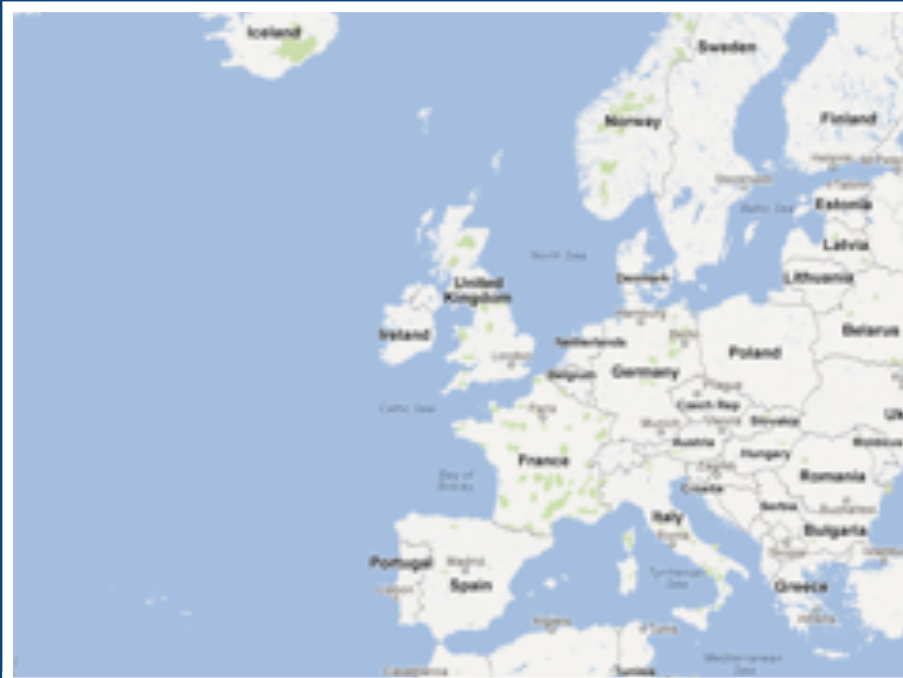
Level 4
Code



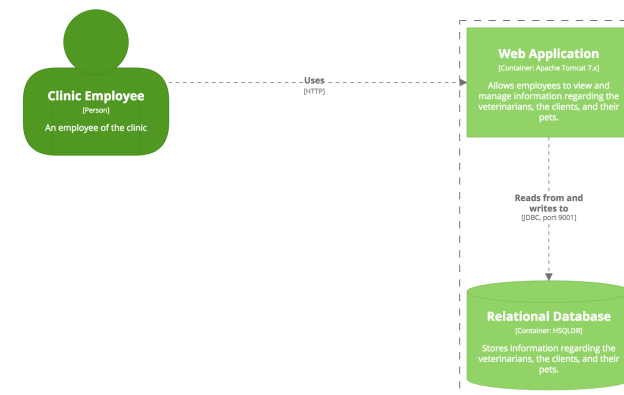
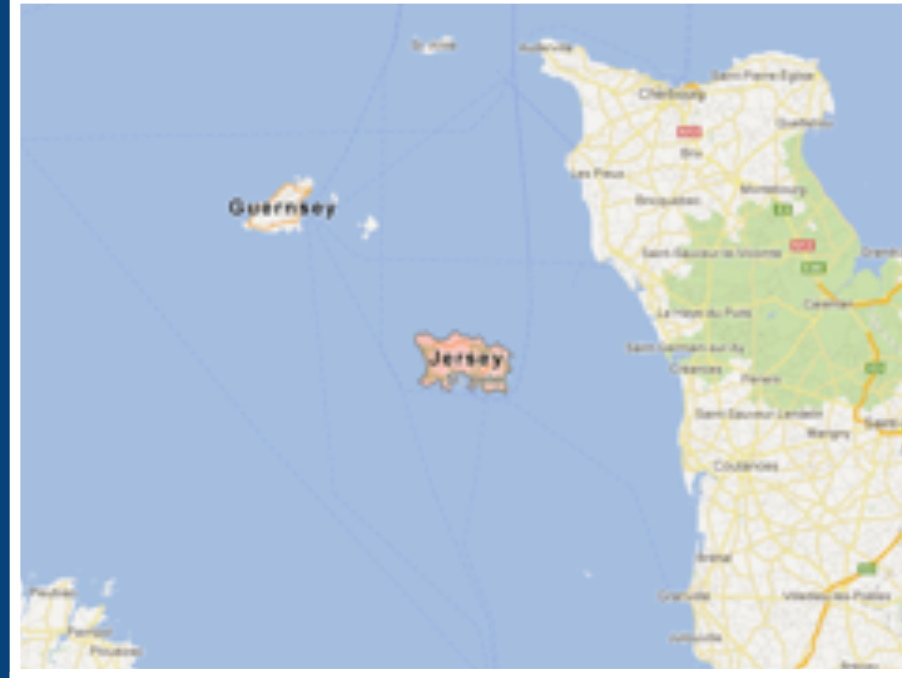
System Context diagram for Spring PetClinic
The System Context diagram for the Spring PetClinic system.
Friday 18 November 2016 22:21 UTC

Diagrams are maps

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



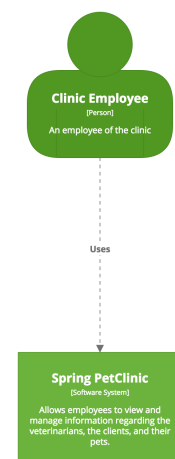
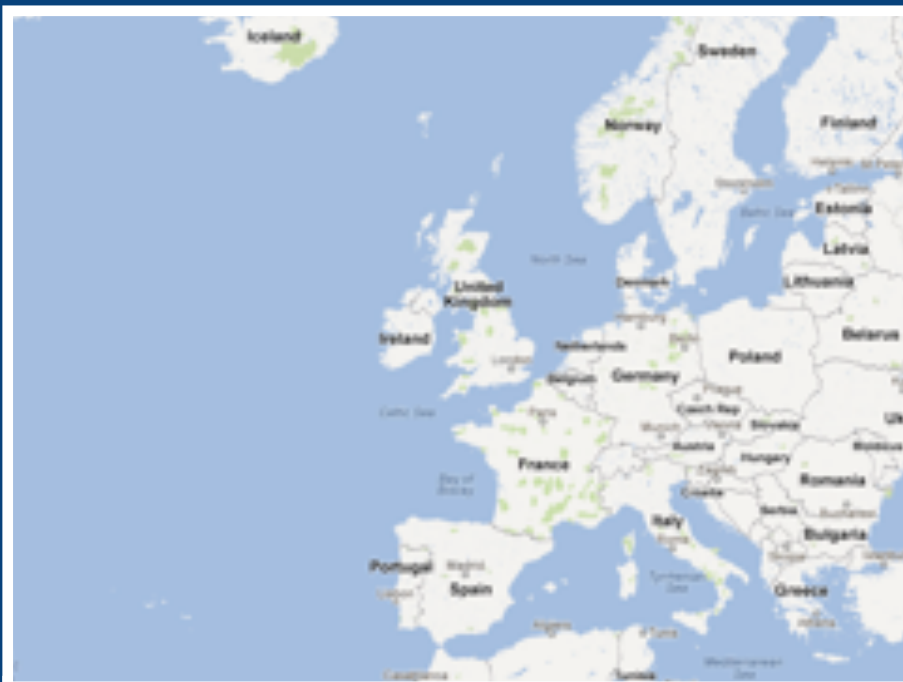
System Context diagram for Spring PetClinic
The System Context diagram for the Spring PetClinic system.
Friday 18 November 2016 22:21 UTC



