

The lost art of software design

Simon Brown

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” 🤔

Big design up front is dumb.

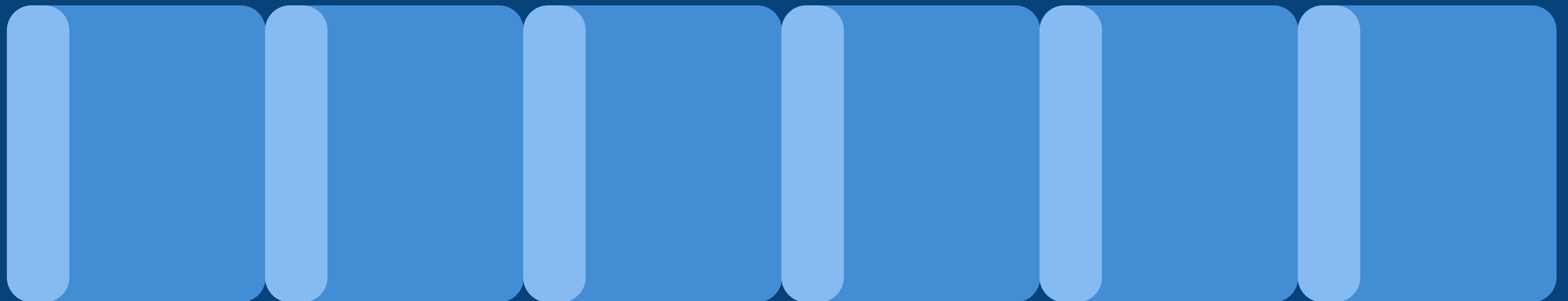
Doing no design up front
is even dumber.

Dave Thomas



Big Design Up Front

Evolutionary Design



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.

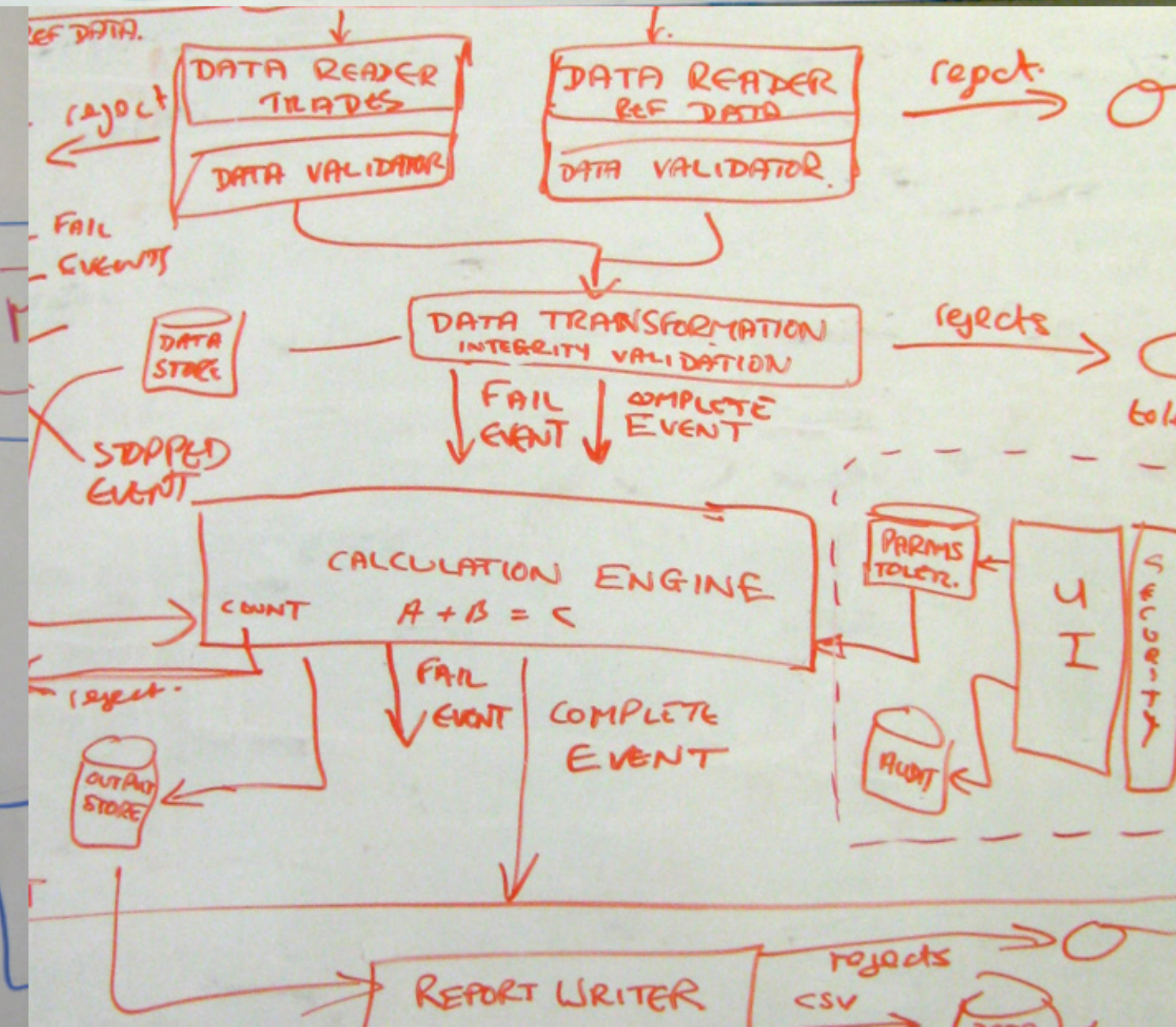
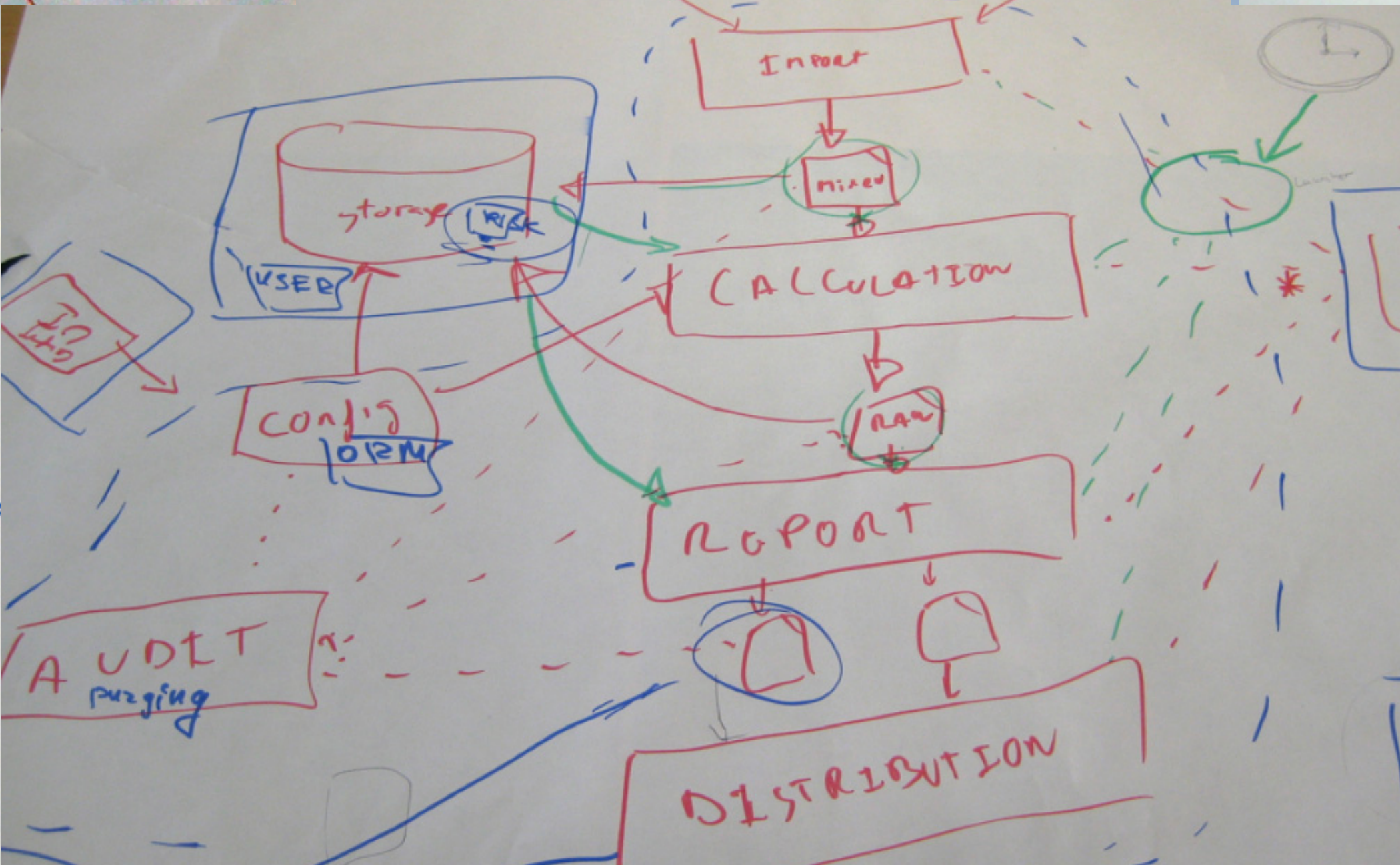
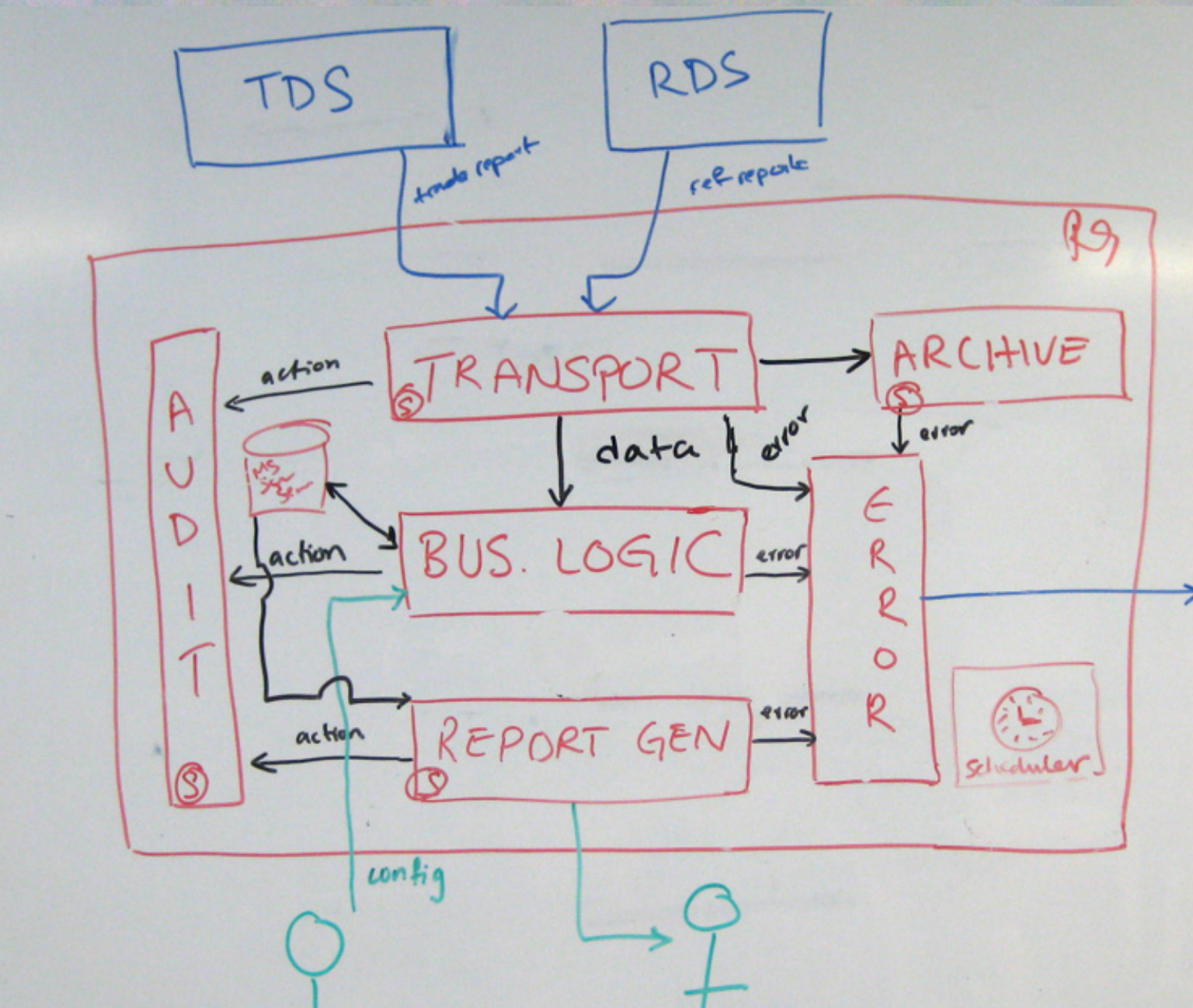
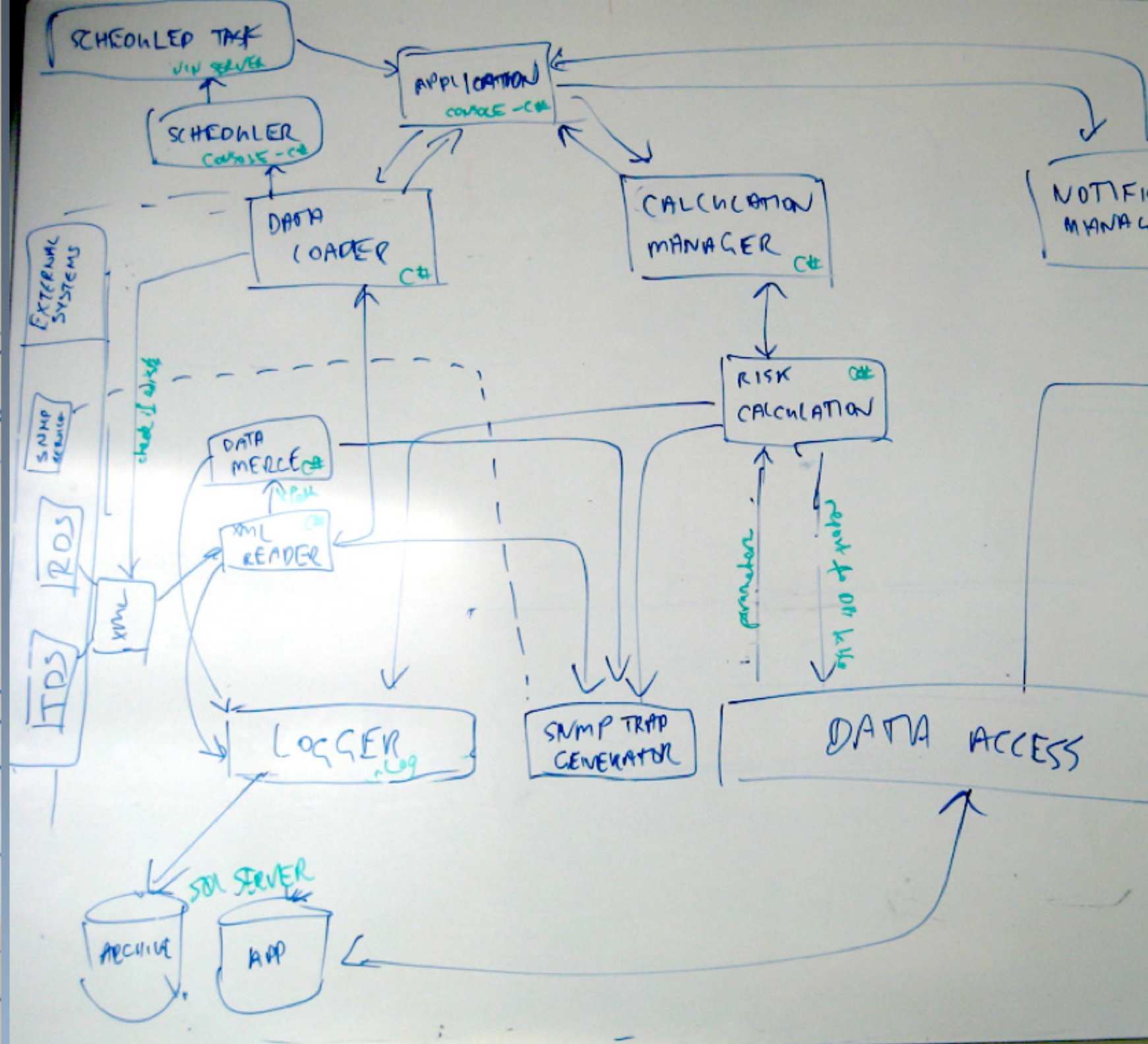
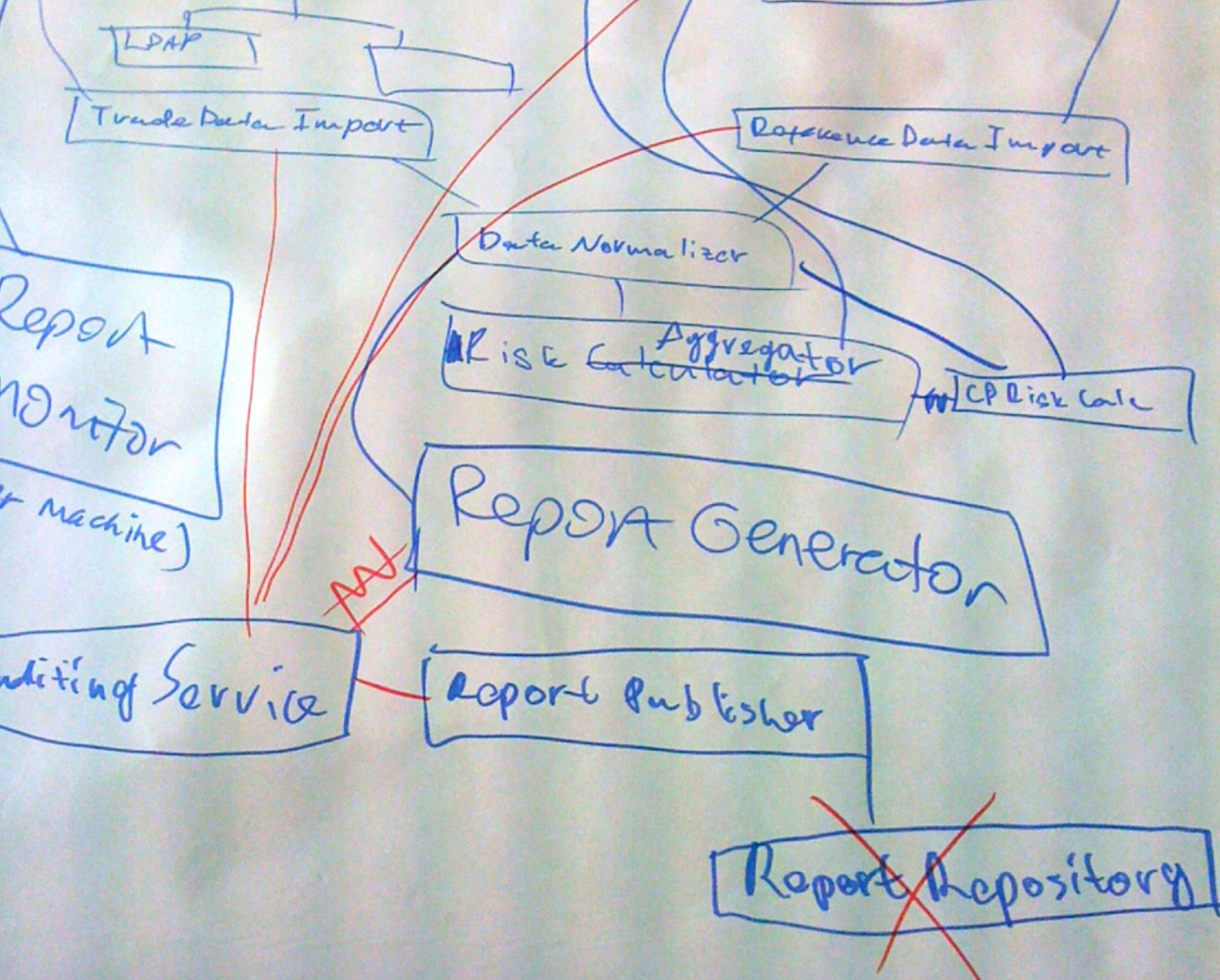


