

Models, Sketches and Everything In-Between

Simon Brown Coding the Architecture
Eoin Woods Artechra

Software Architect 2014
October 2014, London

Welcome

- It's hello from me
 - Simon Brown, Coding the Architecture

- And hello from him
 - Eoin Woods, Artechra

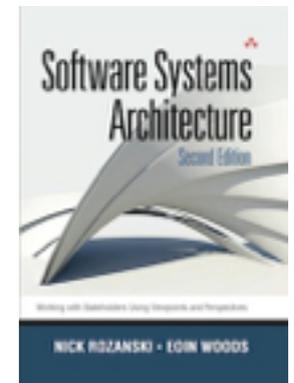


Our Agenda

- Simon Says ...
- Eoin Says ...
- Questions and Queries:
 - Q1. Modelling - Why Bother?
 - Q2. Models and Agility
 - Q3. How to Do It?
 - Q4. UML - Worth the Hassle?
 - Q5. Modelling in the Large vs the Small
- Summary and Conclusions

Background

- We've been talking about software modelling for ages
 - We both think its a good idea (in moderation)
 - Simon likes boxes and lines, Eoin likes UML (sort of)
 - Simon has C4, Eoin has V&P (with Nick Rozanski)
 - We've both inflicted a book on the world ...



- We'd like to work out what the real answer is today
 - We've got questions, but yours are probably better

The Point of Modelling

- Simon:
 - How do you understand what you're building?
 - How do you explain it to the rest of the team?
 - The trick is not getting stuck in analysis paralysis.
- Eoin:
 - Main problem with not modelling is lack of intellectual control
 - Main problem with modelling is believing that modelling is an end in itself

Some Opinions

Simon Says ...

How do we

communicate

software architecture?



9 out of 10 people
don't use UML
(in my experience)

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

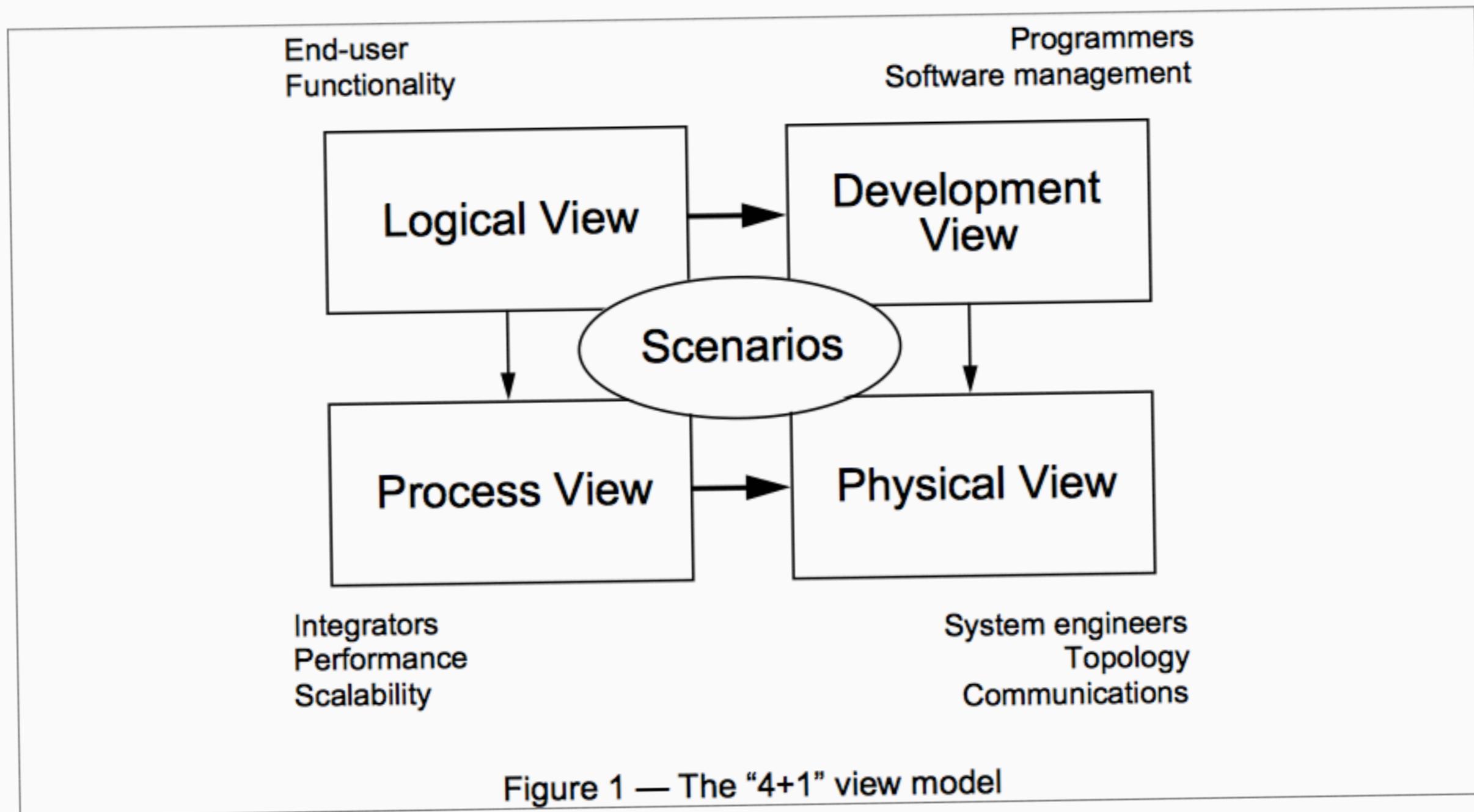


Figure 1 — The "4+1" view model

We apply Perry & Wolf's equation independently on each view, i.e., for each view we define the set of elements to use (components, containers, and connectors), we capture the forms and patterns that work, and we capture the rationale and constraints, connecting the architecture to some of the requirements.

Do the **names**
of those views make
sense?

