Models, Sketches and Everything In-Between

Eoin Woods, Endava

JAX Finance
London, April 2015

Welcome

- Hello from me
  - Eoin Woods, CTO at Endava
  - 10 years at UBS and BGI (BlackRock)
  - Architecture and Delivery
  - Primarily Equities areas

- And thanks to him
  - Simon Brown, Coding the Architecture
  - Co-developed this material

Agenda

- Our Opinions
  - On communicating software architecture
  - On communicating with models
- Two Ways to Do It
  - The C4 Approach
  - The Viewpoints & Perspectives Approach
- Questions and Queries
- Summary and Conclusions

Background

- We have talked software modelling for ages
  - We both think its a good idea (in moderation)
  - Simon likes boxes and lines, Eoin likes UML (a bit)
  - Simon has C4, Eoin has V&P (with Nick Rozanski)
  - We have both inflicted a book on the world …

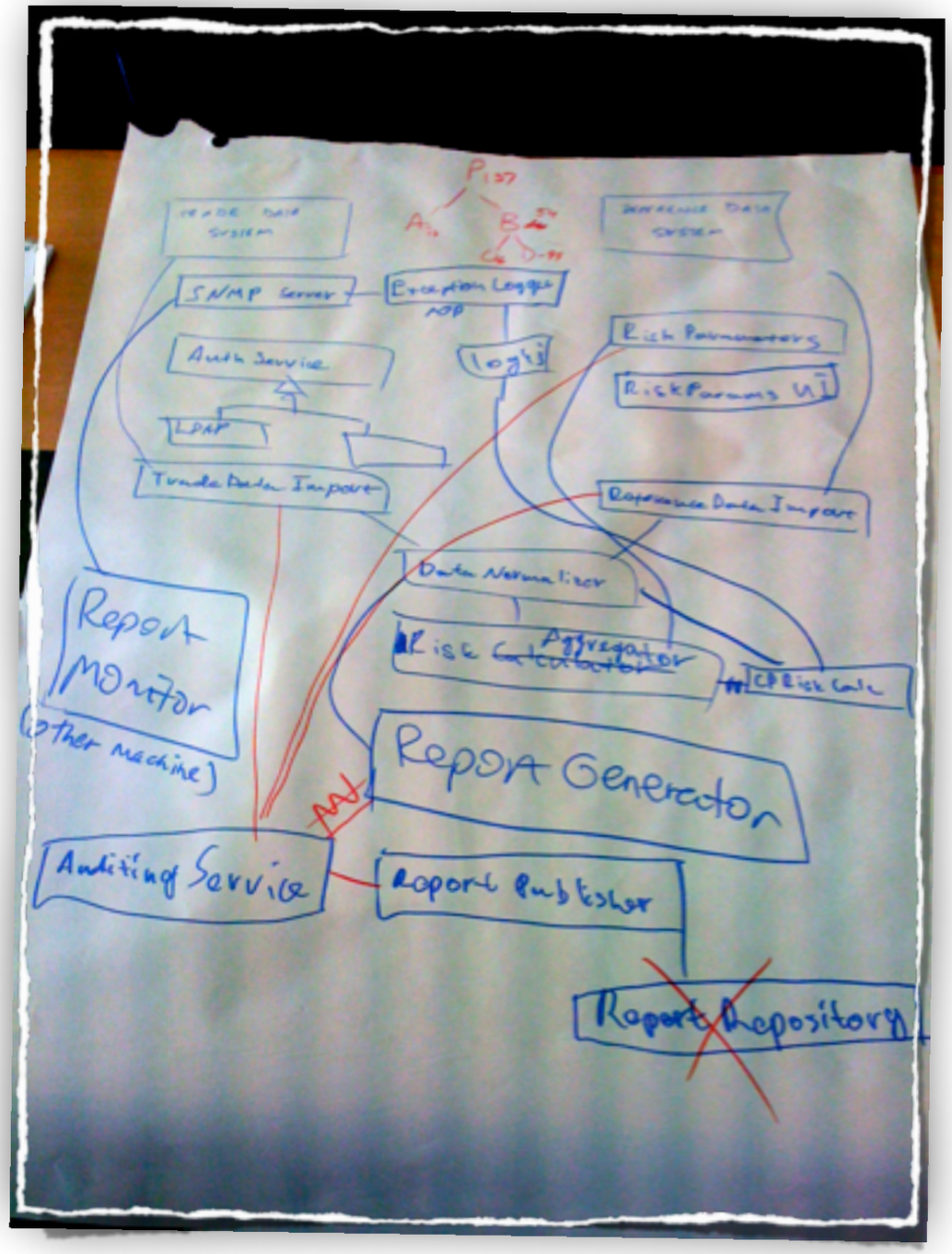
- We tried to work out the story together
  - I've got questions and answers, but let's hear yours too