Design a software solution for the "Financial Risk System", and **draw** one or more architecture diagrams to describe your solution





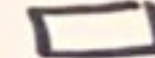

60-90 minutes

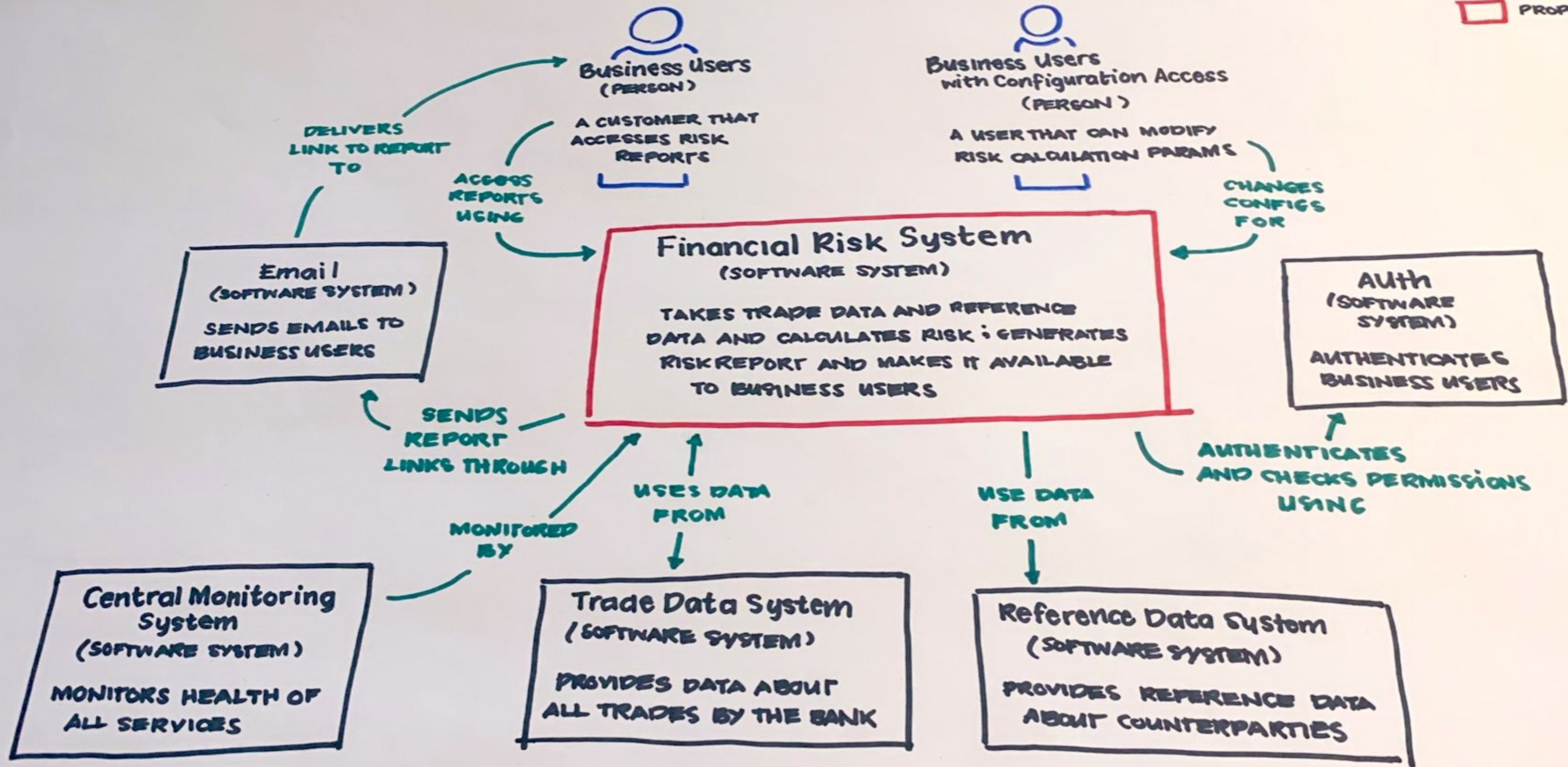
Iteration 1



Iteration 2

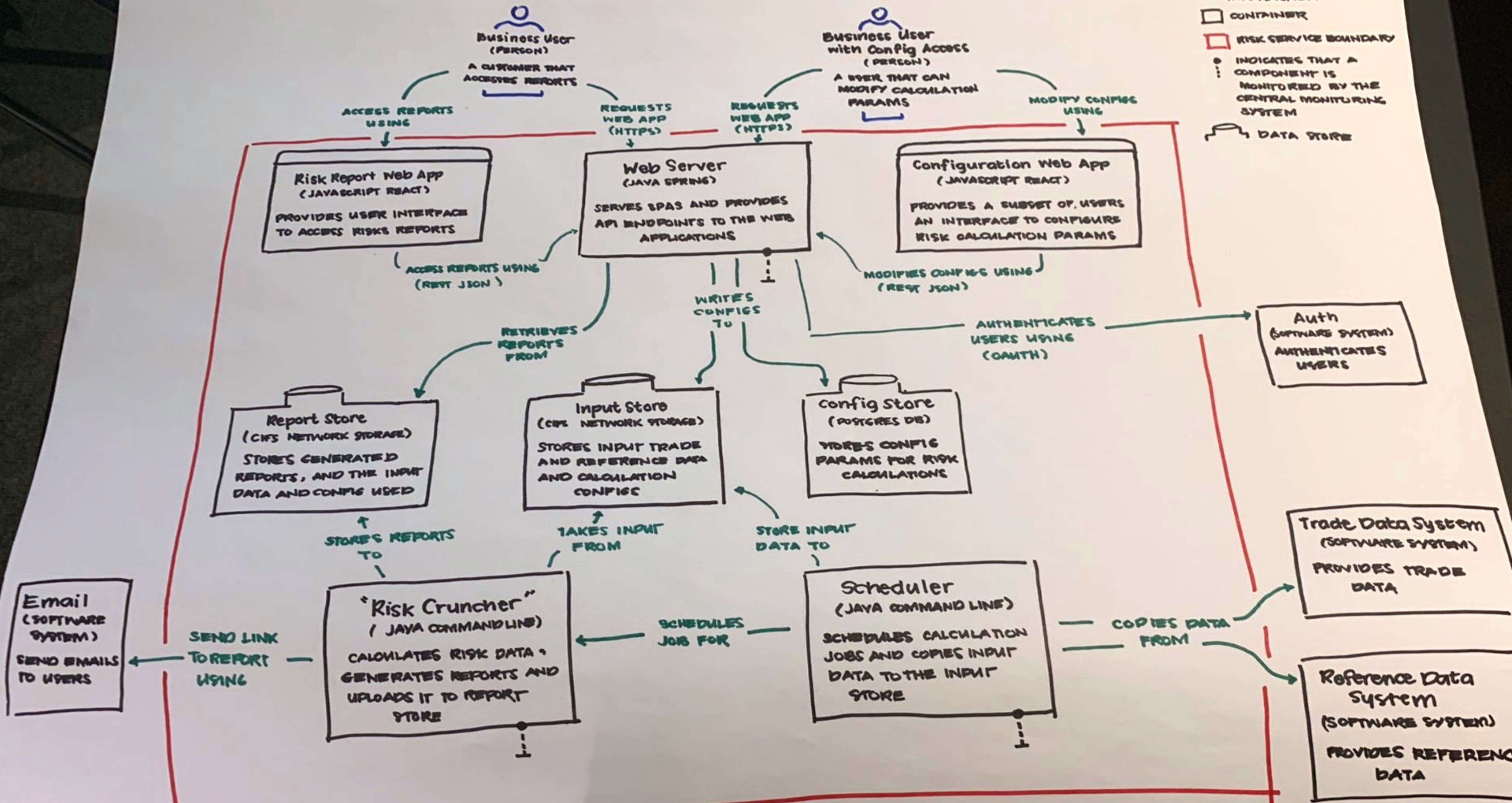
Financial Risk System: Context Diagram

-  USER
-  INTERACTION
-  PRE-EXISTING SYSTEM
-  PROPOSED SYSTEM



Financial Risk System: Container Diagram

- USER
- INTERACTION
- CONTAINER
- RISK SERVICE BOUNDARY
- INDICATES THAT A COMPONENT IS MONITORED BY THE CENTRAL MONITORING SYSTEM
- DATA STORE



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

#12

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



97 Strategies to Avoid
Up Front Design

#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



Simon Brown @simonbrown · Mar 18



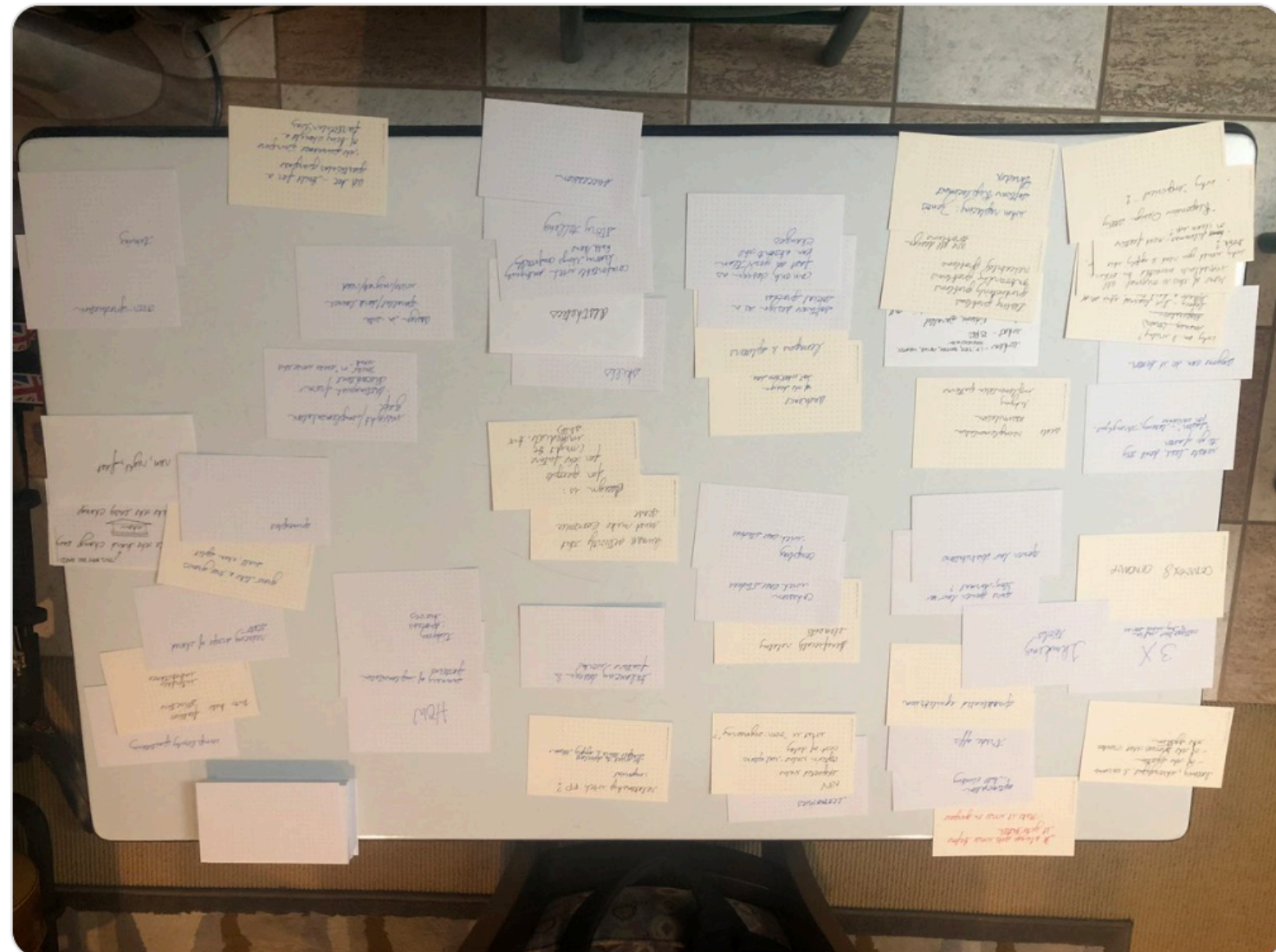
"To write a book I need to see a whole, in part just to reduce my anxiety." ... I'm the same when I'm building software. 😊 #SomeDesignUpFront