Conceptual vs Logical

Process vs Functional

Development vs Physical

Development vs Implementation

Physical vs Implementation

Physical vs Deployment

Logical and
development

views are often

separated

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

Learn more Got It

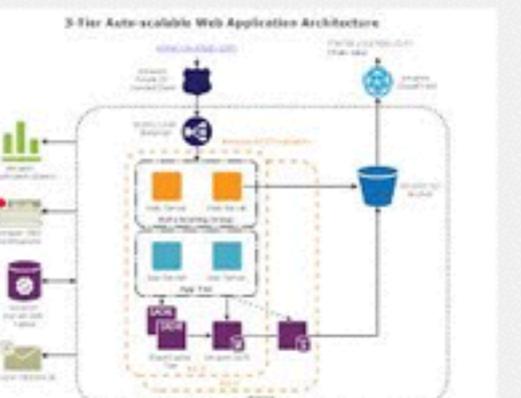
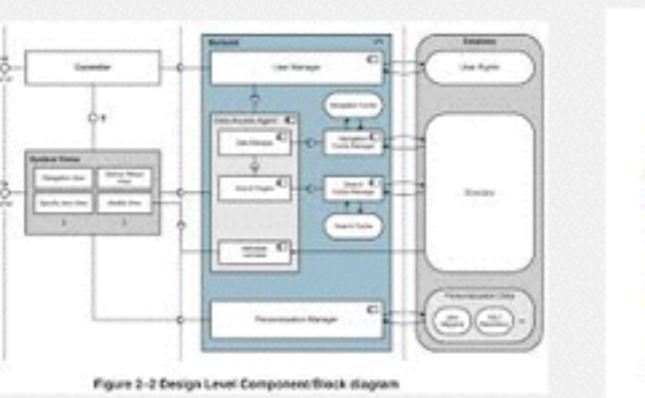
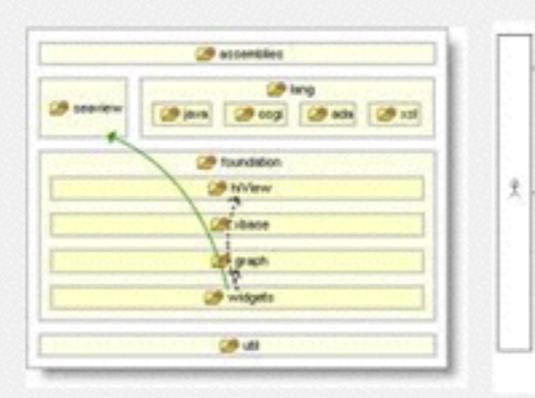
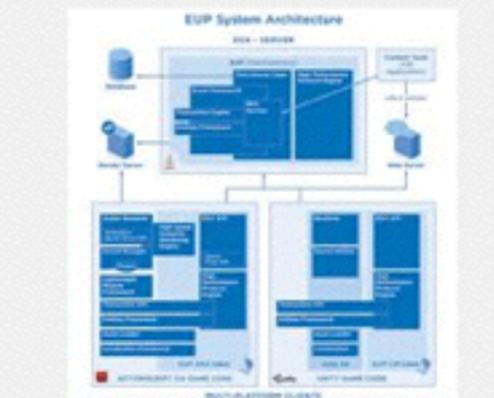
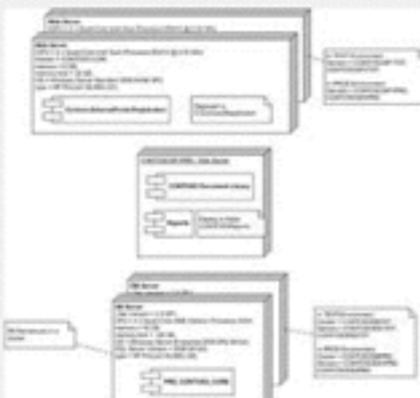
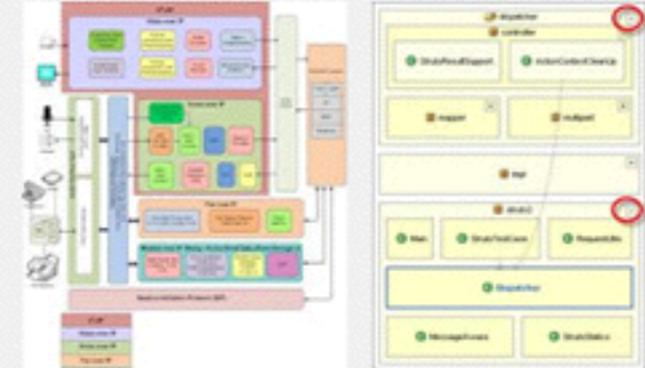
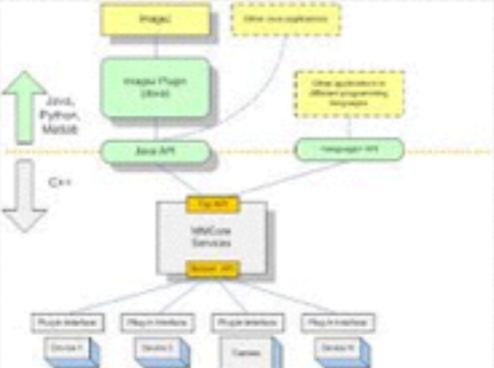
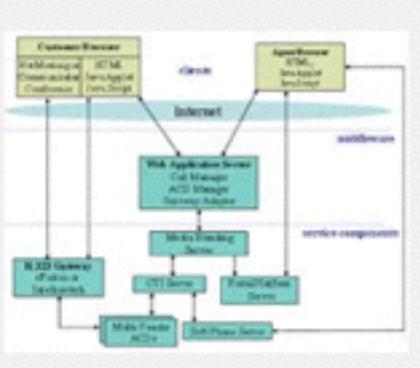
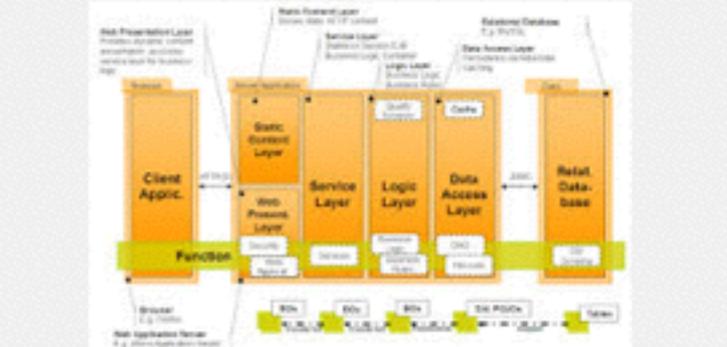
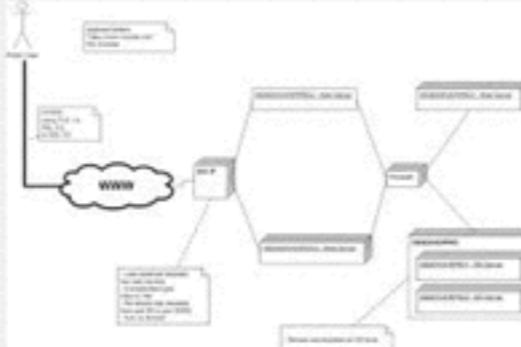
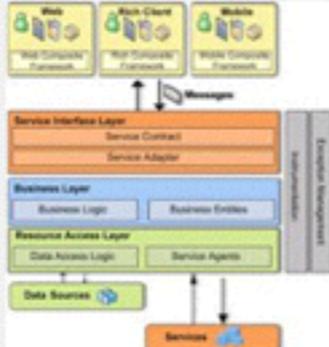
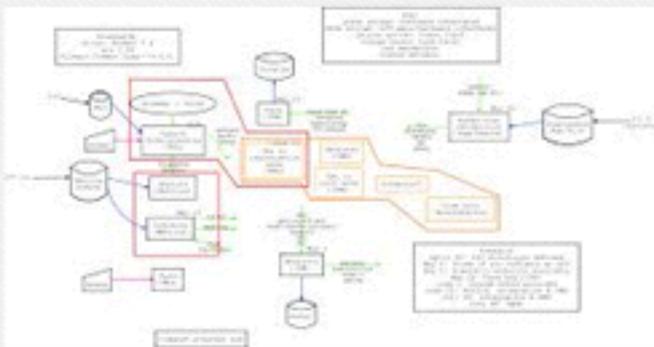
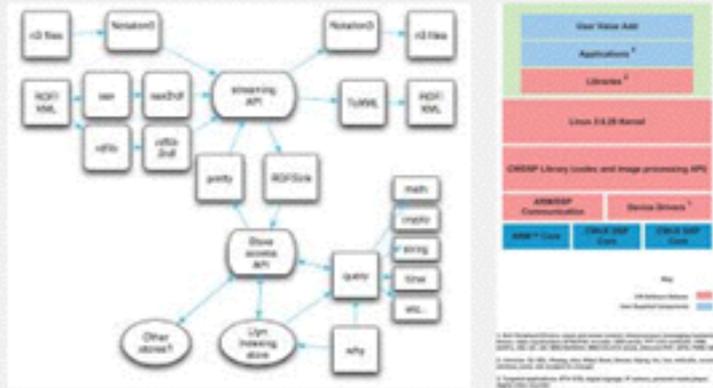
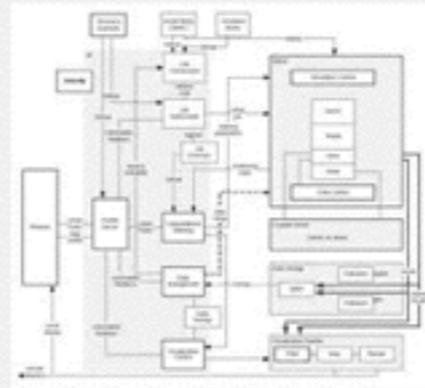
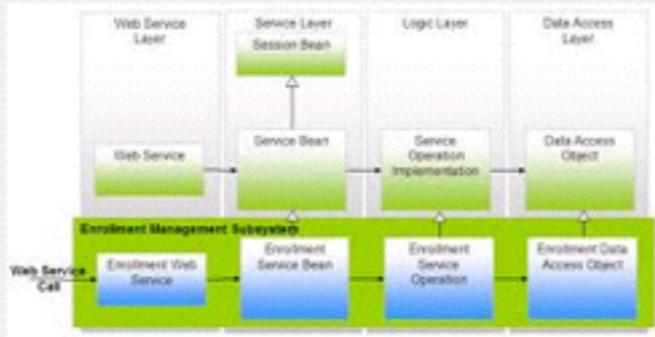
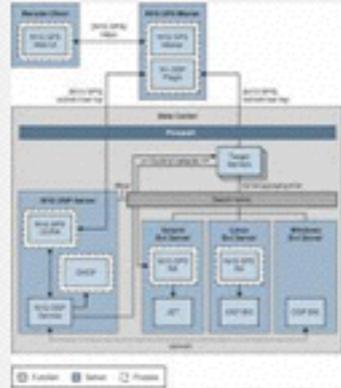
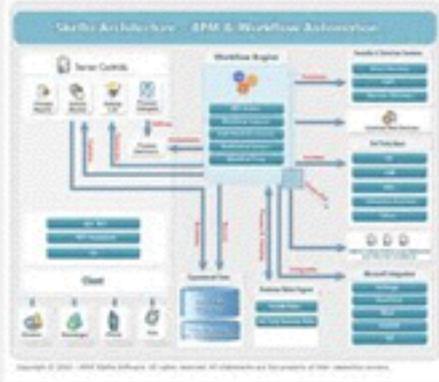
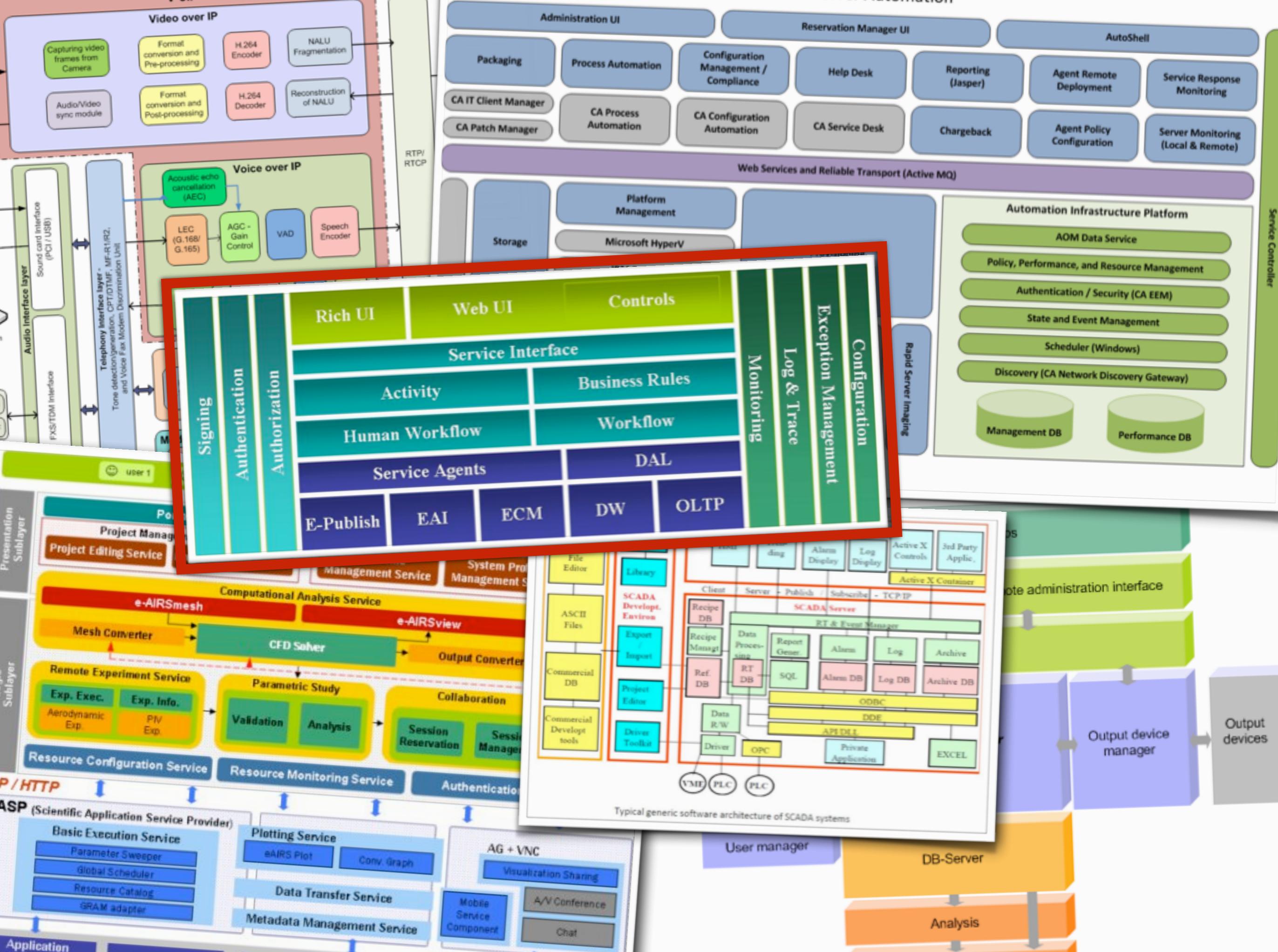
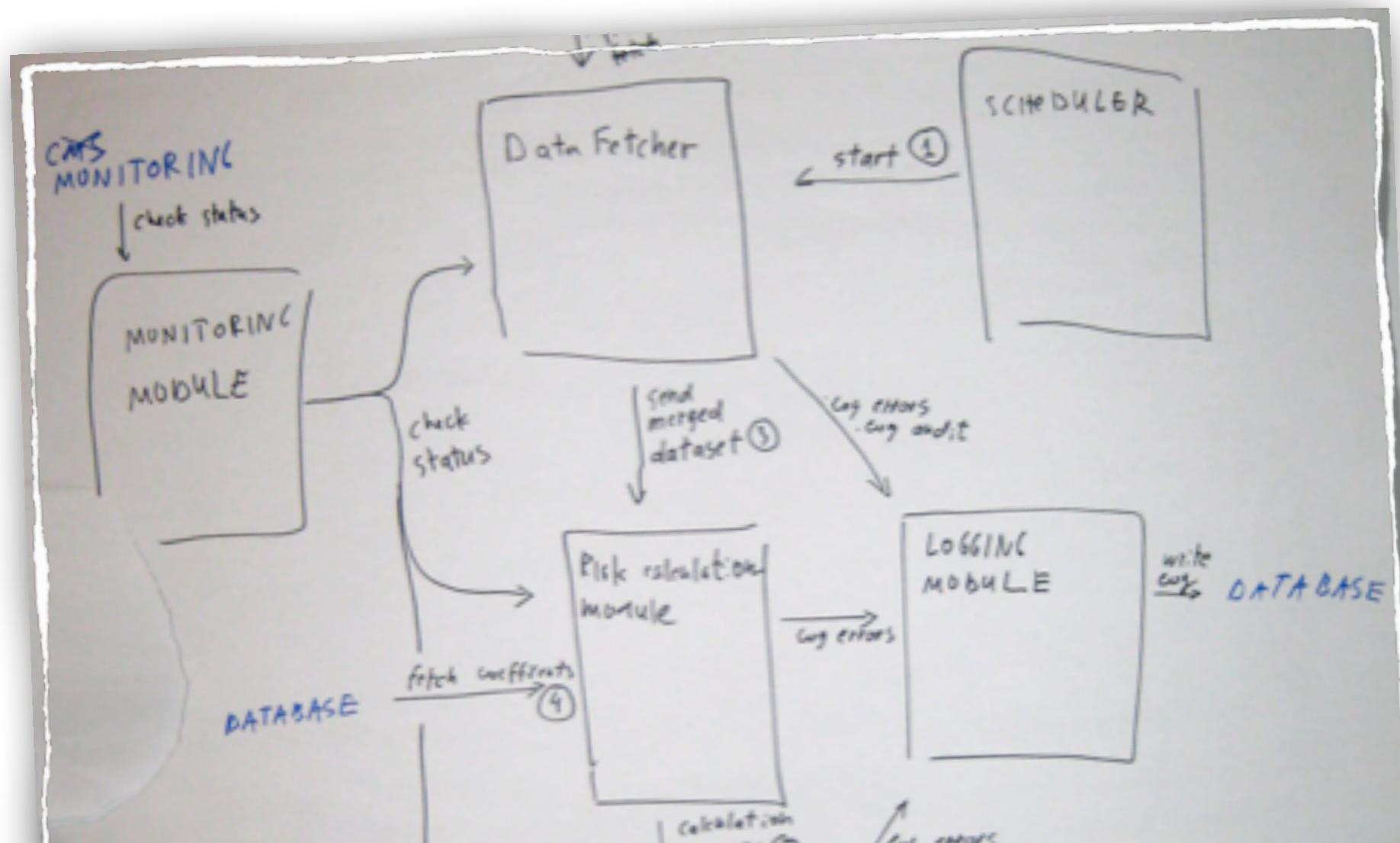