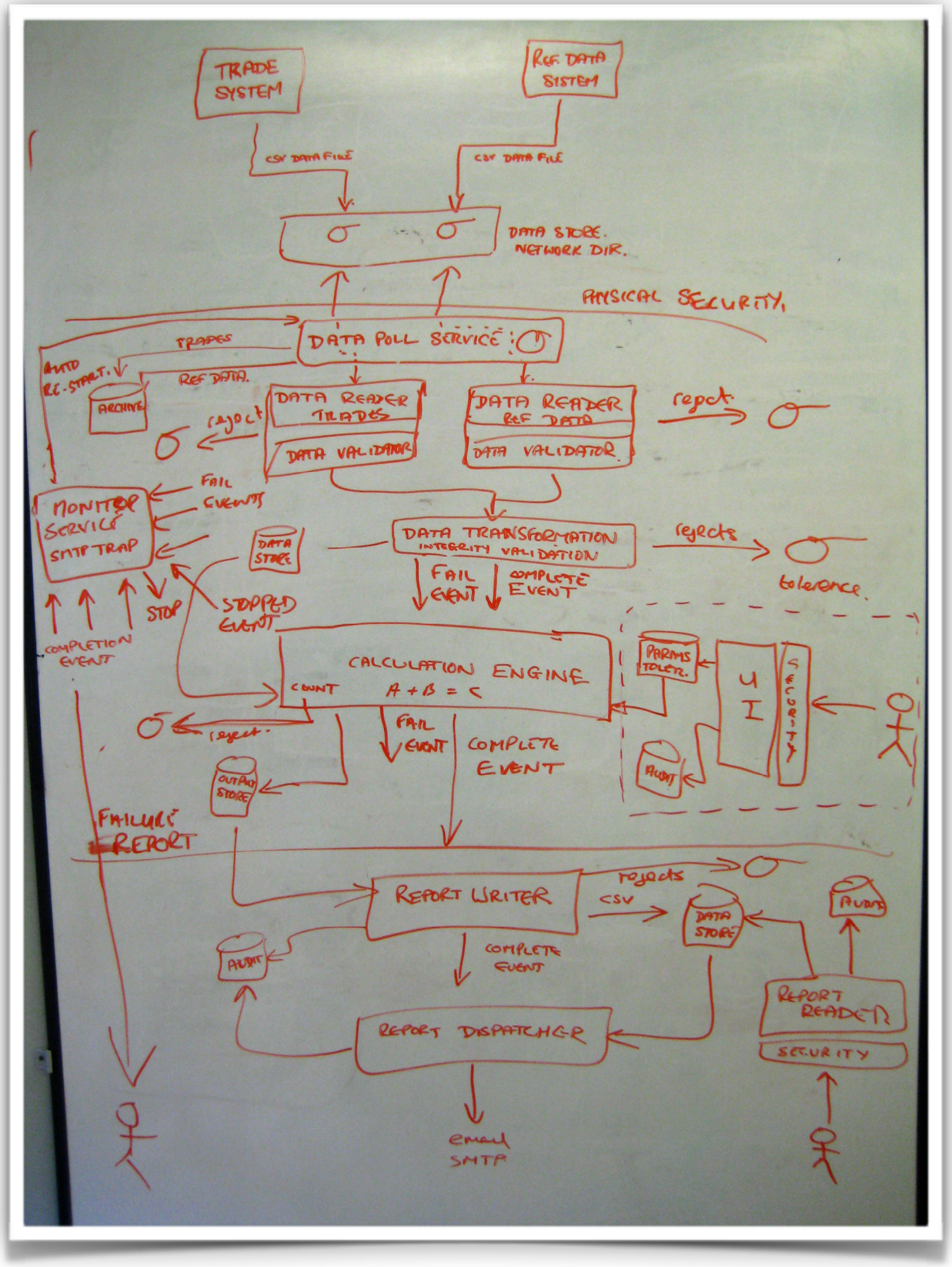
Our Opinions

On Communicating
Software Architecture

endava

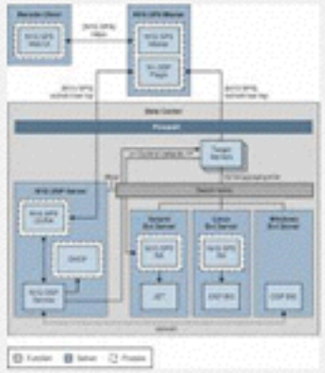
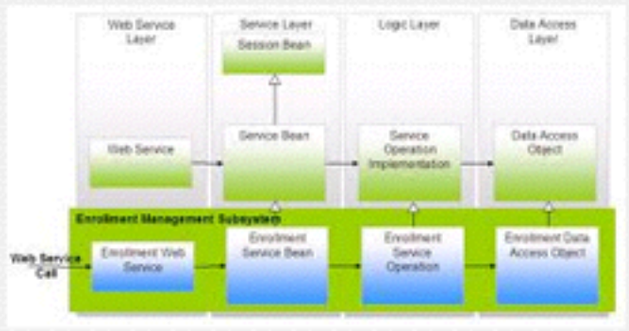
Many teams find it difficult to communicate the architecture of their systems effectively

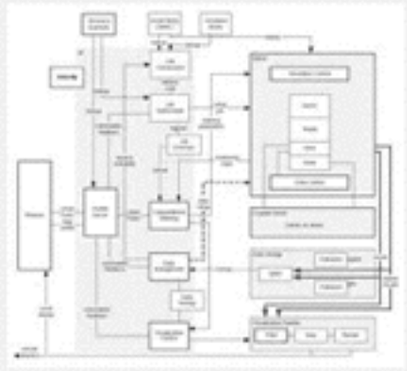
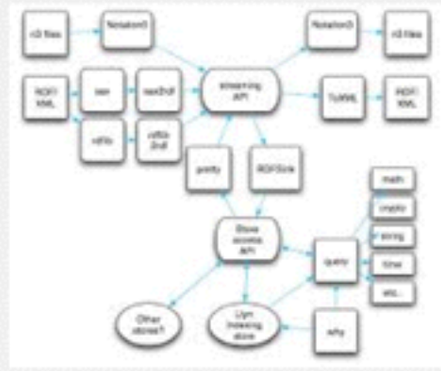


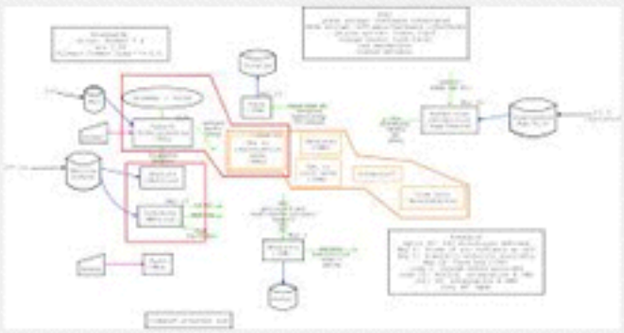
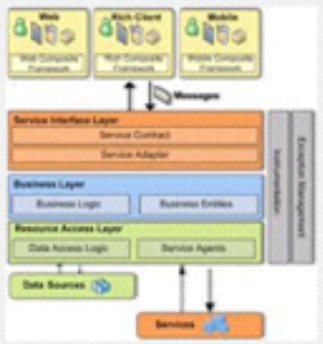


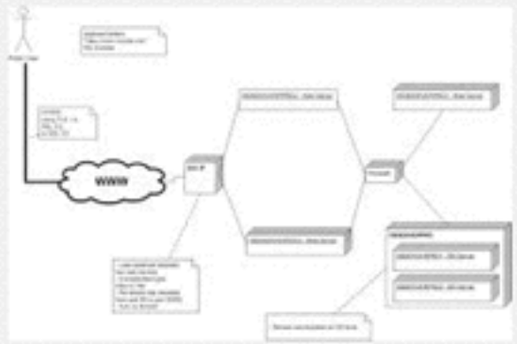
It's usually difficult to show the entire design on a **single** diagram

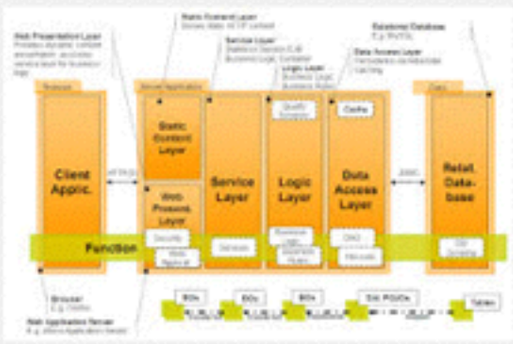
Different **views** of the design can be used to manage complexity and highlight different aspects of the solution

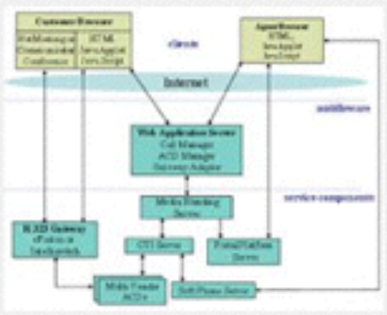




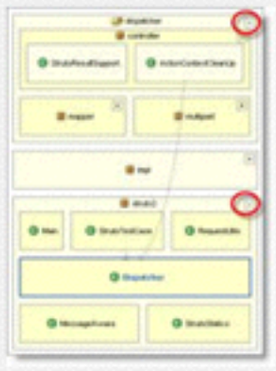



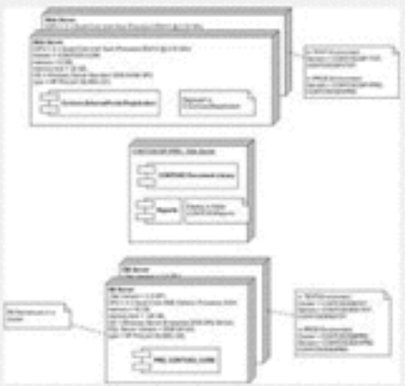
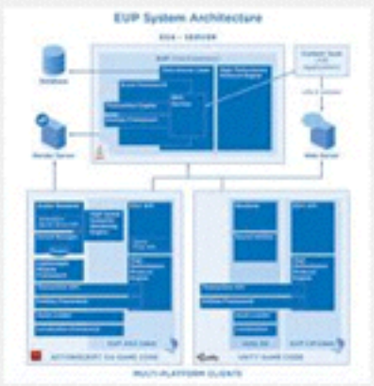



