coding {the} architecture

Firm Foundations

Simon Brown
@simonbrown

Jersey

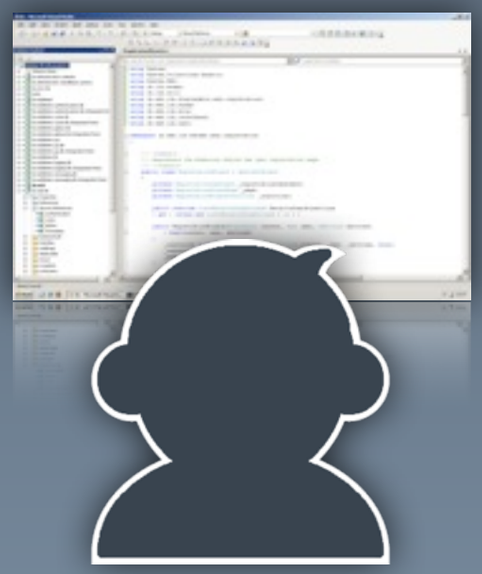
JERSEY FINANCE
VOICE OF THE INTERNATIONAL FINANCE CENTRE

DIGITAL JERSEY

Locate Jersey

I help software teams understand

software architecture, technical leadership and the balance with agility



I code too

⇧ ; - ⇧ 0

Software architecture needs to be more accessible

A developer-friendly guide to software architecture, technical leadership and the balance with agility

Leanpub

10 out of 10

"Highly recommended reading"

Junilu Lacar, JavaRanch

http://leanpub.com/software-architecture-for-developers

In previous ASAS editions...

2012

The conflict between agile and architecture

The conflict between agile and architecture

Myth or reality?

1. A conflict in team structure

vs

Dedicated
software architect

Single point of responsibility for the technical aspects of the software project

Everybody is a
software architect

Joint responsibility for the technical aspects of the software project

2. A conflict in process

VS

```
/// <summary>
/// Represents the behaviour behind the ...
/// </summary>
public class SomeWizard : AbstractWizard
{
    private DomainObject _object;
    private WizardPage _page;
    private WizardController _controller;

    public SomeWizard()
    {
    }

    ...

}
```

Big up front design

Requirements capture, analysis and design complete before coding starts

Evolutionary architecture

The architecture evolves secondary to the value created by early regular releases of working software

2013

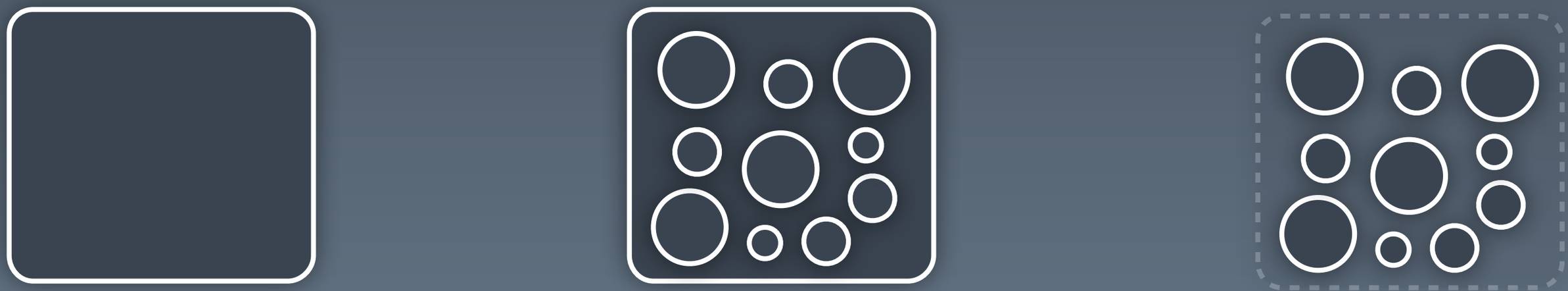
Agility and the essence of software architecture

A good architecture enables agility

Monolithic architecture

Service-based architecture

(SOA, micro-services, etc)



Something in between

(components)

2014

Firm foundations

“Big design up front
is dumb.

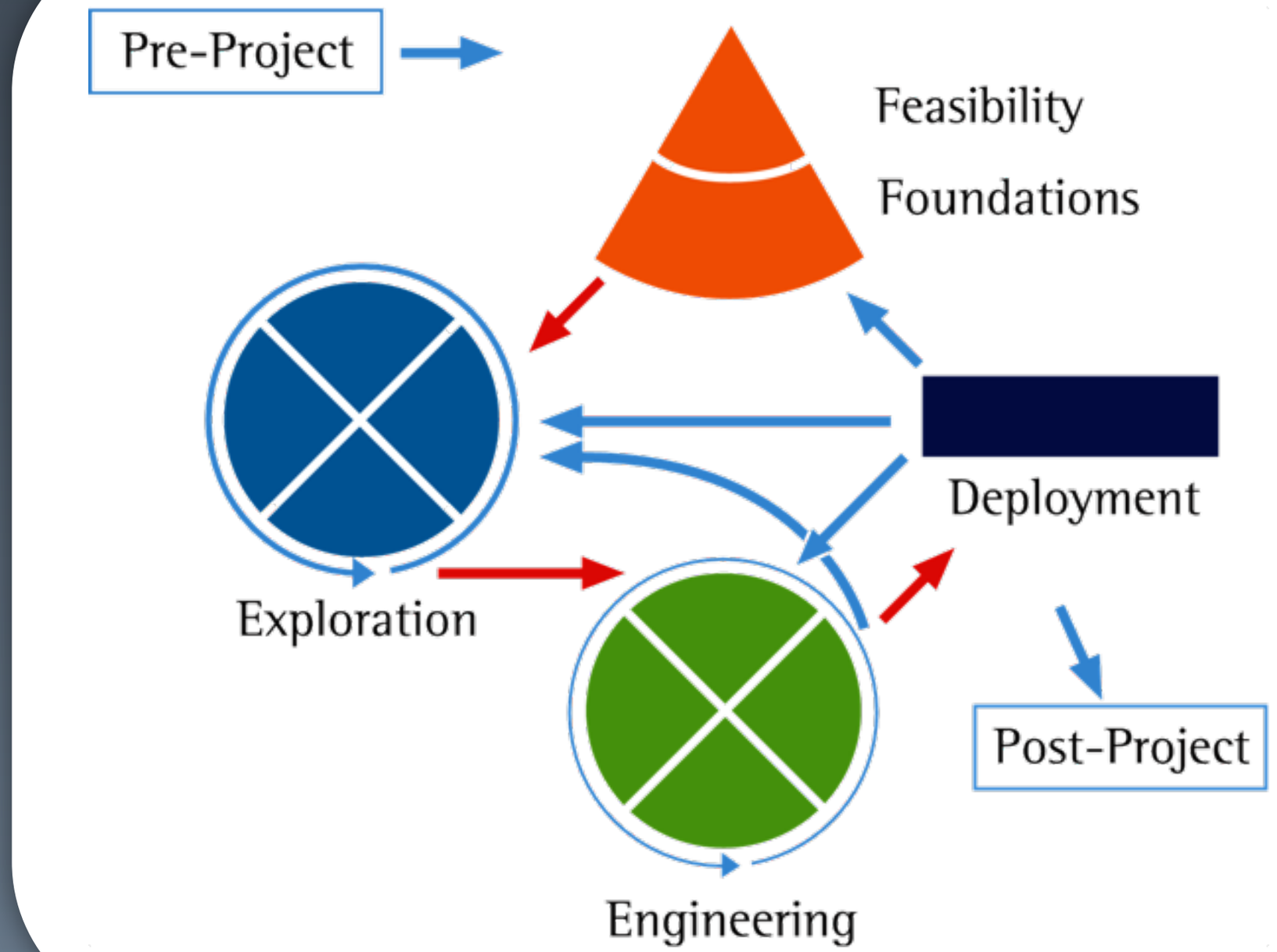
Doing no design up front
is even dumber.”

(Dave Thomas)

A design revival?

Beyond Scrum

DSDM Atern
Disciplined Agile Delivery (DAD)
Scaled Agile Framework (SAFe)



“The Atern lifecycle process”

(from http://dsdm.org/content/6-lifecycle)

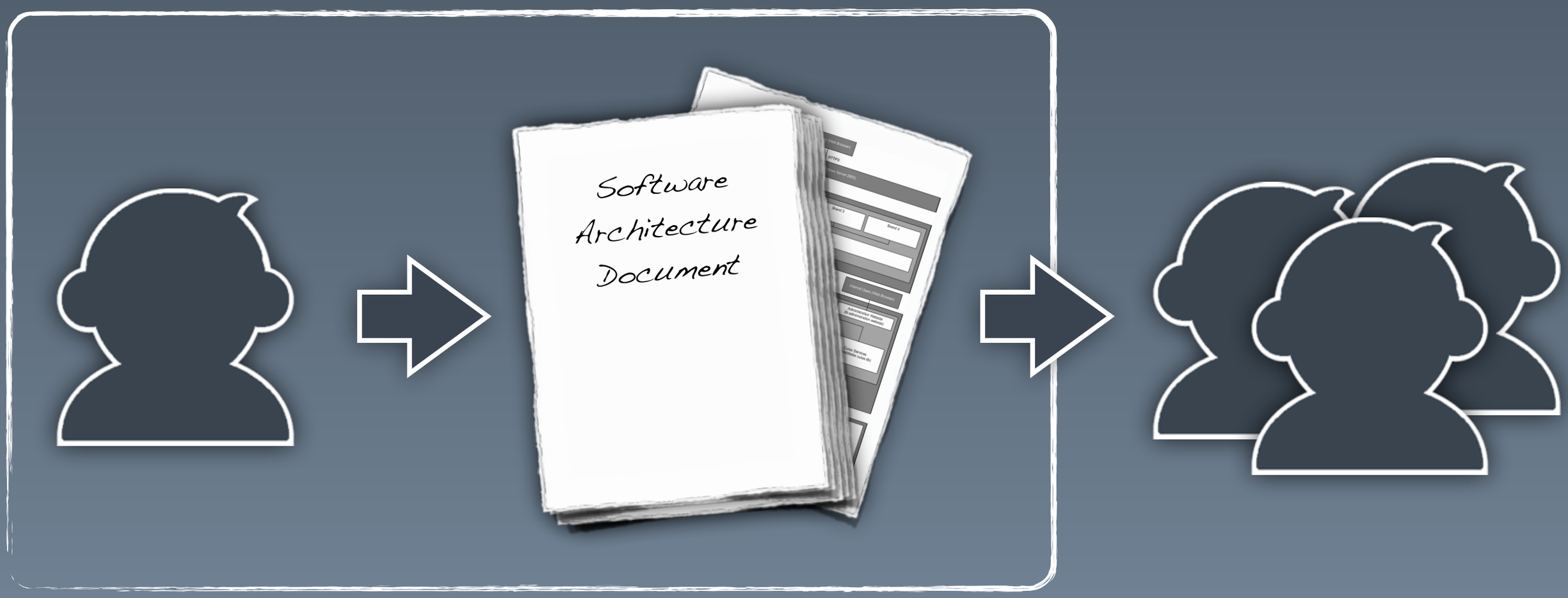
The Foundations phase is aimed at establishing

firm and enduring foundations

for the project.

Software development is not a

relay sport



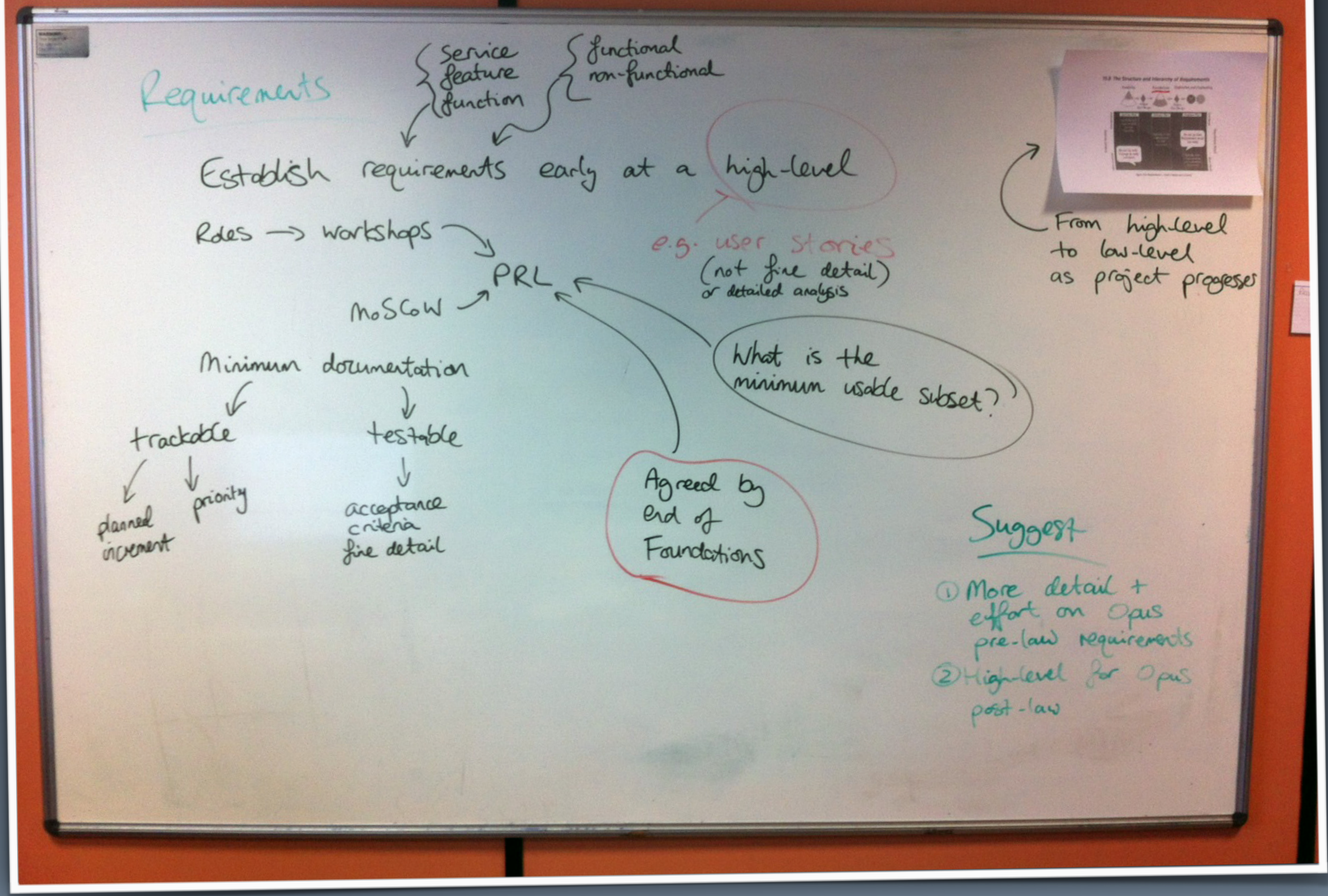
AaaS ... architecture as a service

It’s **not** about creating a perfect end-state, framework or complete architecture