Figure 2-2 Design Level Component/Block Diagram



Abstraction

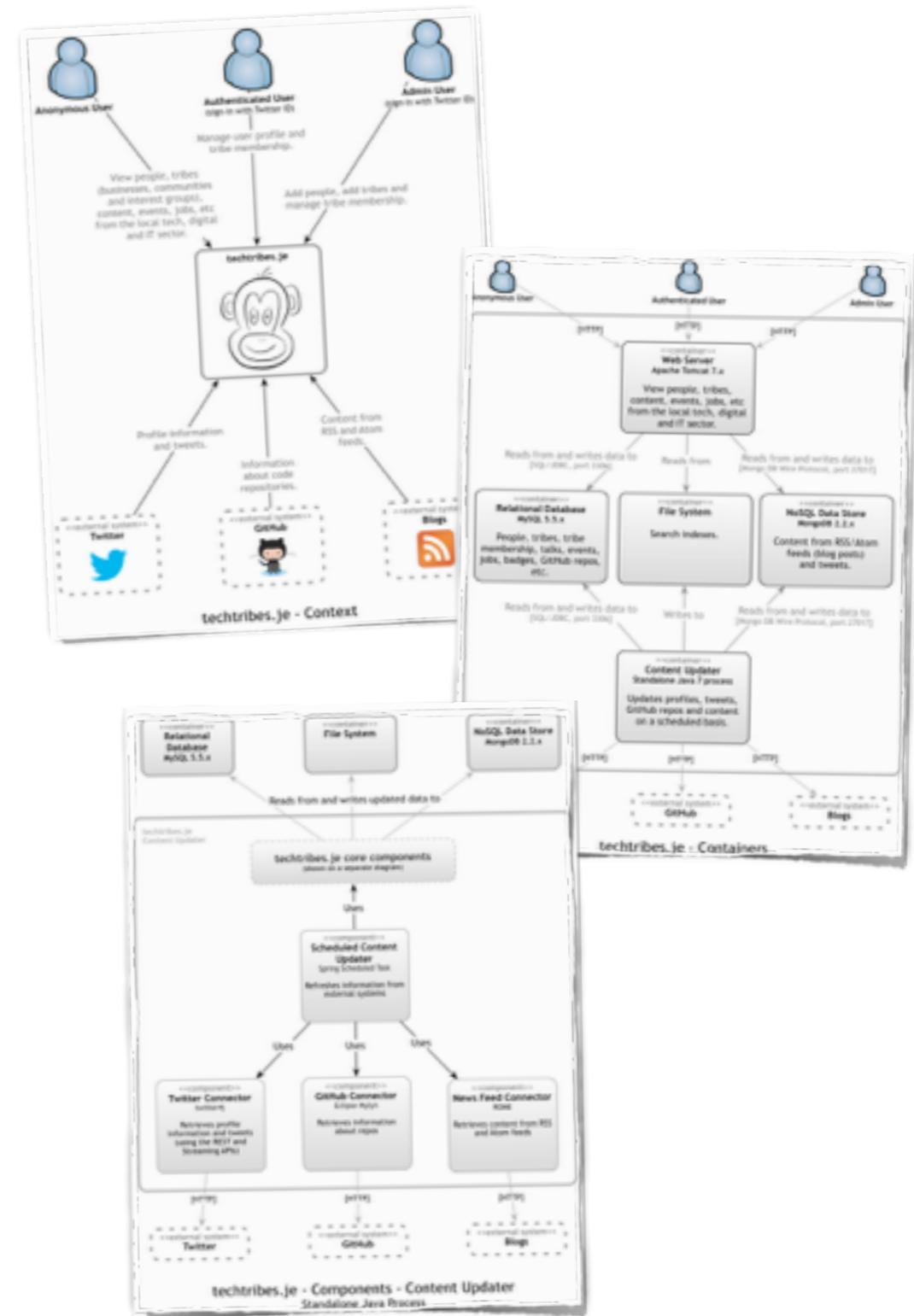
is about reducing detail
rather than creating a different representation