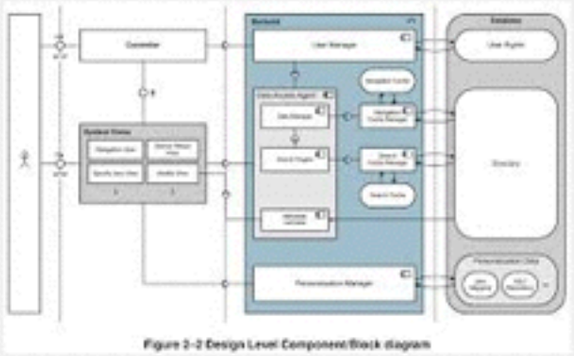


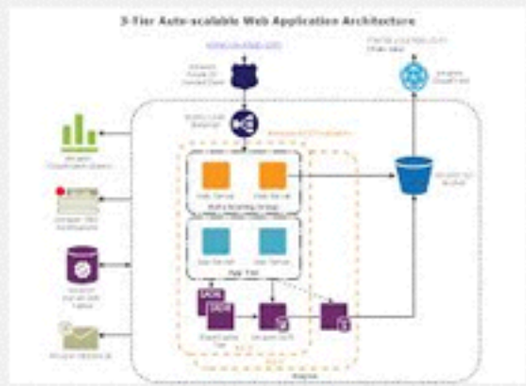





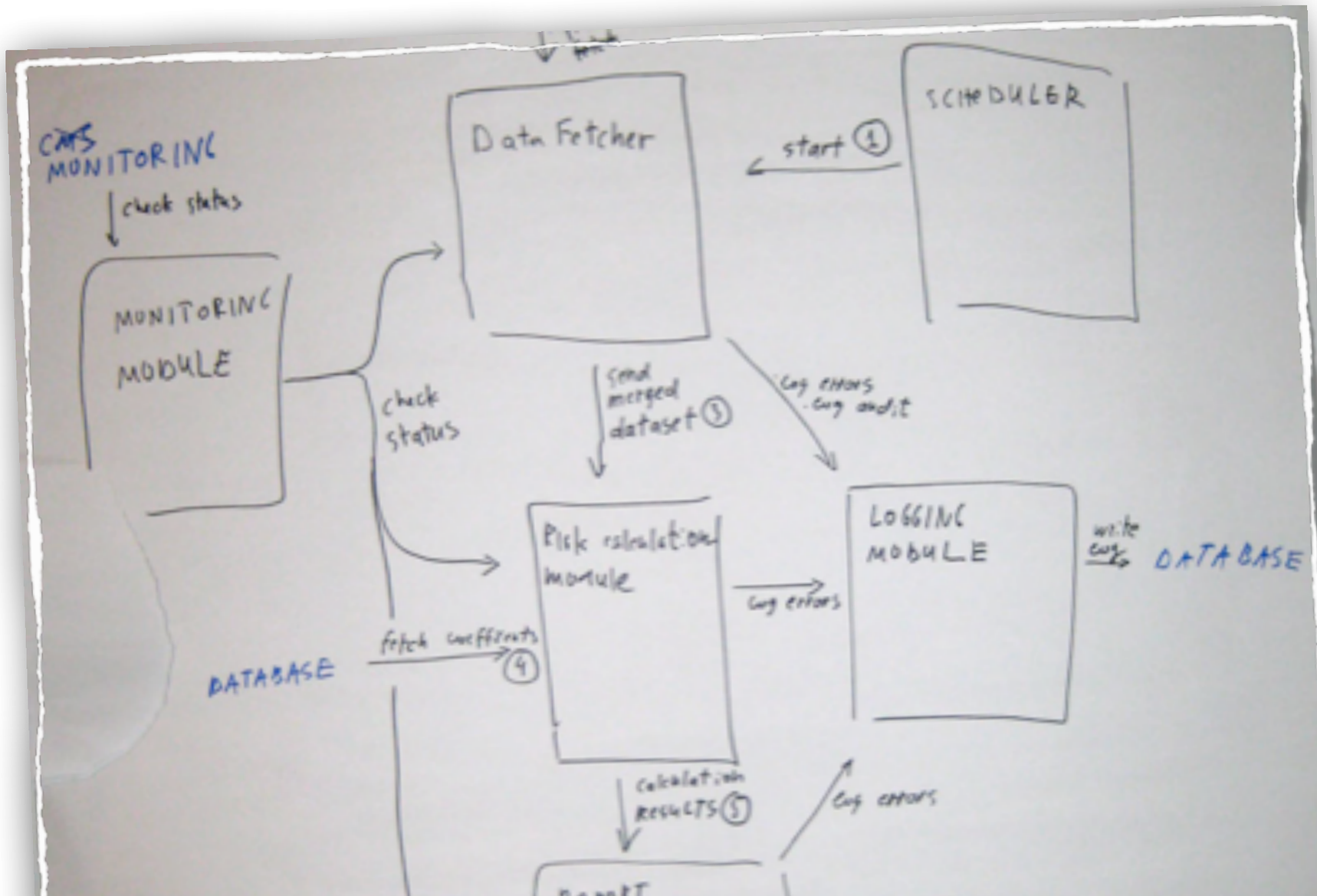








Video over IP
Capturing video frames from Camera
Format conversion and Pre-processing
H.264 Encoder
NALU Fragmentation
Audio/Video sync module
Format conversion and Post-processing
H.264 Decoder
Reconstruction of NALU
RTP/ RTCP
Voice over IP
Acoustic echo cancellation (AEC)
LEC (G.168/ G.165)
AGC - Gain Control
VAD
Speech Encoder
Audio Interface layer
Sound card interface (PCI / USB)
FXS/TDM Interface
Telephony interface layer - Tone detection/generation, CPT/DTMF, MF-R1/R2, and Voice Fax Modem Discrimination Unit
Administration UI
Reservation Manager UI
AutoShell
Packaging
Process Automation
Configuration Management / Compliance
Help Desk
Reporting (Jasper)
Agent Remote Deployment
Service Response Monitoring
CA IT Client Manager
CA Patch Manager
CA Process Automation
CA Configuration Automation
CA Service Desk
Chargeback
Agent Policy Configuration
Server Monitoring (Local & Remote)
Web Services and Reliable Transport (Active MQ)
Storage
Platform Management
Microsoft HyperV
Automation Infrastructure Platform
AOM Data Service
Policy, Performance, and Resource Management
Authentication / Security (CA EEM)
State and Event Management
Scheduler (Windows)
Discovery (CA Network Discovery Gateway)
Management DB
Performance DB
Service Controller
Rapid Server Imaging
Rich UI
Web UI
Controls
Service Interface
Activity
Business Rules
Human Workflow
Workflow
Service Agents
DAL
E-Publish
EAI
ECM
DW
OLTP
Signing
Authentication
Authorization
Monitoring
Log & Trace
Exception Management
Configuration
user 1
Project Manag
Project Editing Service
Management Service
Computational Analysis Service
e-AIRSmesh
e-AIRSview
Mesh Converter
CFD Solver
Output Converter
Remote Experiment Service
Exp. Exec.
Exp. Info.
Aerodynamic Exp.
PIV Exp.
Parametric Study
Validation
Analysis
Collaboration
Session Reservation
Resource Configuration Service
Resource Monitoring Service
P / HTTP
ASP (Scientific Application Service Provider)
Basic Execution Service
Parameter Sweeper
Global Scheduler
Resource Catalog
GRAM adapter
Plotting Service
eAIRS Plot
Conv. Graph
Data Transfer Service
Metadata Management Service
AG + VNC
Visualization Sharing
Mobile Service Component
A/V Conference
Chat
Application
Presentation Sublayer
Sublayer
File Editor
Library
ASCII Files
SCADA Develop. Environ
Export / Import
Commercial DB
Project Editor
Commercial Develop tools
Driver Toolkit
Client
Server
Publish
Subscribe
TCP/IP
SCADA Server
Recipe DB
Recipe Managt
Ref. DB
Data Proces-sing
RT DB
Report Gener.
SQL
Alarm
Alarm DB
Log
Log DB
Archive
Archive DB
RT & Event Manager
ODBC
DDE
API DLL
Data R/W
Driver
OPC
Private Application
EXCEL
VME
PLC
PLC
Alarm Display
Log Display
Active X Controls
3rd Party Applic.
Active X Container
Typical generic software architecture of SCADA systems
ote administration interface
Output device manager
Output devices
User manager
DB-Server
Analysis