You need a

starting

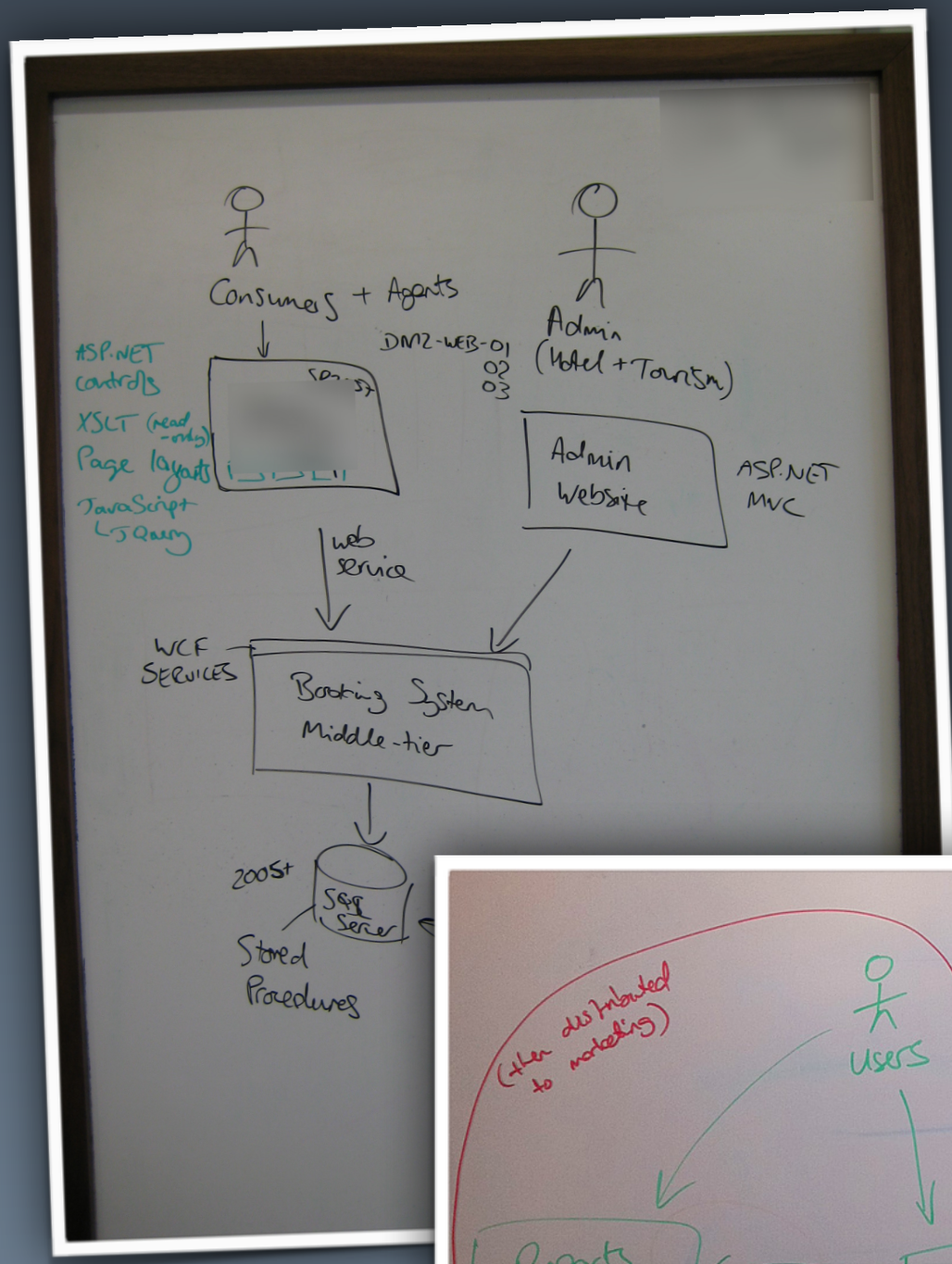
point

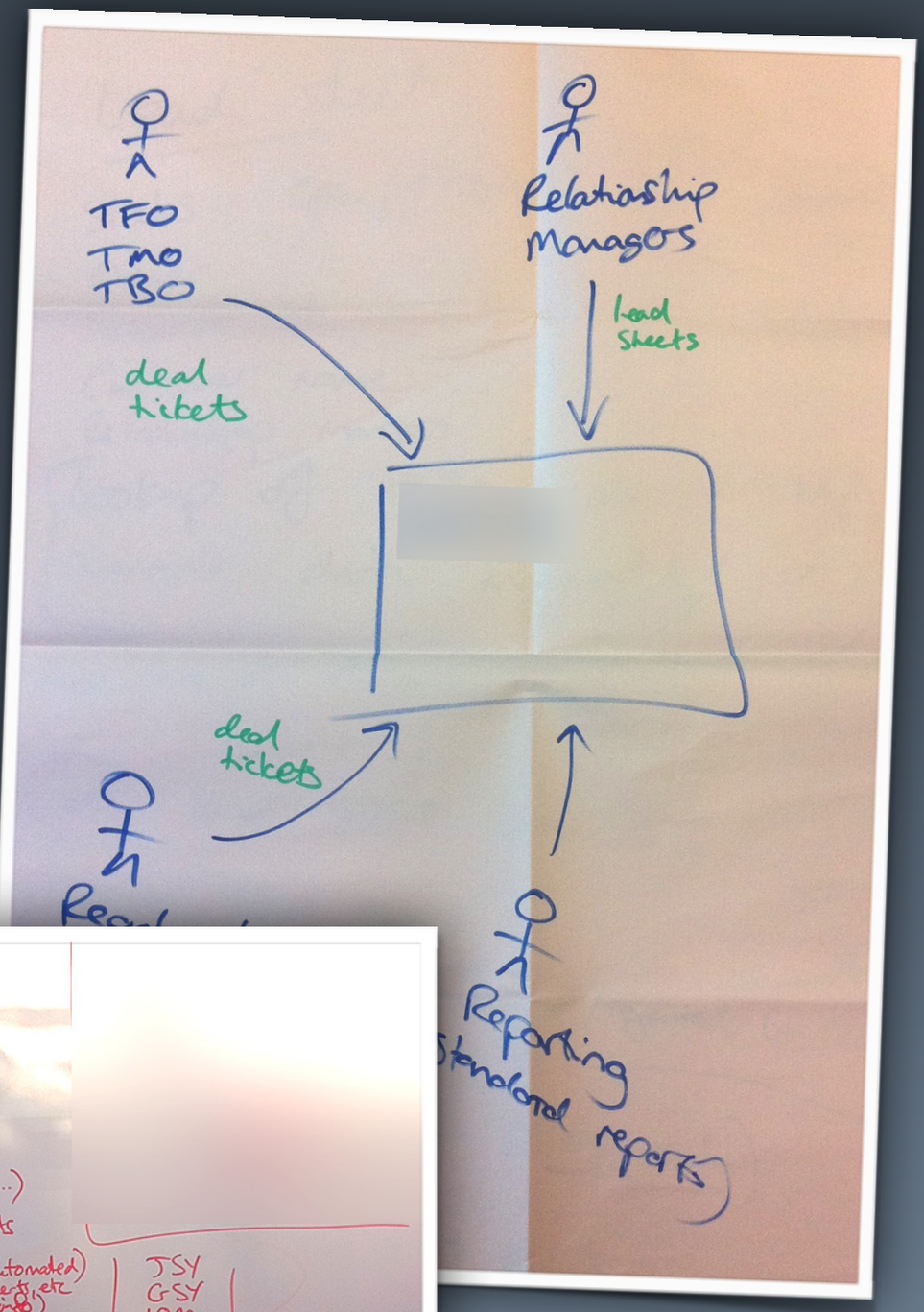


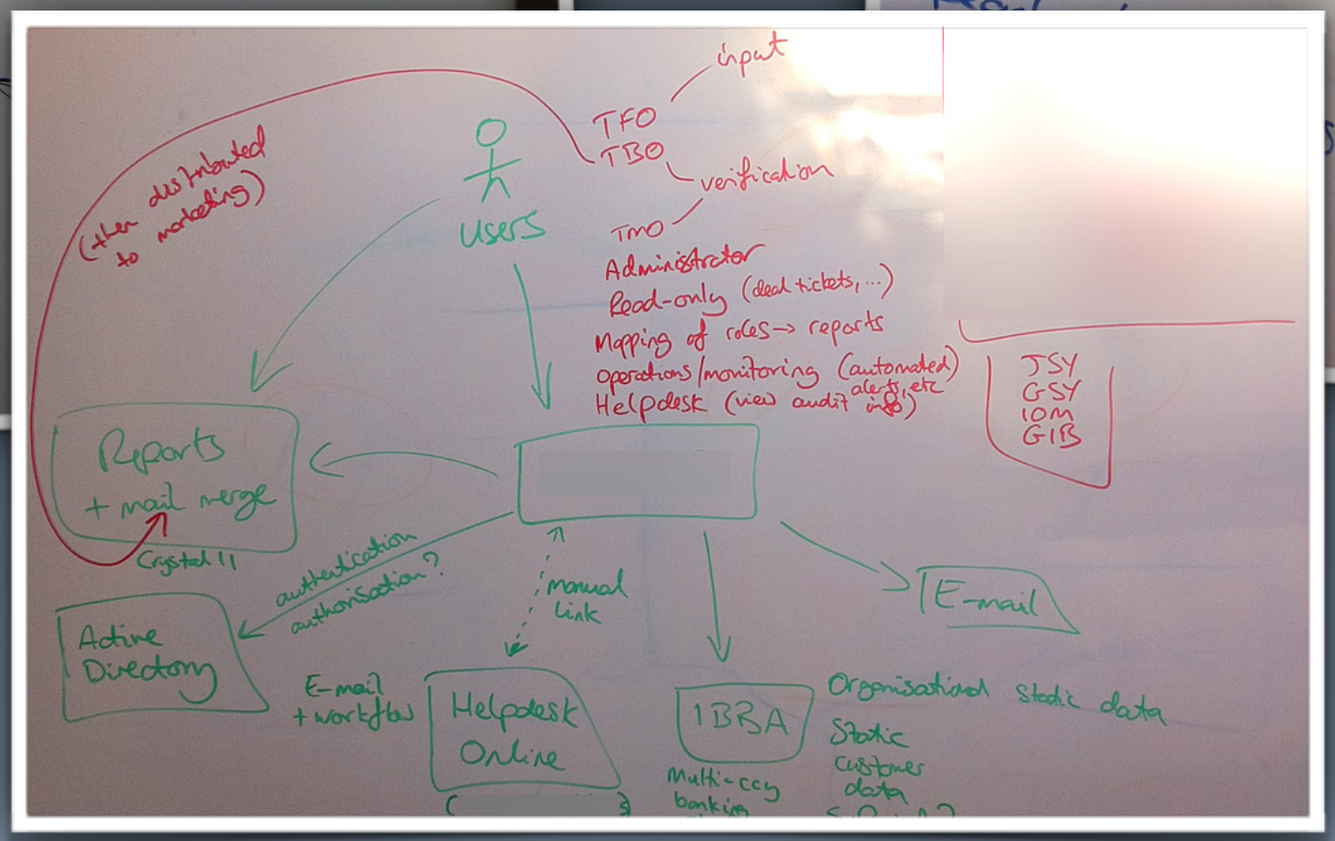
I often need to be the process guy…

Structure

Whiteboards and context diagrams


Consumers + Agents
Admin
(Hotel + Tourism)
DMZ-WEB-01
02
03
ASP.NET
controls
XSLT (read-only)
Page layouts
JavaScript
jQuery
Admin
Website
ASP.NET
MVC
web
service
WCF
SERVICES
Booking System
Middle-tier
2005+
SQL
Server
Stored
Procedures