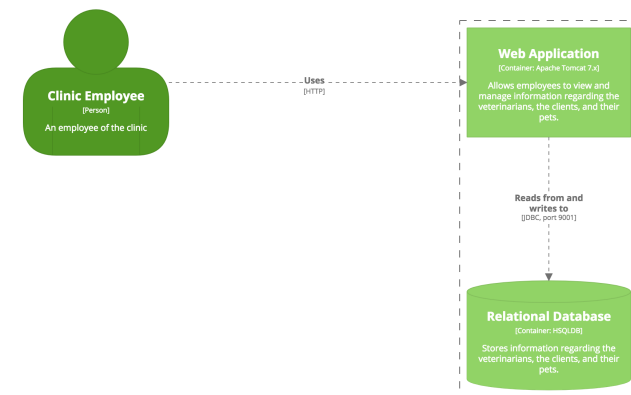
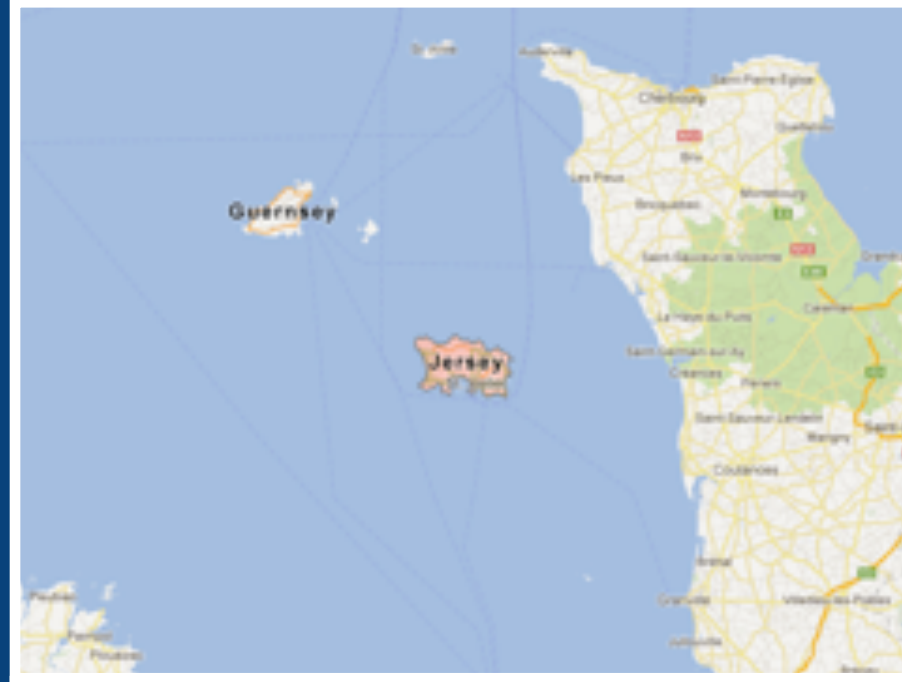
Container diagram for Spring PetClinic
The Container diagram for the Spring PetClinic system.
Friday 18 November 2016 22:21 UTC

Diagrams are maps

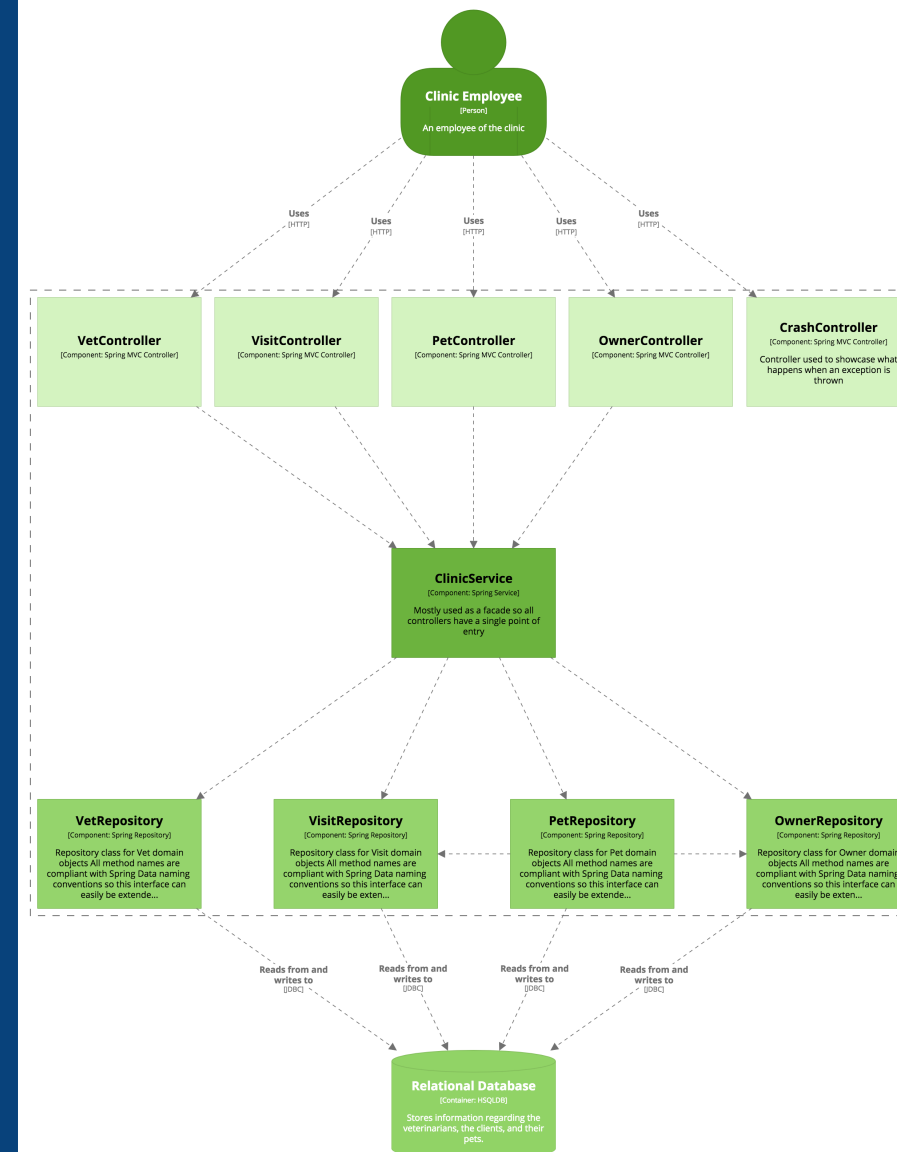
that help software developers navigate a large and/or complex codebase



System Context diagram for Spring PetClinic
The System Context diagram for the Spring PetClinic system.
Friday 18 November 2016 22:21 UTC