Abstraction
should allow you to remove (unnecessary) detail



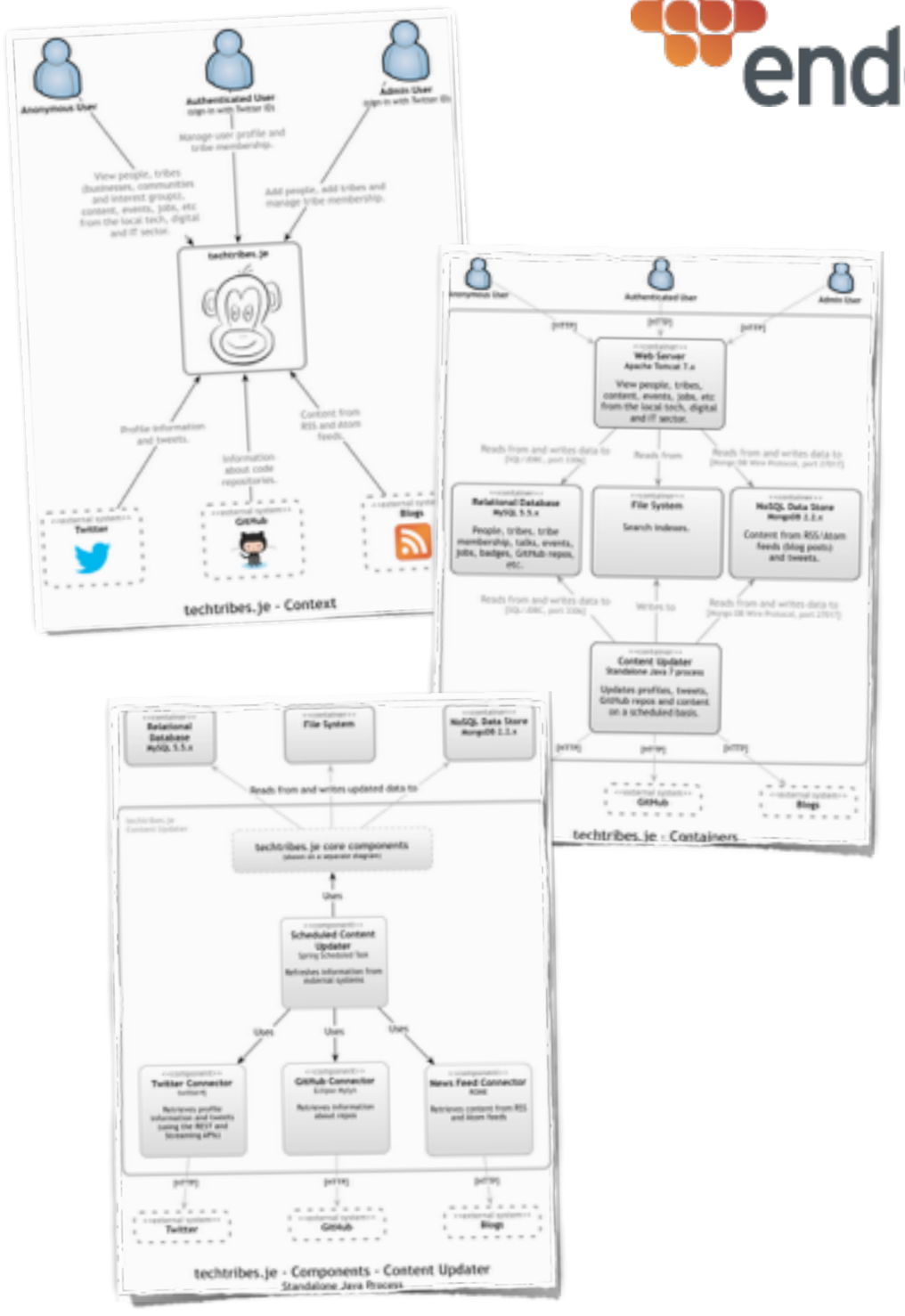
endava

Abstractions help us

reason about

large and complex

software systems



Sketches can work as maps

that help a team navigate a complex codebase

Does your code reflect the

abstractions

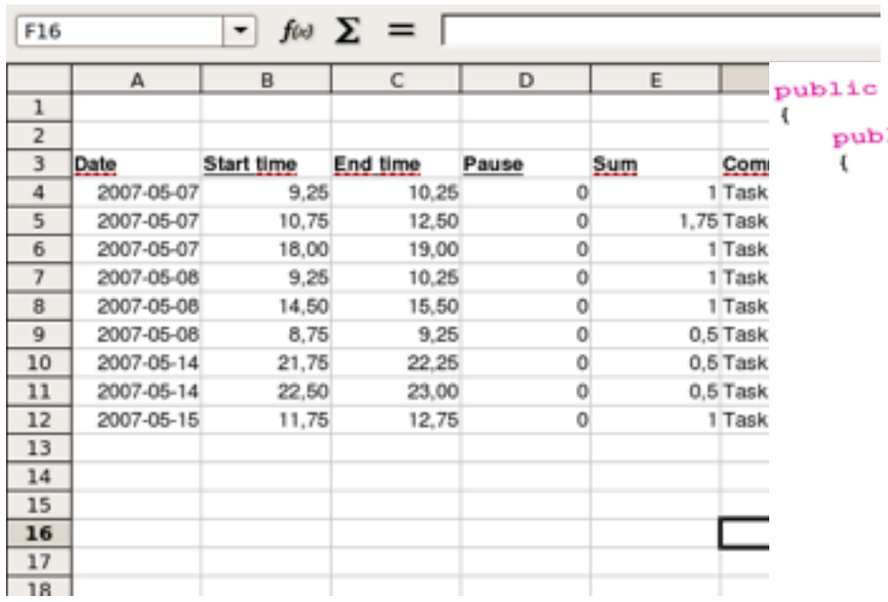
that you think about?

(many don’t)

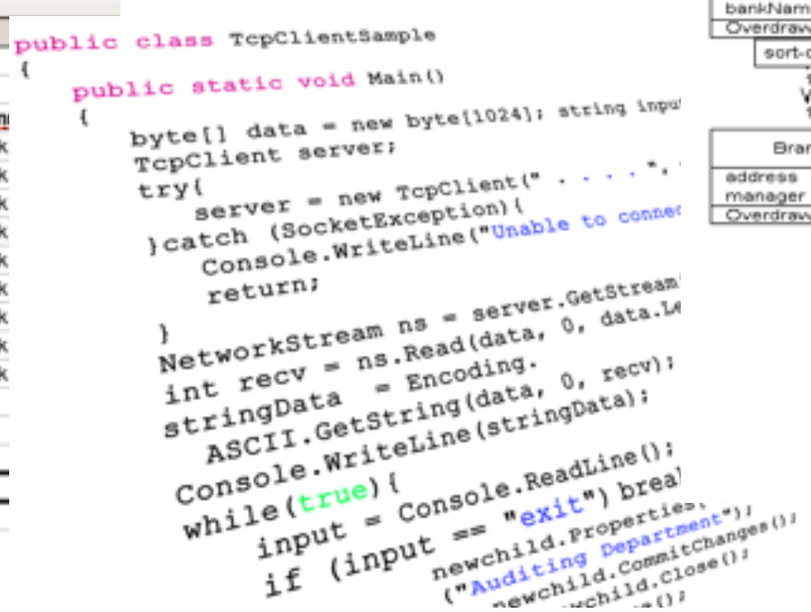
On Communicating with Models

What is modelling?

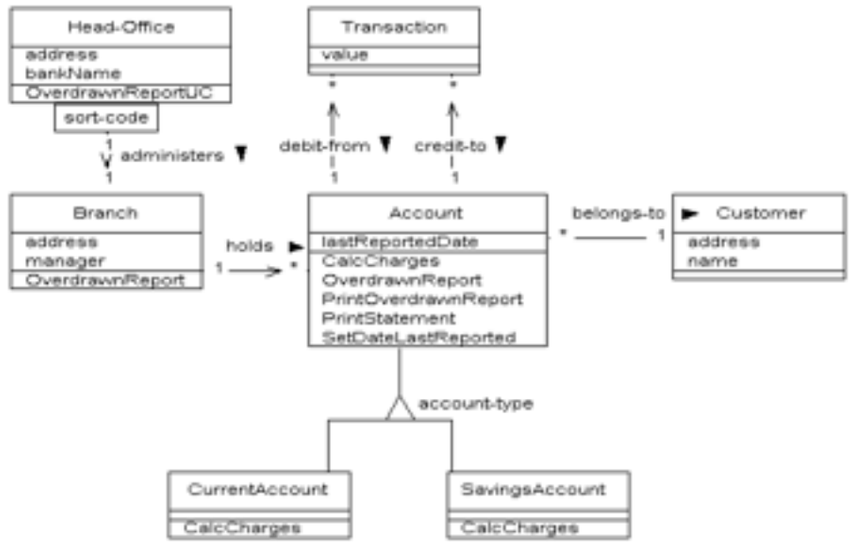
- A model is a simplified depiction of reality
  - a spreadsheet, a Java domain model, a UML model
- Modelling represents concepts to allow some aspect of them to be understood



	A	B	C	D	E	
1						
2						
3	Date	Start time	End time	Pause	Sum	Com
4	2007-05-07	9,25	10,25	0	1	Task
5	2007-05-07	10,75	12,50	0	1,75	Task
6	2007-05-07	18,00	19,00	0	1	Task
7	2007-05-08	9,25	10,25	0	1	Task
8	2007-05-08	14,50	15,50	0	1	Task
9	2007-05-08	8,75	9,25	0	0,5	Task
10	2007-05-14	21,75	22,25	0	0,5	Task
11	2007-05-14	22,50	23,00	0	0,5	Task
12	2007-05-15	11,75	12,75	0	1	Task
13						
14						
15						
16						
17						
18						



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



Why create models?