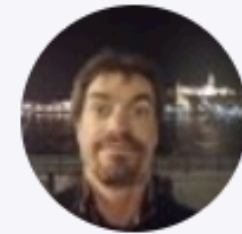
Kent Beck ✓

@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

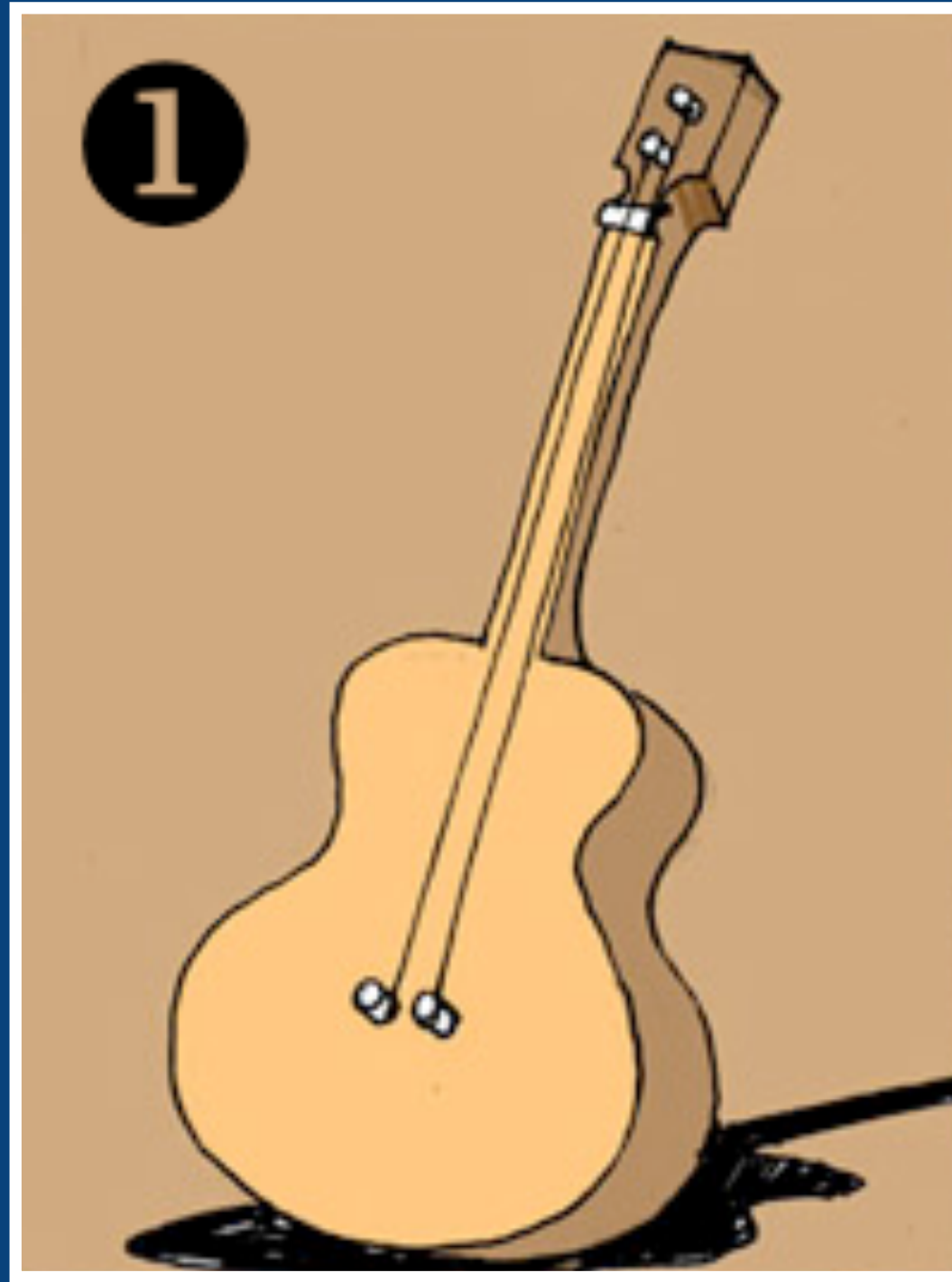


Up front design is not
necessarily about creating a
perfect end-state or
complete architecture



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

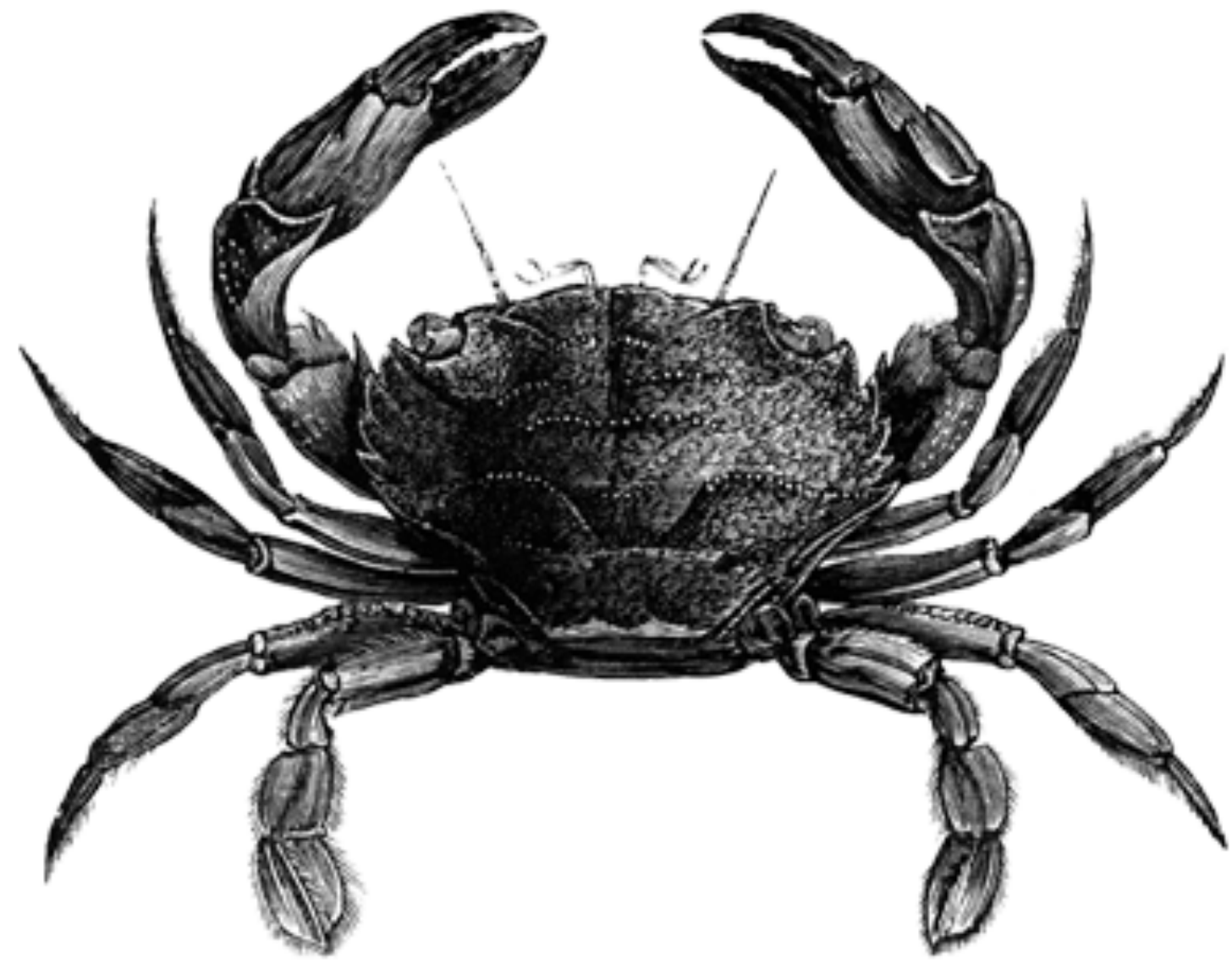
(irrespective of team size)

**Incomprehensible software
architecture diagrams**

UML?



UML usage is low



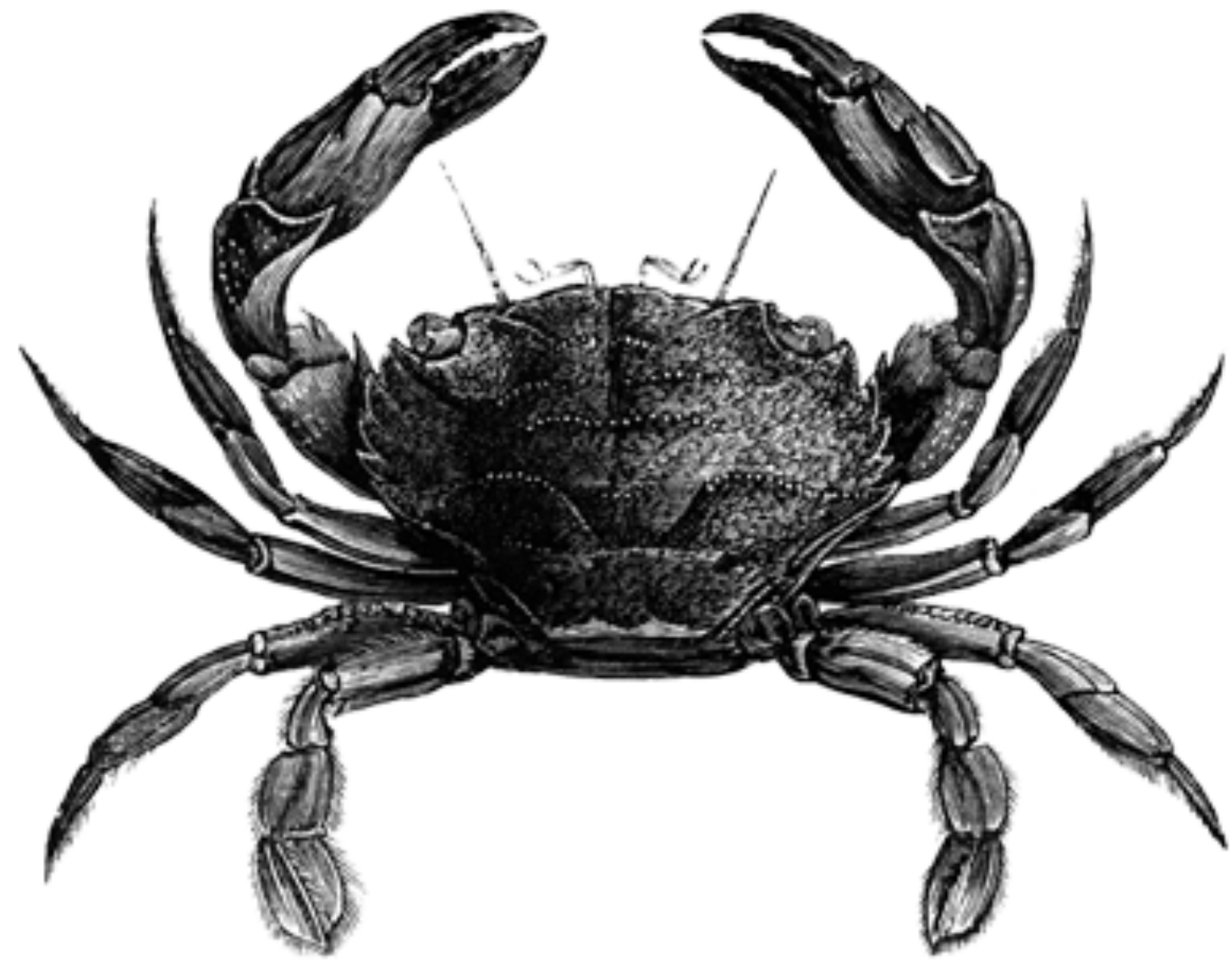
97 Ways to Sidestep UML

O RLY?

Knowfa Mallity

#36

“You’ll be seen as old.”



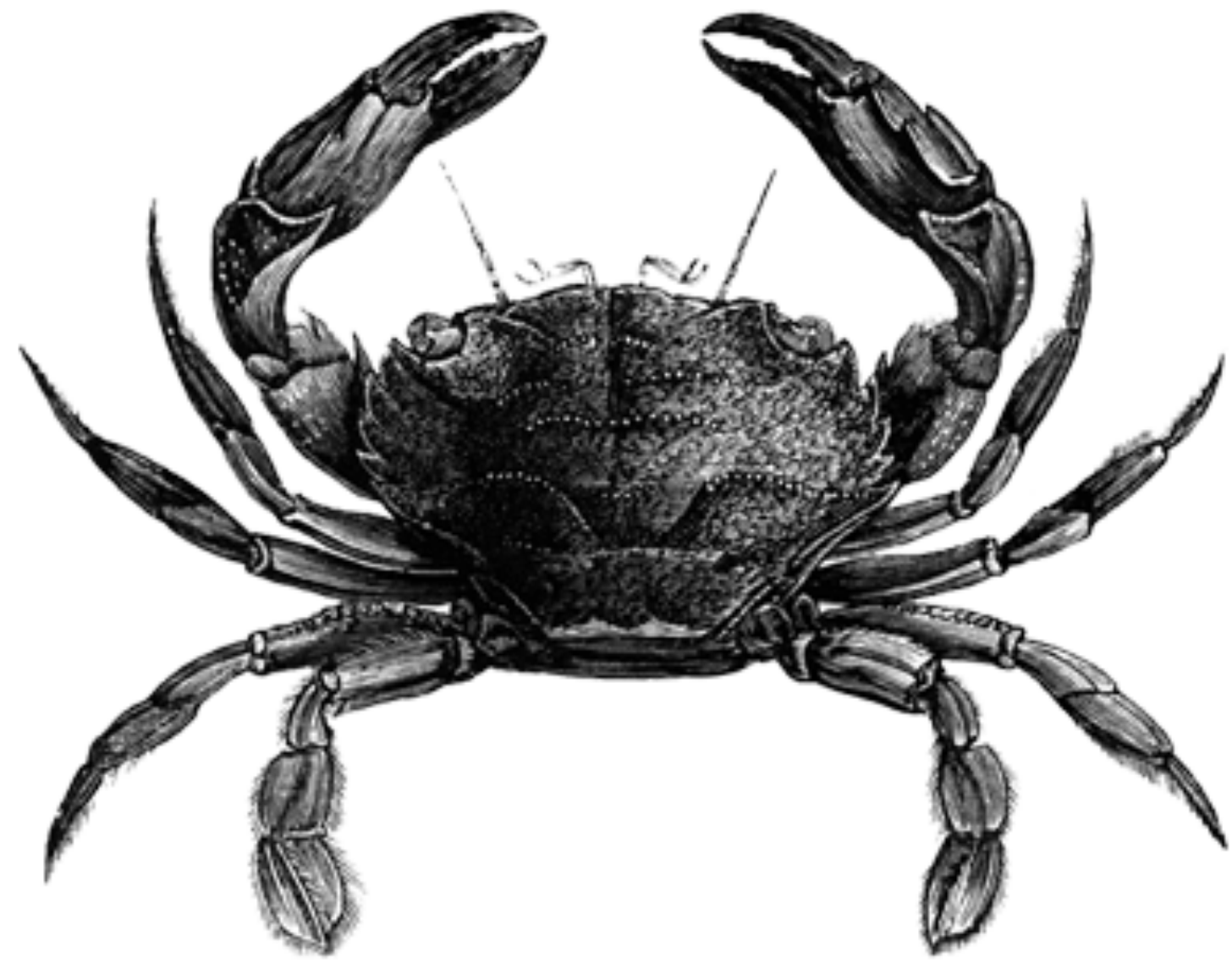
97 Ways to Sidestep UML

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.”