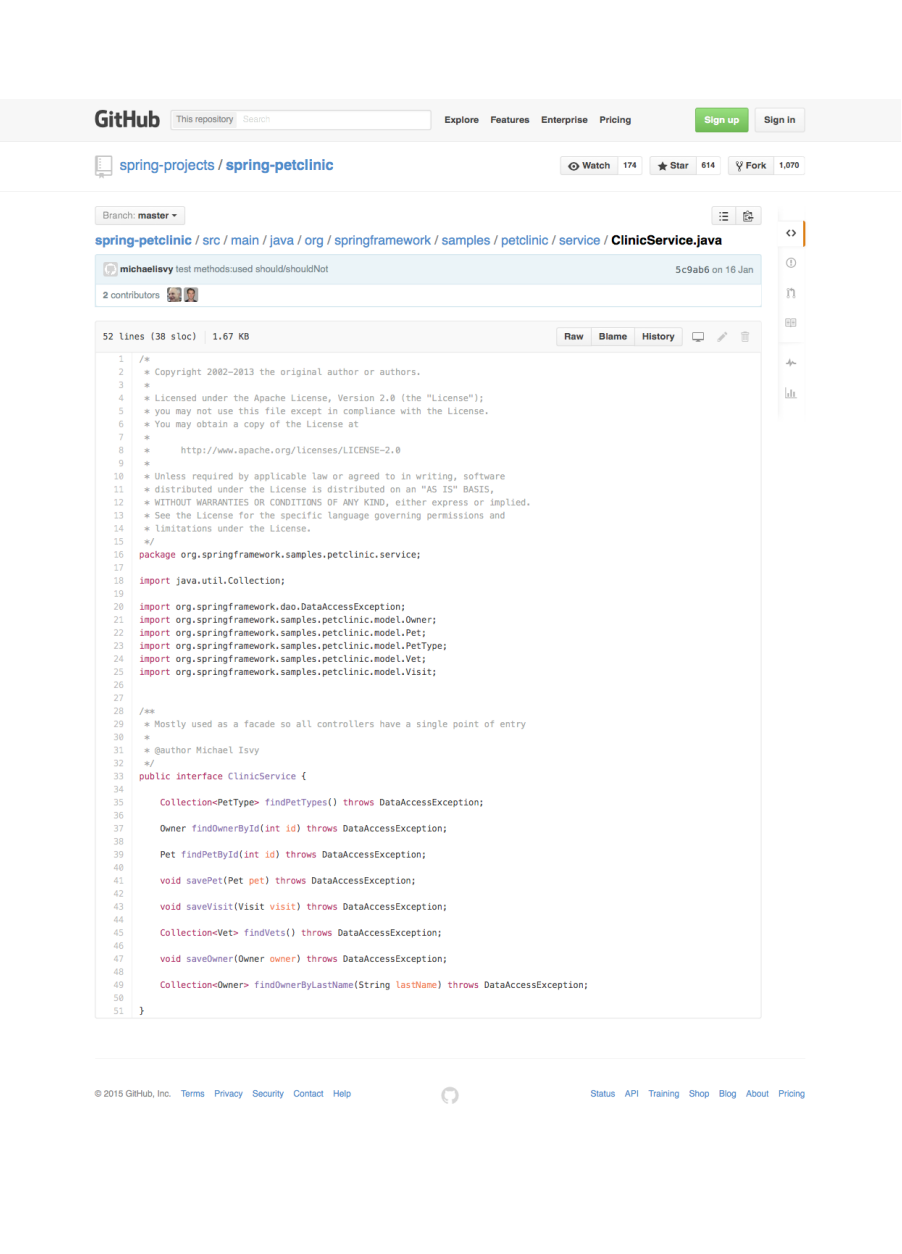
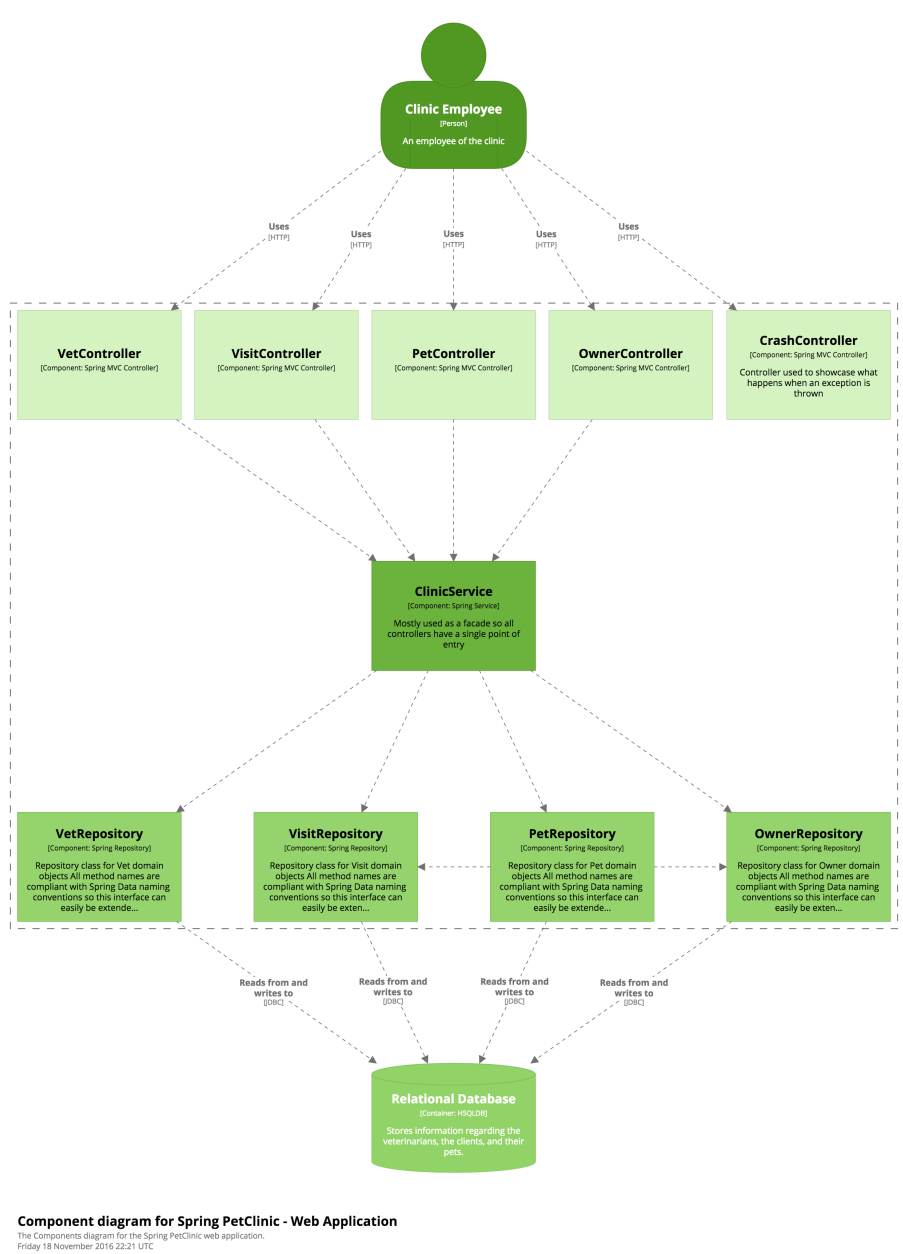
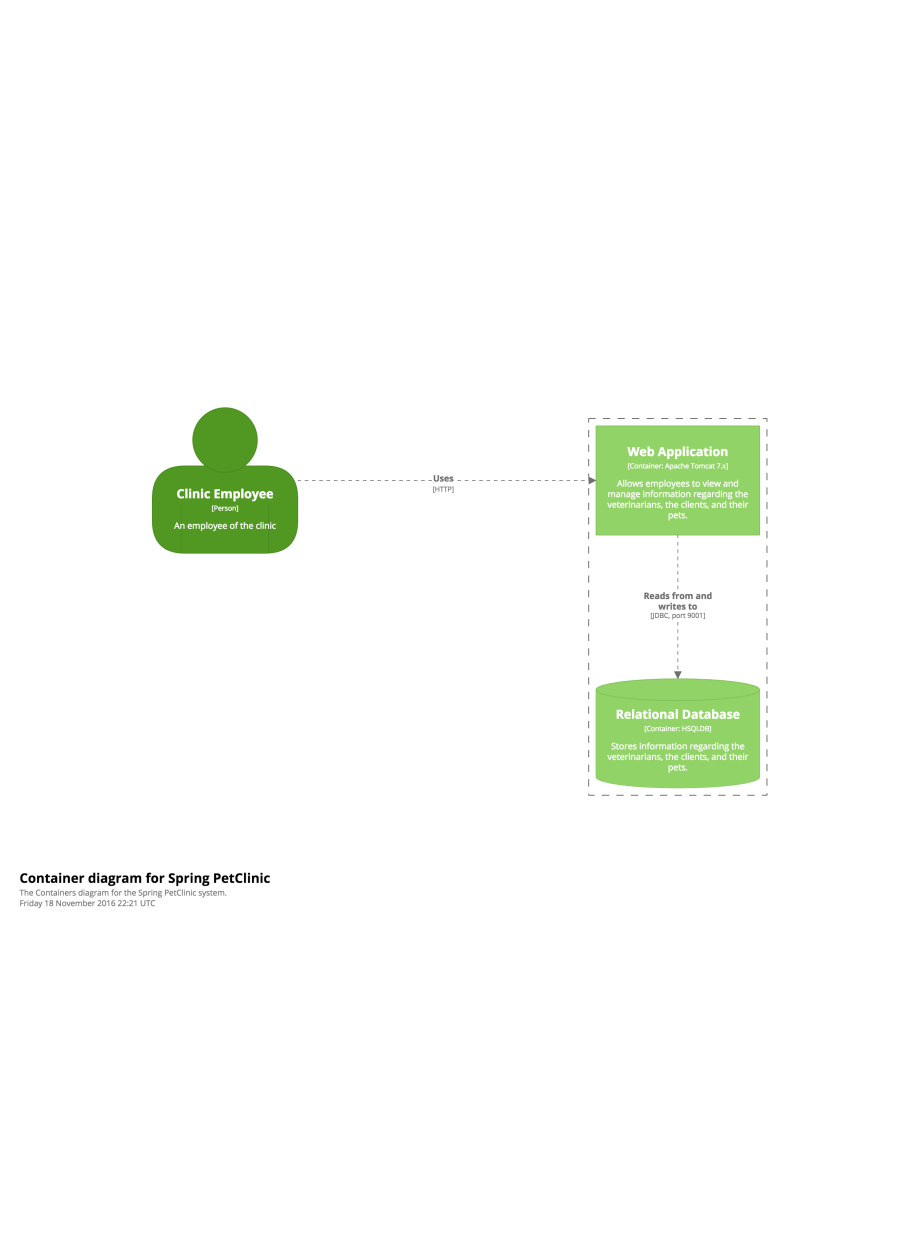
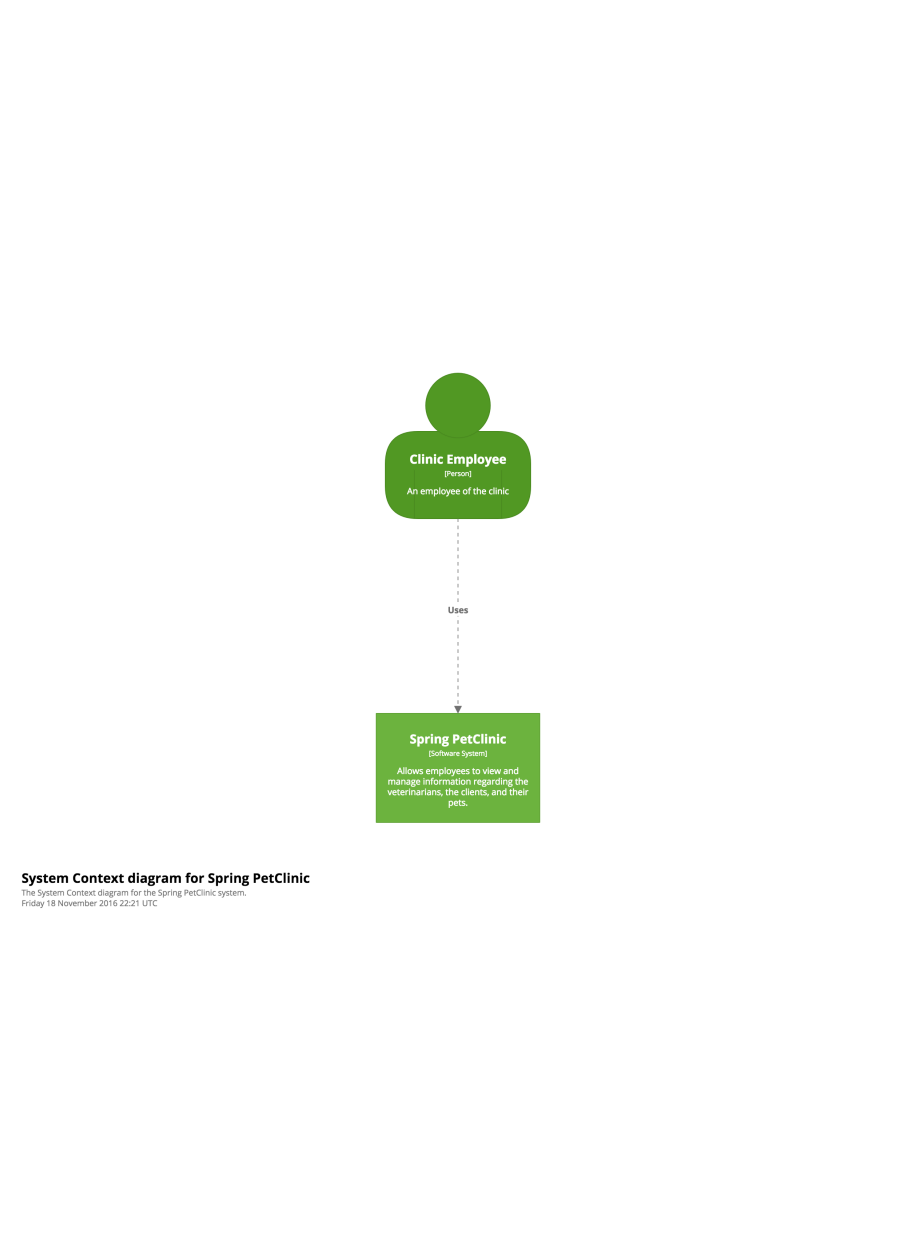