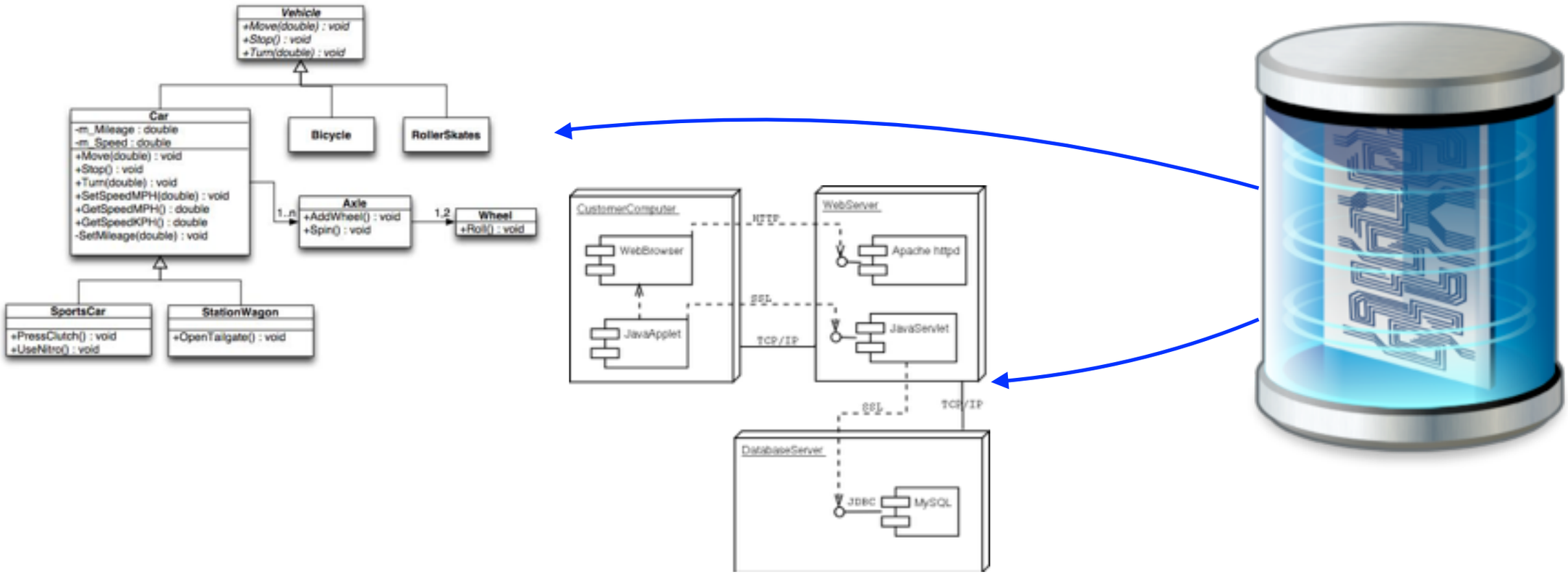
Communicate

Record

Understand

Models vs diagrams

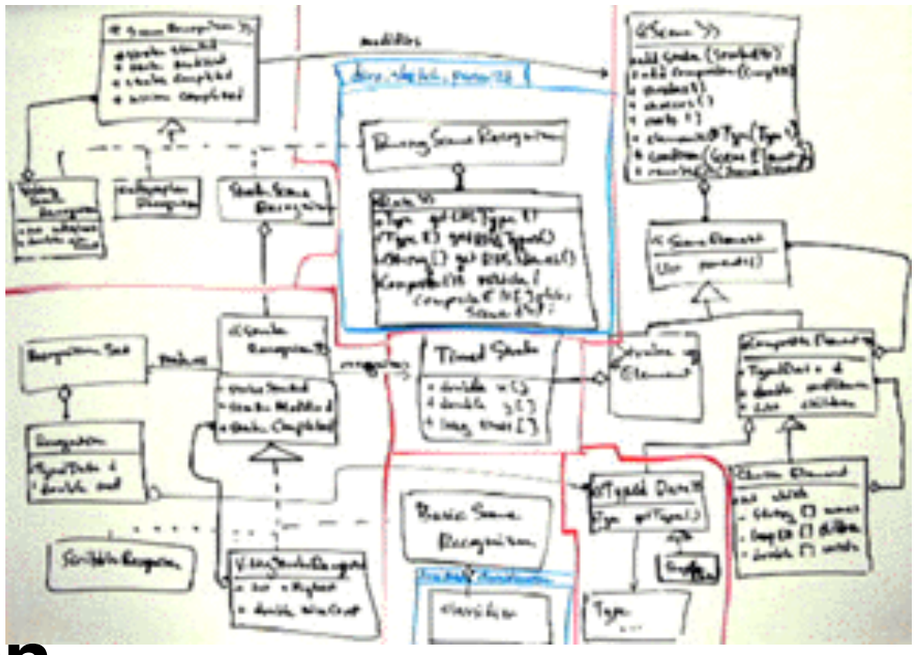
- A **diagram** is a purely **visual** representation
- A **model** has **definitions** (and diagram?)
  - In UML terms diagrams provide views of a model



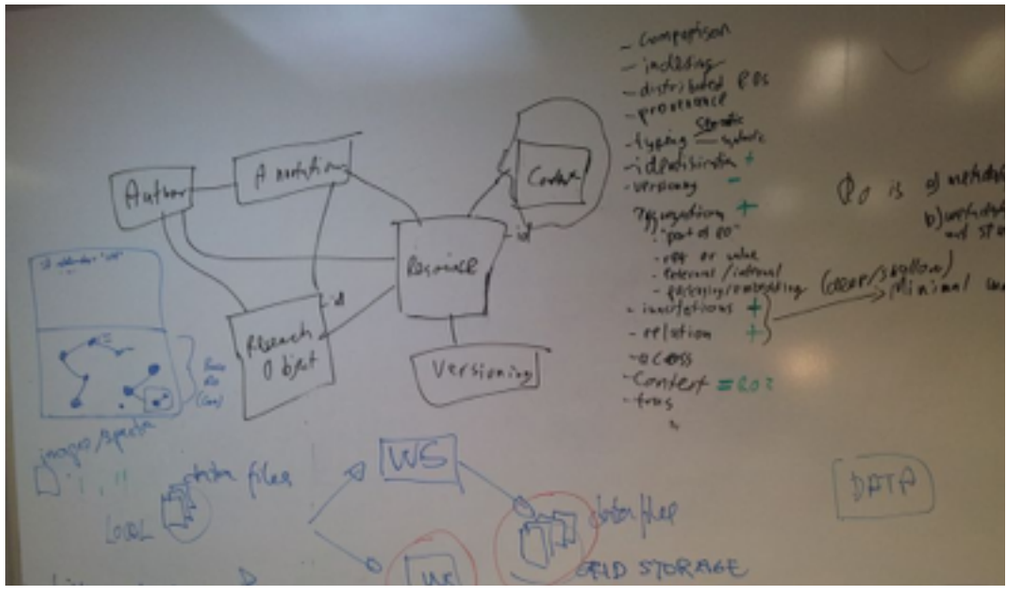
Types of model

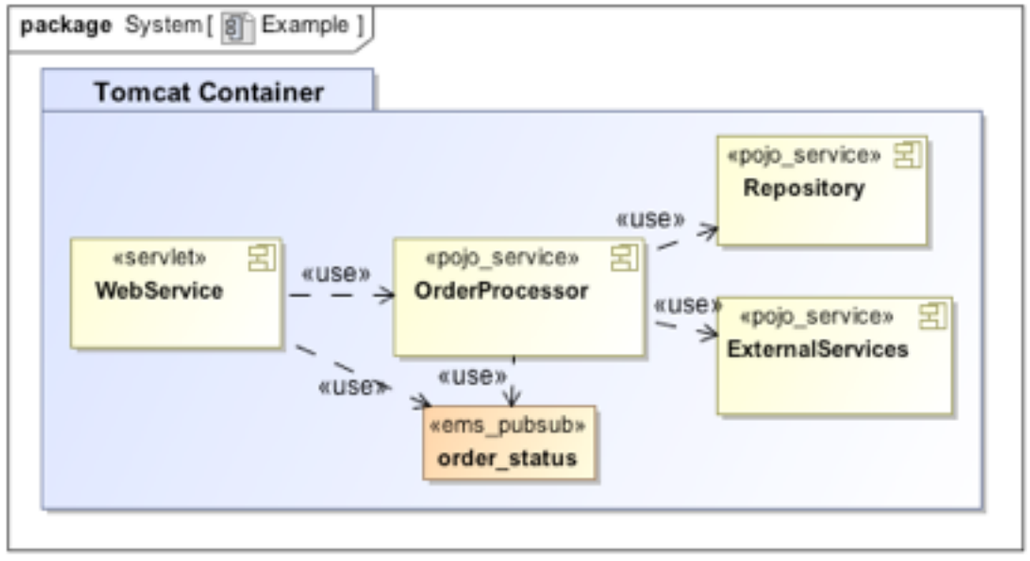
High Detail

Low Precision



High Precision





Low Detail

Uses for models

- **Consistency**
  - change once, its changed everywhere
- **Reporting**
  - “what is connected to the Flange Modulator Service?”
- **Checking** and **Validation**
  - do I have a deployment node for every piece of the system?
- **Sharing** information
  - generate many views of a single model (PPT, wiki, …)

Would you ever use UML?