Abstractions help us
reason about
a big and/or complex
software system

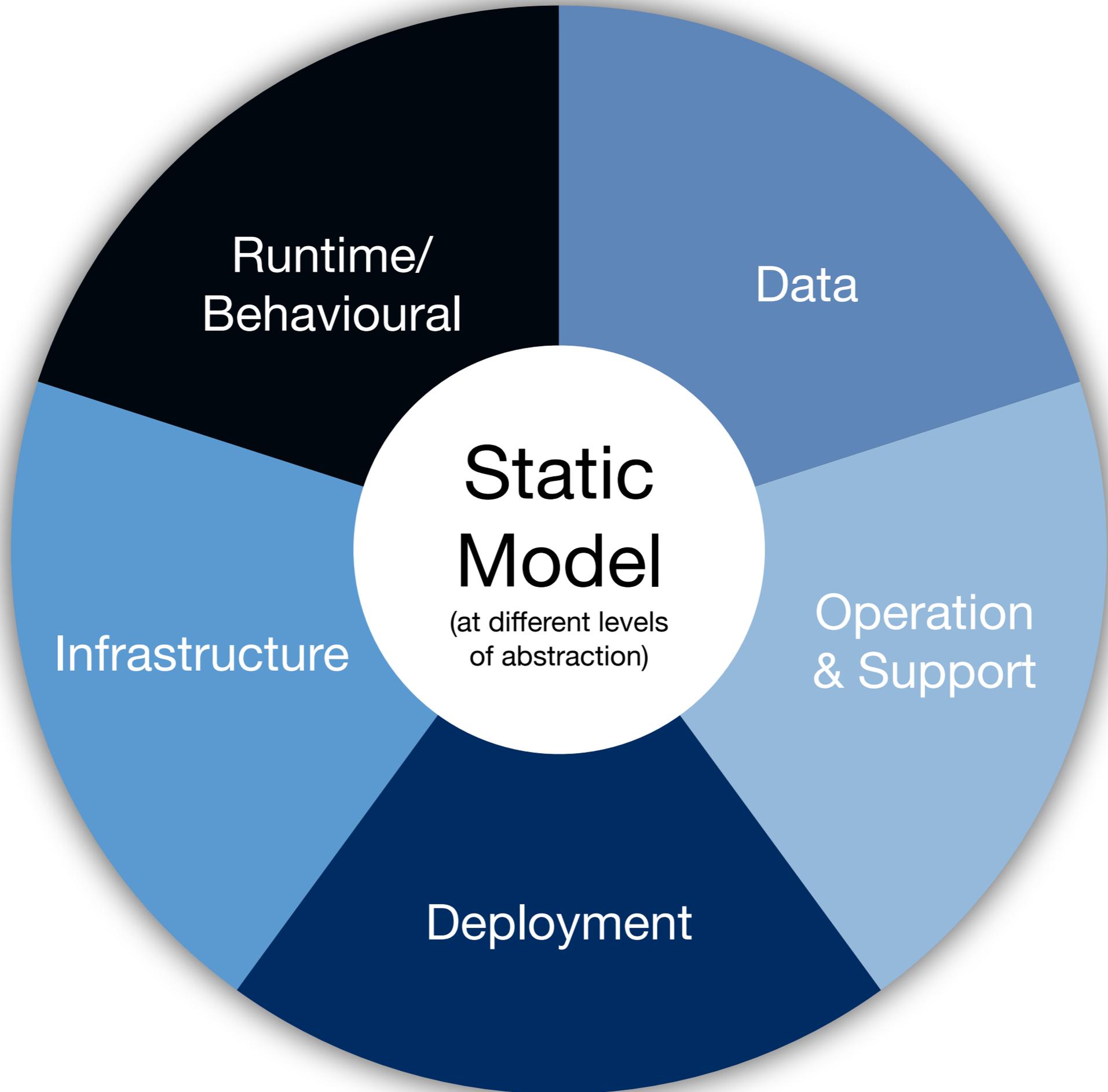
A common set of
abstractions

is more important than
a common notation



Sketches are maps

that help a team navigate a complex codebase



Runtime/
Behavioural

Data

Static
Model

(at different levels
of abstraction)

Operation
& Support

Infrastructure

Deployment

Does your code reflect the
abstractions
that you think about?

My focus is primarily on the

static structure

of software, which is ultimately about

code

Software developers are
the most important
stakeholders
of software architecture



Eoin Says ...

The point is that ...

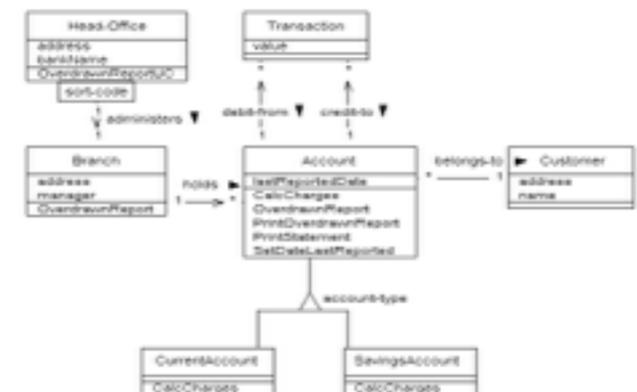
- Some models worth creating are worth preserving
- Models capture things that code can't
- Sketches the place to start ... but limited
- Models communicate, so ground rules are useful - UML is a good *base* to work from

What is modelling?

- A model is any simplified representation of reality
 - a spreadsheet of data
 - a Java domain model
 - a UML model
- Modelling represents concepts to allow some aspect of them to be understood

	A	B	C	D	E	F
1						
2						
3	Date	Start time	End time	Pause	Sum	Comment
4	2007-06-07	9.25	10.25	0	0	1 Task 1
5	2007-06-07	10.75	12.50	0	1.75	Task 1
6	2007-06-07	18.00	19.00	0	0	1 Task 2
7	2007-06-08	9.25	10.25	0	0	1 Task 2
8	2007-06-08	14.50	15.50	0	0	1 Task 3
9	2007-06-08	8.75	9.25	0	0.5	Task 3
10	2007-06-14	21.75	22.25	0	0.5	Task 3
11	2007-06-14	22.50	23.00	0	0.5	Task 3
12	2007-06-15	11.75	12.75	0	0	1 Task 3
13						
14						
15						
16						
17						
18						

```
public class TcpClientSample
{
    public static void Main()
    {
        byte[] data = new byte[1024]; string input, strOutput;
        TcpClient server;
        try
        {
            server = new TcpClient("...", port);
        } catch (SocketException)
        {
            Console.WriteLine("Unable to connect to server");
            return;
        }
        NetworkStream ns = server.GetStream();
        int rcv = ns.Read(data, 0, data.Length);
        stringData = Encoding.ASCII.GetString(data, 0, rcv);
        Console.WriteLine(stringData);
        while (true)
        {
            input = Console.ReadLine();
            if (input == "exit") break;
            if (input == "Auditing Department")
            {
                newChild.CommitChanges();
                newChild.Close();
            }
        }
    }
}
```



Why create models?



Communicate



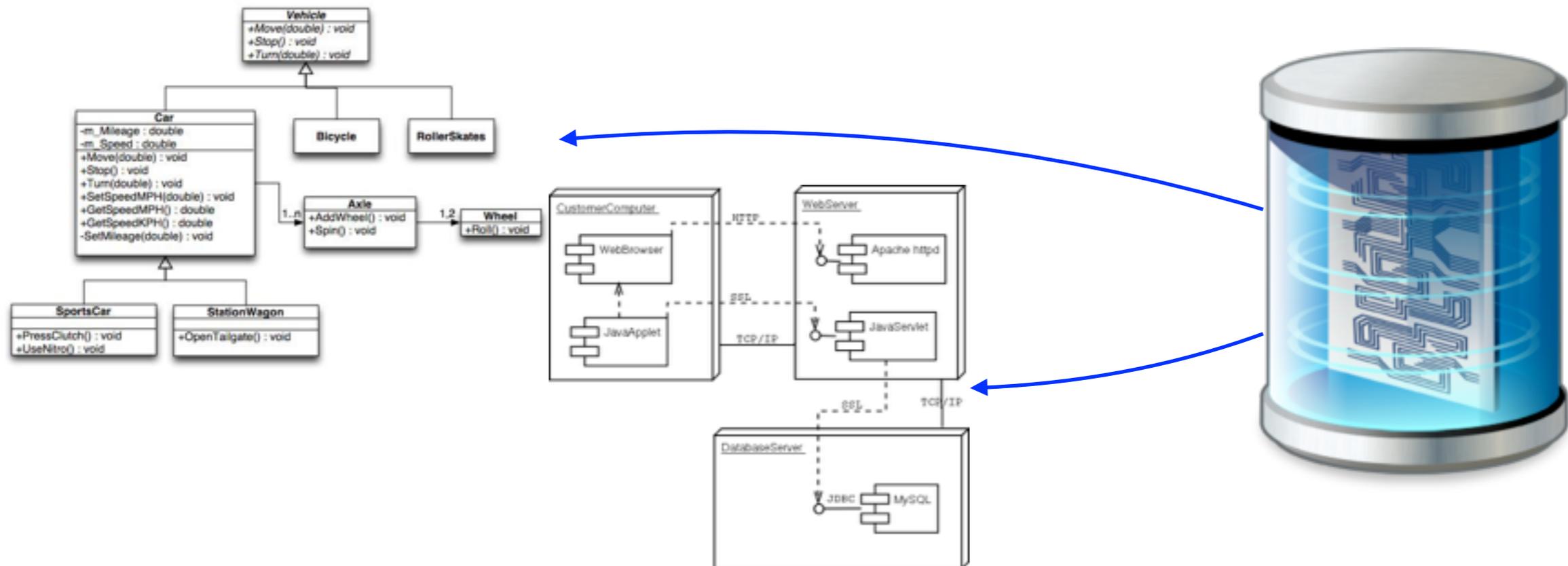
Record



Understand

Models vs diagrams

- A diagram is a purely visual representation
- A model contains definitions (and possibly a diagram)
 - In UML terms diagrams provide views of a model



Types of Model

High Detail



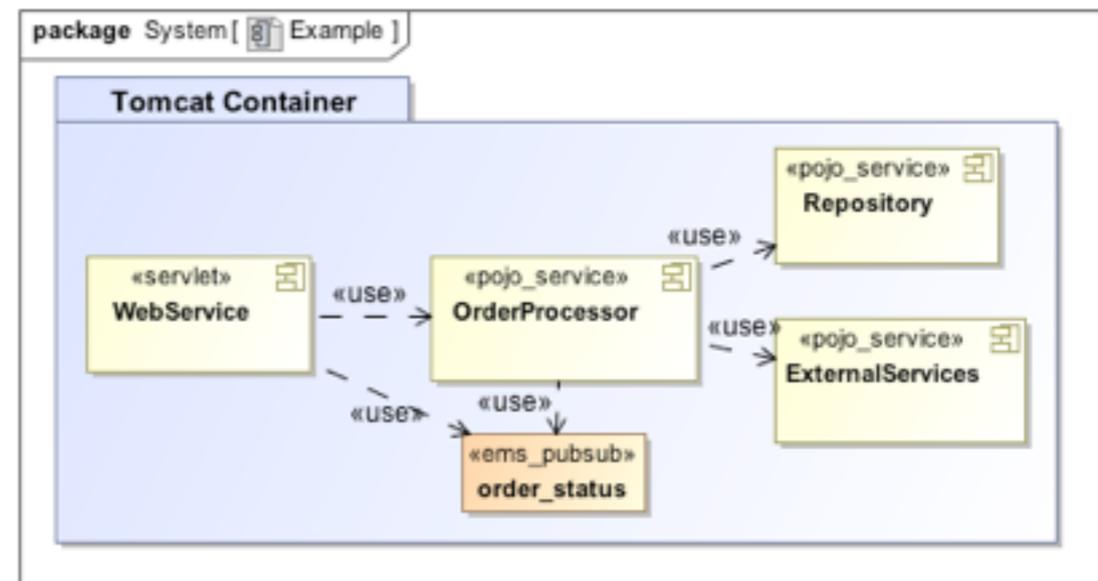
Low Precision



High Precision



Low Detail



Uses for models

- Consistency
 - change once, its changed everywhere
- Reporting
 - ask your model a question
 - “what is connected to the Flange Modulator Service?”
- Checking and Validation
 - do I have a deployment node for every piece of the system?
 - how complicated is the system going to be?
- Sharing information
 - generate many views of a single model
 - Powerpoint, wiki, tables, ...