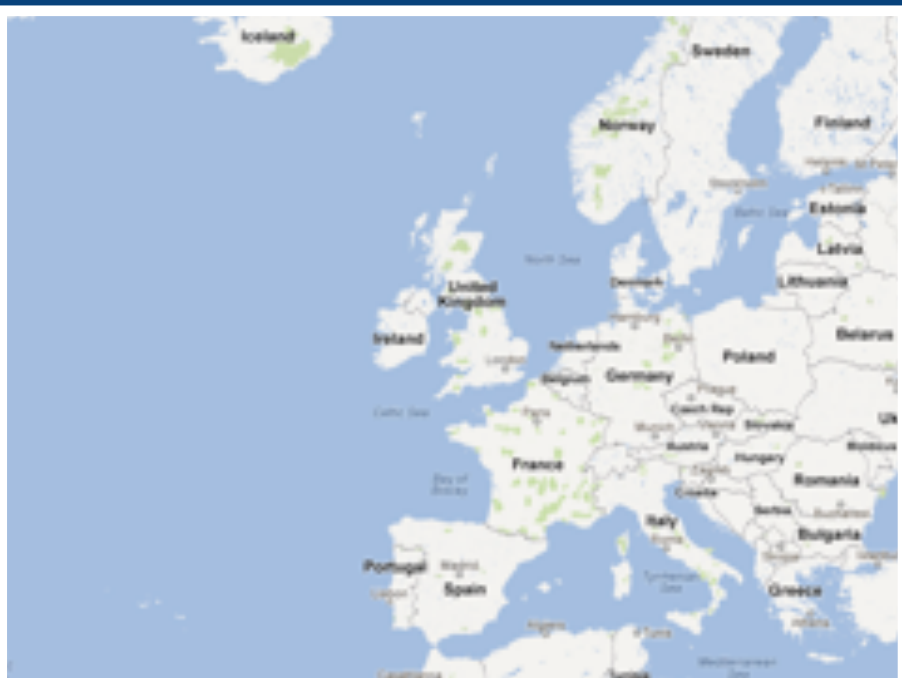
Container diagram for Spring PetClinic
The Container diagram for the Spring PetClinic system.
Friday 18 November 2016 22:21 UTC