- **Some** models are worth **preserving**
  - some aren't and are "of the moment"
- Models **capture** things that **code can't**
- **Sketches** are a **start** … but are **limited**
- **Models communicate**, so rules are useful
  - UML is a good *base* to work from

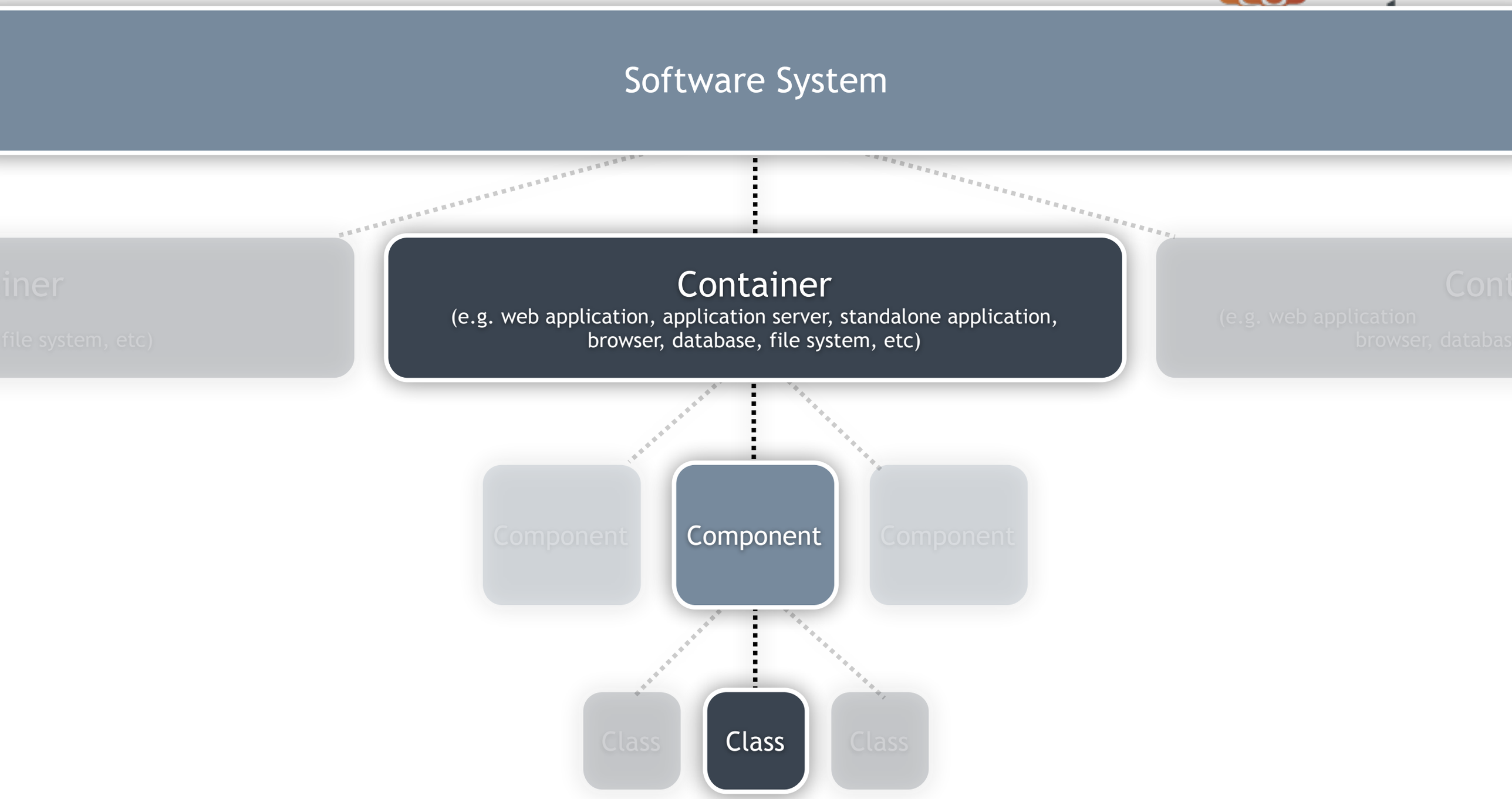
An Analogy

- Would you use **JSON** for 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 as **data**
  - UML is a practical option for machine readable models
  - But so is Excel, or a database … or Structurizr

How to Do It - Two Approaches

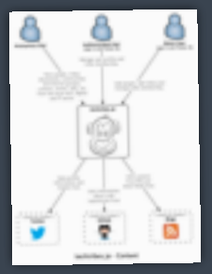
The C4 Approach

- Simon Brown



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

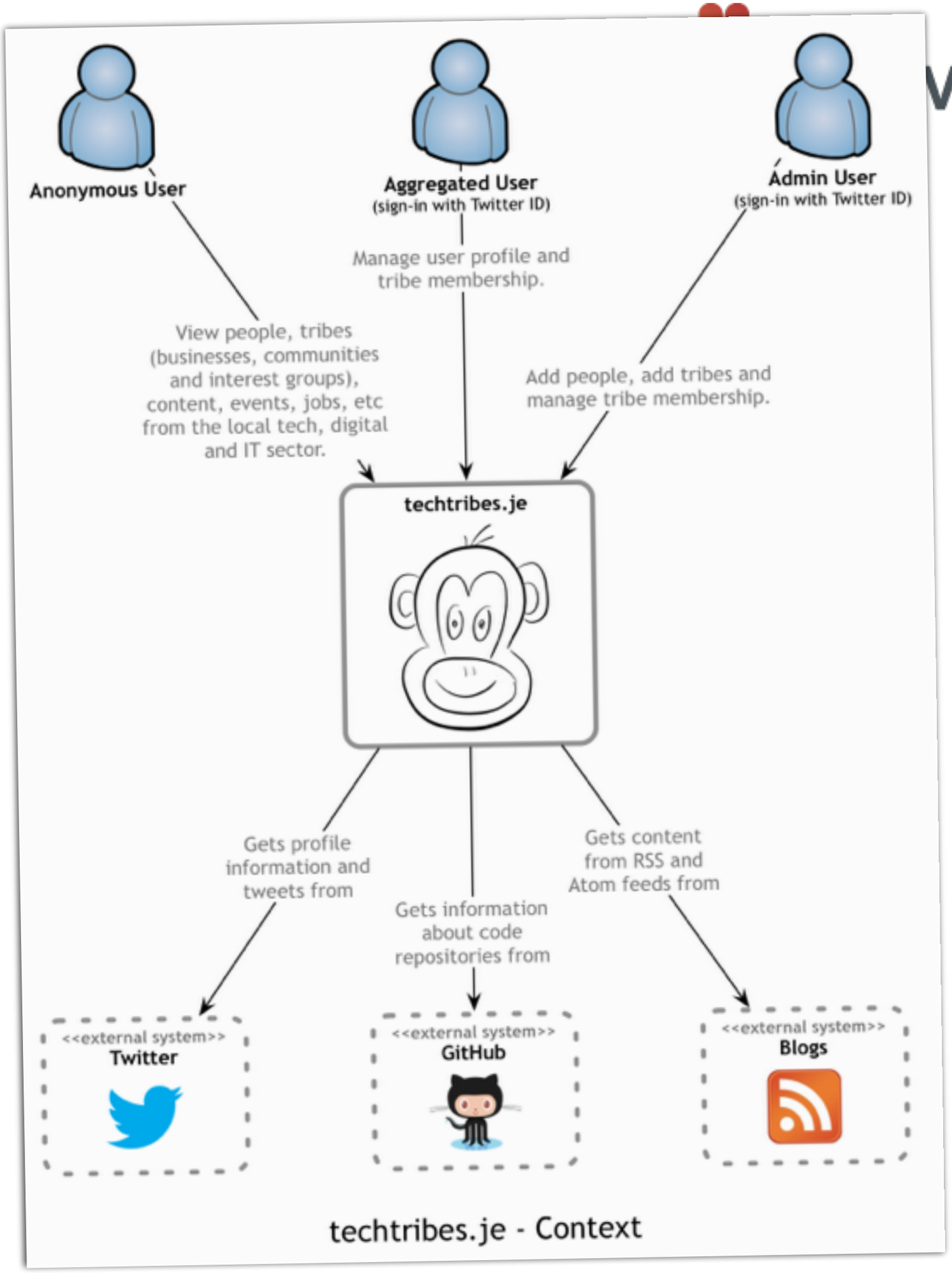
Logical components and their interactions within a container