Very elaborate waste of time

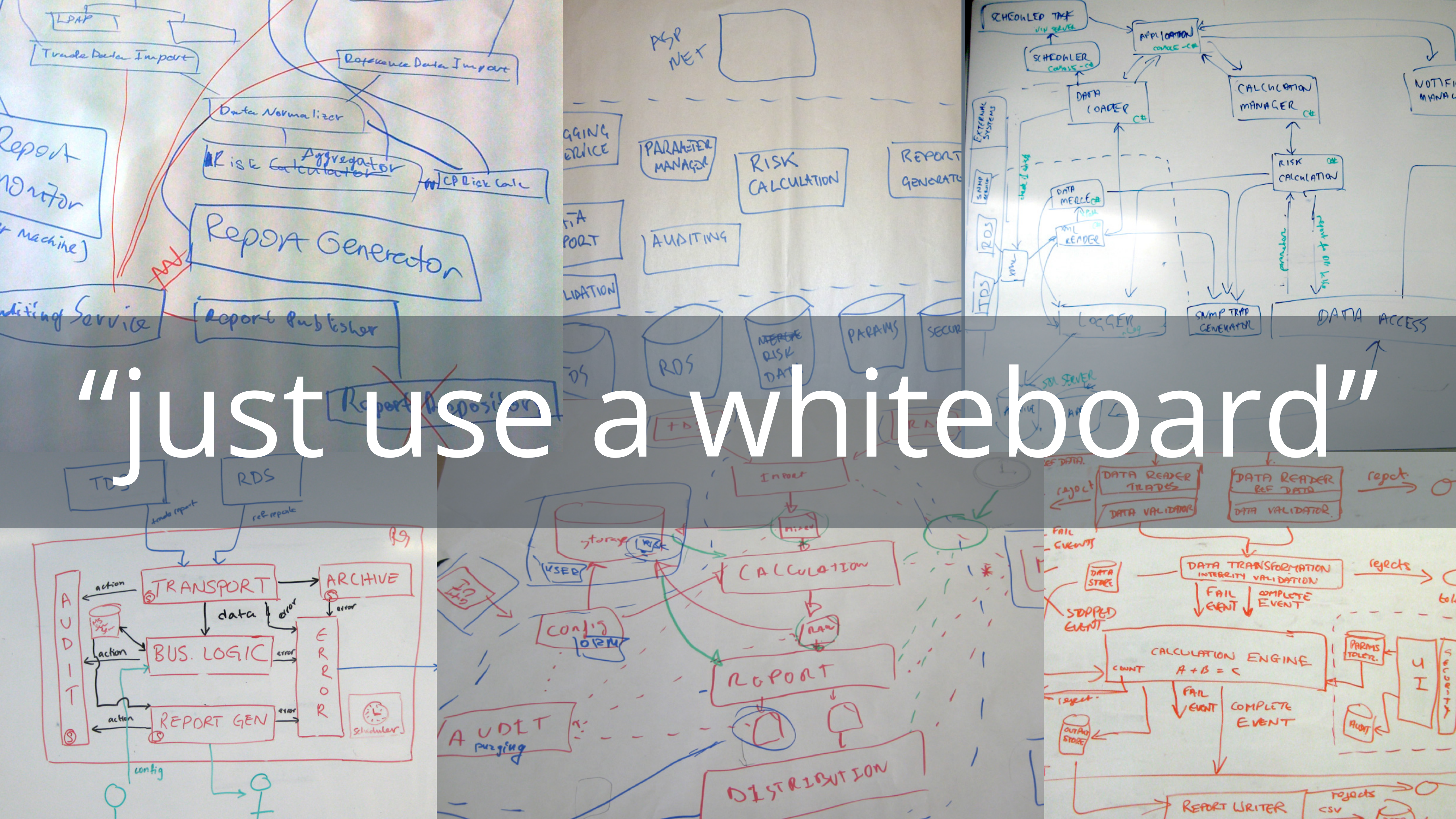




Just use a whiteboard!



“just use a whiteboard”




What's wrong these diagrams?

Swap and review your diagrams

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





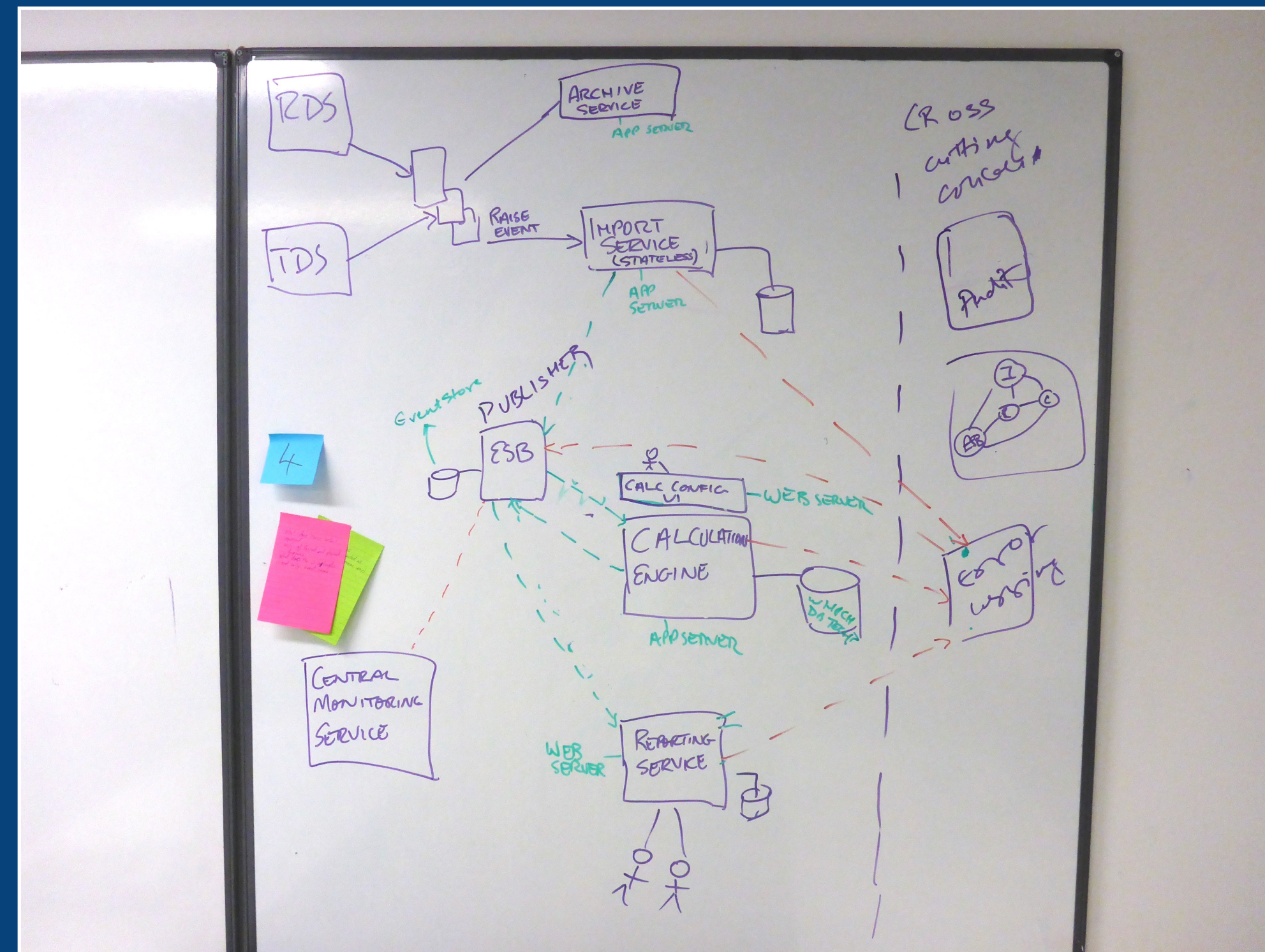
It's impossible to
answer those questions