An Analogy

- Would you use JSON to represent your shopping list?
 - I personally use a PostIt™ note
- Would you hold system configuration in free text?
 - I personally would rather XML or JSON
- Long lived models are valuable ... store them as data
 - UML is a practical option for machine readable models

Some Questions and Answers

Q1. Modelling - Why Bother?

- Simon:

- A model makes it easy to step back and see the big picture.
- A model aids communication, inside and outside of the team.
- Modelling provides a ubiquitous language with which to describe software.

- Eoin:

- Modelling helps you understand what you have and need
- You can't understand all of the detail anyway
- Code is in fact a model, we just don't think of it as such

Q2. Modelling and Agility

- Simon:
 - Good communication helps you move fast.
 - A model provides long-lived documentation.
 - A model provides the basis for structure, vision and risks.
- Eoin:
 - No fundamental conflict - “*model with a purpose*” (Daniels)
 - Working software *over* comprehensive documentation
 - Agility should be for the long haul, not this sprint
 - Can you know all the feed dependencies from your system?

Q3. How to Do It?

- Simon:
 - Start with the big picture, and work into the detail.
 - Stop when you get to a “sufficient” level of detail.
 - Include technology choices!
- Eoin:
 - Start small, start with a definite purpose
 - Start with a whiteboard or a napkin or an A4 sheet
 - Skip Visio and Omnigraffle ... get a tool, get a model

Q4. UML - Is It Worth the Hassle?

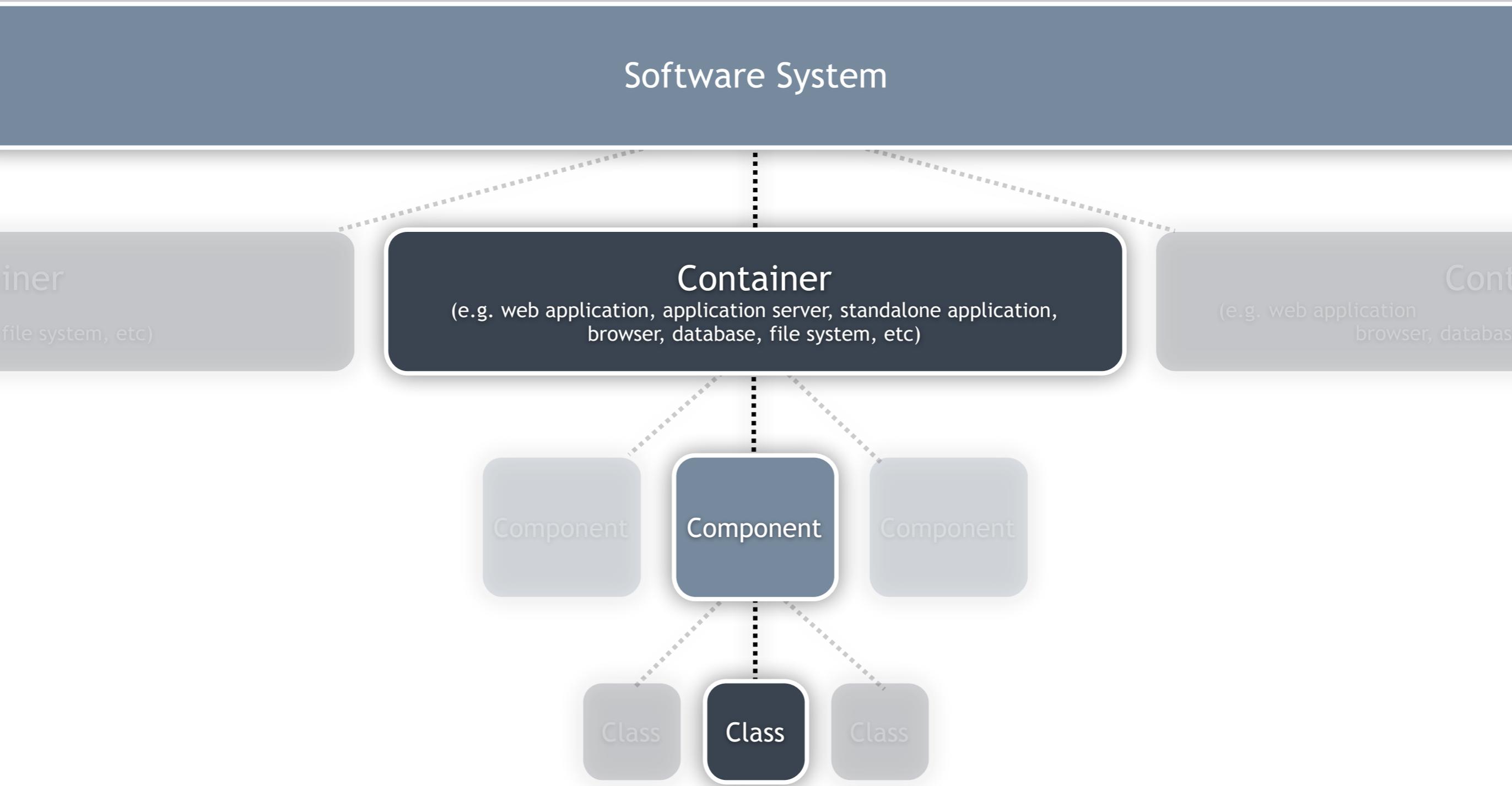
- Simon:
 - No.
- Eoin:
 - Maybe ... depends what you need
 - Would you write a shopping list in JSON? Would you store configuration settings in a free text file?
 - If you have long lived models and want to use the data then *yes, highly tailored* UML is worth the effort

Q5. Modelling in the Large vs the Small

- Simon:
 - Sketches will quickly become out of date.
 - Reverse-engineering tends to lead to cluttered diagrams.
 - Many small diagrams are better than one uber-diagram.
- Eoin:
 - A large system means you need help from a computer to understand it
 - However large your model, the code is still “the truth”
 - Modelling languages scale like programming languages