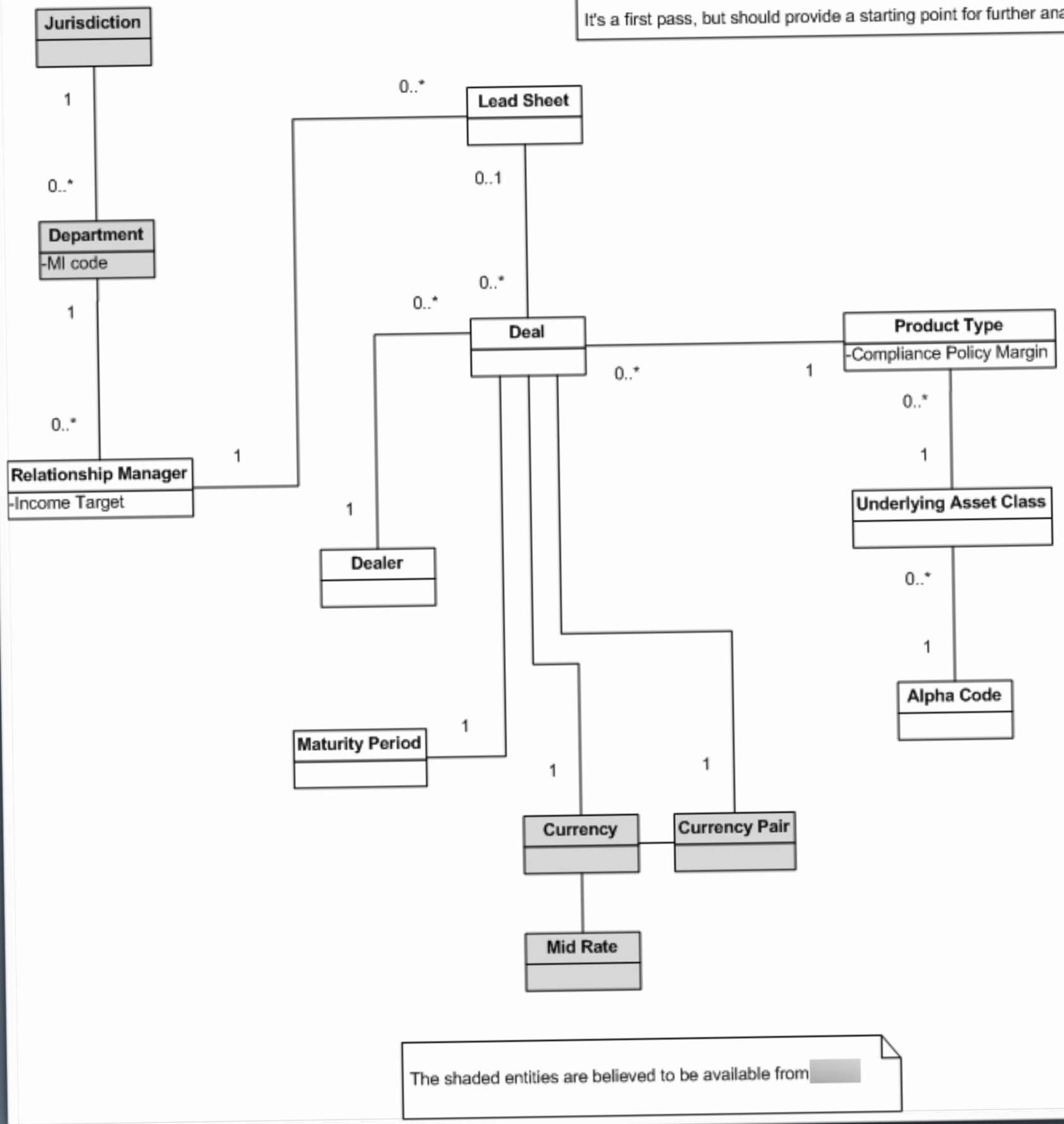

TFO
TMO
TBO
deal
tickets
Relationship
Managers
load
sheets
deal
tickets
Reporting
Standard reports


input
TFO
TBO
verification
TMO
Administrator
Read-only (deal tickets, ...)
Mapping of roles → reports
operations/monitoring (automated alerts, etc
Helpdesk (view audit info)
Users
(then distributed to marketing)
JSY
GSY
IOM
GIB
Reports
+ mail merge
Crystal ??
authentication
authorisation?
Active
Directory
manual
link
E-mail
E-mail
+ workflow
Helpdesk
Online
IBBA
Multi-ccy
banking
Organisational static data
Static
customer
data

Whiteboards and domain diagrams

This is a summary of the major entities and their relationships.

It's a first pass, but should provide a starting point for further analysis.



The shaded entities are believed to be available from

Wireframes

(e.g. Balsamiq)

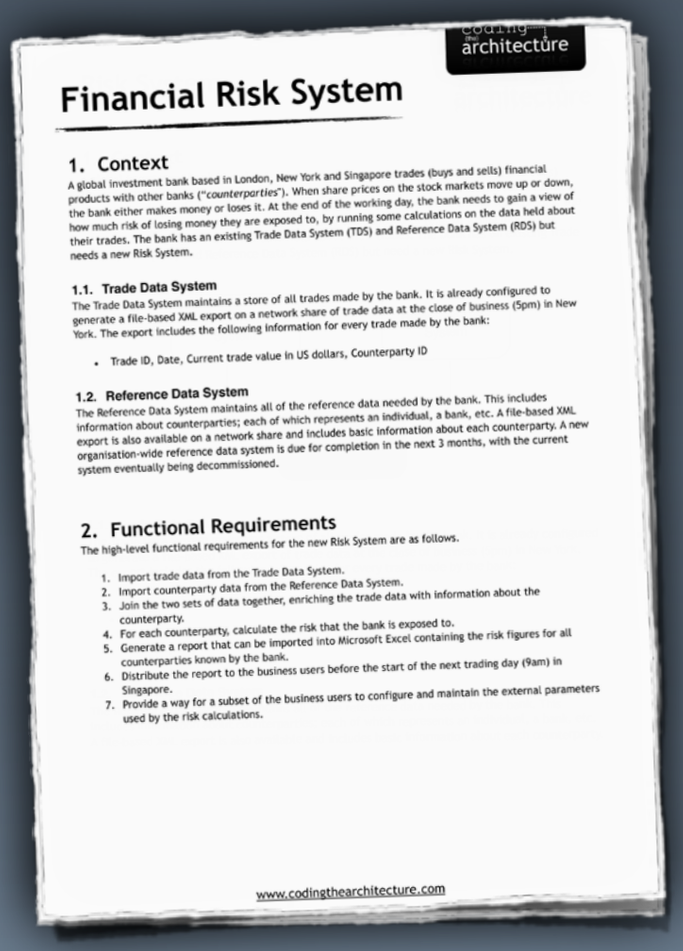
Pair* architecting

* two or more people

We’re trying to design some software;

not create a fluffy, conceptual solution

Should software design be technology agnostic?



Financial Risk System

coding the architecture

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 on a network share of trade data at the close of business (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 maintains all of the reference data needed by the bank. This includes information about counterparties; each of which represents an individual, a bank, etc. A file-based XML export is also available on a network share and includes basic information about each counterparty. A new organisation-wide reference data system is due for completion in the next 3 months, with the current system eventually being decommissioned.

2. Functional Requirements

The high-level functional requirements for the new Risk System are as follows.

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.

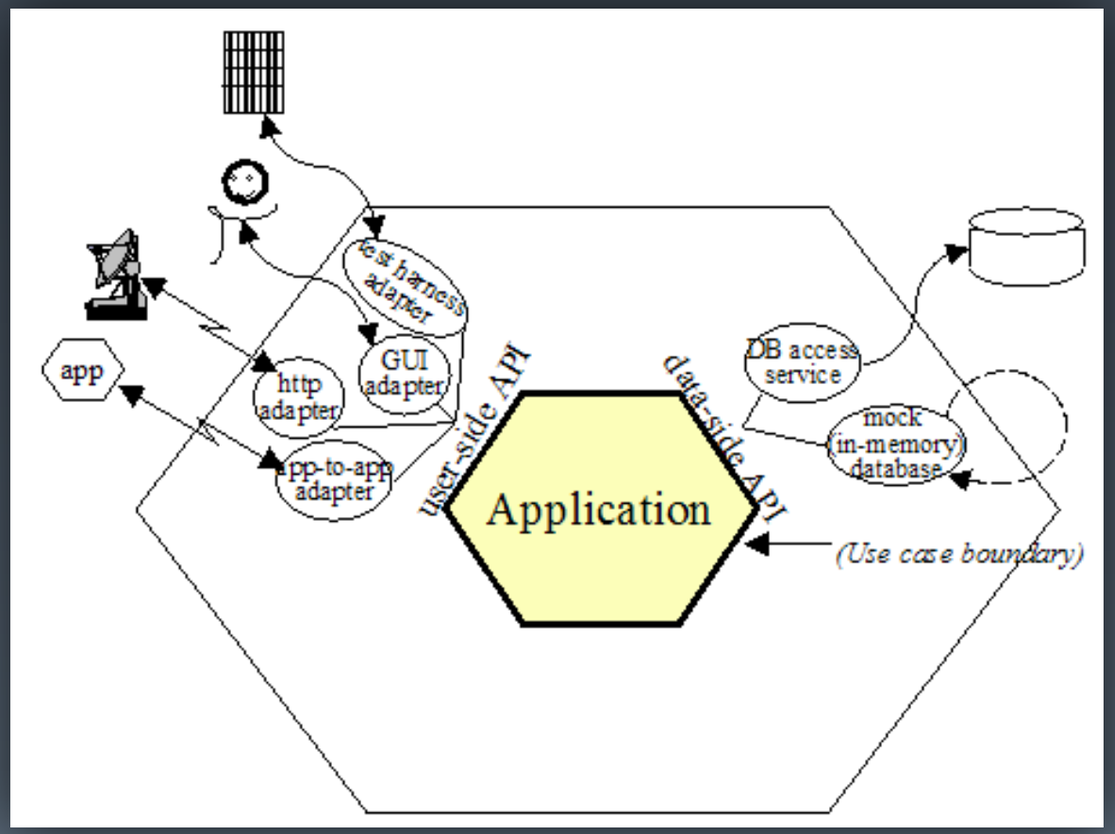
www.codingthearchitecture.com

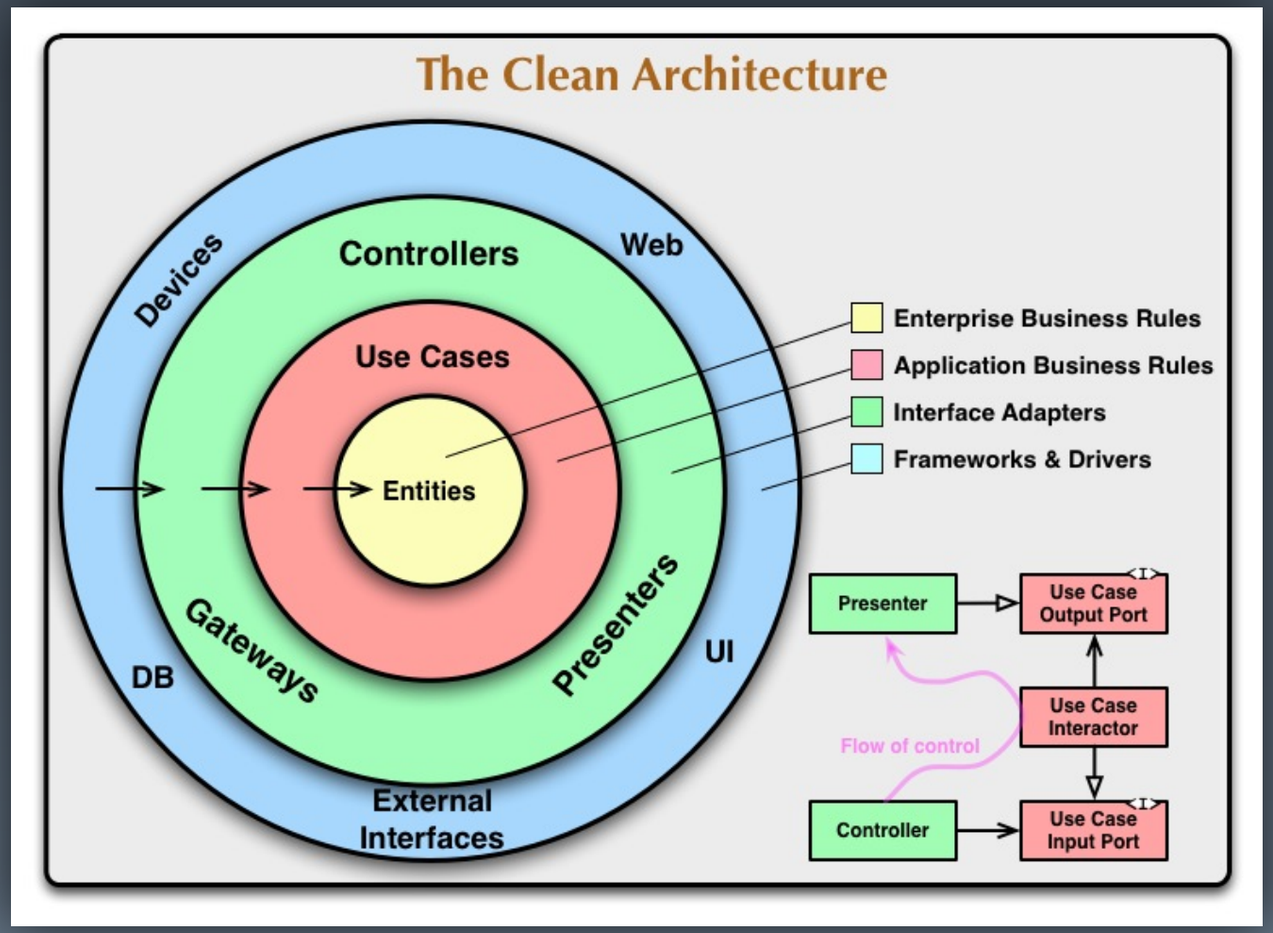
"the solution is simple and can be built with any technology"