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

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

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

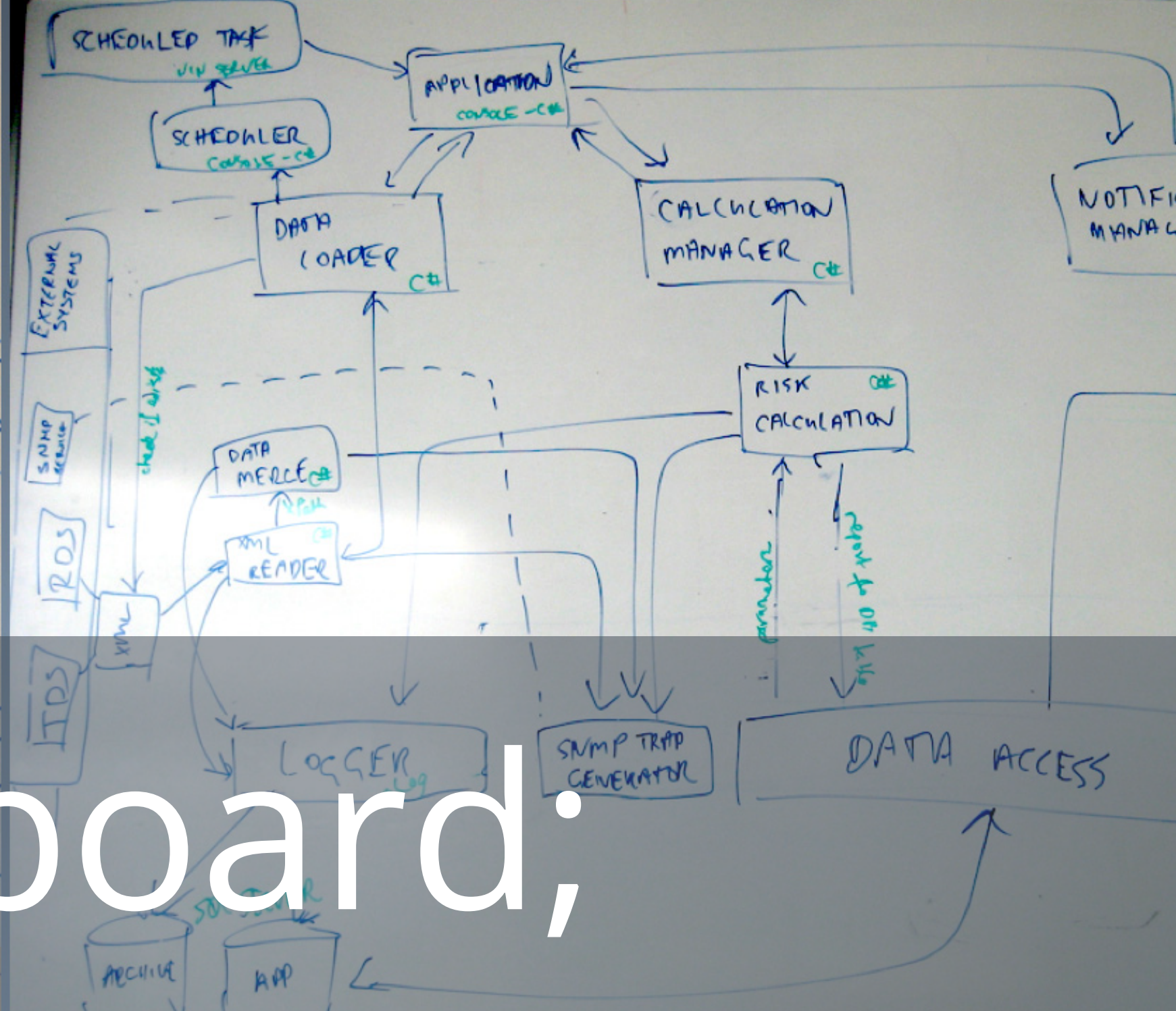
Implementation

Curly braces on the same or next line
Whitespace vs tabs

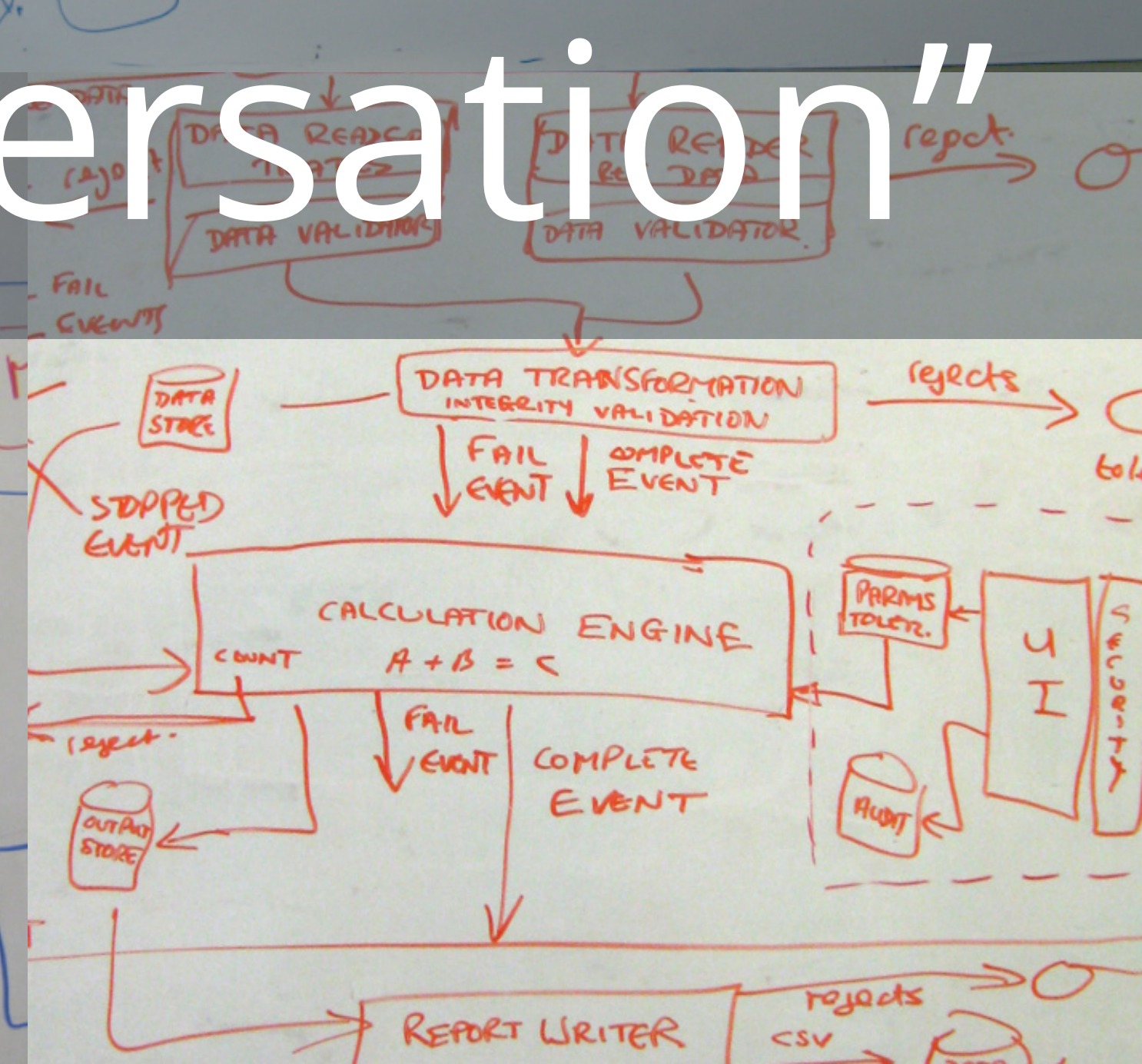
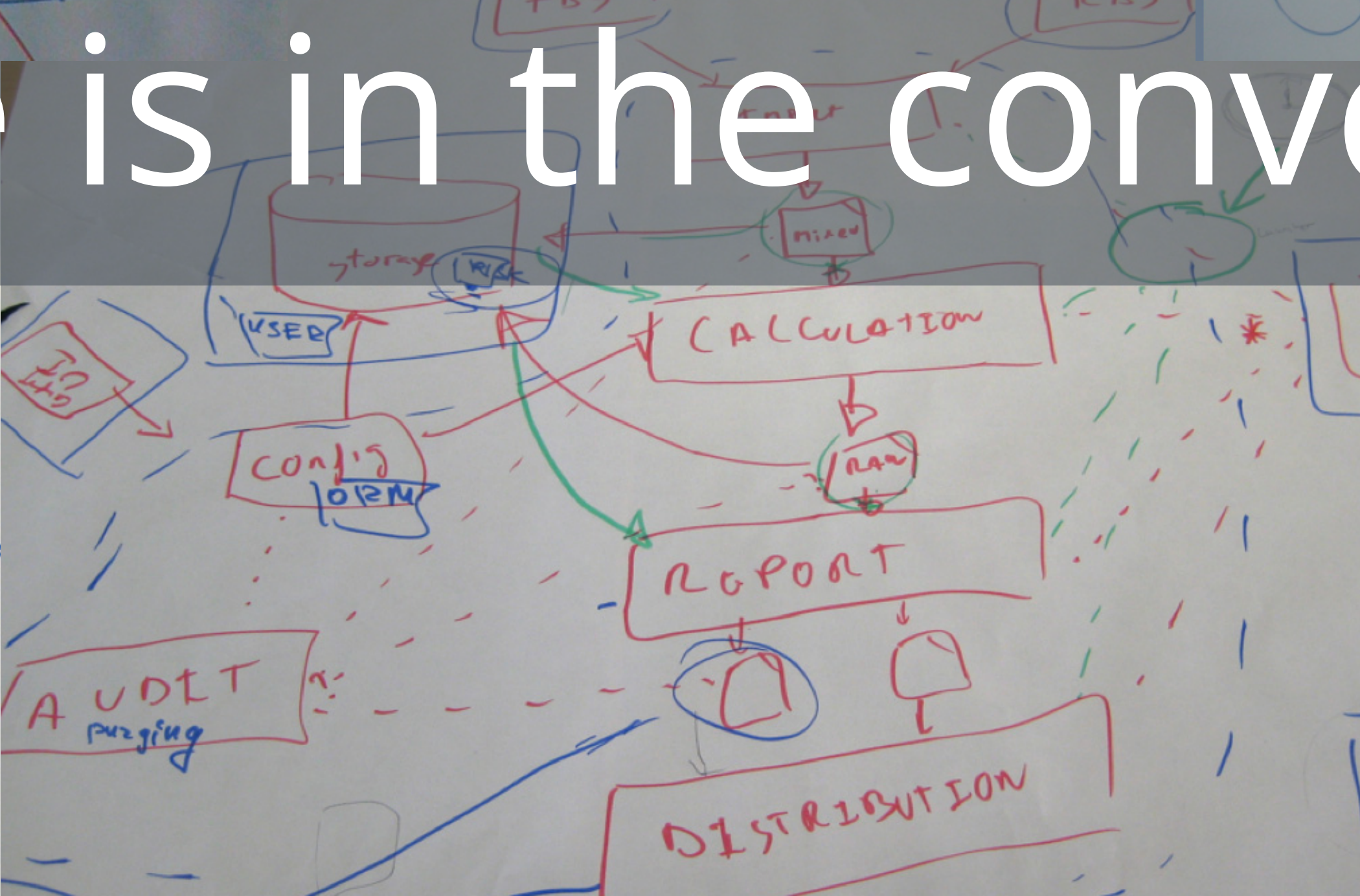
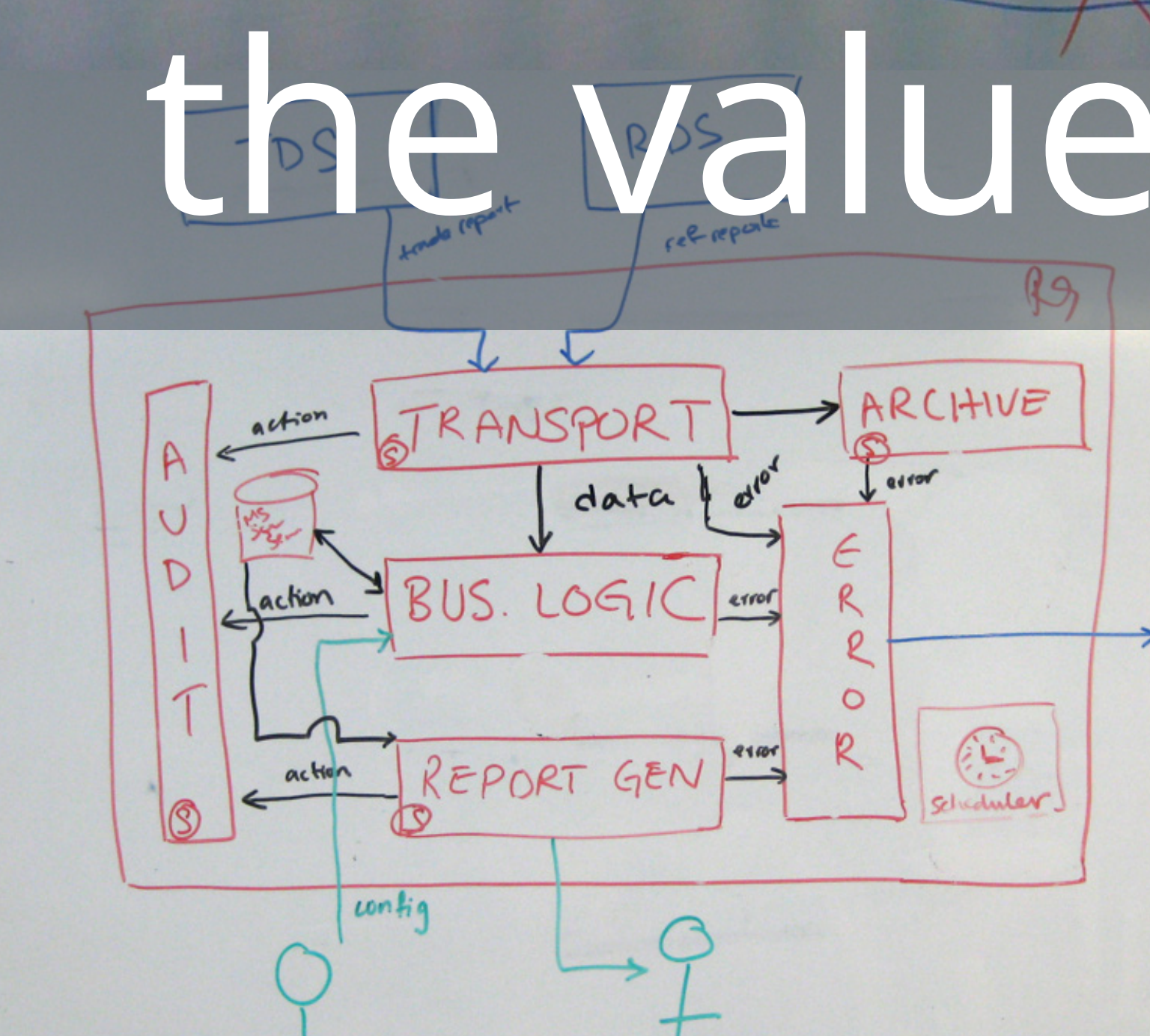
I think there is a role for a **broad starting point architecture**. Such things as stating early on how to layer the application, how you'll interact with the database (if you need one), what approach to use to handle the web server.

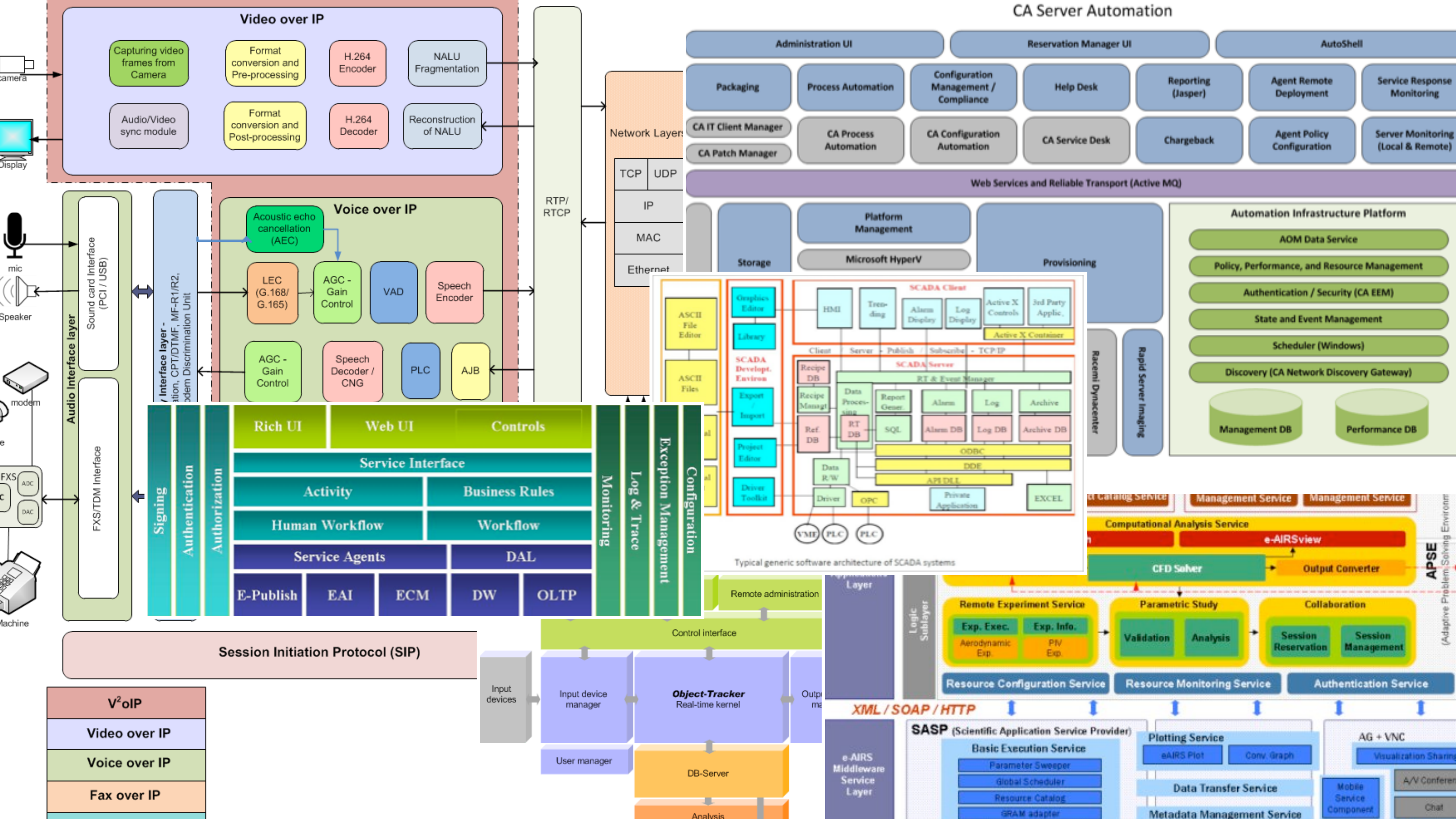
Martin Fowler

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

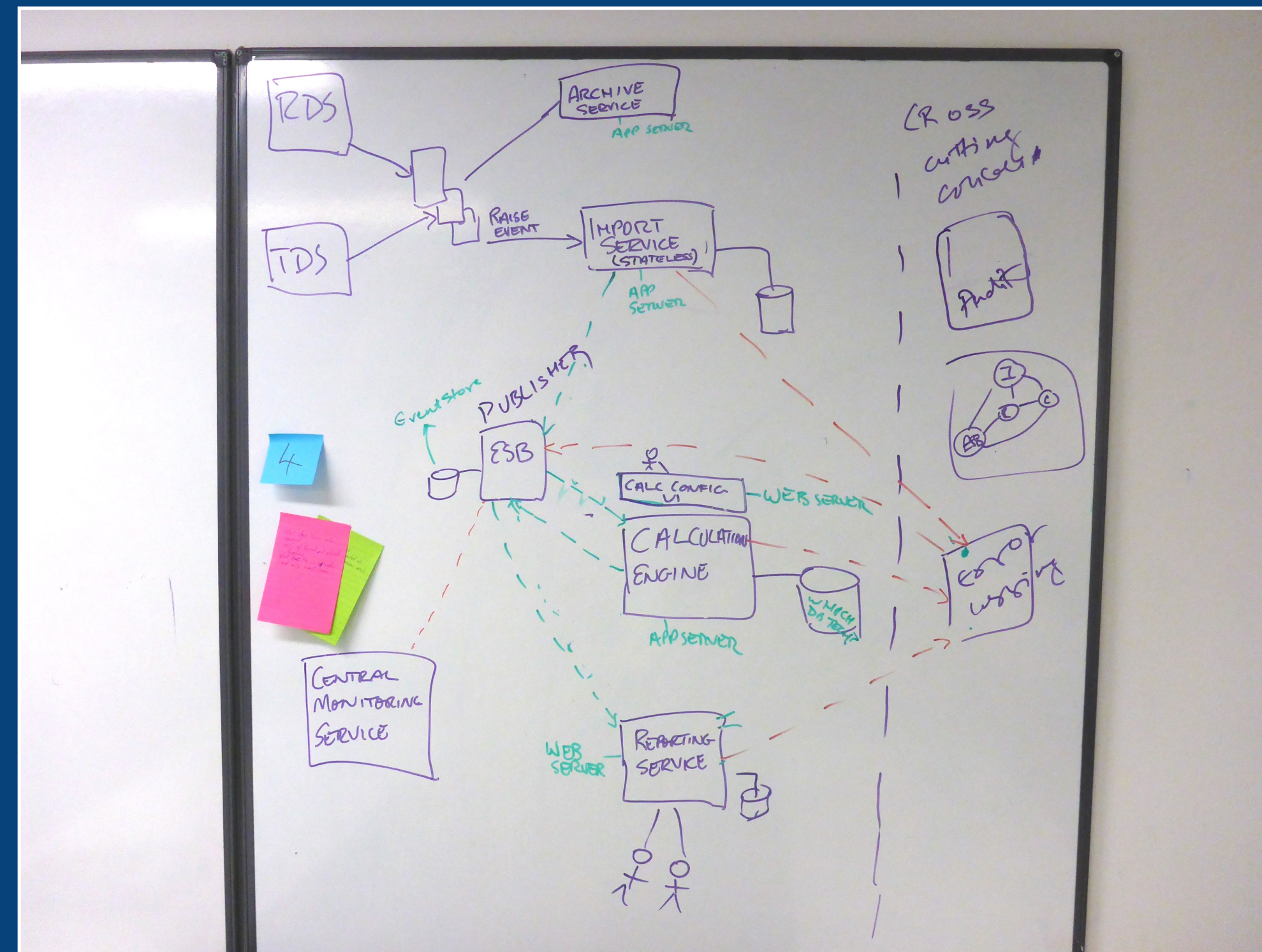


“just use a whiteboard; the value is in the conversation”





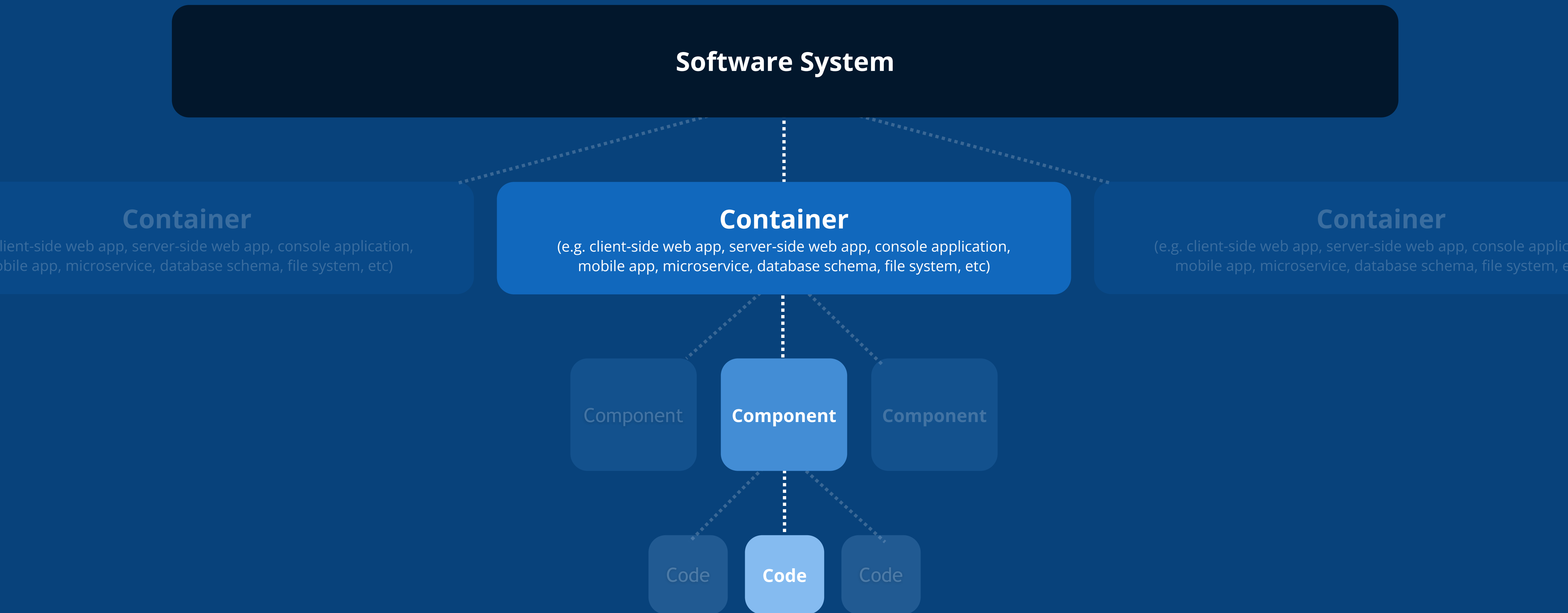
1. Is that what we're going to build?