Component diagram for Spring PetClinic - Web Application
The Component diagram for the Spring PetClinic web application.
Friday 18 November 2016 22:21 UTC

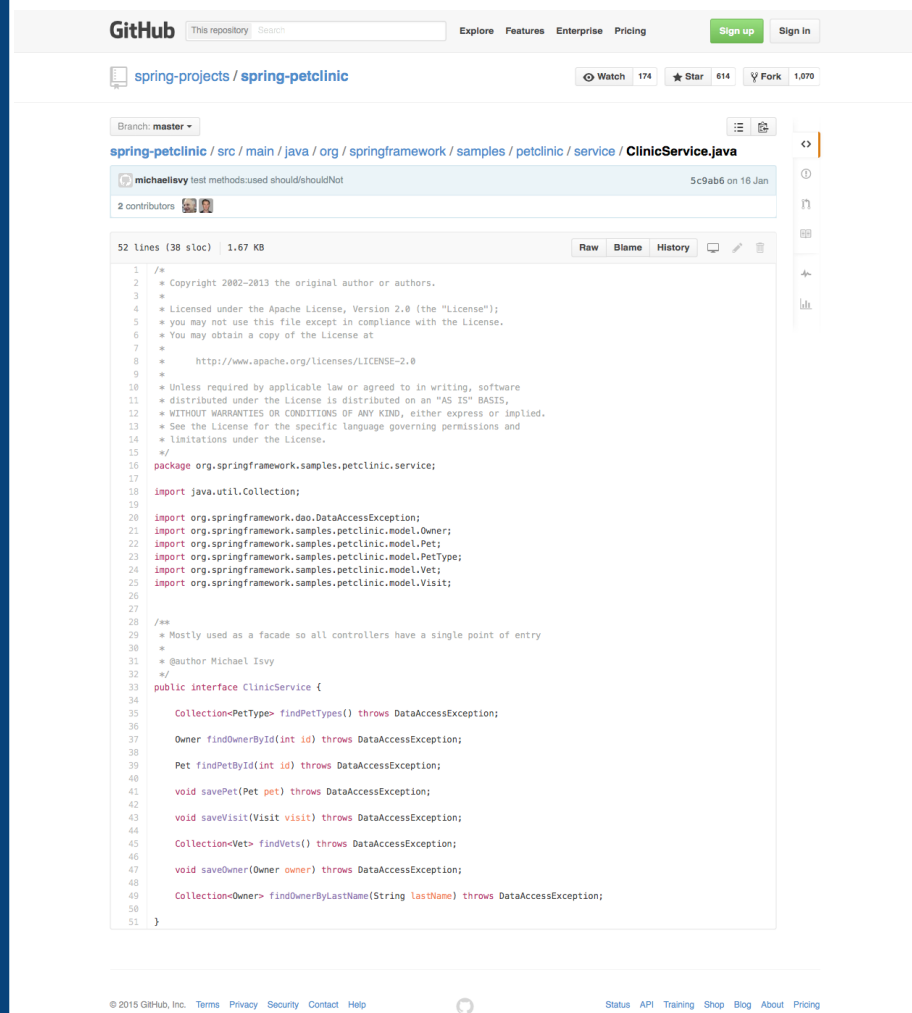
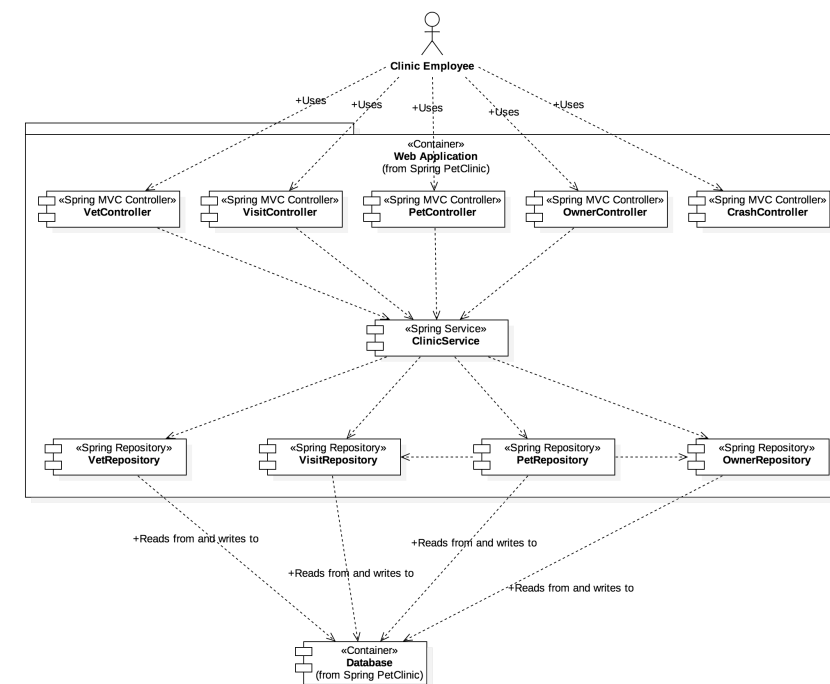
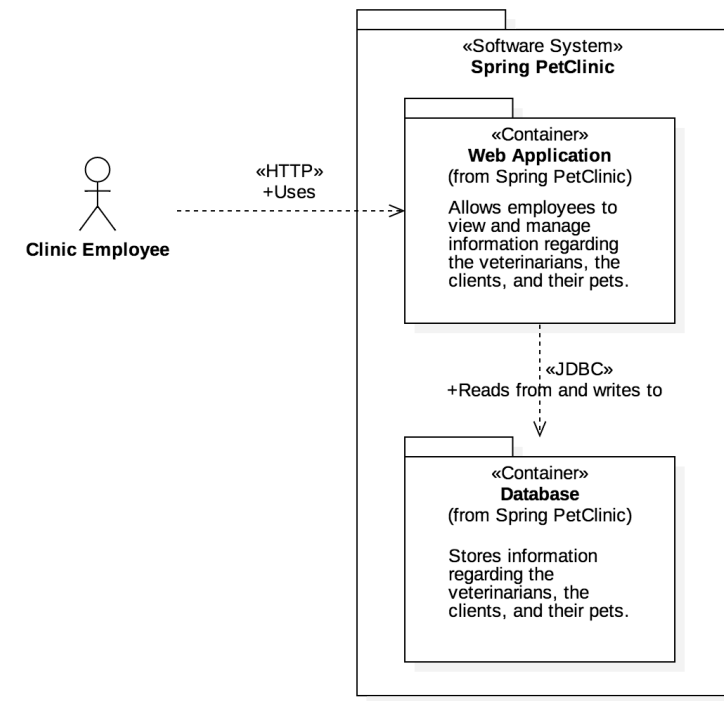
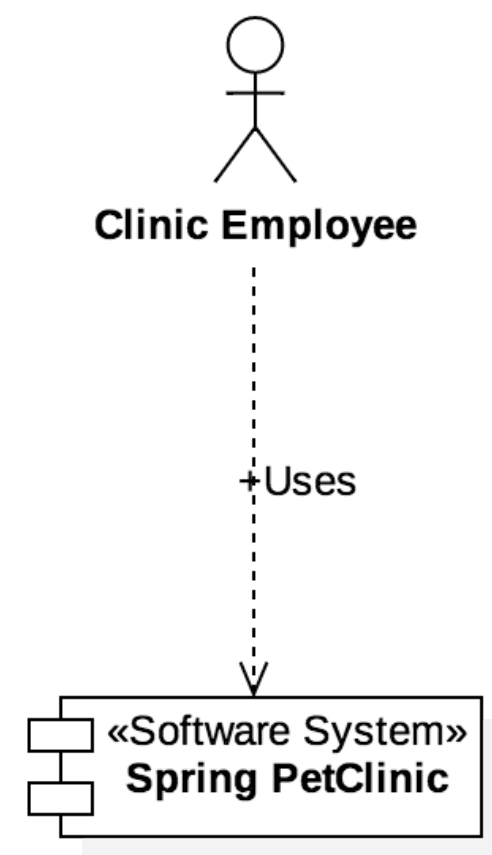
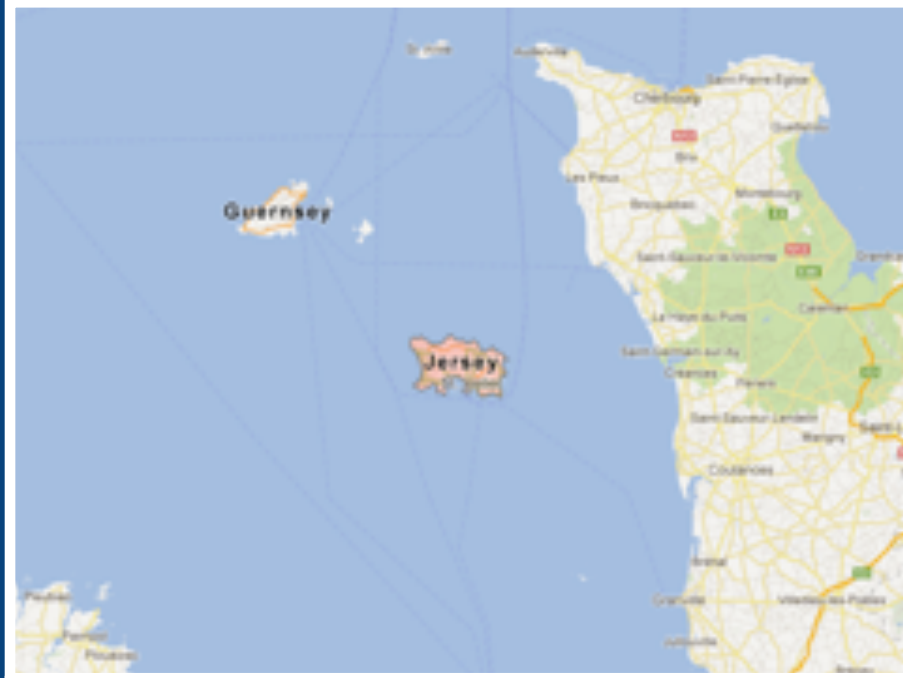
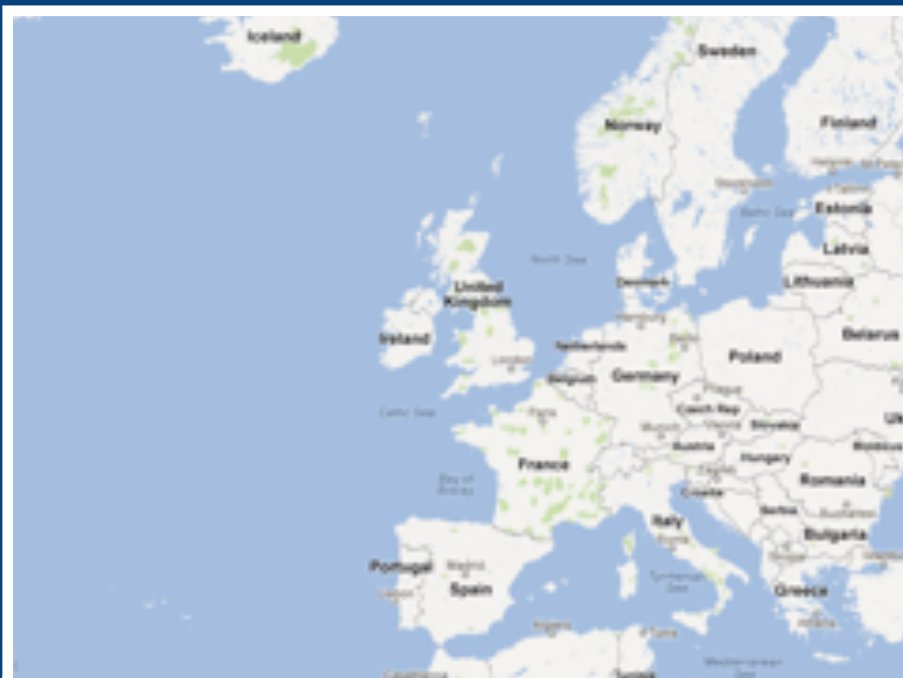
Diagrams are maps