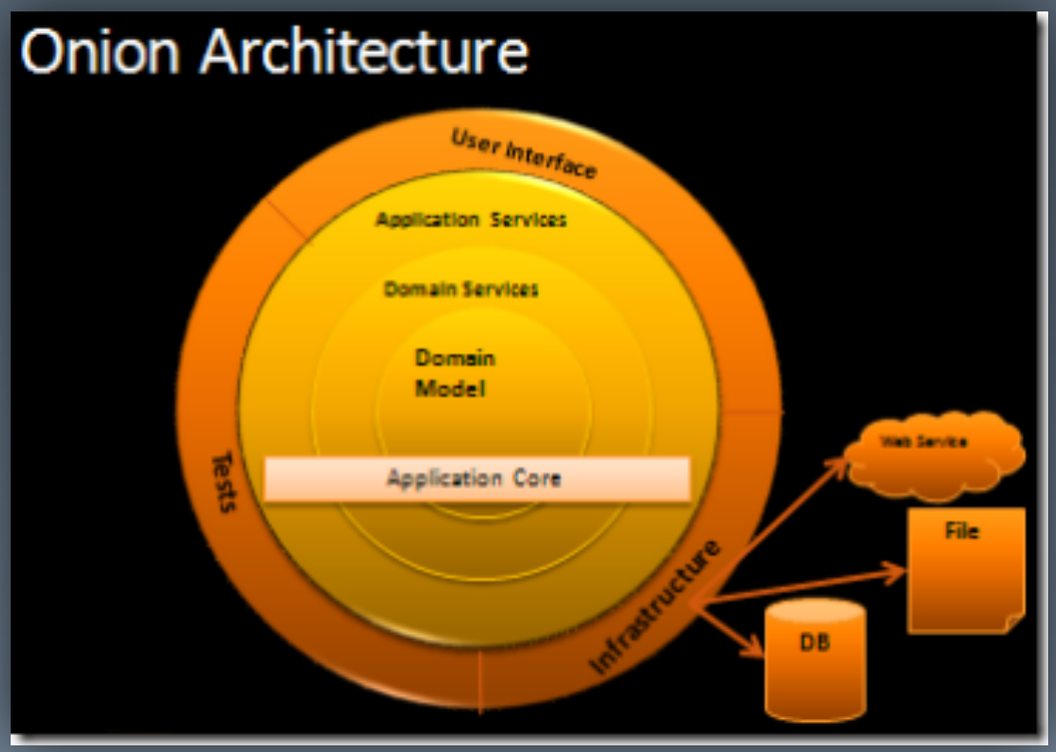
"we don't want to force a solution on developers"

"it's an implementation detail"

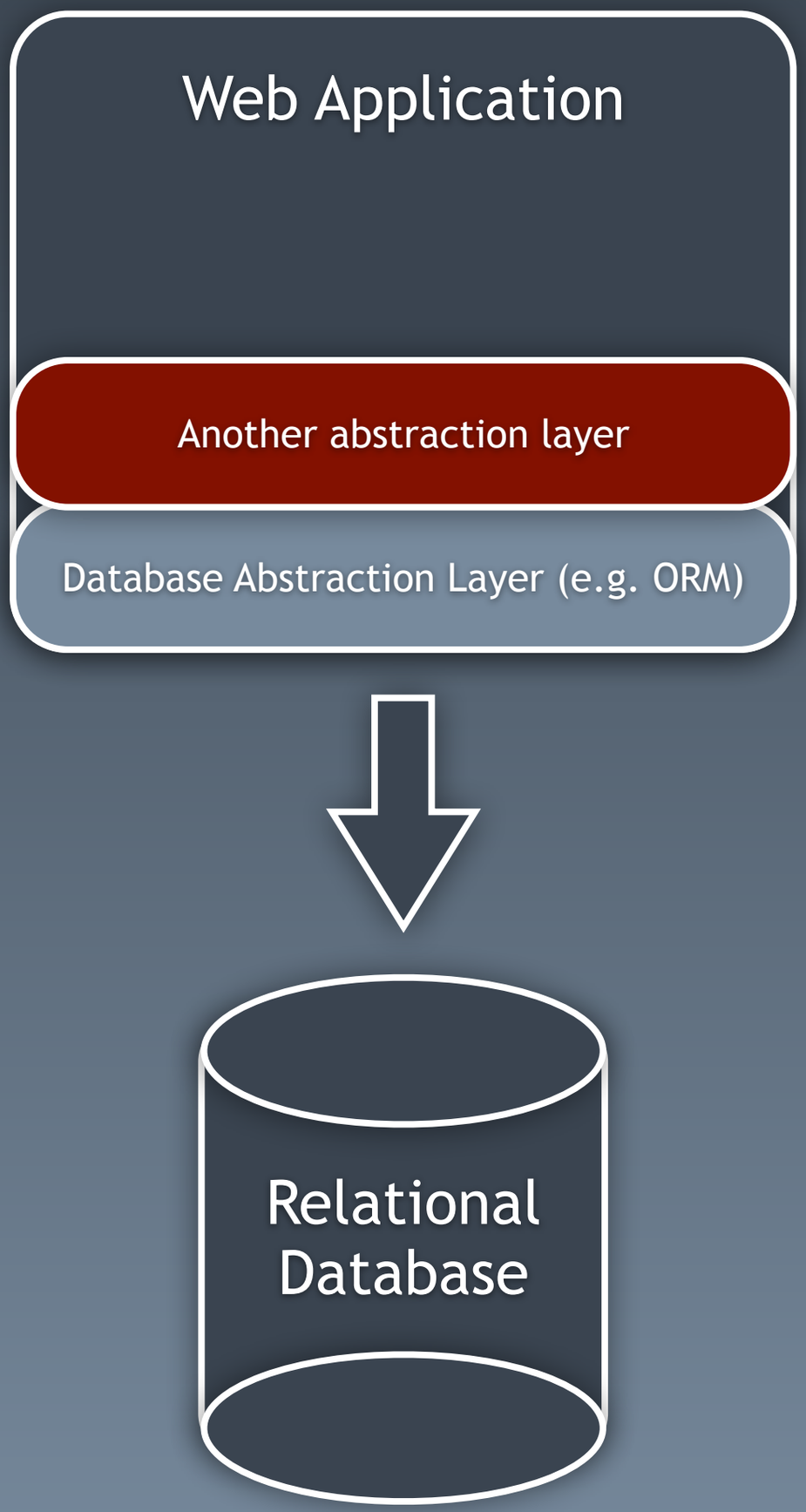
"we follow the 'last responsible moment' principle"







Hexagons and onions



Significant decisions can be moved but rarely eliminated

Technology is

not

an

“implementation detail”

A common set of abstractions

is more important than
a common notation

Software System

iner
tion server, standalone
e, file system, etc)

Container
(e.g. web server, application server, standalone application, browser, database, file system, etc)

Cont
(e.g. web server, applic
application, databa

Component

Component

Component

Class

Class

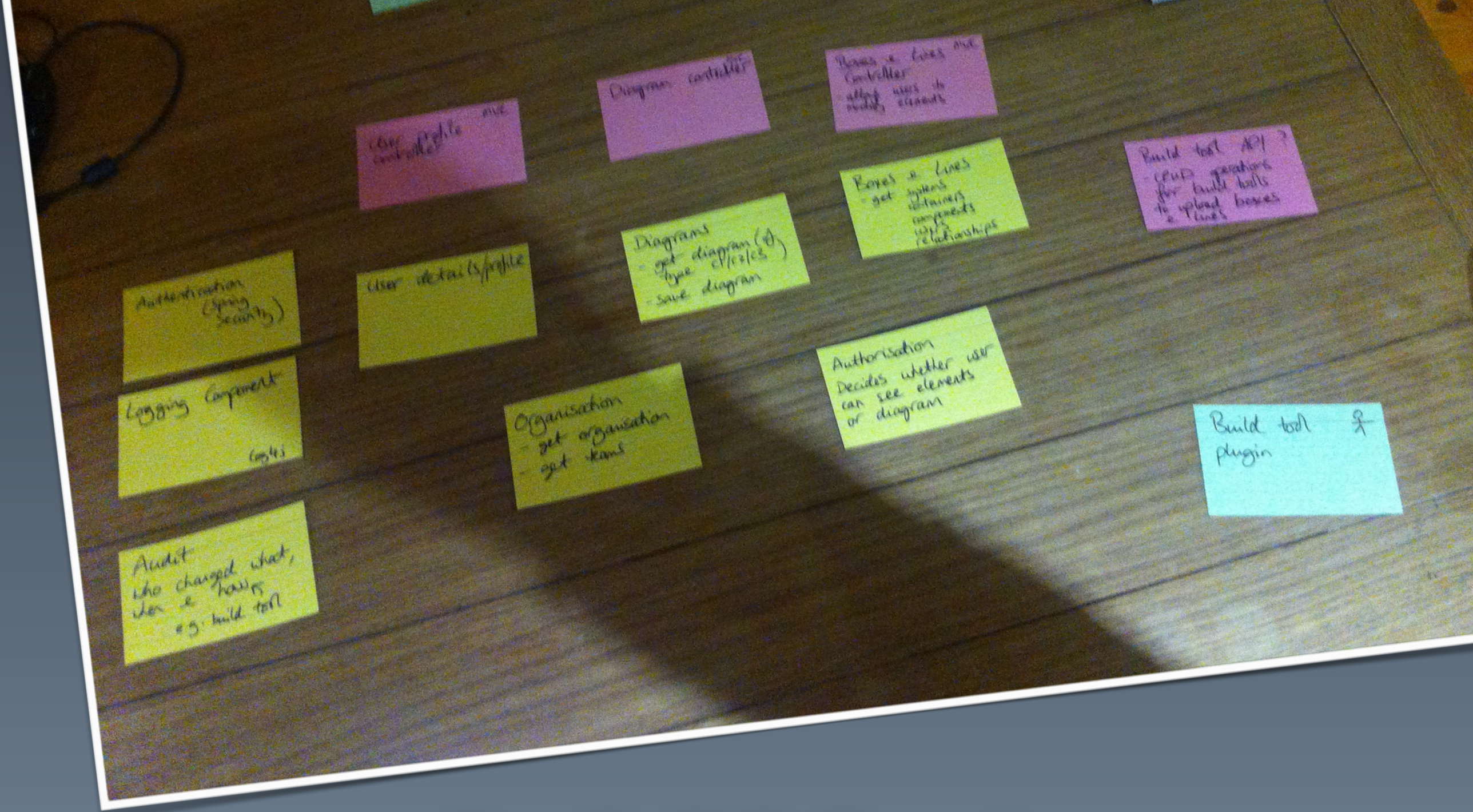
Class

Agree on a simple set of abstractions that the whole team can use to communicate

Components

~~Classes,~~

Responsibilities, Collaborations

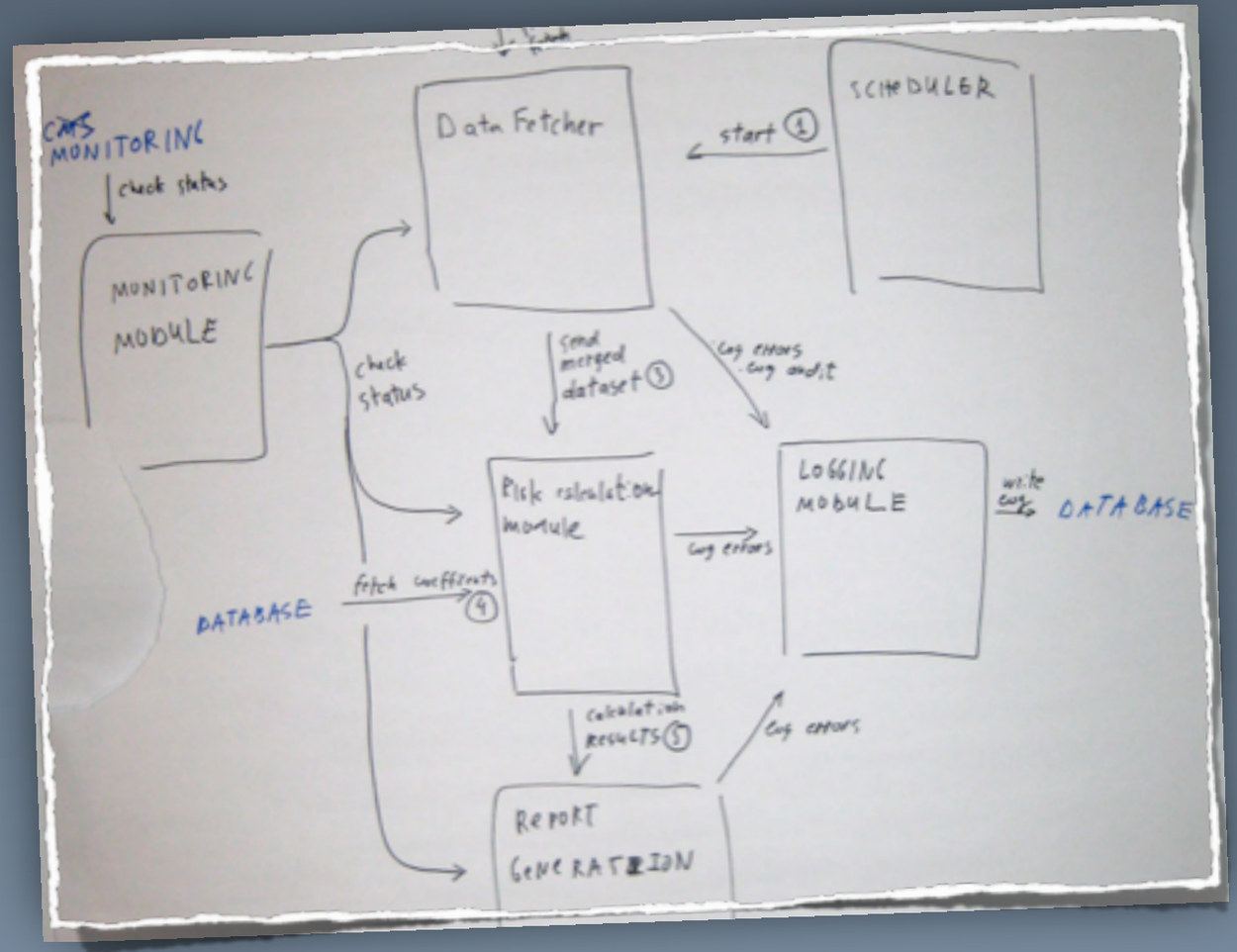


An initial set of components

Abstraction

is about reducing detail

rather than creating a different representation



Software architecture

vs

code

Does your code reflect the

abstractions

that you think about?