How We Do It

Simon



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

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

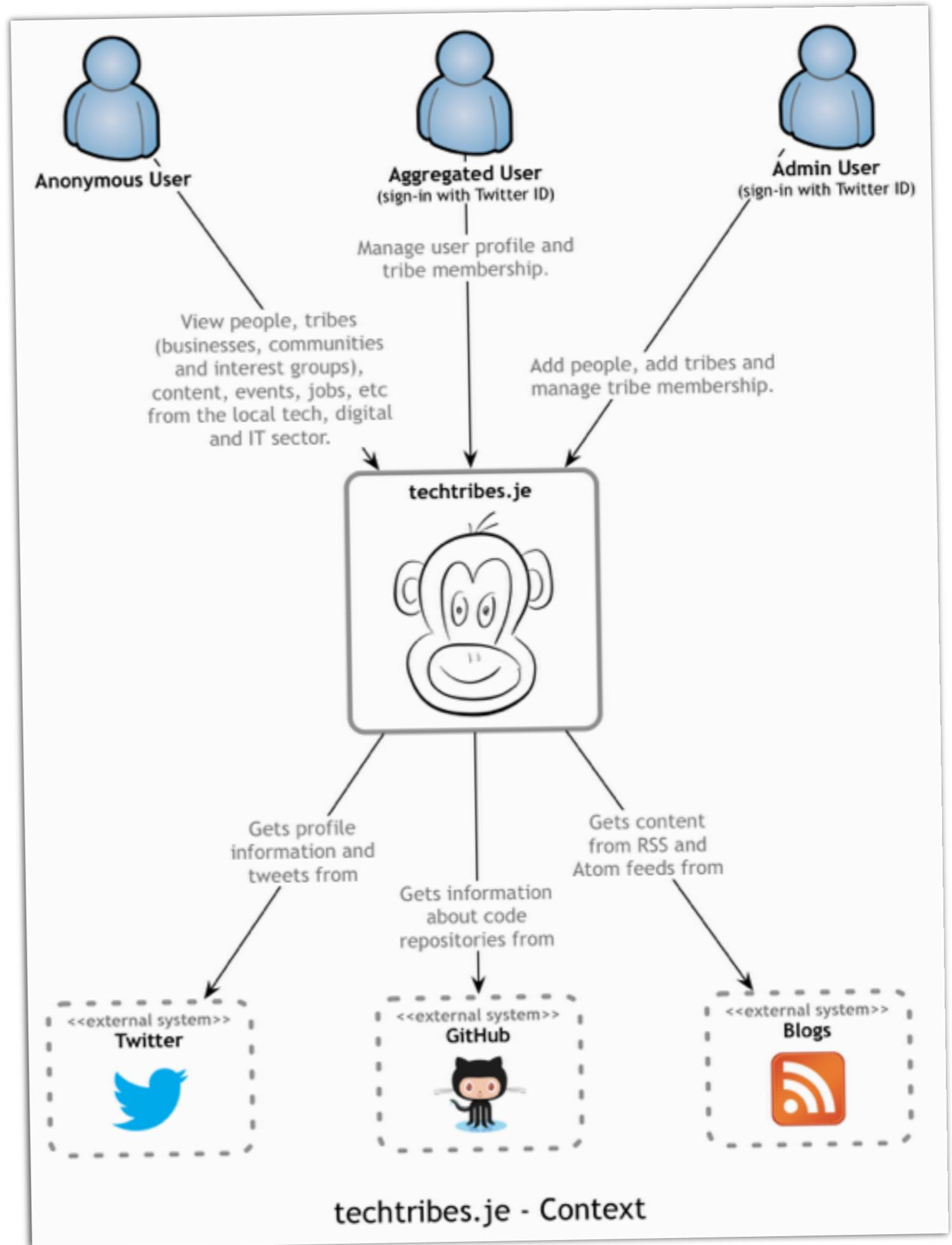


Classes

Component or pattern implementation details

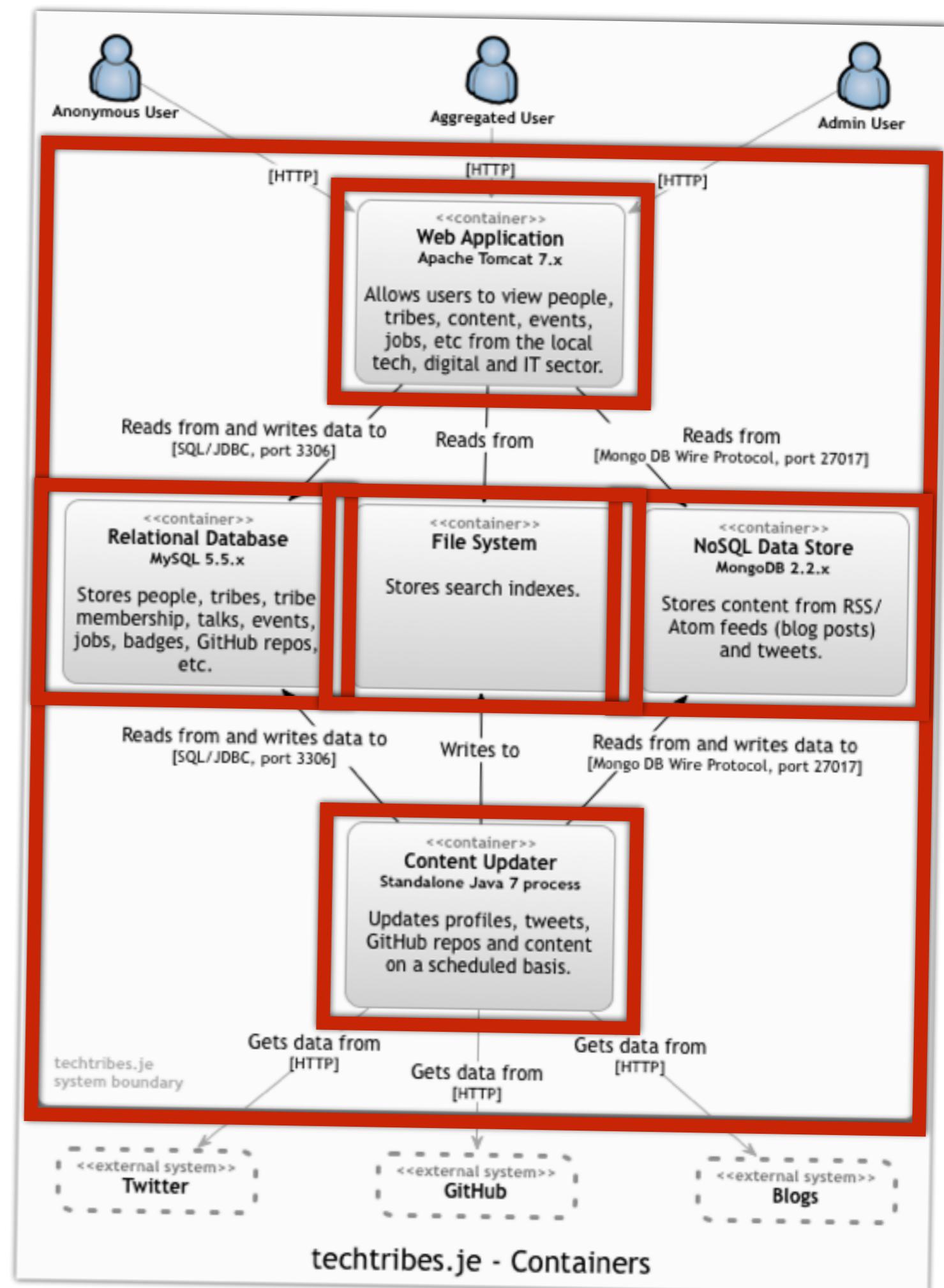
Context

- 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)



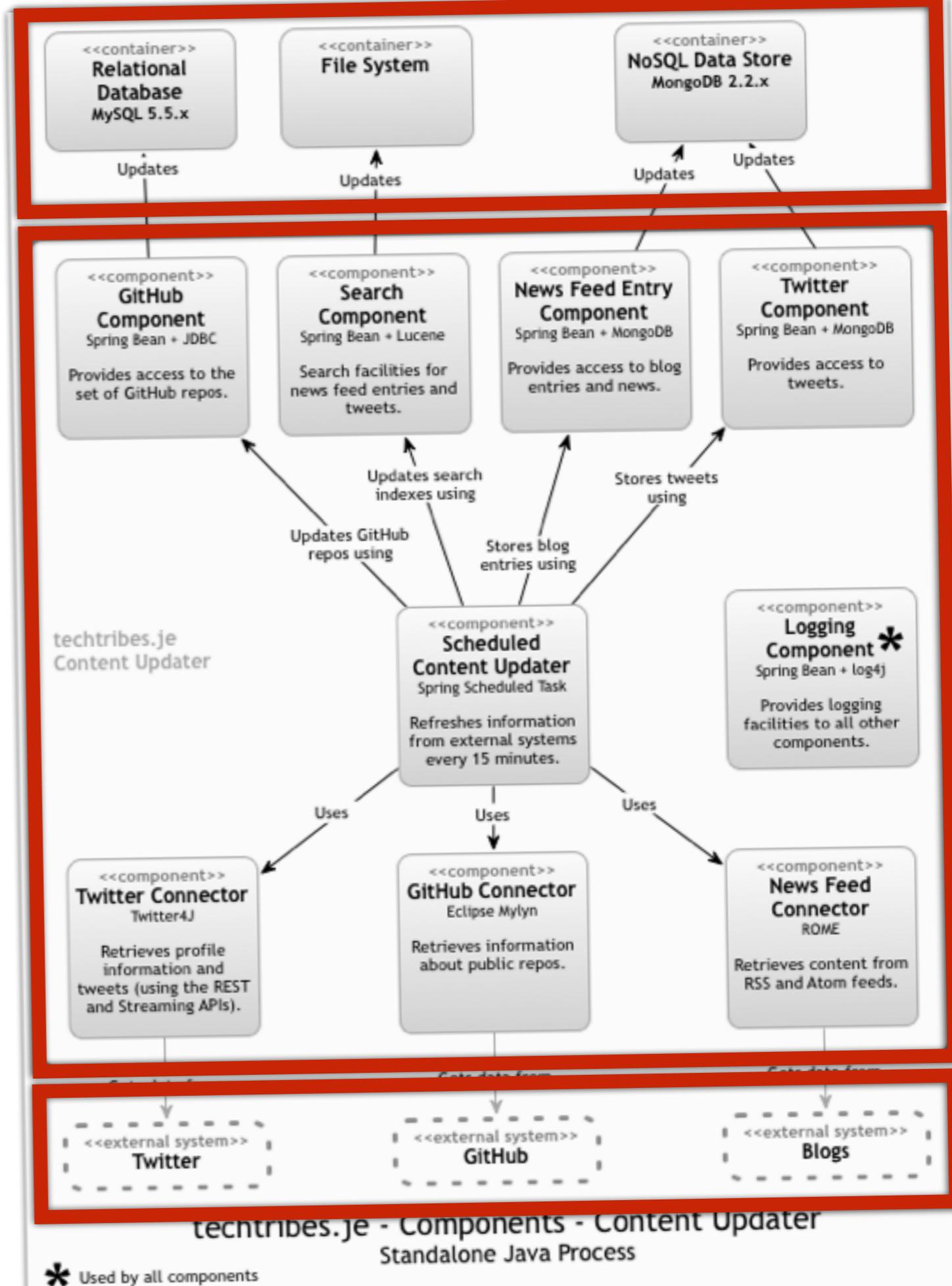
Containers

- 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?



structurizr.com

```
/**  
 * This is a C4 representation of the Spring PetClinic sample app  
 */
```

```
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.com");
```

```
        // 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 Target 7.x");
```

```
        Container relationalDatabase = springPetClinic.addContainer("Relational Database Target 7.x");
```

```
        user.uses(webApplication, "Uses");
```

```
        webApplication.uses(relationalDatabase, "Uses");
```

```
        // and now automatically find components
```

```
        ComponentFinder componentFinder = new ComponentFinder(model);
```

```
        new SpringComponentFinder(componentFinder);
```

```
        componentFinder.findComponents();
```

```
        // connect the user to all of the components
```

```
        webApplication.getComponents().forEach(component -> user.uses(component, "Uses"));
```

```
        // connect all of the repositories to the web application
```

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?