2. Is it going to work?

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

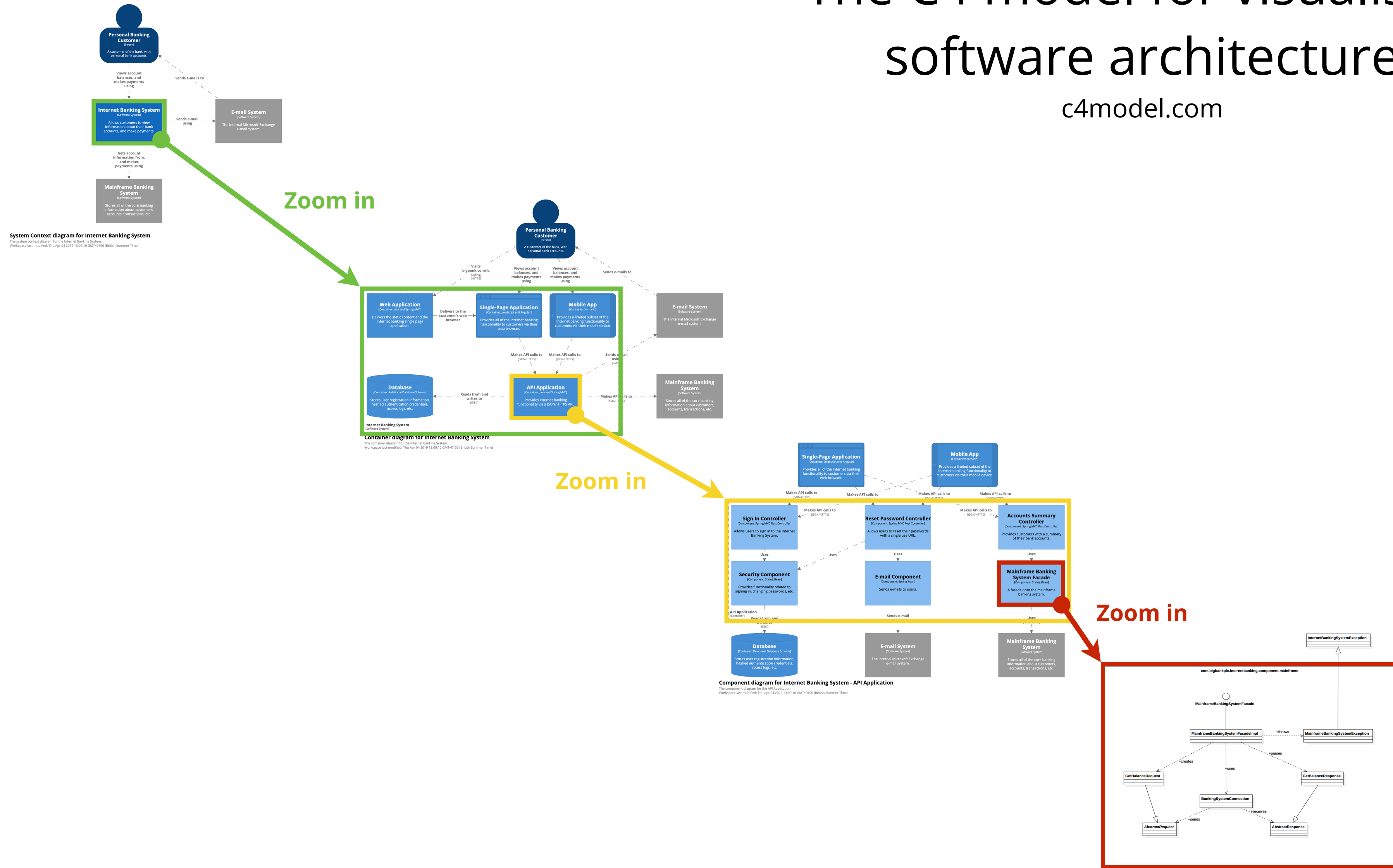
A common set of abstractions
is more important
than a common notation



A **software system** is made up of one or more **containers** (web applications, mobile apps, desktop applications, databases, file systems, etc), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (e.g. classes, interfaces, objects, functions, etc).

The C4 model for visualising software architecture

c4model.com

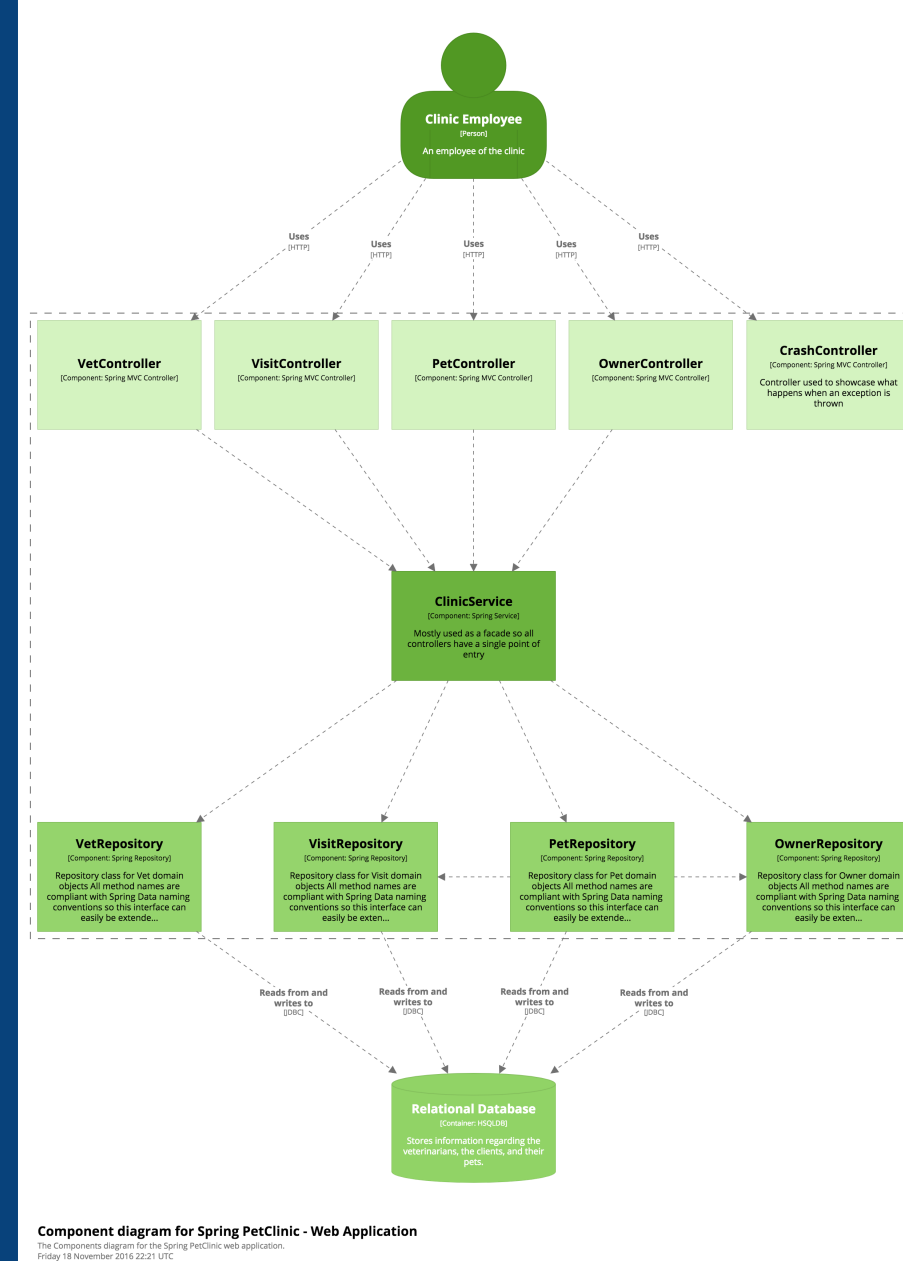
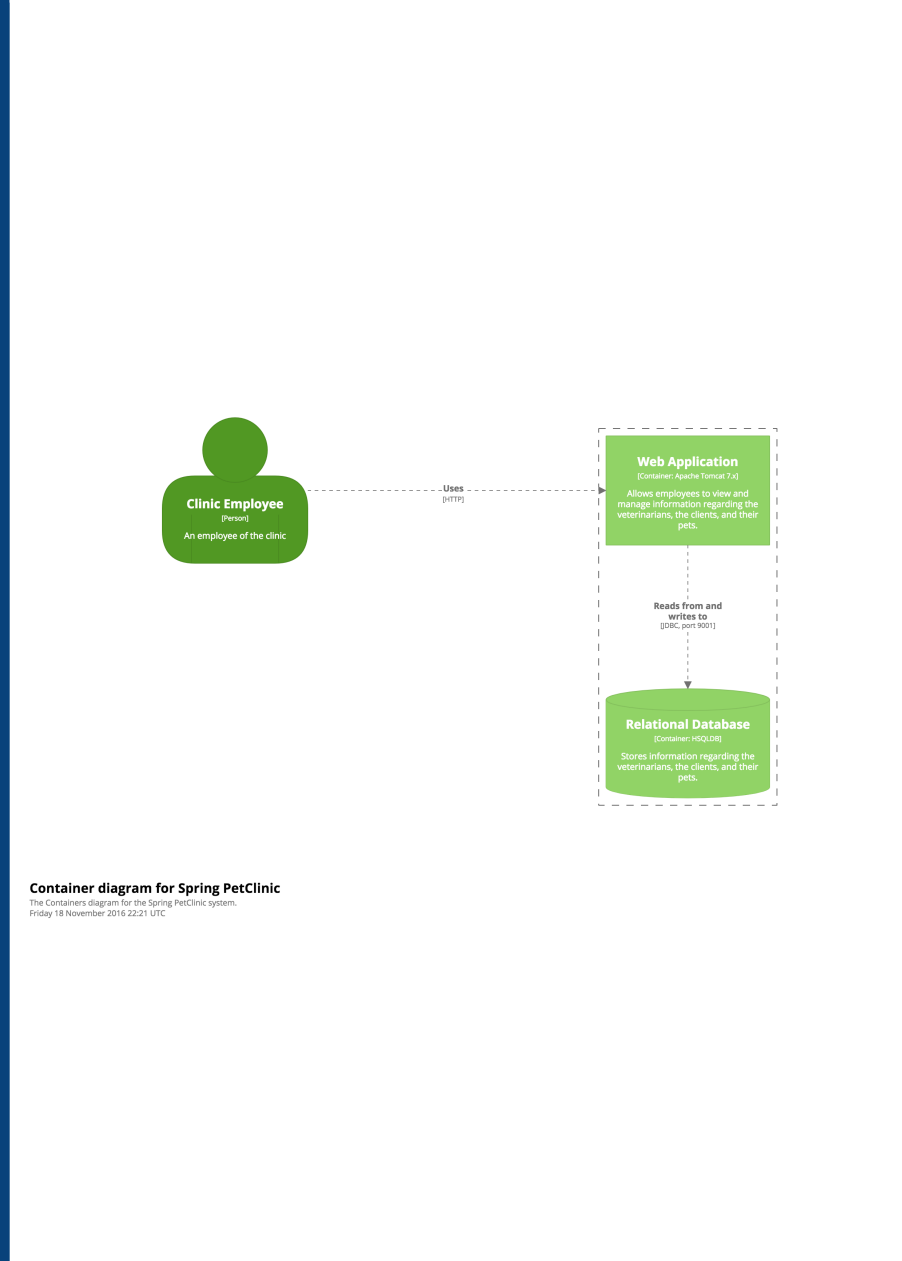
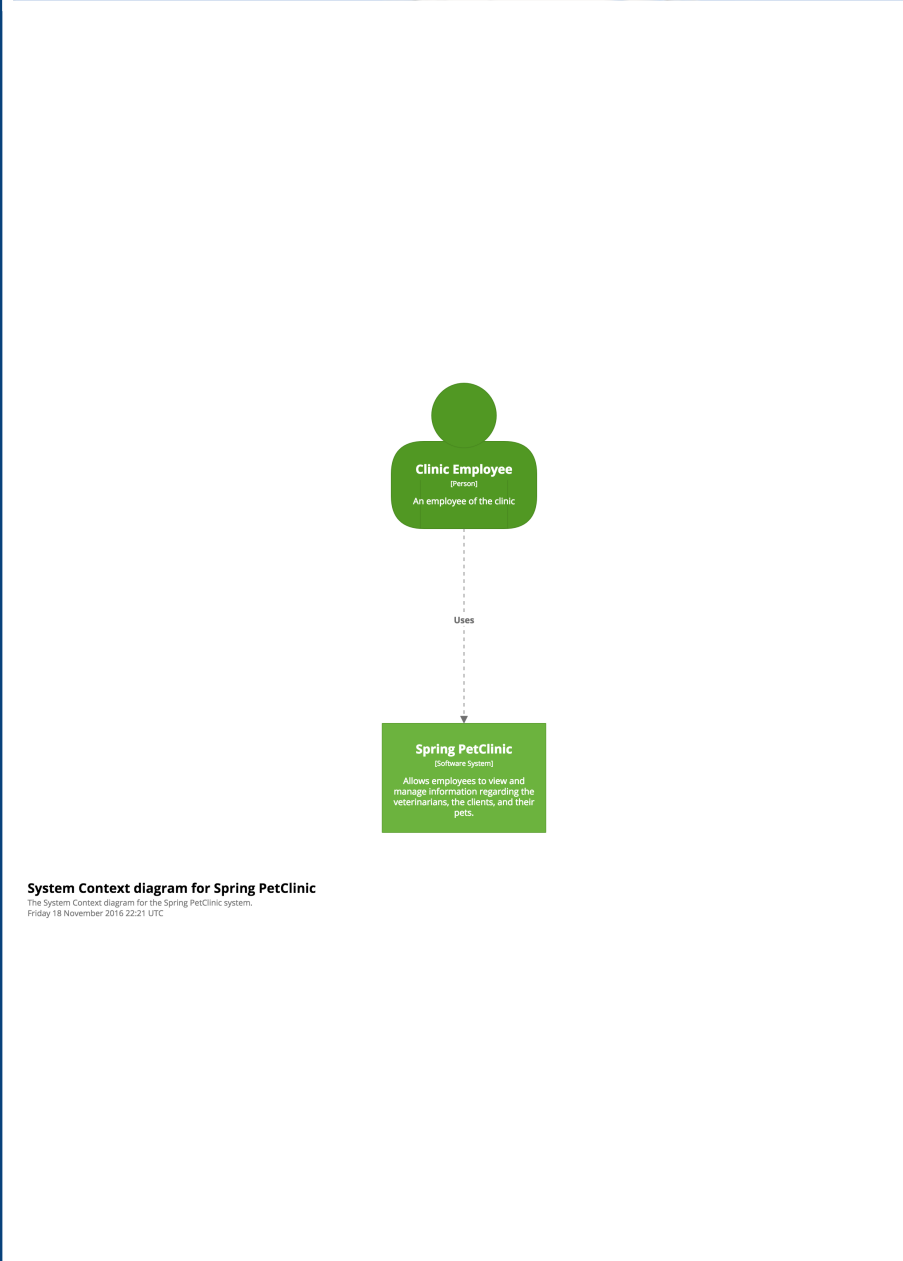