Agree upon some principles

If you genuinely don’t know what you’re building (and therefore can’t create a component model), agree on some principles that you will use to structure your software

Structure

Components,
technologies,
principles

Vision

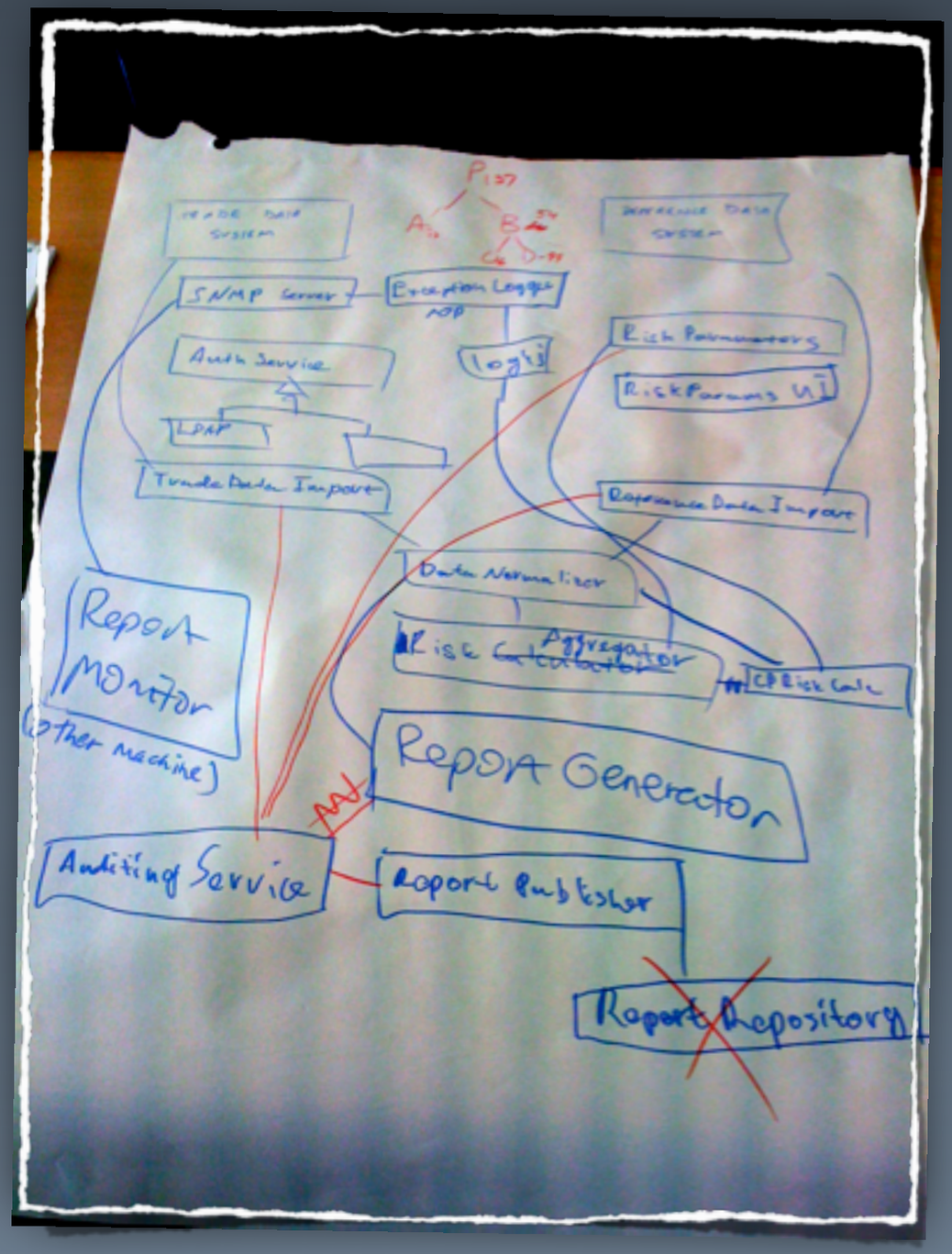
Detailed blueprints

vs

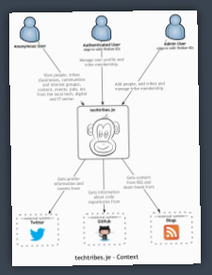
setting a direction

In my experience,

software teams aren't able to effectively communicate the software architecture of their systems

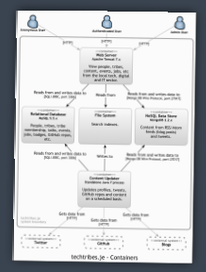


The C4 model



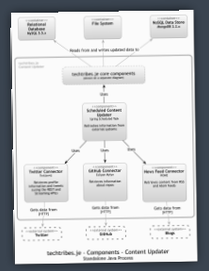
System Context

The system plus users and system dependencies



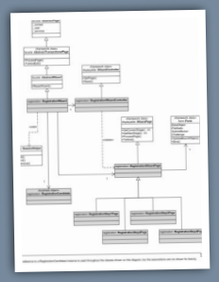
Containers

The overall shape of the architecture and technology choices



Components

Logical components and their interactions within a container



Classes

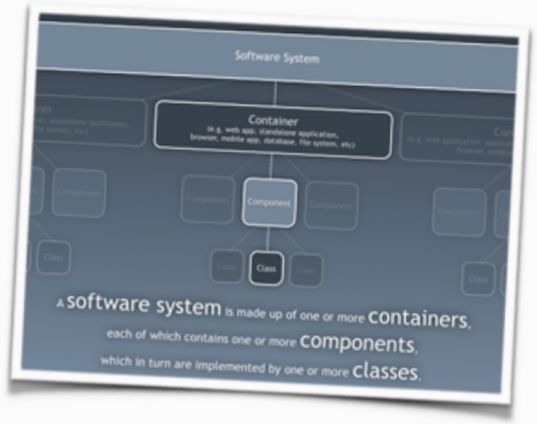
Component or pattern implementation details

Software architecture and the C4 model

coding {the} architecture

1 Agree on a simple set of abstractions that the whole team can use to communicate.

A common set of abstractions is more important than a common notation,
but do ensure that your notation (shapes, colours, line styles, acronyms, etc) is understandable. Add a key/legend if in doubt.



2 Draw a number of simple diagrams at different levels of abstraction.
C4: context, containers, components, classes

1. System context diagram

A context diagram can be a useful starting point for diagramming and documenting a software system, allowing you to step back and look at the big picture. Draw a simple block diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interfaces with.

Detail isn't important here as this is your zoomed out view showing a big picture of the system landscape. The focus should be on people (actors, roles, personas, etc) and software systems rather than technologies, protocols and other low-level details. It's the sort of diagram that you could show to non-technical people.

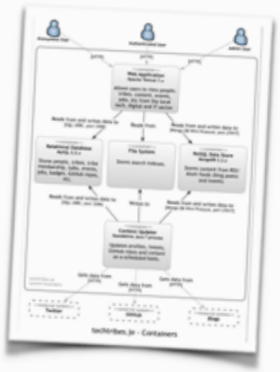
Naming
If naming is one of the hardest things in software development, resist the temptation to have a diagram full of boxes that only contain names!

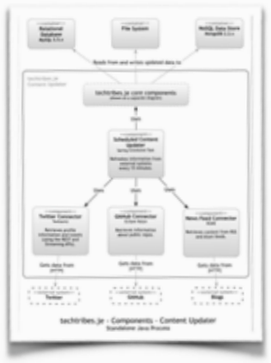
Adding a brief statement of responsibilities to containers and components is a simple way to remove ambiguity. It provides a nice "at a glance" view too.

2. Container diagram

Once you understand how your system fits in to the overall IT environment with a context diagram, a really useful next step can be to illustrate the high-level technology choices with a containers diagram. By "container" I mean something like a web application, desktop application, mobile app, database, file system, etc. Essentially, what I call a container is anything that can host code or data.

A container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another. It's a simple, high-level technology focussed diagram that is useful for software developers and support/operations staff alike.





3. Component diagram(s)

Following on from a container diagram showing the high-level technology decisions, I'll then start to zoom in and decompose each container further. However you decompose your system is up to you, but I tend to identify the major logical components and their interactions. This is about partitioning the functionality implemented by a software system into a number of distinct components, services, subsystems, layers, workflows, etc.

The components diagram shows how a container is divided into components, what each of those components are, their responsibilities and the technology/implementation details.

How would you code it?
Ask yourself this question if you can't figure out how to draw something.

4. Class diagram(s)

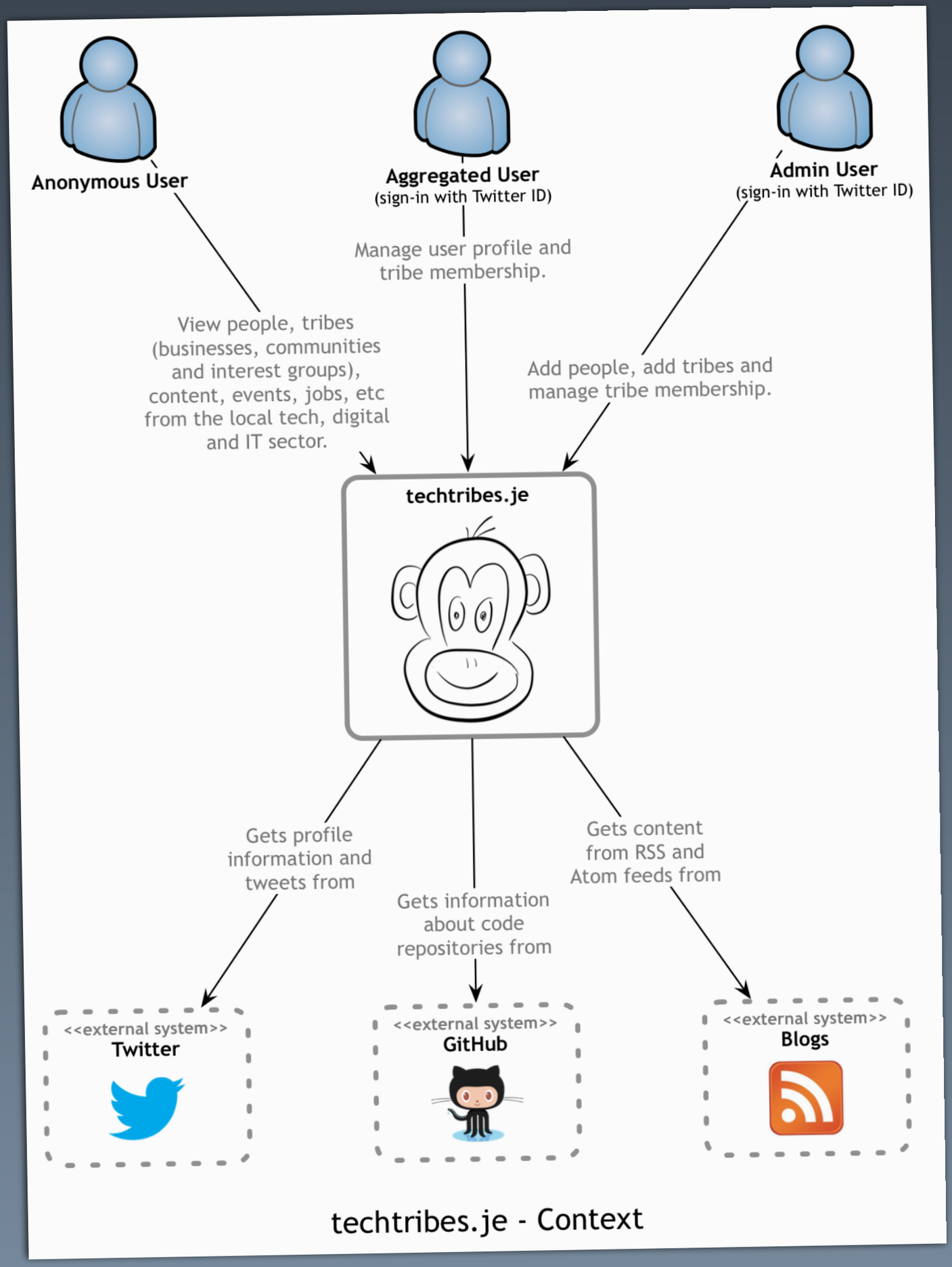
This is optional, but I sometimes draw one or more UML class diagrams if I want to illustrate specific component implementation details.