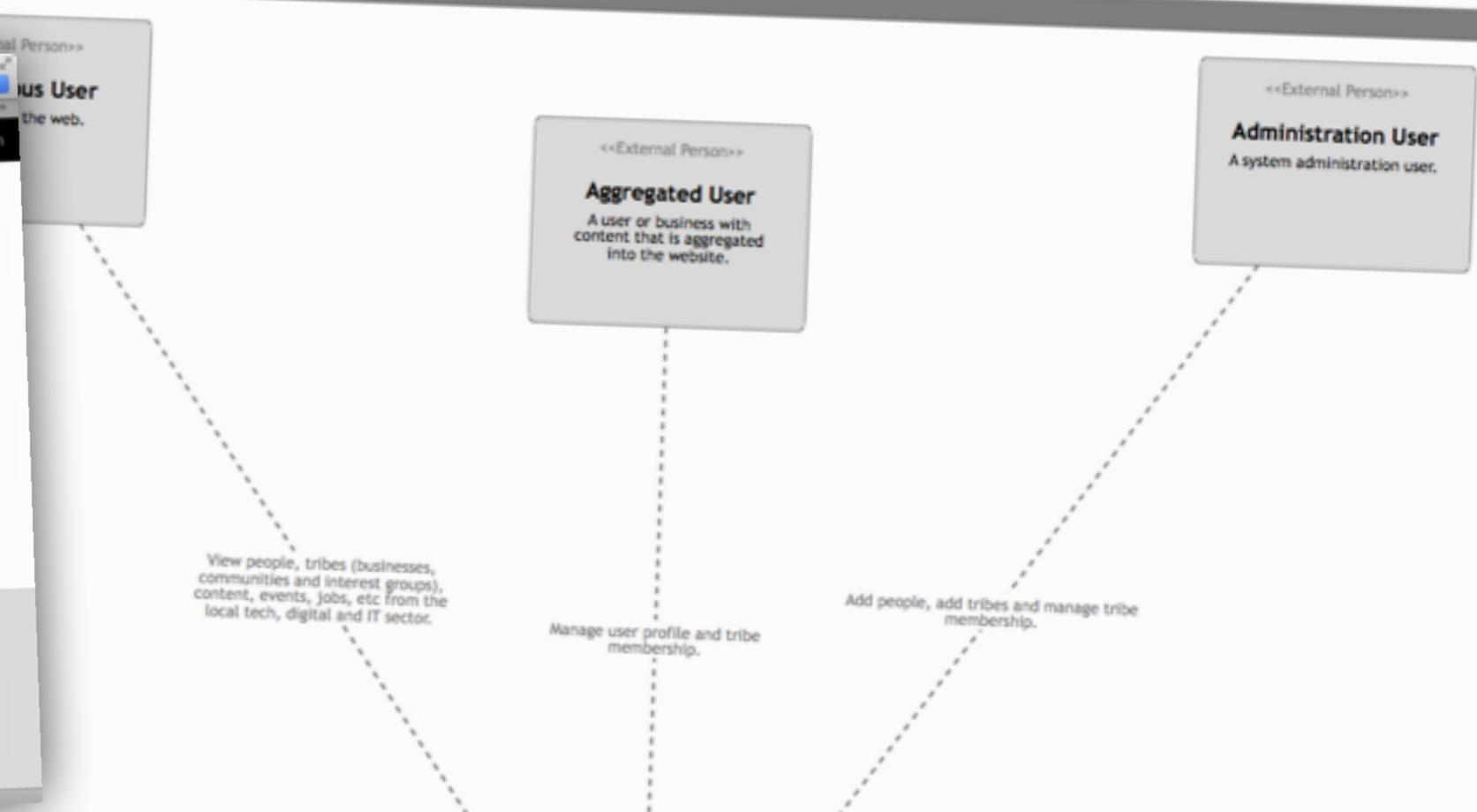
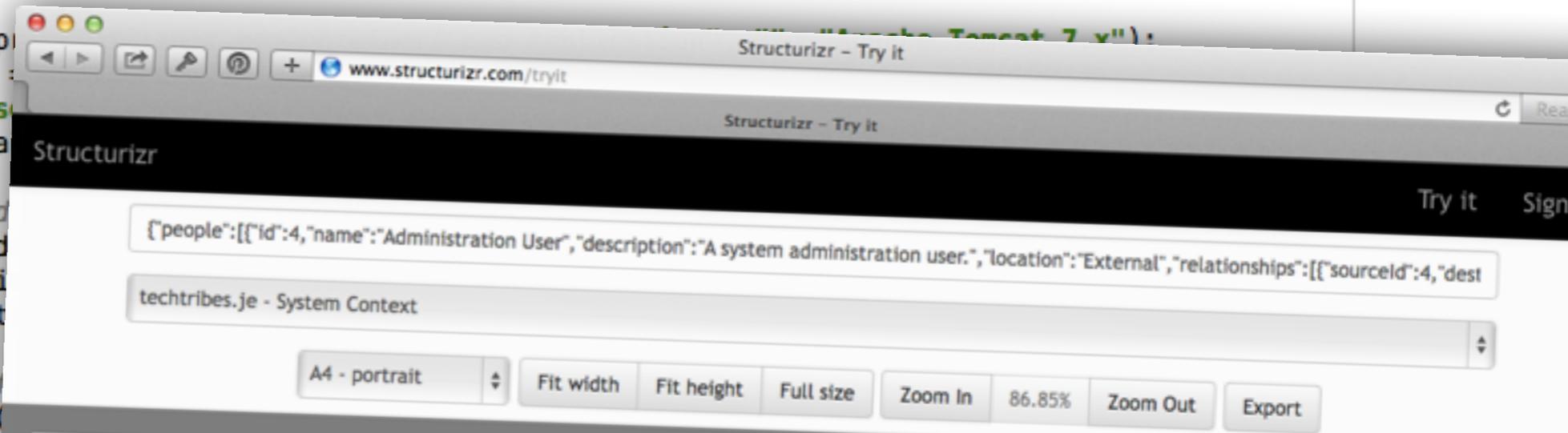
A Java library to create a JSON model can be found on GitHub, as can Mike Minutillo's high-level DSL for creating a C4 model in .NET called *ArchitectureScript*.



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.

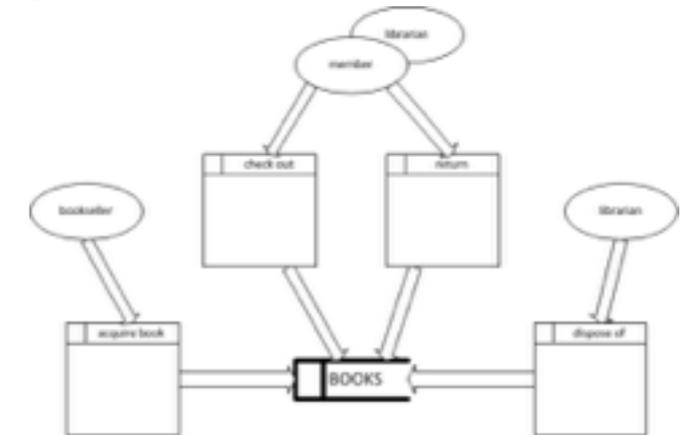
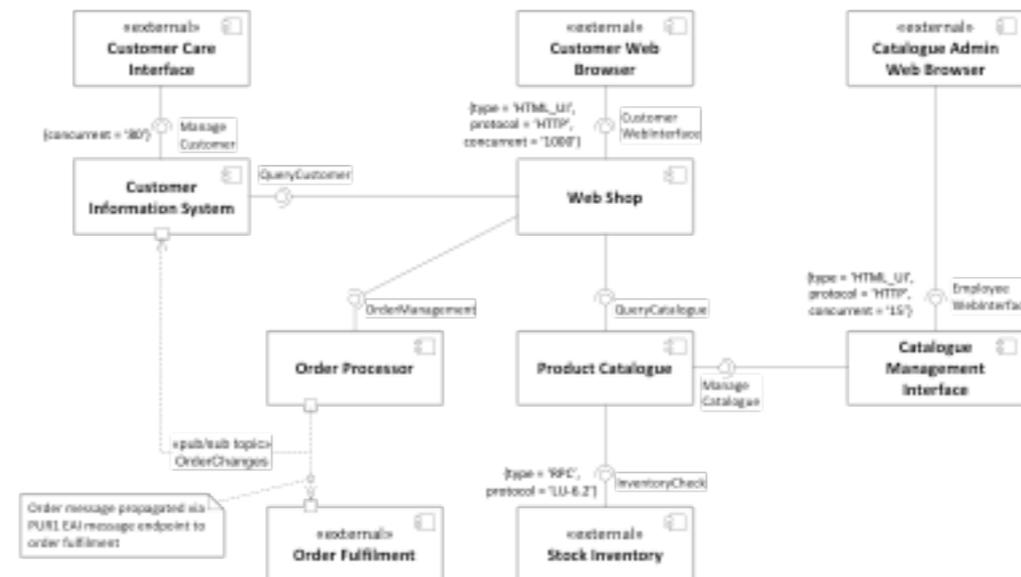
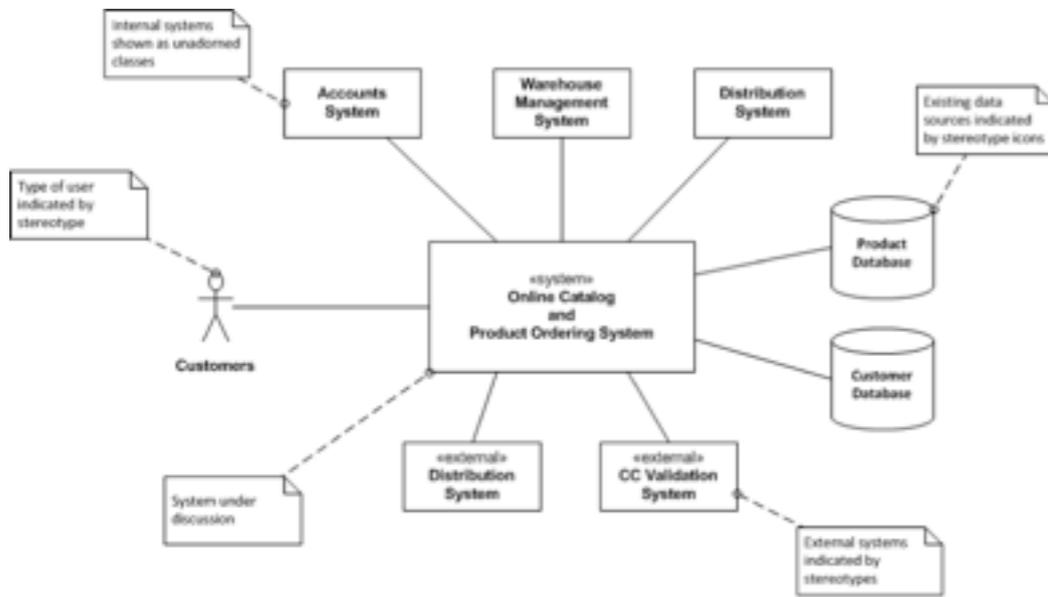
Detail isn't important here as this is your zoomed out view showing a big picture of the