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



Diagrams are maps

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



Diagrams are maps

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

Diagrams are a visual checklist
for design decisions

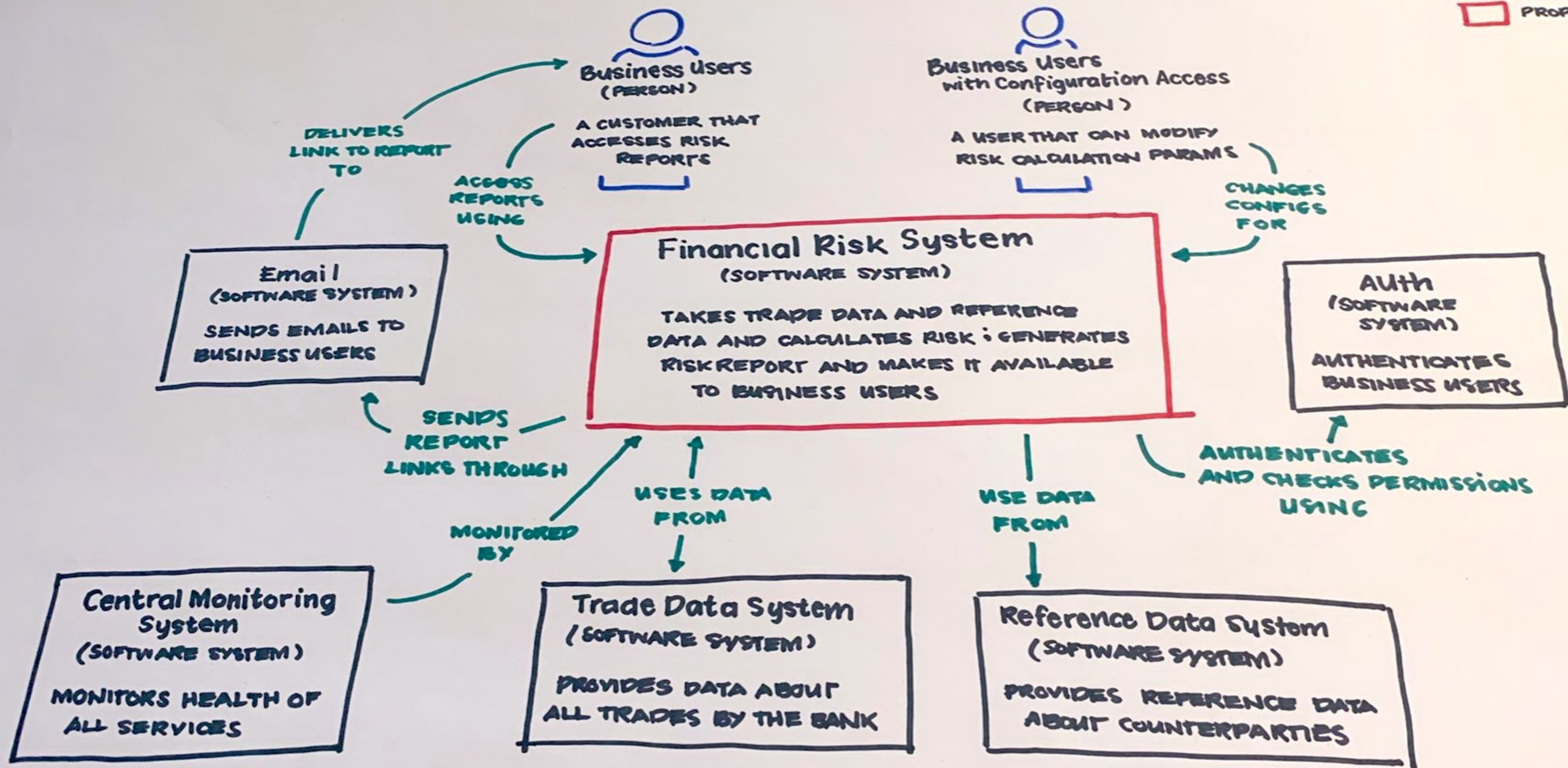
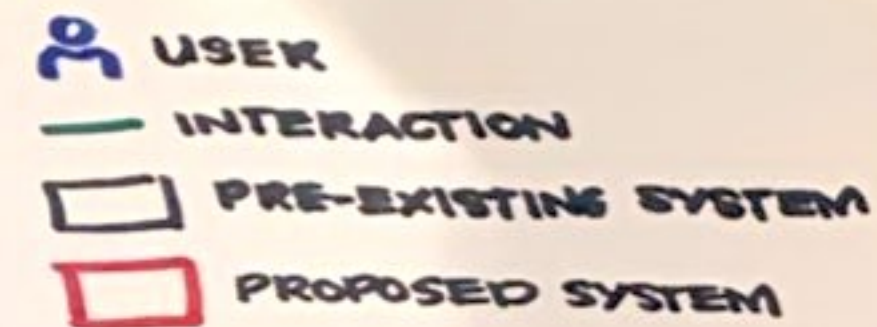
System Context diagram