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)

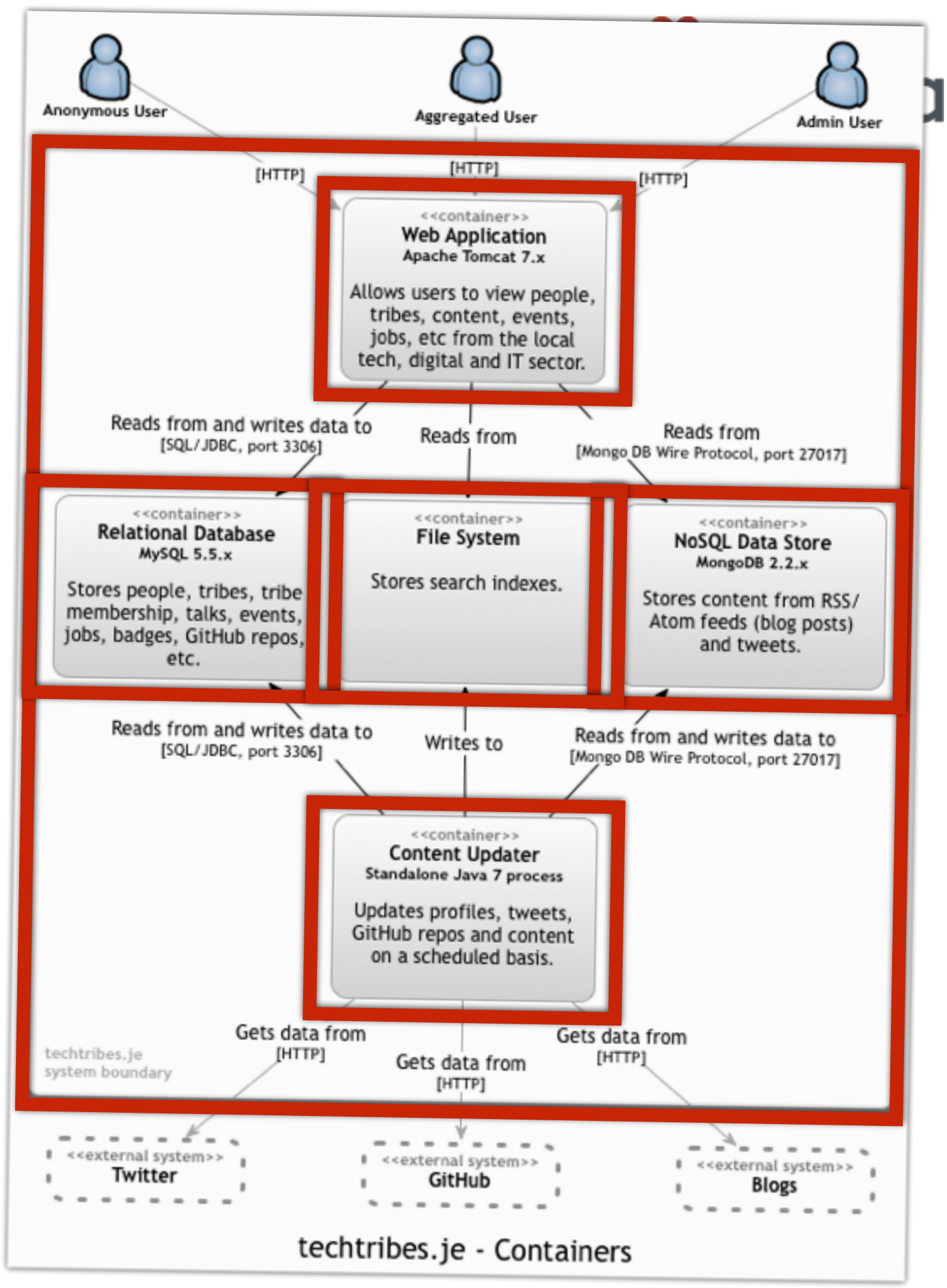


Containers

- What are the high-level technology decisions?
  (including responsibilities)

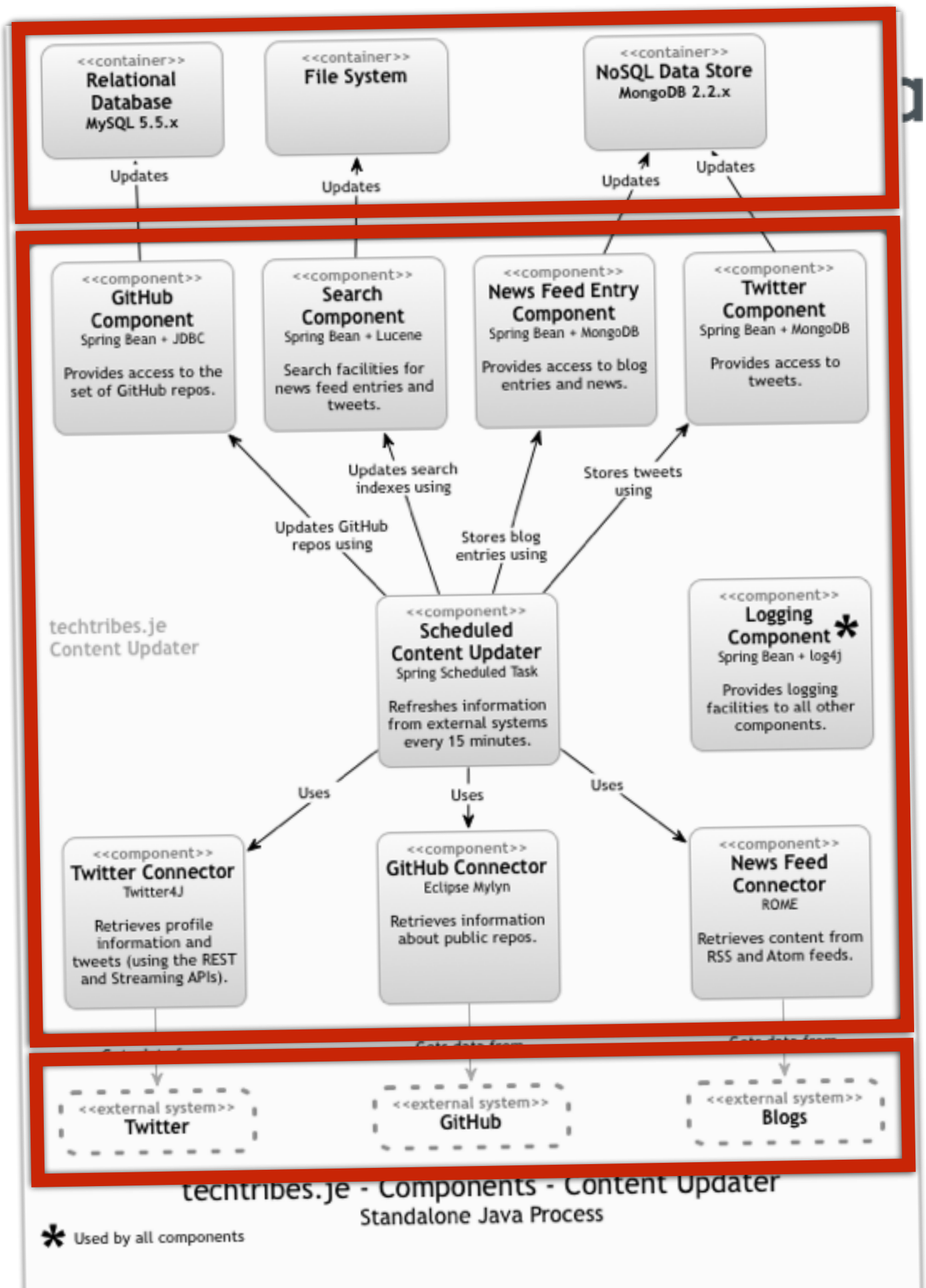
- How do containers communicate with one another?