More information about software architecture and the C4 model can be found in "Software Architecture for Developers", available as an ebook from leanpub.com

@simonbrown | www.codingthearchitecture.com

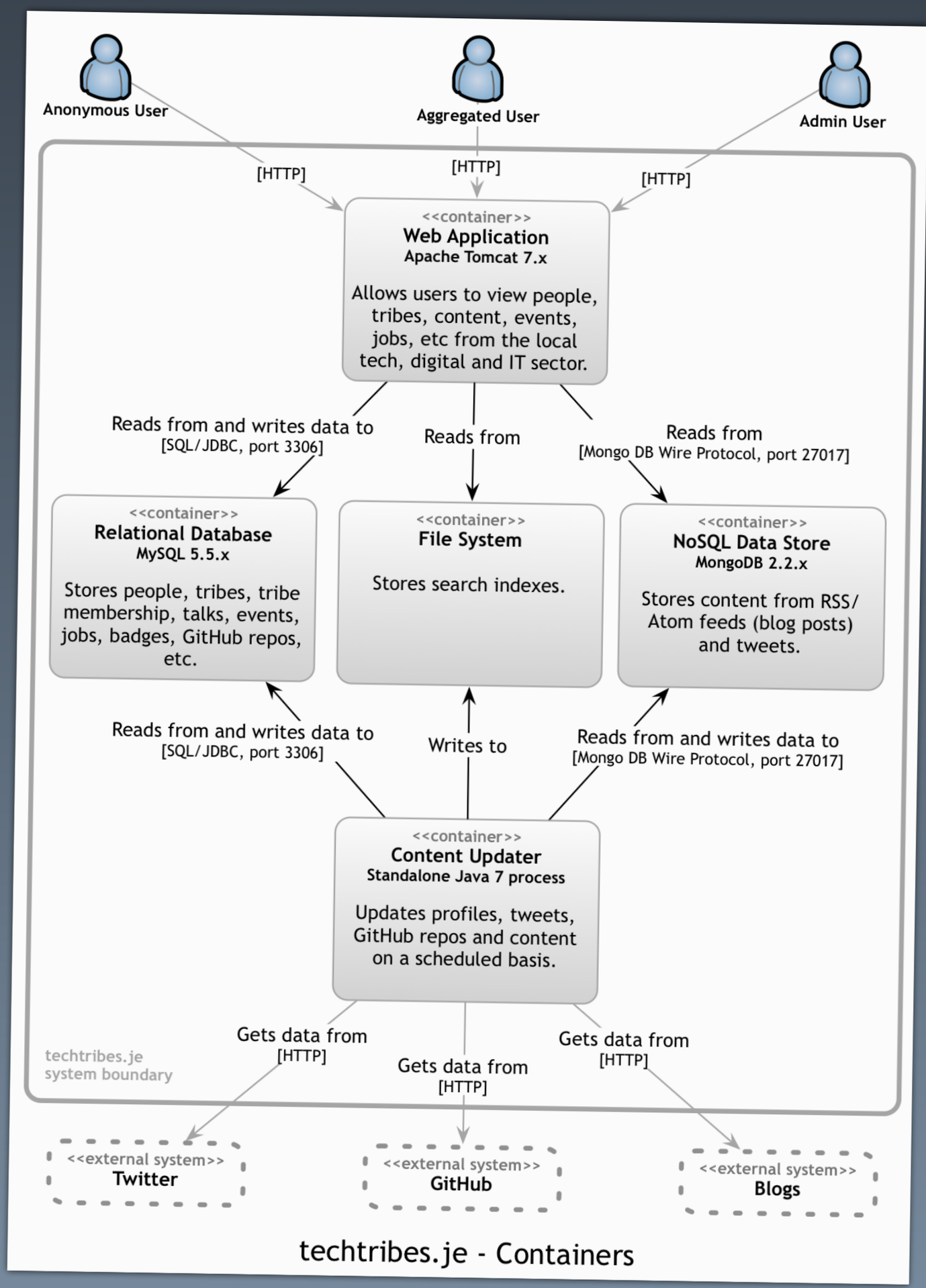
Context

- What are we building?
- Who is using it?
  (users, actors, roles, personas, etc)
- How does it fit into the existing IT environment?
  (systems, services, etc)



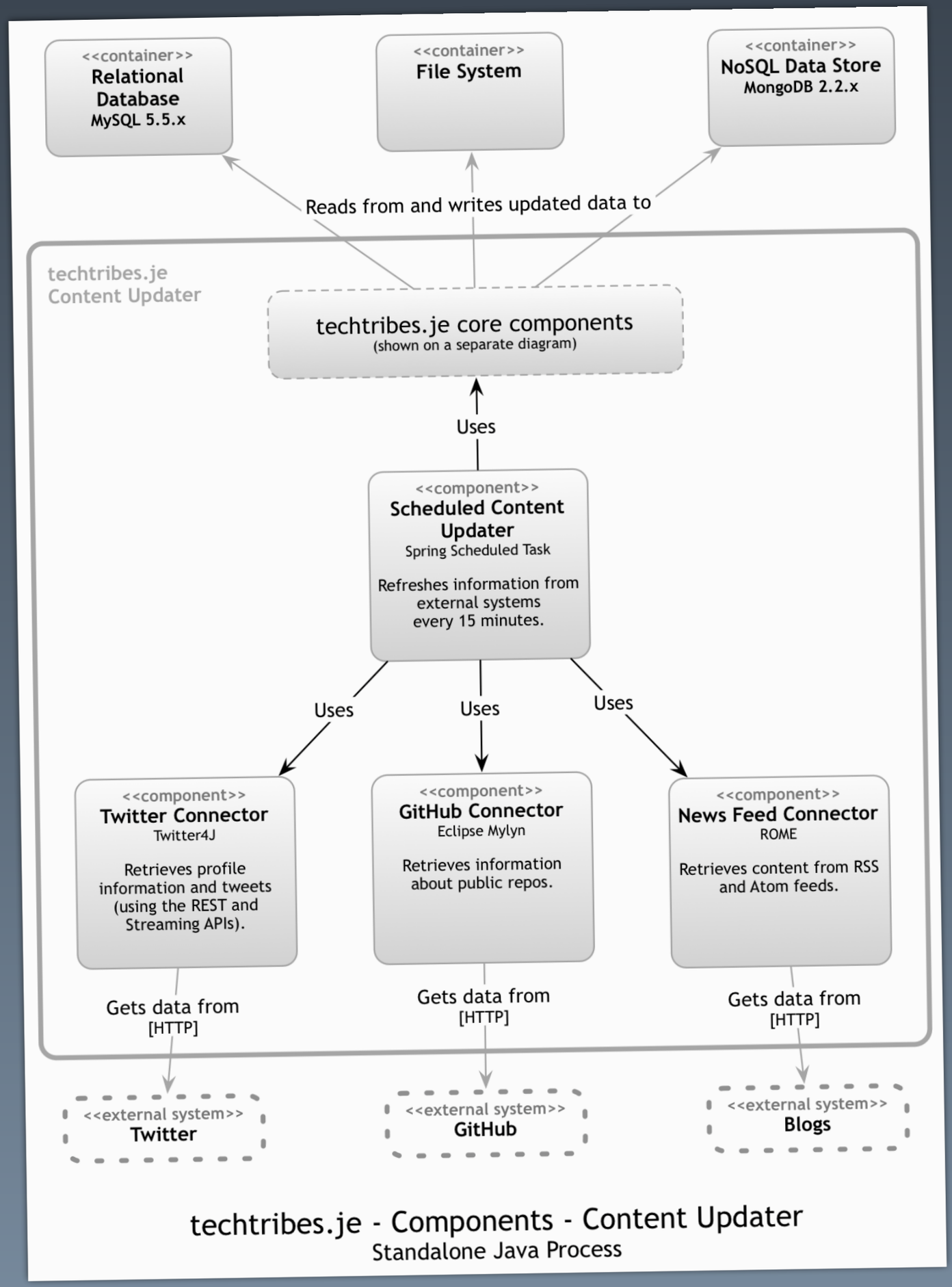
Containers

- What are the high-level technology decisions? (including responsibilities)
- How do containers communicate with one another?
- As a developer, where do I need to write code?



Components

- What components/ services is the container made up of?
- Are the technology choices and responsibilities clear?



The conversations change

Sketches

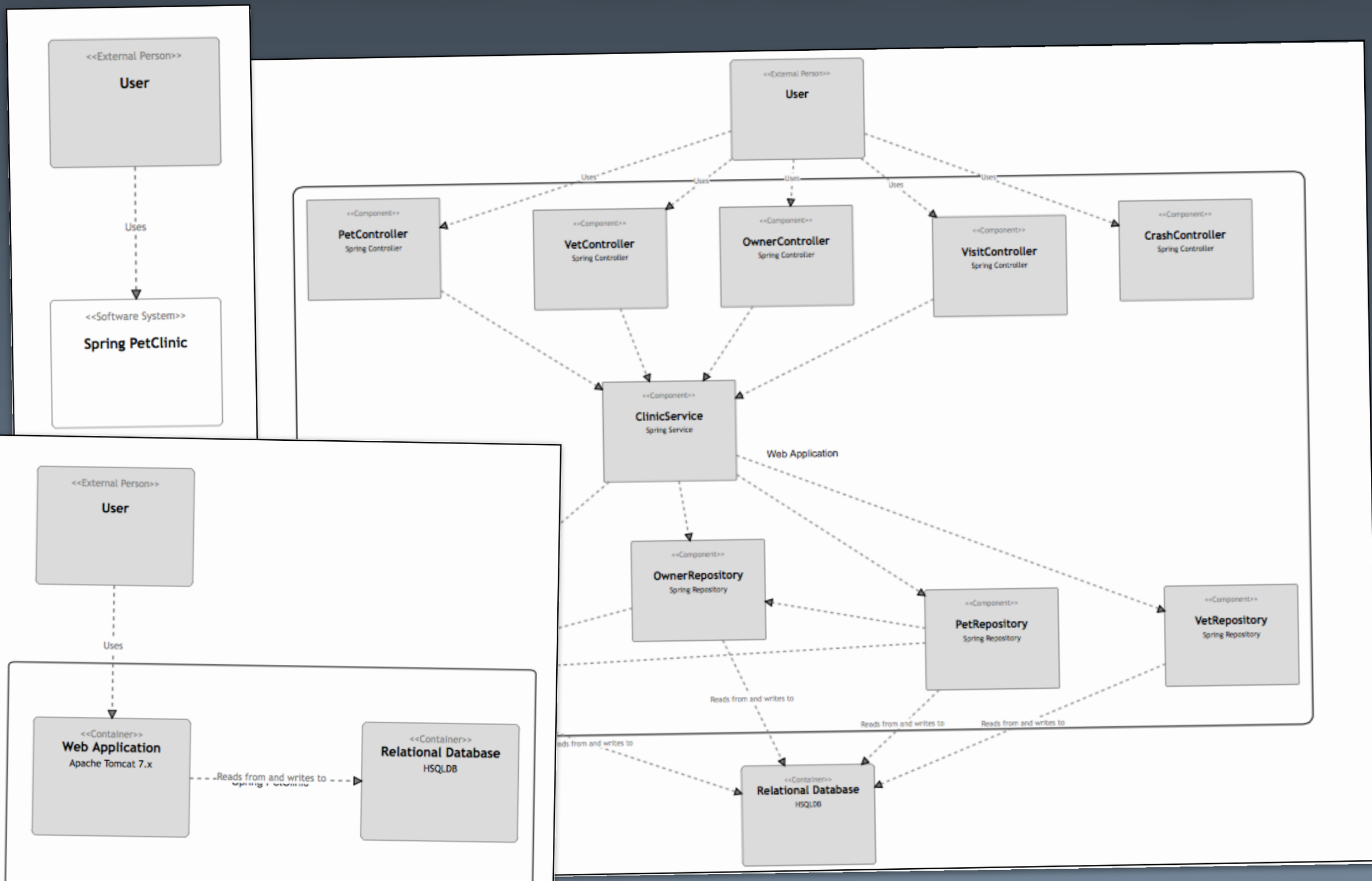
will become

out of date

Software architecture as code

```java
/**
 * This is a C4 representation of the Spring PetClinic sample app (https://github.com/spring-projects/spring-petclinic/).
 */
public class SpringPetClinic {

    public static void main(String[] args) throws Exception {
        Model model = new Model("Spring PetClinic", "This is a C4 representation of the Spring PetClinic sample app (https://github.co

        // create the basic model (the stuff we can't get from the code)
        SoftwareSystem springPetClinic = model.addSoftwareSystem(Location.Internal, "Spring PetClinic", "");
        Person user = model.addPerson(Location.External, "User", "");
        user.uses(springPetClinic, "Uses");

        Container webApplication = springPetClinic.addContainer("Web Application", "", "Apache Tomcat 7.x");
        Container relationalDatabase = springPetClinic.addContainer("Relational Database", "", "HSQLDB");
        user.uses(webApplication, "Uses");
        webApplication.uses(relationalDatabase, "Reads from and writes to");

        // and now automatically find all Spring @Controller, @Component, @Service and @Repository components
        ComponentFinder componentFinder = new ComponentFinder(webApplication, "org.springframework.samples.petclinic",
                new SpringComponentFinderStrategy());
        componentFinder.findComponents();

        // connect the user to all of the Spring MVC controllers
        webApplication.getComponents().stream().filter(c -> c.getTechnology().equals("Spring Controller")).forEach(c -> user.uses(c, "

        // connect all of the repository components to the relational database
        webApplication.getComponents().stream().filter(c -> c.getTechnology().equals("Spring Repository")).forEach(c -> c.uses(relatio

        // finally create some views
        SystemContextView contextView = model.createContextView(springPetClinic);
        contextView.addAllSoftwareSystems();
        contextView.addAllPeople();

        ContainerView containerView = model.createContainerView(springPetClinic);
        containerView.addAllPeople();
        containerView.addAllSoftwareSystems();
        containerView.addAllContainers();

        ComponentView componentView = model.createComponentView(springPetClinic, webApplication);
        componentView.addAllComponents();
        componentView.addAllPeople();
        componentView.add(relationalDatabase);

        System.out.println(JsonUtils.toJson(model, true));
    }
```