What is the scope of the software system we're building?

Who is using it? What are they doing?

What system integrations does it need to support?

Financial Risk System: Context Diagram



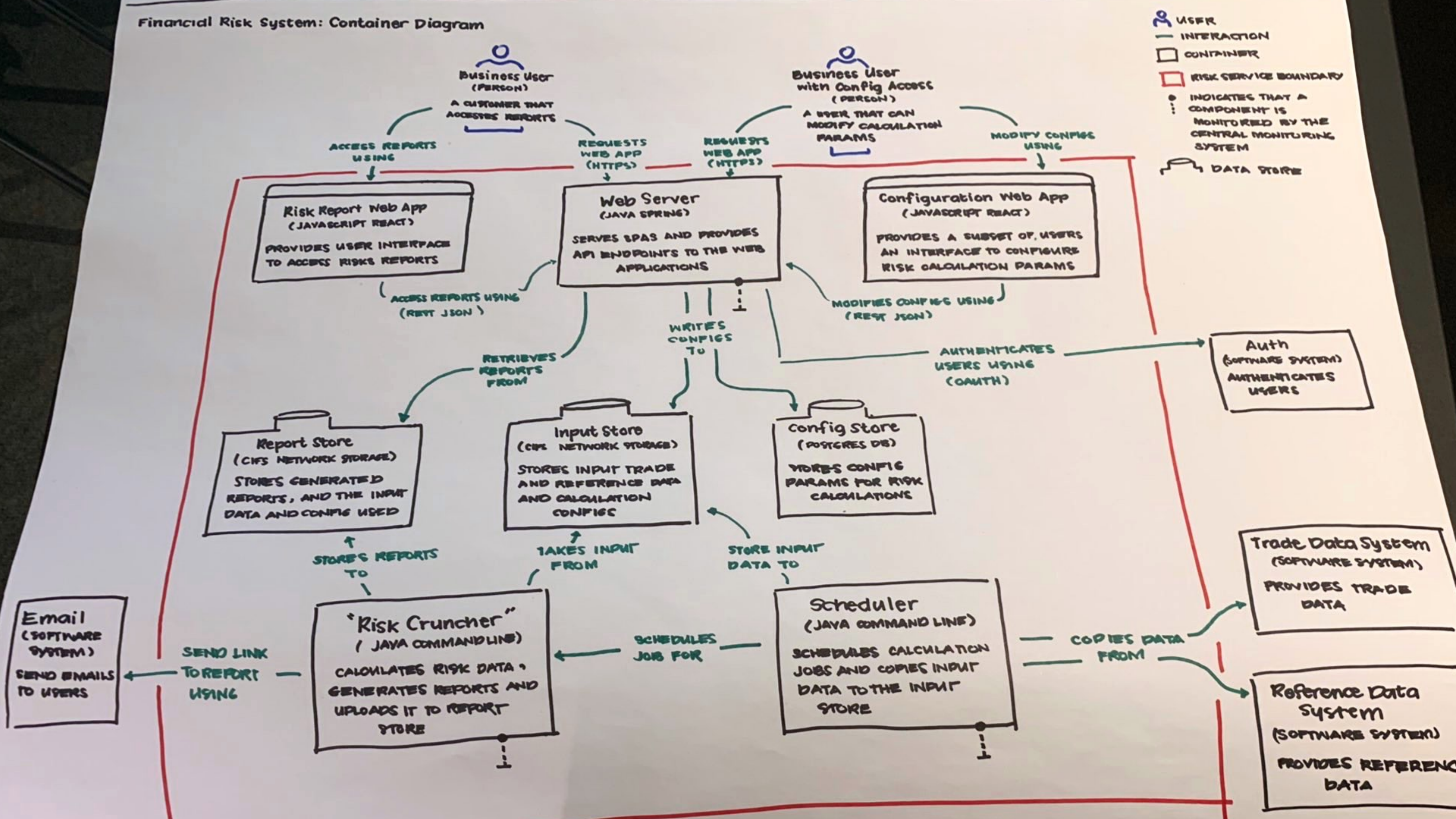
Container diagram

What are the major technology building blocks?

What are their responsibilities?

How do they communicate?

Financial Risk System: Container Diagram



The diagrams should spark
meaningful questions

No

“What does that arrow mean?”

“Why are some boxes red?”

“Is that a Java application?”

“Is that a monolithic application, or a collection of microservices?”

“How do the users get their reports?”

Yes

“What protocol are your two Java applications using to communicate with each other?”

“Why do you have two separate C# applications instead of one?”

“Why are you using MongoDB?”

“Why are you using MySQL when our standard is Oracle?”

“Should we really build new applications with .NET Framework rather than .NET Core?”