Eoin

Common Types of Models

- System Environment - context view
- Run Time Structure - functional view
- Software meets Infrastructure - deployment view
- Stored and In-Transit Data - information view



The Viewpoints and Perspectives model

Context View
(where the system lives)

Functional View
(runtime structure)

Information View
(data moving & at rest)

Concurrency View
(processes and threads)

Development View
(code structures)

Deployment View
(system meets infra)

Operational View
(keeping it running)

Context View

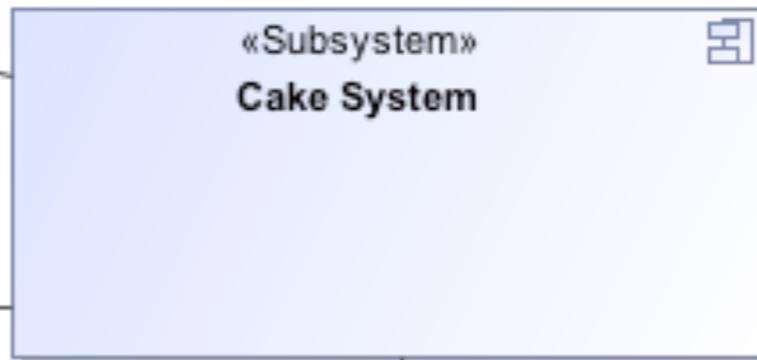
package Context [ CakeSystem]



Customer

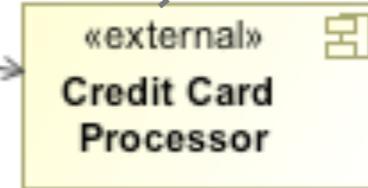


Baker



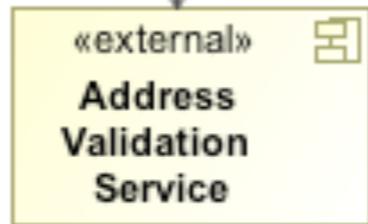
«Subsystem»
Cake System

authorisation_requests



«external»
Credit Card
Processor

address_lookups



«external»
Address
Validation
Service

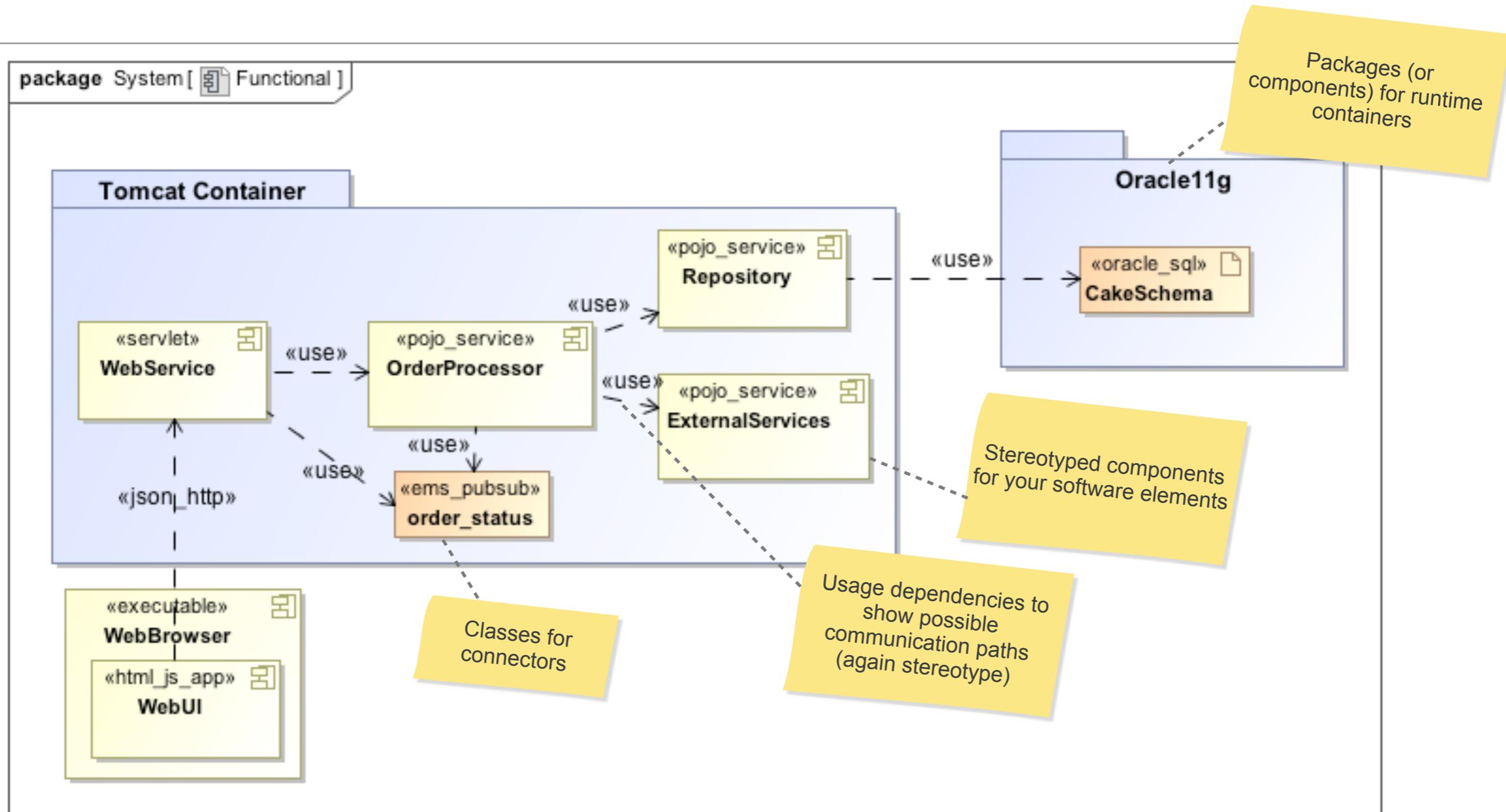
Component diagram with a single "component" - your system

External systems represented as «external» components

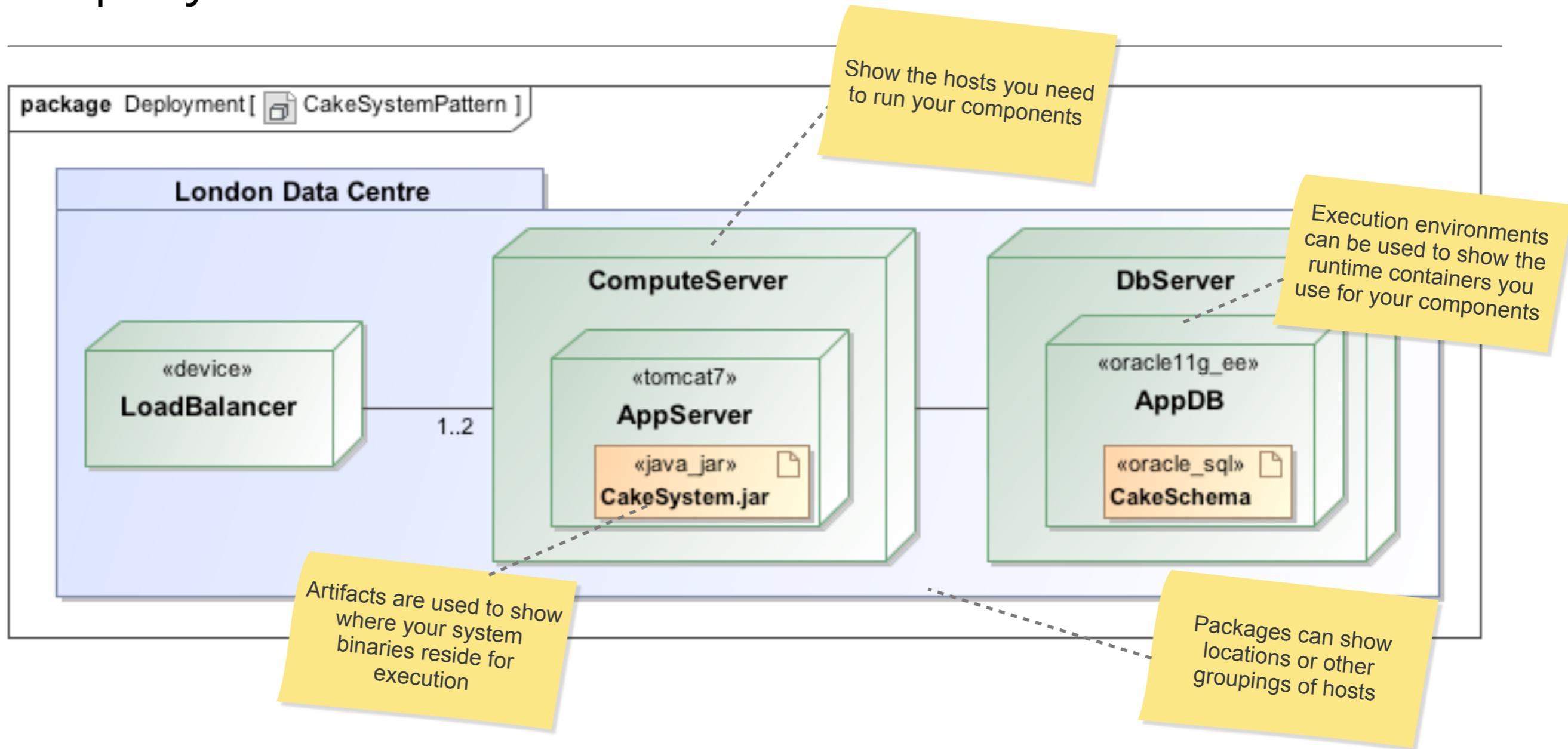
User groups represented by actors

Interactions with external systems using named associations

Functional View



Deployment View



Summary and Conclusions

What We Have Talked About

- Modelling is terrifically useful
 - communication
 - clarity
 - analysis
- Many ways of doing it
 - napkins to UML tools
- The key point is to get value from what you do
 - don't get stuck in “analysis paralysis”

Questions?

Simon Brown

www.codingthearchitecture.com

@simonbrown

Eoin Woods

www.eoinwoods.info

@eoinwoodz