Level 1
Context

Level 2
Containers

Level 3
Components

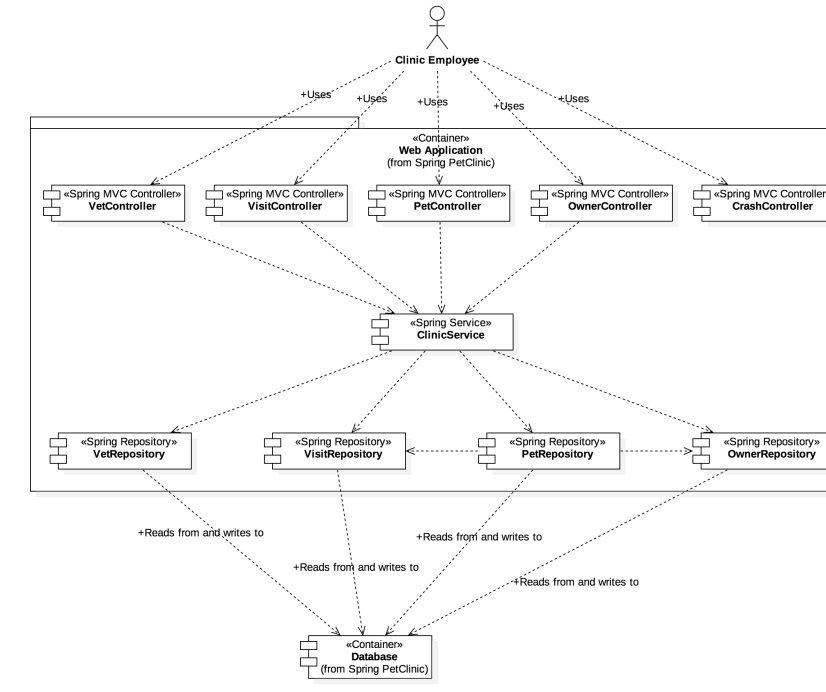
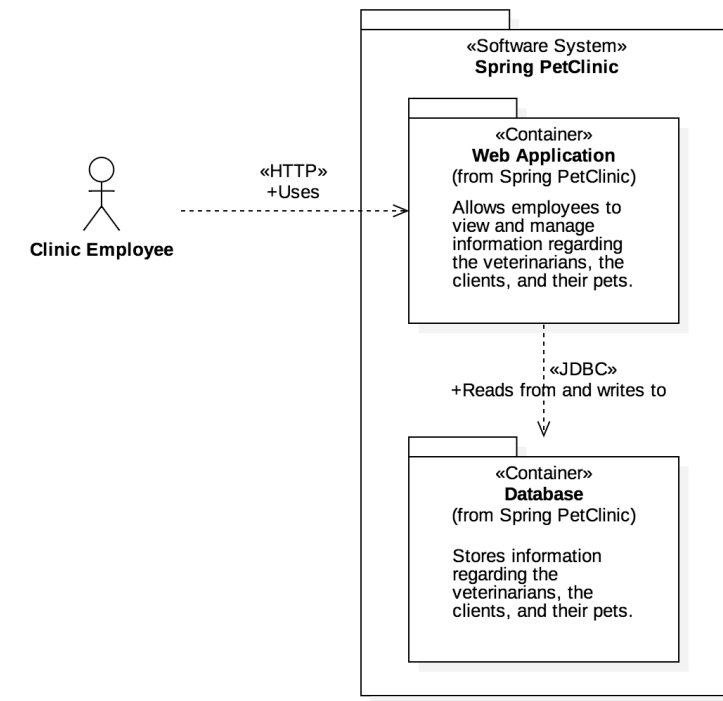
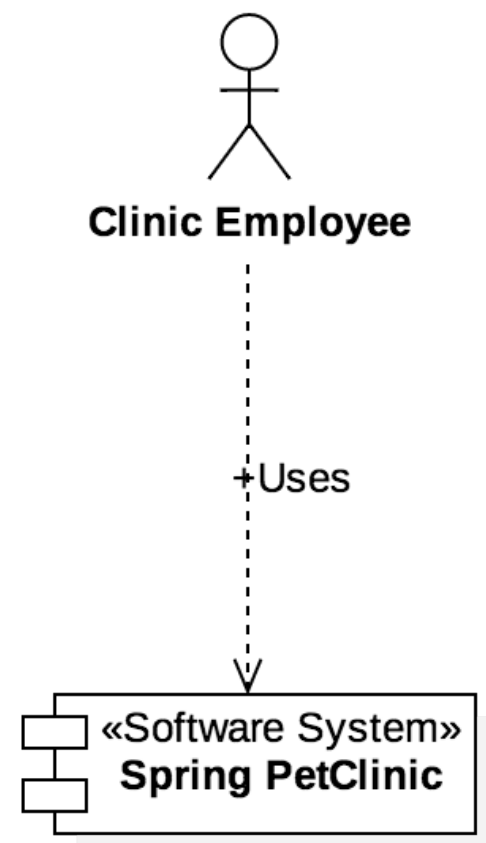
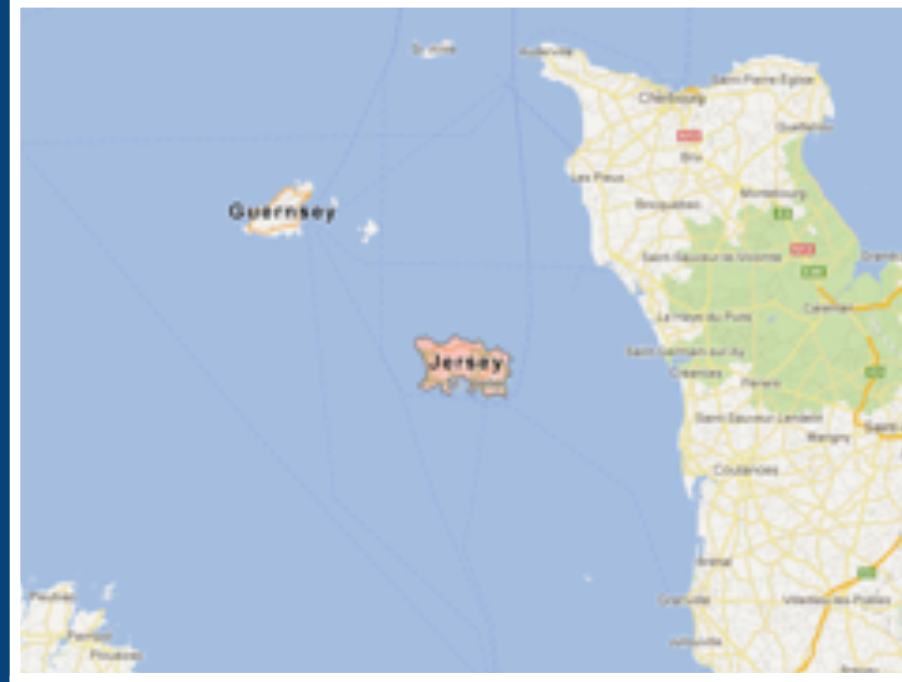
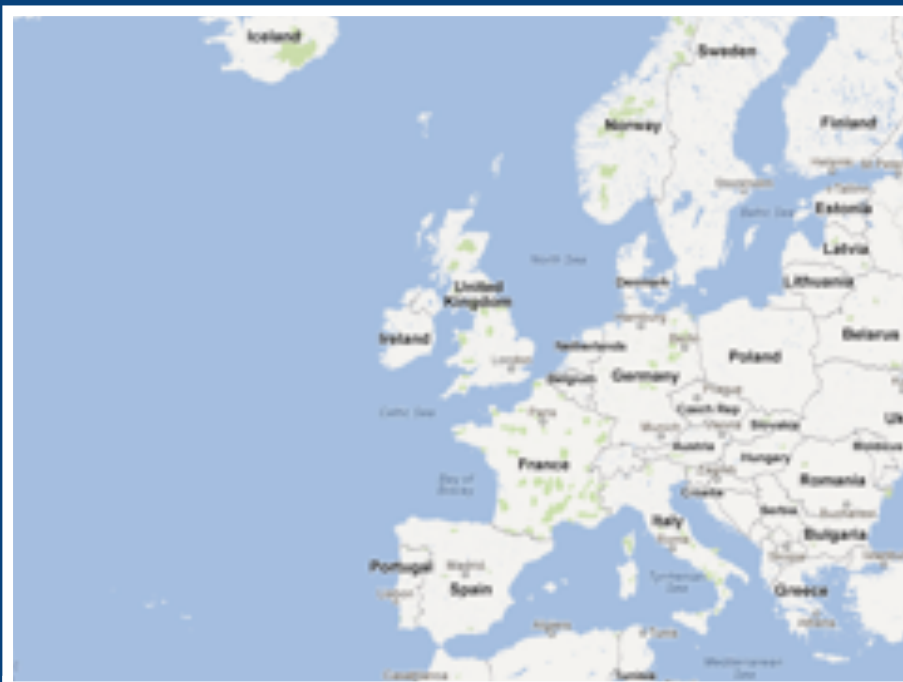
Level 4
Code



```
1 //
2 * Copyright 2002-2013 the original author or authors.
3 * Licensed under the Apache License, Version 2.0 (the "License");
4 * you may not use this file except in compliance with the License.
5 * You may obtain a copy of the License at
6 * http://www.apache.org/licenses/LICENSE-2.0
7 *
8 * Unless required by applicable law or agreed to in writing, software
9 * distributed under the License is distributed on an "AS IS" BASIS,
10 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
11 * See the License for the specific language governing permissions and
12 * limitations under the License.
13
14 package org.springframework.samples.petclinic.service;
15
16 import java.util.Collection;
17
18 import org.springframework.dao.DataAccessExceptions;
19 import org.springframework.samples.petclinic.model.Owner;
20 import org.springframework.samples.petclinic.model.Pet;
21 import org.springframework.samples.petclinic.model.PetTypes;
22 import org.springframework.samples.petclinic.model.Vet;
23 import org.springframework.samples.petclinic.model.Visit;
24
25
26
27
28
29 // Mostly used as a facade so all controllers have a single point of entry
30
31
32
33 public interface ClinicService {
34
35     Collection<PetType> findPetTypes() throws DataAccessExceptions;
36
37     Owner findById(int id) throws DataAccessExceptions;
38
39     Pet findById(int id) throws DataAccessExceptions;
40
41     void savePet(Pet pet) throws DataAccessExceptions;
42
43     void saveVisit(Visit visit) throws DataAccessExceptions;
44
45     Collection<Vet> findVets() throws DataAccessExceptions;
46
47     void saveOwner(Owner owner) throws DataAccessExceptions;
48
49     Collection<Owner> findOwnerByLastName(String lastName) throws DataAccessExceptions;
50 }
51 }
```

Diagrams are maps

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



```

    Copyright 2002-2013 the original author or authors.
    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.
    package org.springframework.samples.petclinic.service;
    import java.util.Collection;
    import org.springframework.dao.DataAccessExceptions;
    import org.springframework.samples.petclinic.model.Owner;
    import org.springframework.samples.petclinic.model.Pet;
    import org.springframework.samples.petclinic.model.PetType;
    import org.springframework.samples.petclinic.model.Visit;
    import org.springframework.samples.petclinic.model.Visit;

    /**
     * Mostly used as a facade so all controllers have a single point of entry
     * @author Michael Isvy
     */
    public interface ClinicService {
        Collection<PetType> findPetTypes();
        Owner findOwnerById(int id) throws DataAccessExceptions;
        Pet findPetById(int id) throws DataAccessExceptions;
        void savePet(Pet pet) throws DataAccessExceptions;
        void saveVisit(Visit visit) throws DataAccessExceptions;
        Collection<Visit> findVisits() throws DataAccessExceptions;
        void saveOwner(Owner owner) throws DataAccessExceptions;
        Collection<Owner> findOwnersByName(String lastName) throws DataAccessExceptions;
    }
  
```

Diagrams are maps

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

Diagrams are a visual checklist
for design decisions

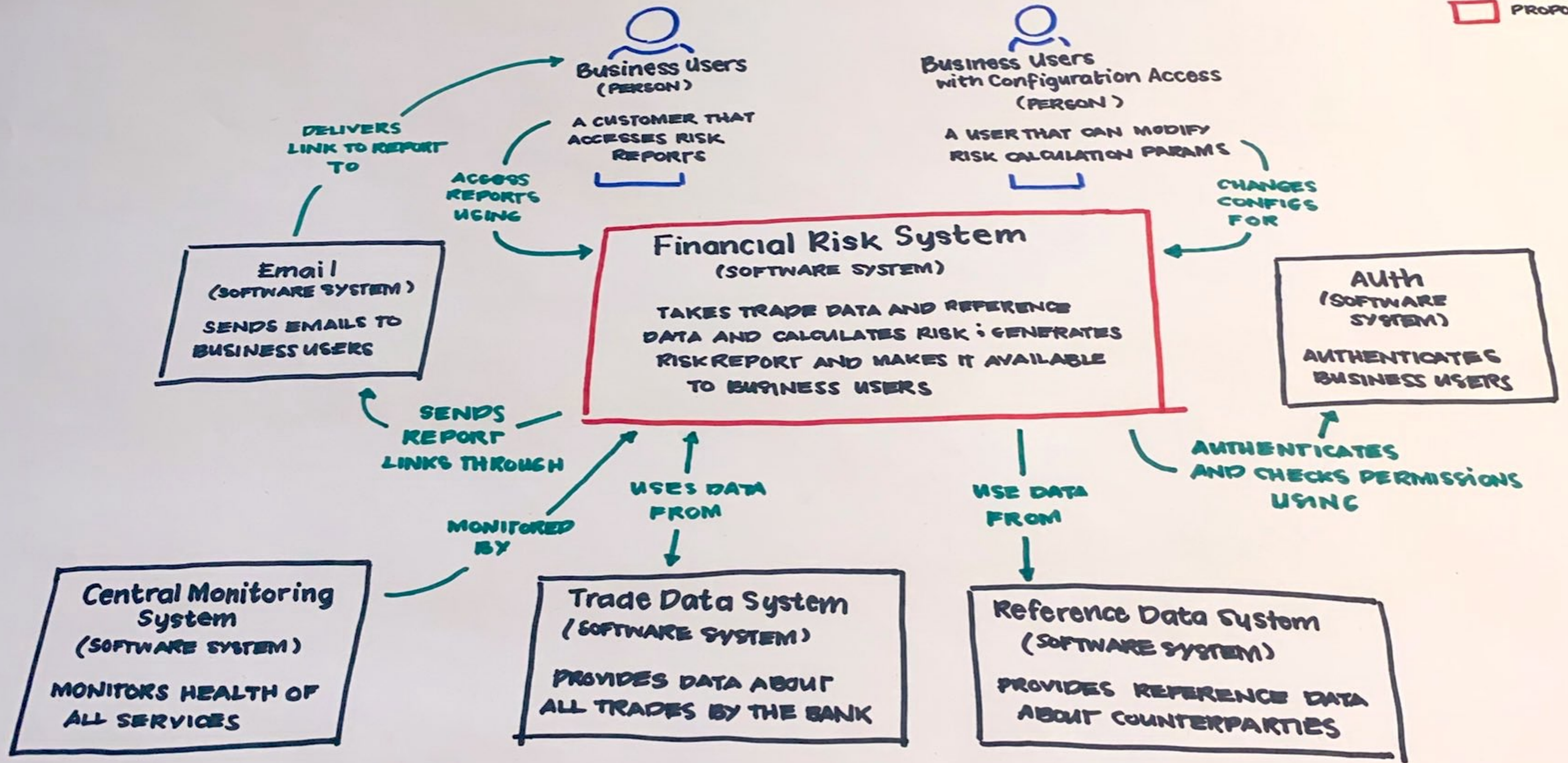
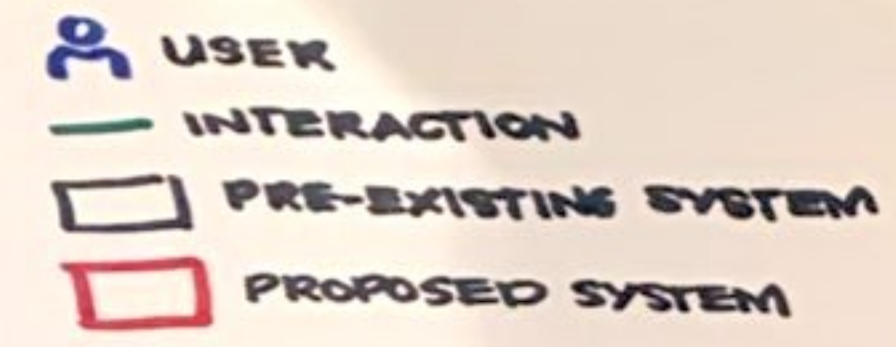
System Context diagram