Software architecture model as code -> JSON -> diagrams



structurizr.com

www.structurizr.com

Reader

www.structurizr.com

Structurizr

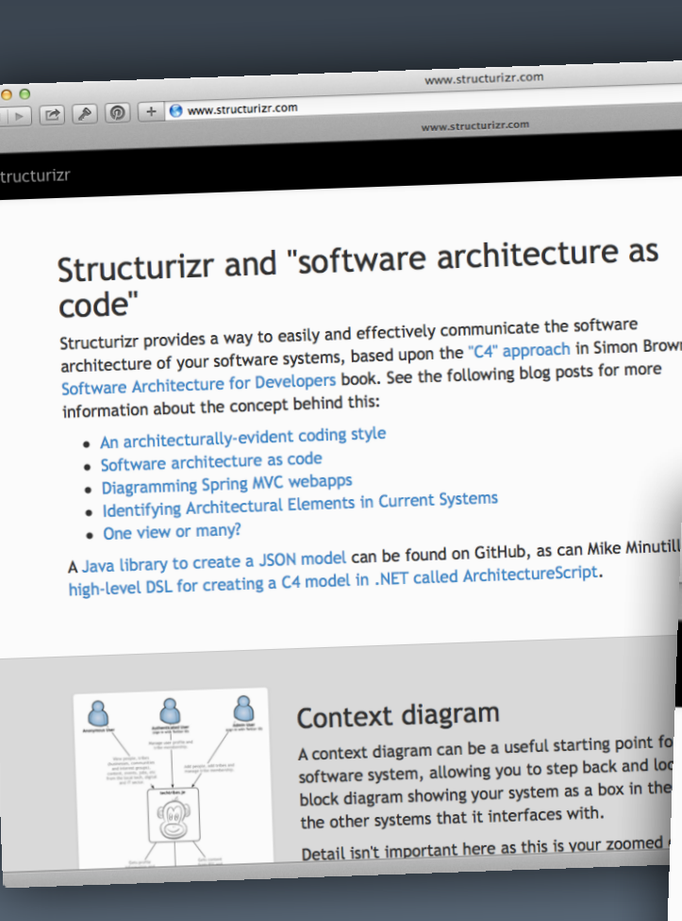
Try it Sign in

Structurizr and "software architecture as code"

Structurizr provides a way to easily and effectively communicate the software architecture of your software systems, based upon the "C4" approach in Simon Brown's Software Architecture for Developers book. See the following blog posts for more information about the concept behind this:

- An architecturally-evident coding style
- Software architecture as code
- Diagramming Spring MVC webapps
- Identifying Architectural Elements in Current Systems
- One view or many?

A Java library to create a JSON model can be found on GitHub, as can Mike Minutill high-level DSL for creating a C4 model in .NET called ArchitectureScript.



Context diagram

A context diagram can be a useful starting point fo software system, allowing you to step back and loo block diagram showing your system as a box in the the other systems that it interfaces with.

Detail isn't important here as this is your zoomed

Structurizr – Try it

www.structurizr.com/tryit

Structurizr – Try it

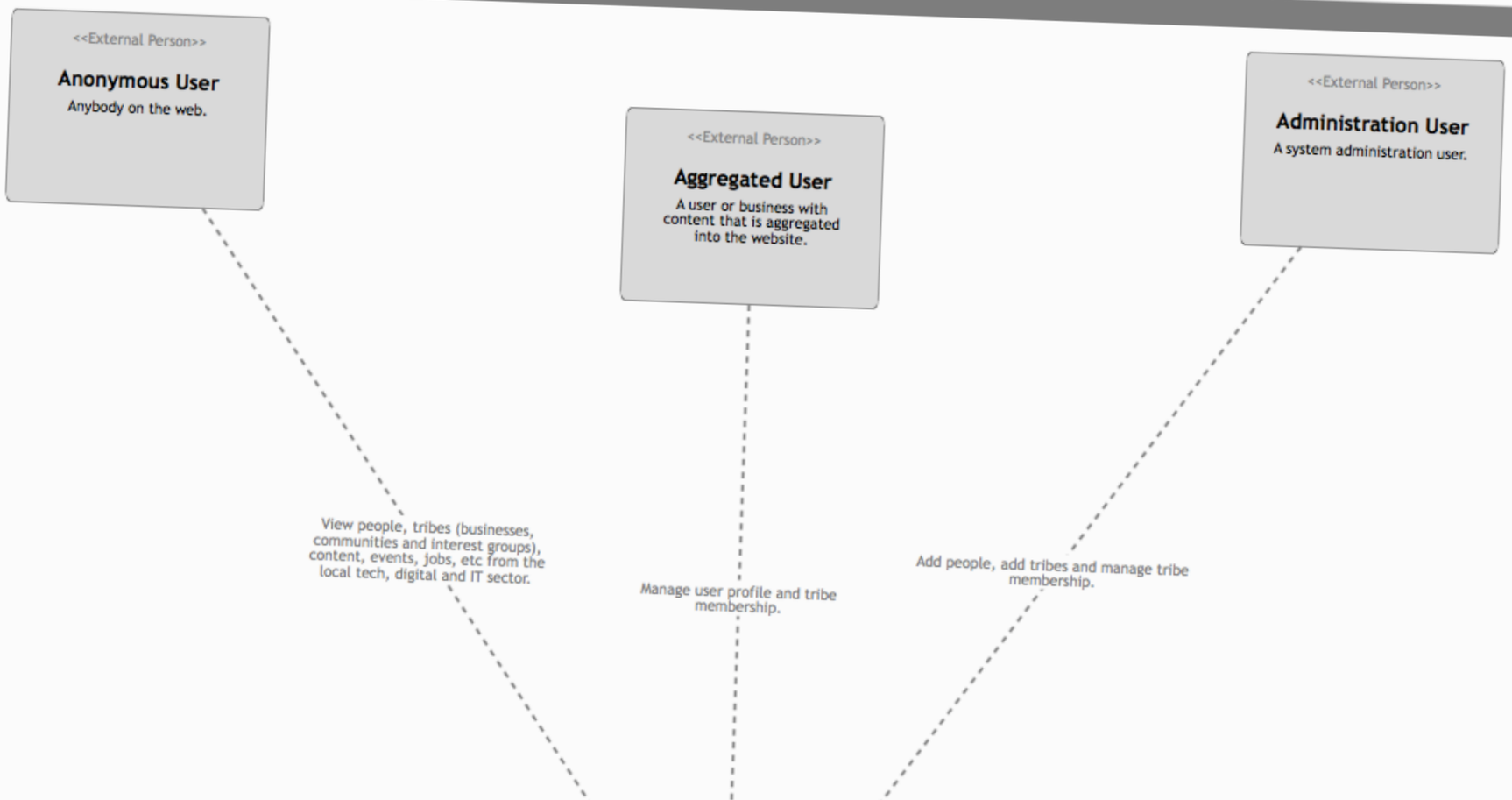
Structurizr

Try it Sign

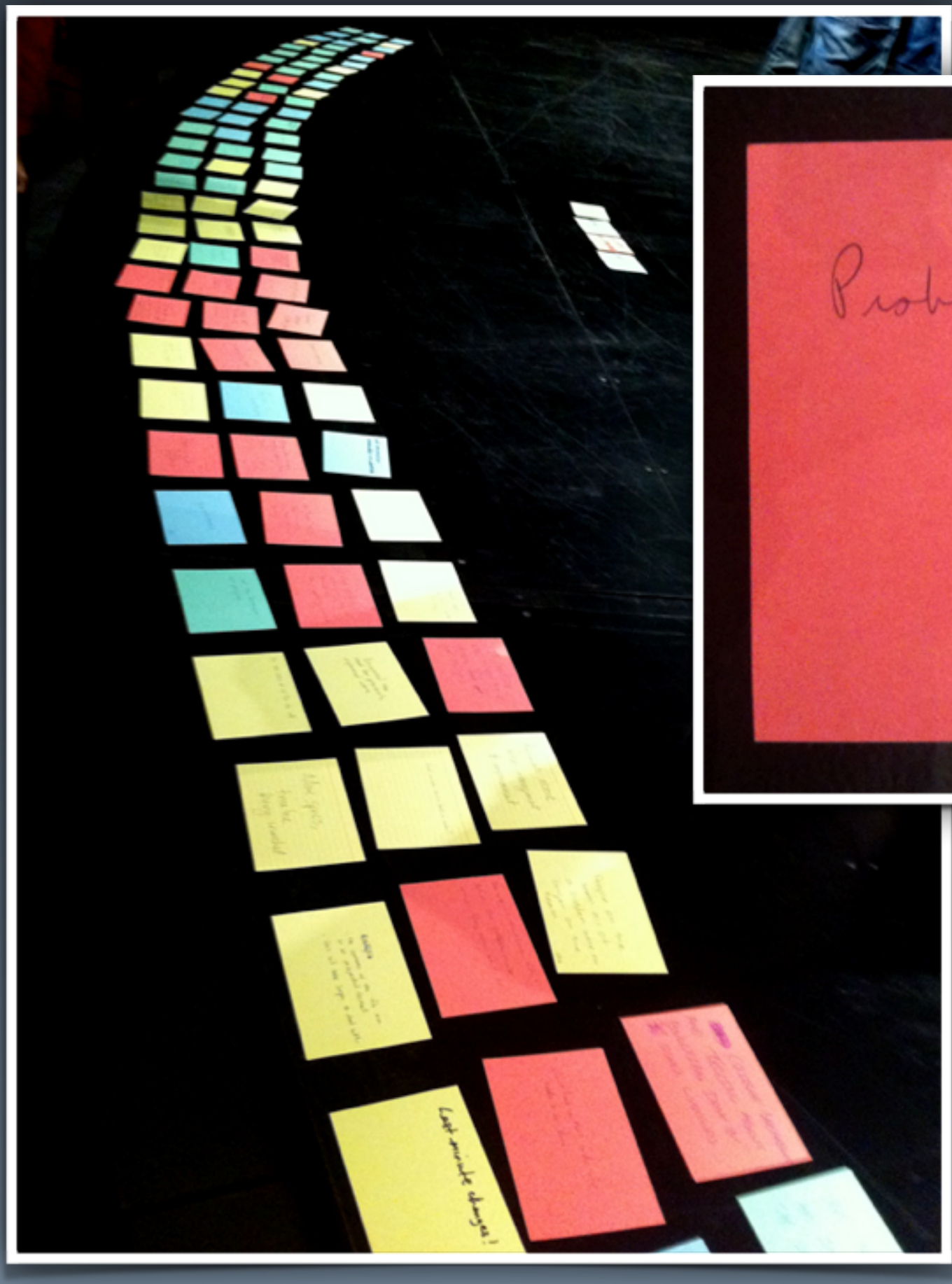
{"people":[{"id":4,"name":"Administration User","description":"A system administration user.","location":"External","relationships":[{"sourceId":4,"dest