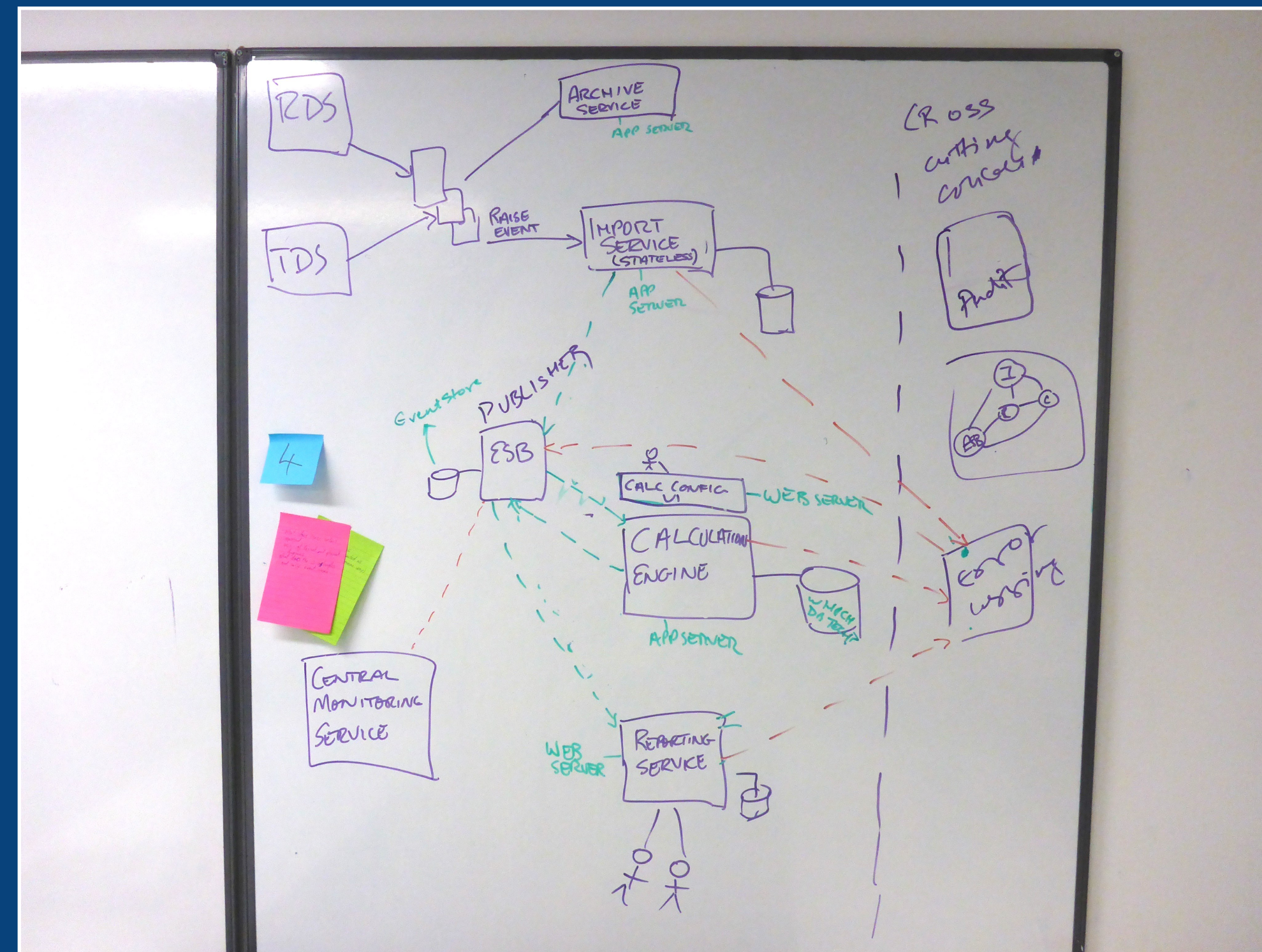
Richer diagrams lead to
richer design discussions

Richer diagrams lead to
better communication,
making it easier to scale teams

The diagrams should provide
meaningful feedback

We're trying to diagram a
[microservices | serverless] architecture,
but the diagram is getting complicated.

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



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

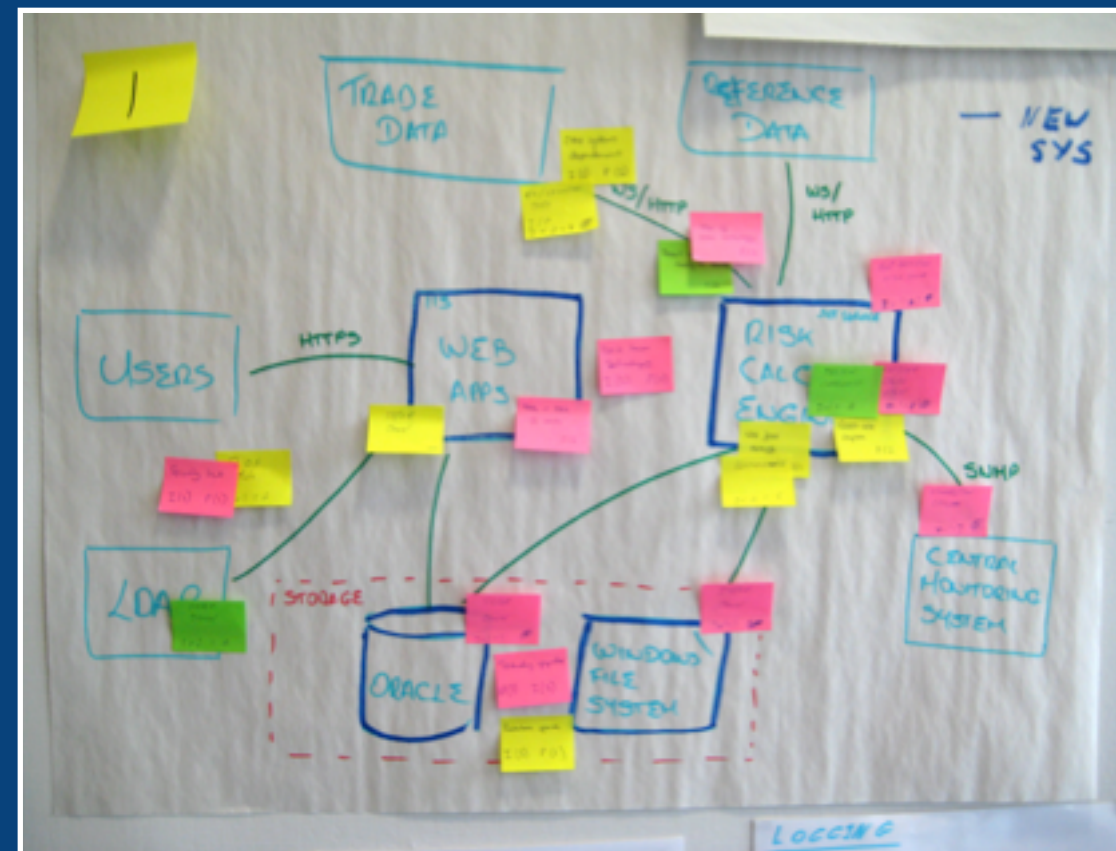
Base your architecture on
requirements, travel light
and prove your architecture
with concrete experiments.

Agile Architecture: Strategies for Scaling Agile Development

Scott Ambler

Identify and mitigate
your highest priority risks

Like estimates,
risks are subjective



Risk-storming

A visual and collaborative technique for identifying risk

Threat modelling

(STRIDE, LINDDUN, Attack Trees, etc)

Title These documents have names that are short noun phrases. For example, "ADR 1: Deployment on Ruby on Rails 3.0.10" or "ADR 9: LDAP for Multitenant Integration"

Context This section describes the forces at play, including technological, political, social, and project local. These forces are probably in tension, and should be called out as such. The language in this section is value-neutral. It is simply describing facts.

Decision This section describes our response to these forces. It is stated in full sentences, with active voice. "We will ..."

Status A decision may be "proposed" if the project stakeholders haven't agreed with it yet, or "accepted" once it is agreed. If a later ADR changes or reverses a decision, it may be marked as "deprecated" or "superseded" with a reference to its replacement.

Consequences This section describes the resulting context, after applying the decision. All consequences should be listed here, not just the "positive" ones. A particular decision may have positive, negative, and neutral consequences, but all of them affect the team and project in the future.

“Architecture Decision Record”

A short description of an
architecturally significant decision

<http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions> (Michael Nygard)

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)



97 Strategies to Avoid Up Front Design

O RLY?

Vera Gile

#52

“I’m good with
maybe a day
for a one-year
effort.”

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

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



You understand the significant architectural drivers (requirements, quality attributes, constraints).

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.

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).

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.



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.



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.



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.

Some up front design to create a
starting point and direction
for further **evolutionary design**

Adopt an agile mindset

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

Thank you!



Simon Brown



@simonbrown