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

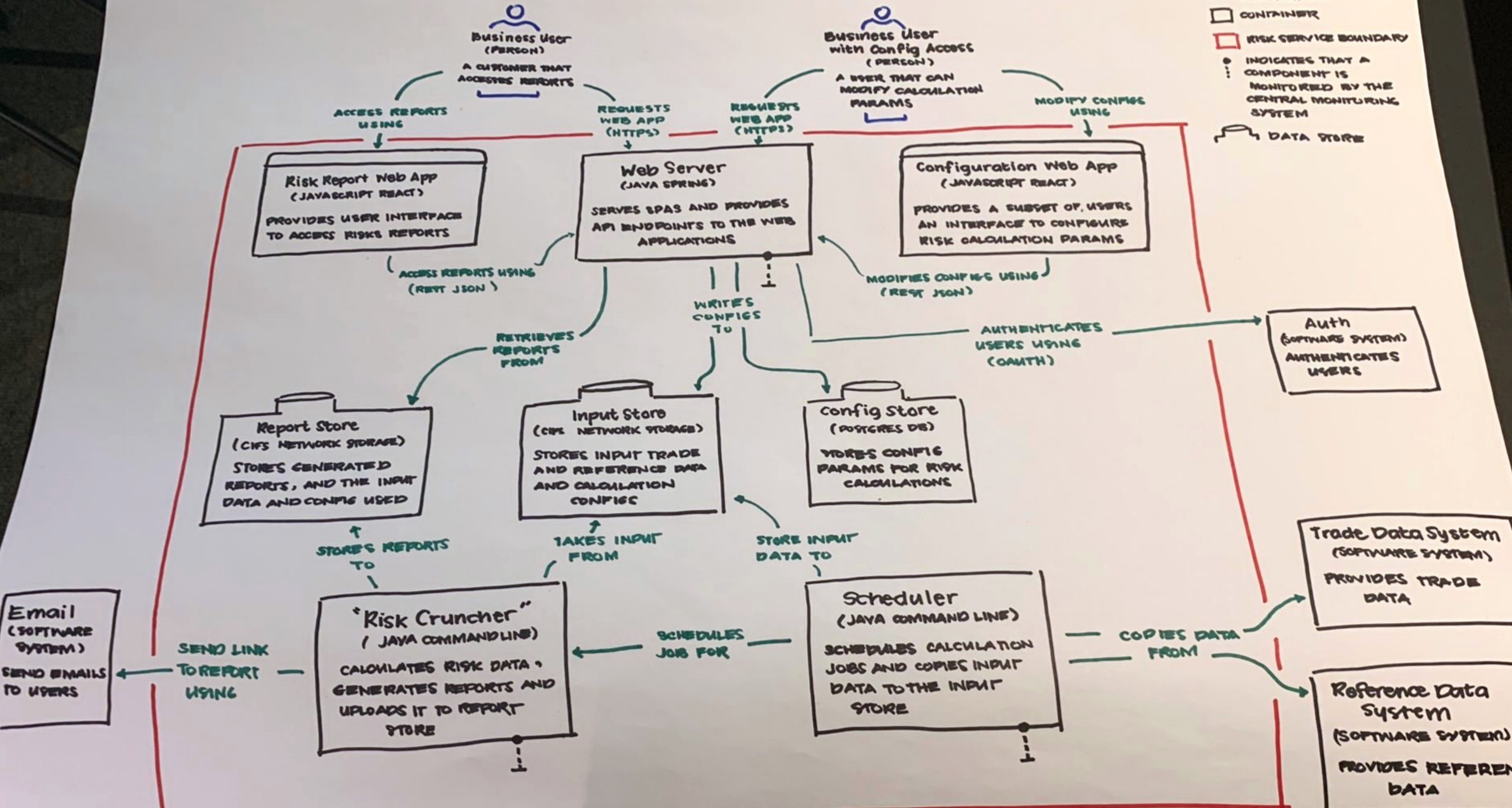
What are the major technology building blocks?

What are their responsibilities?

How do they communicate?

Financial Risk System: Container Diagram

- USER
- INTERACTION
- CONTAINER
- RISK SERVICE BOUNDARY
- INDICATES THAT A COMPONENT IS MONITORED BY THE CENTRAL MONITORING SYSTEM
- DATA STORE



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?”

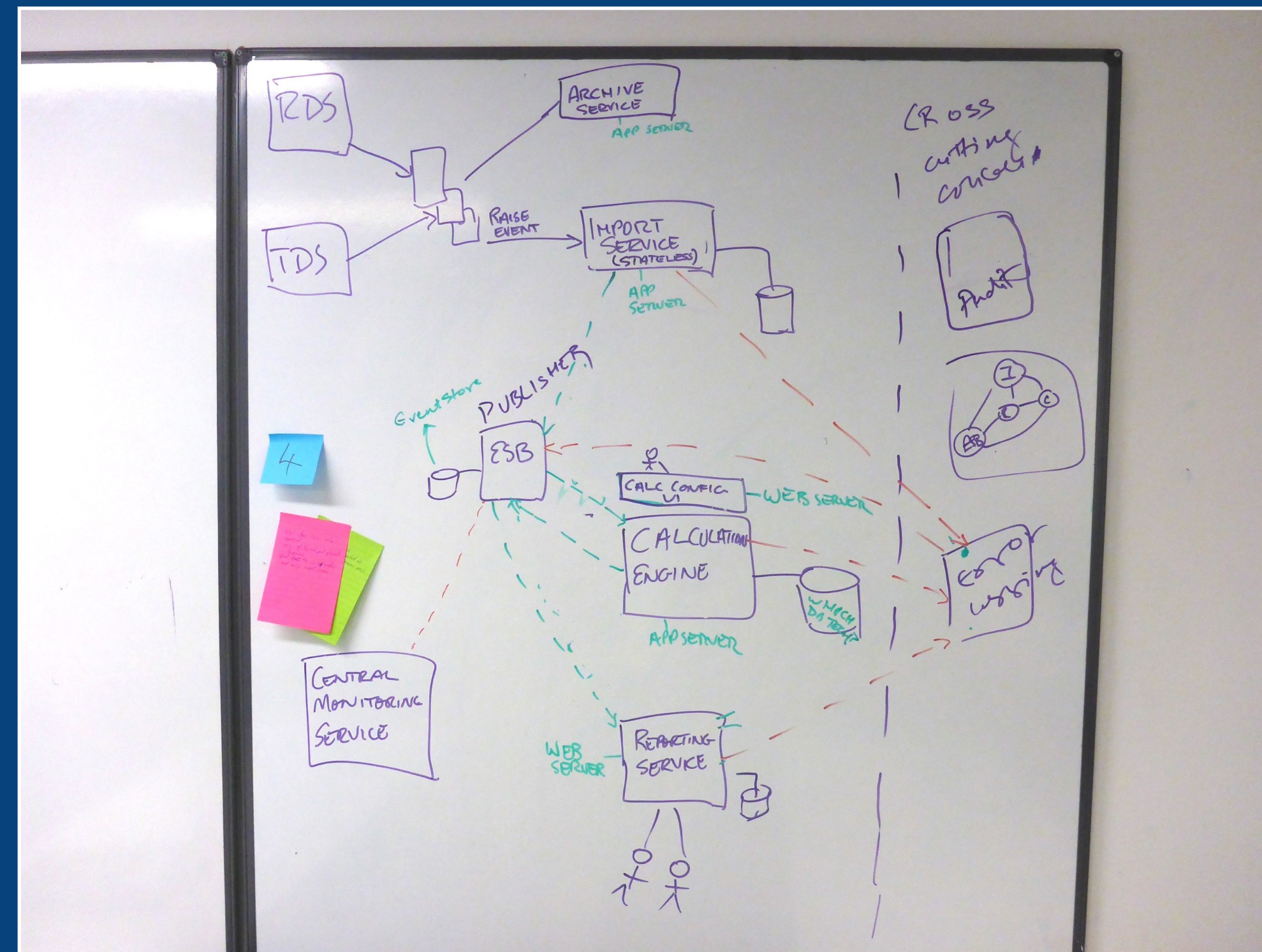
Richer diagrams lead to
richer **design discussions**

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?

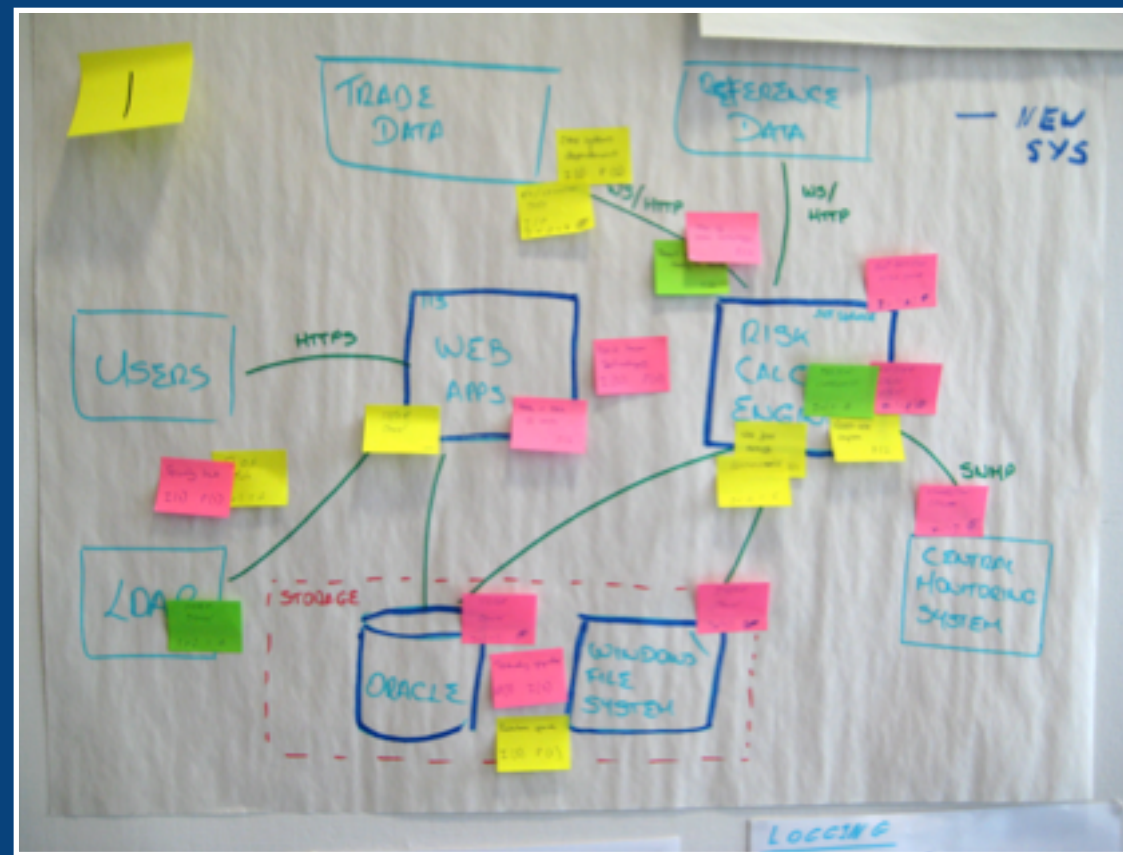
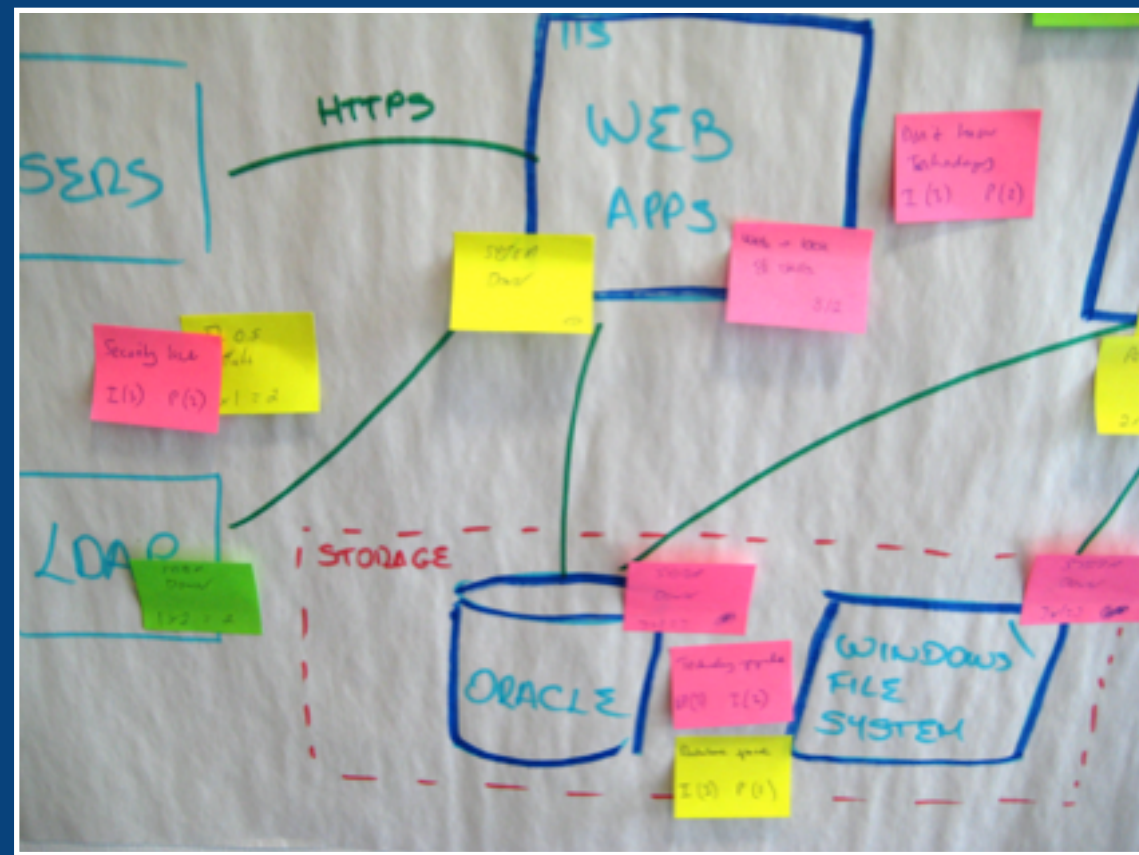
Base your architecture on requirements, travel light and prove your architecture with concrete experiments.

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

#52

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

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



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

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



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

You are confident that your design satisfies the key architectural drivers.



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

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**

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”

Adopt an agile mindset

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

Thank you!

Simon Brown