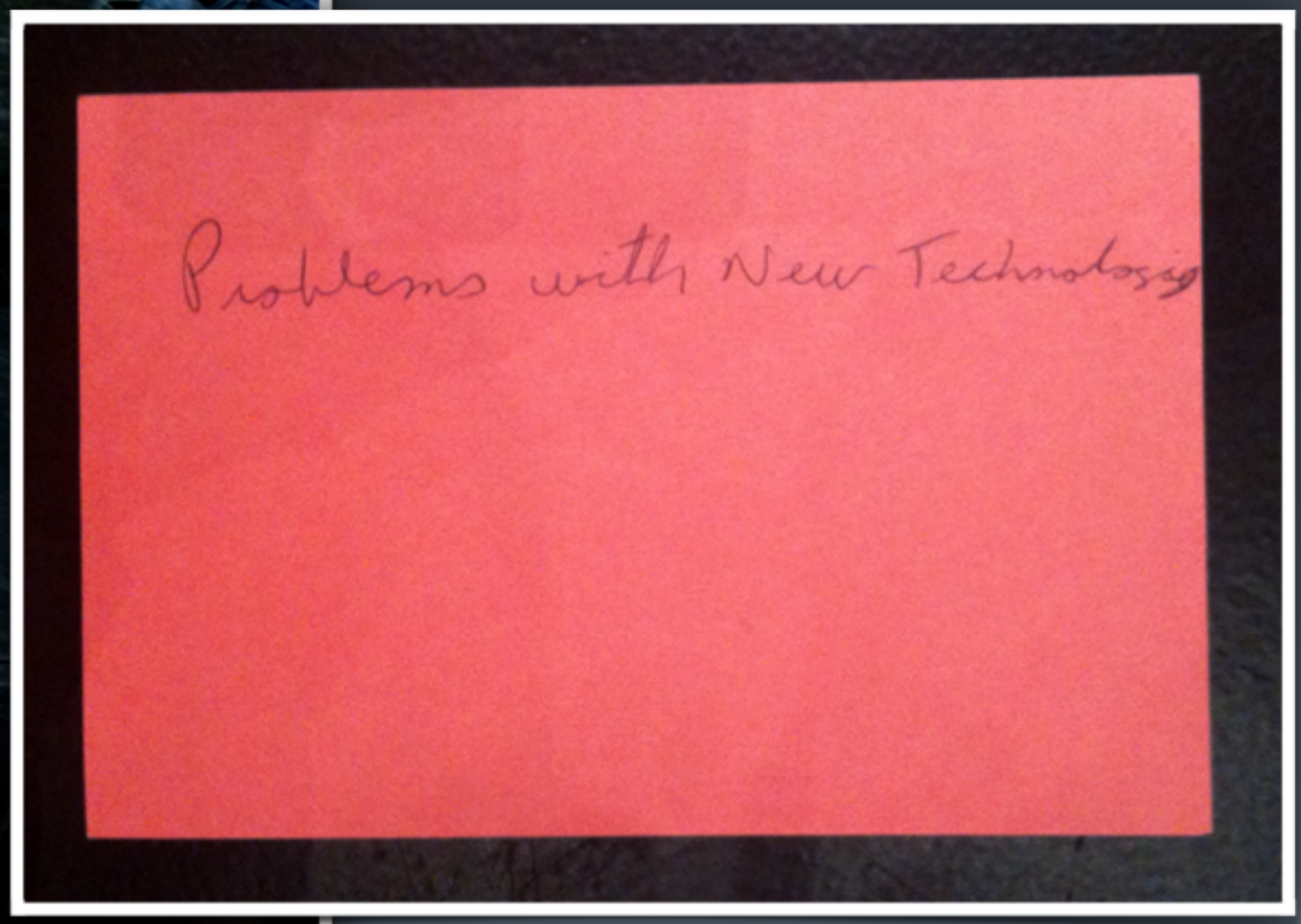
techtribes.je - System Context

A4 - portrait | Fit width | Fit height | Full size | Zoom In | 86.85% | Zoom Out | Export



Risks





An example timeline from

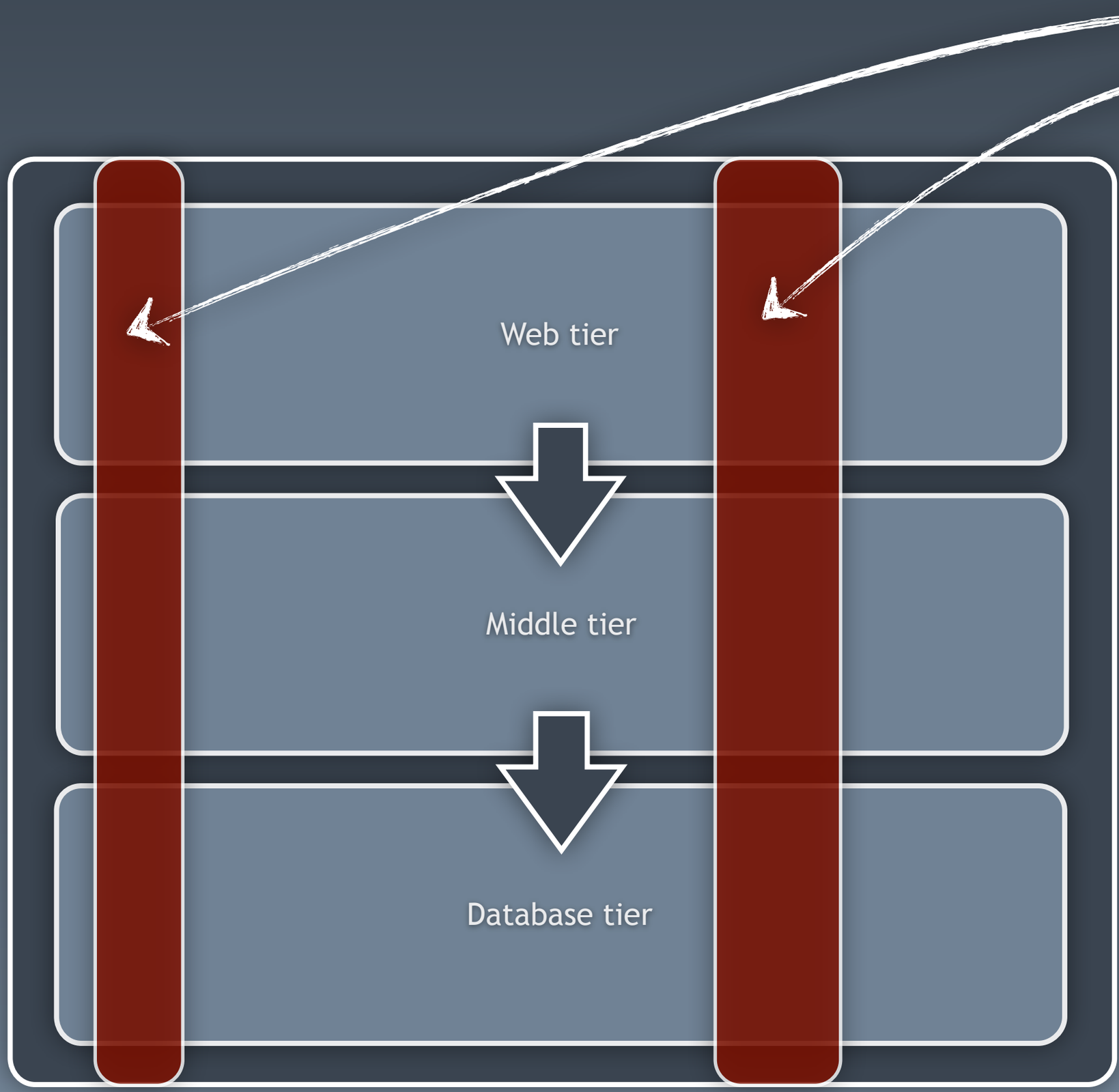
“Beyond Retrospectives”

by Linda Rising

#gotocon Aarhus 2011

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

Scott Ambler
http://www.agilemodeling.com/essays/agileArchitecture.htm



Concrete

experiments

(prototype, proof of concept or production code)

What is architecturally **significant**?

Costly to change | Complex | New

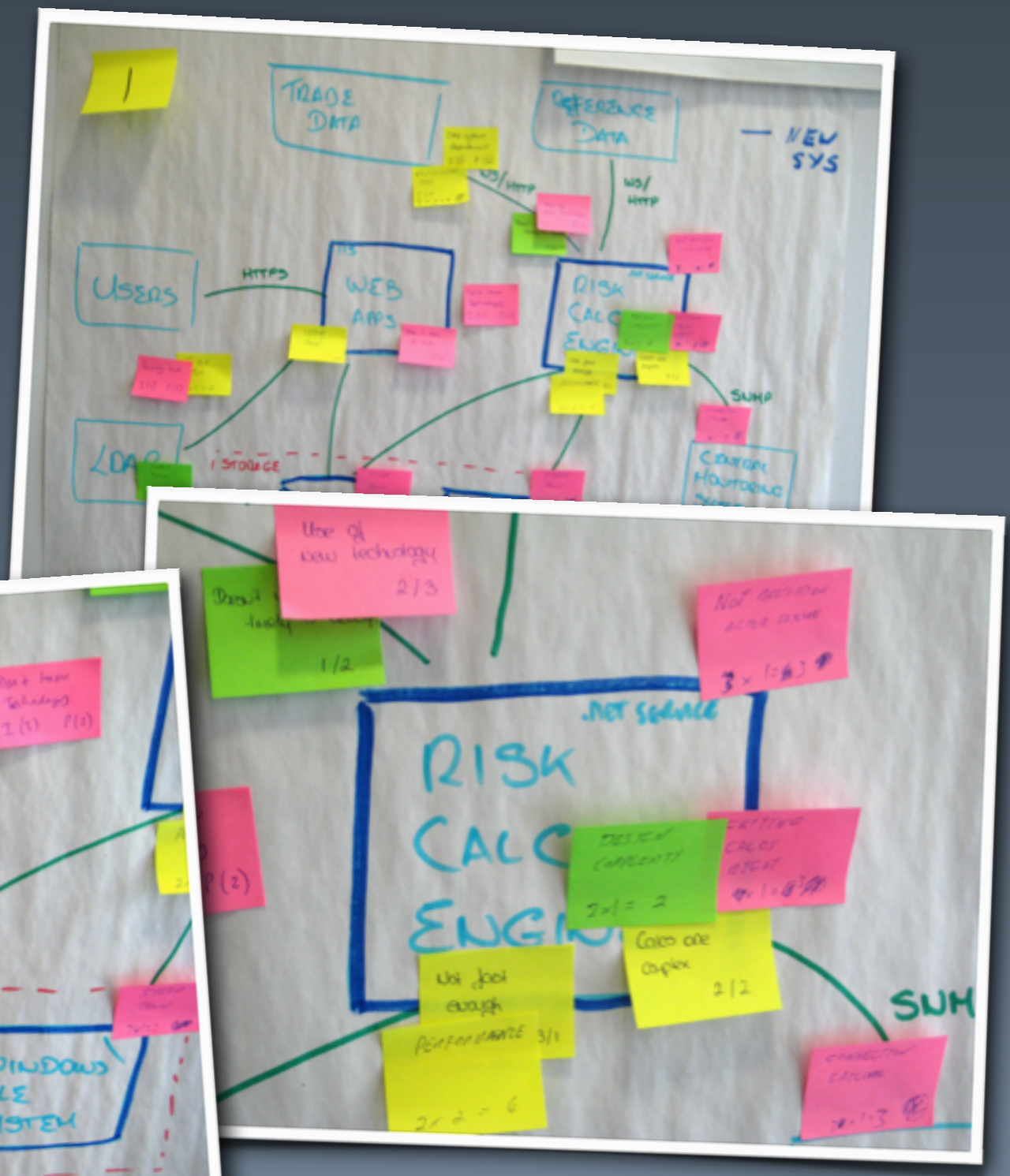
You need to

identify and mitigate your highest priority risks

Security holes/breaches, data loss, network outages, server outages, poor performance, poor scalability, deployment complexity, lack of skills, unproven technology, external interfaces changing, users doing things they shouldn't...

Like estimates, risks are subjective

Risk-storming



A collaborative and visual technique for identifying risk

You still need to deal with the risks

(mitigation strategies include hiring people,
undertaking proof of concept
and changing your architecture)

Does your architecture

work?

(write code and test early if needed)

For structurizr.com

Rendering a dynamic diagram in JavaScript
Exporting that diagram to PNG
Rackspace vs Pivotal Web Services

You can still get it wrong though!

In summary…

Structure

Understand the significant structural elements and how they fit together, based upon the architectural drivers.

Design and decomposition down to containers and components.

Vision

Create and communicate a vision for the team to work with.

Context, container and component diagrams.

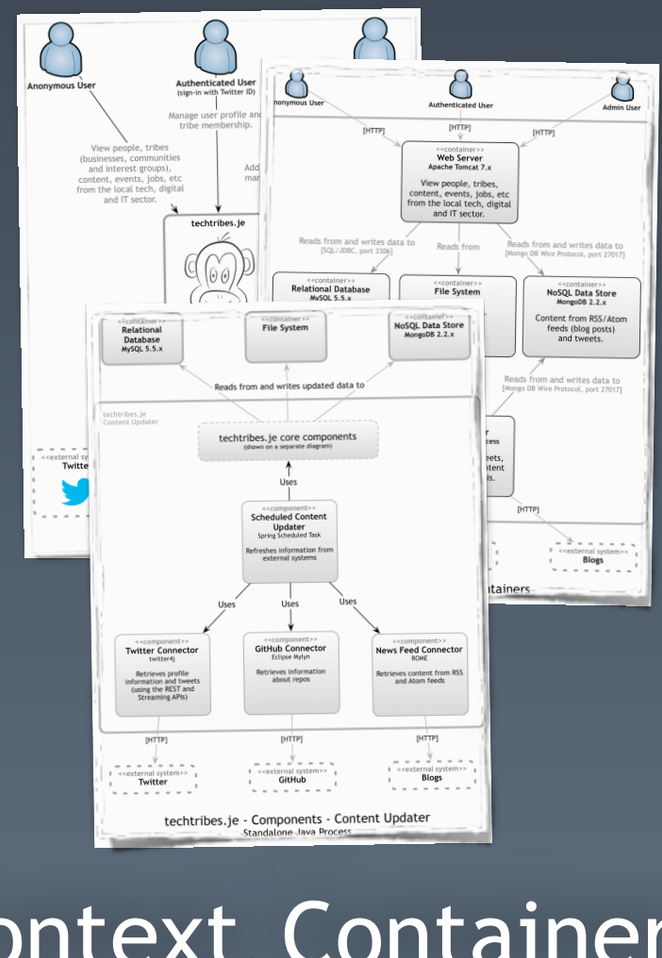
Risks

Identify and mitigate the highest priority risks.

Risk-storming and concrete experiments.

Just enough up front design
to create firm foundations
for the software product and its delivery

Requirements



Context, Containers and Components

How long?

Firm foundations

Create a *sufficient starting point*

and *set a direction*,

in a *minimal amount of time*,

to *stack the odds of success*

in your favour

simon.brown@codingthearchitecture.com
@simonbrown on Twitter