- As a developer, where do I need to write code?



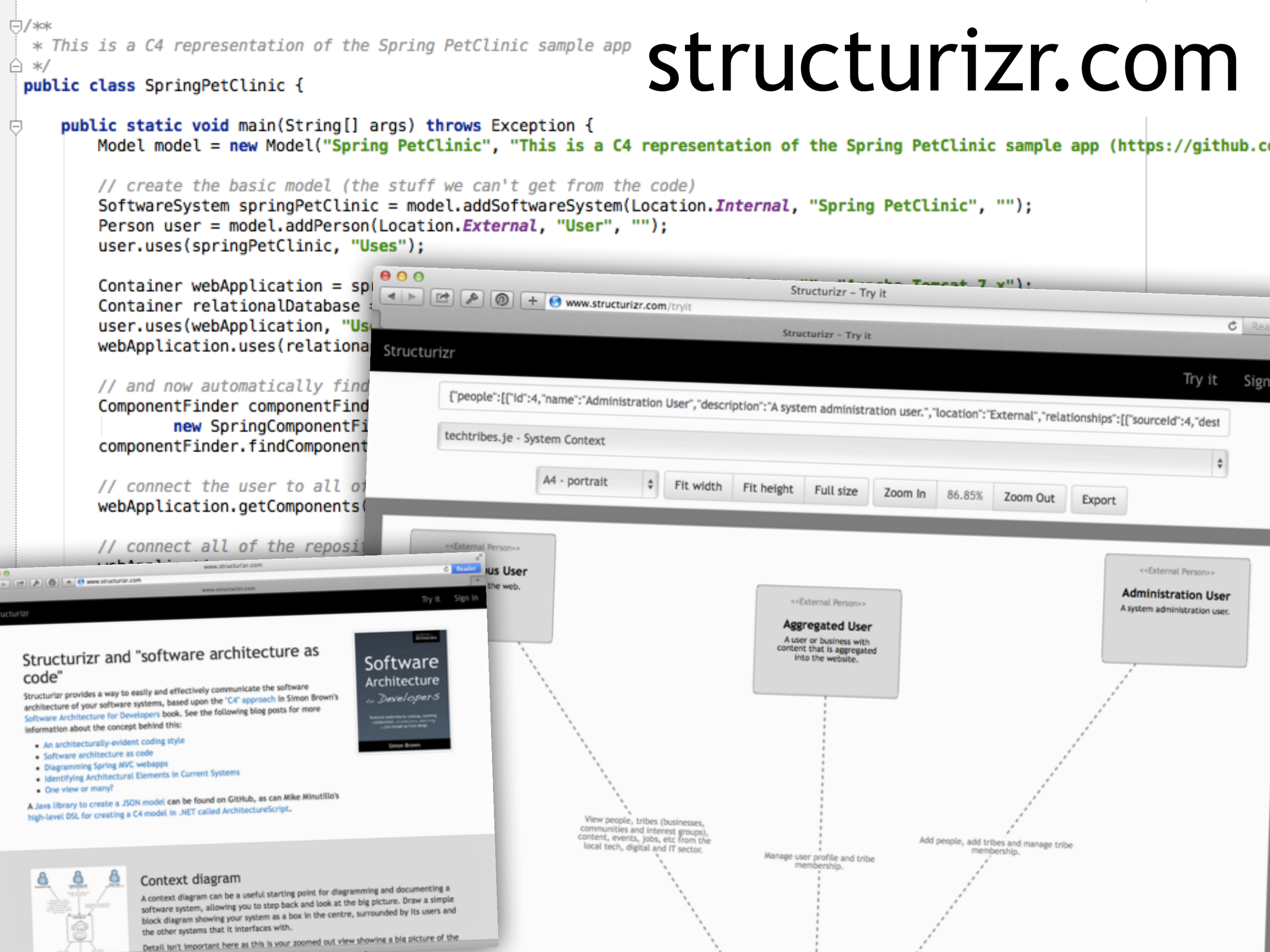
Components

- 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.c

        // 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 = sp
        Container relationalDatabase
        user.uses(webApplication, "Us
        webApplication.uses(relationa

        // and now automatically find
        ComponentFinder componentFind
                new SpringComponentFi
        componentFinder.findComponent

        // connect the user to all of
        webApplication.getComponents(

        // connect all of the reposit
```



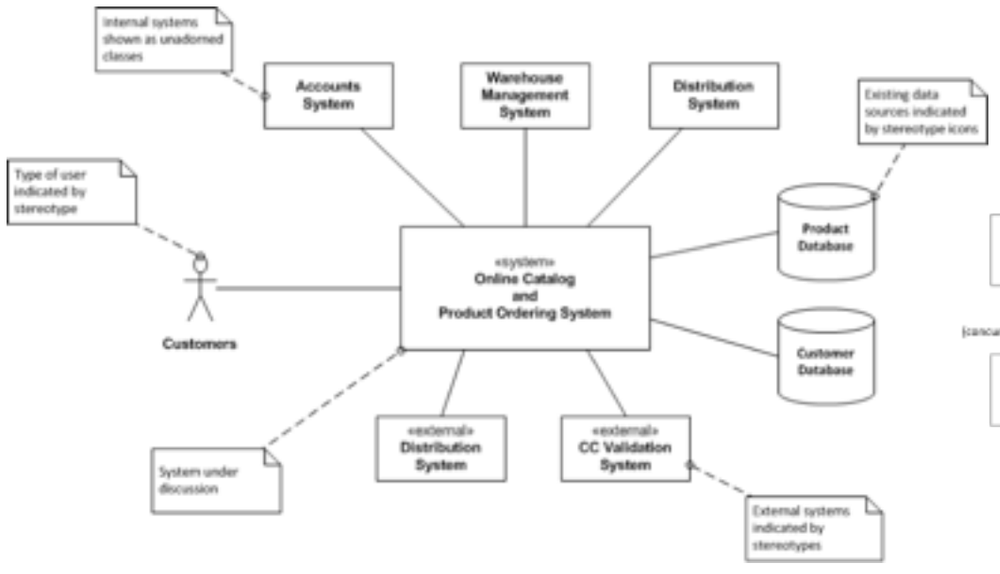
endava

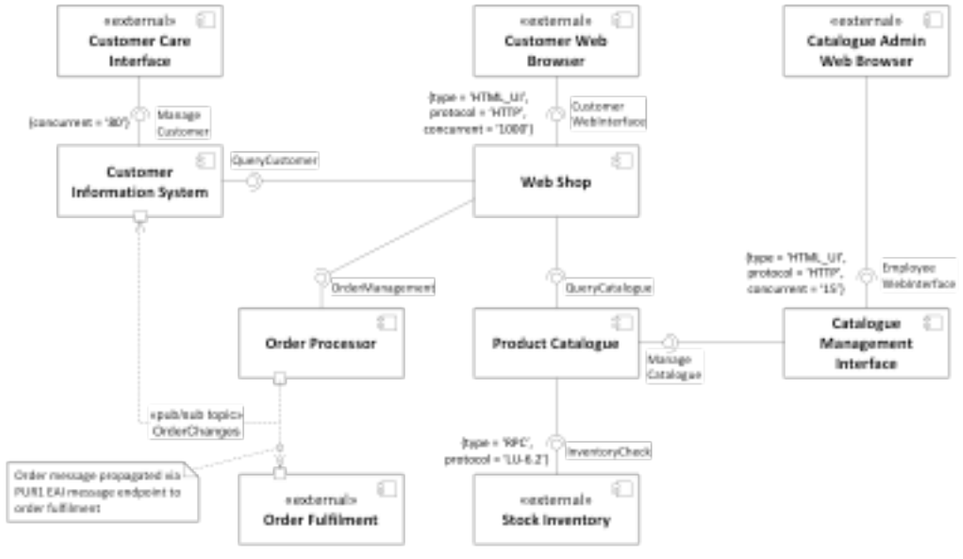
Viewpoints and Perspectives

- Nick Rozanski & Eoin Woods

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)

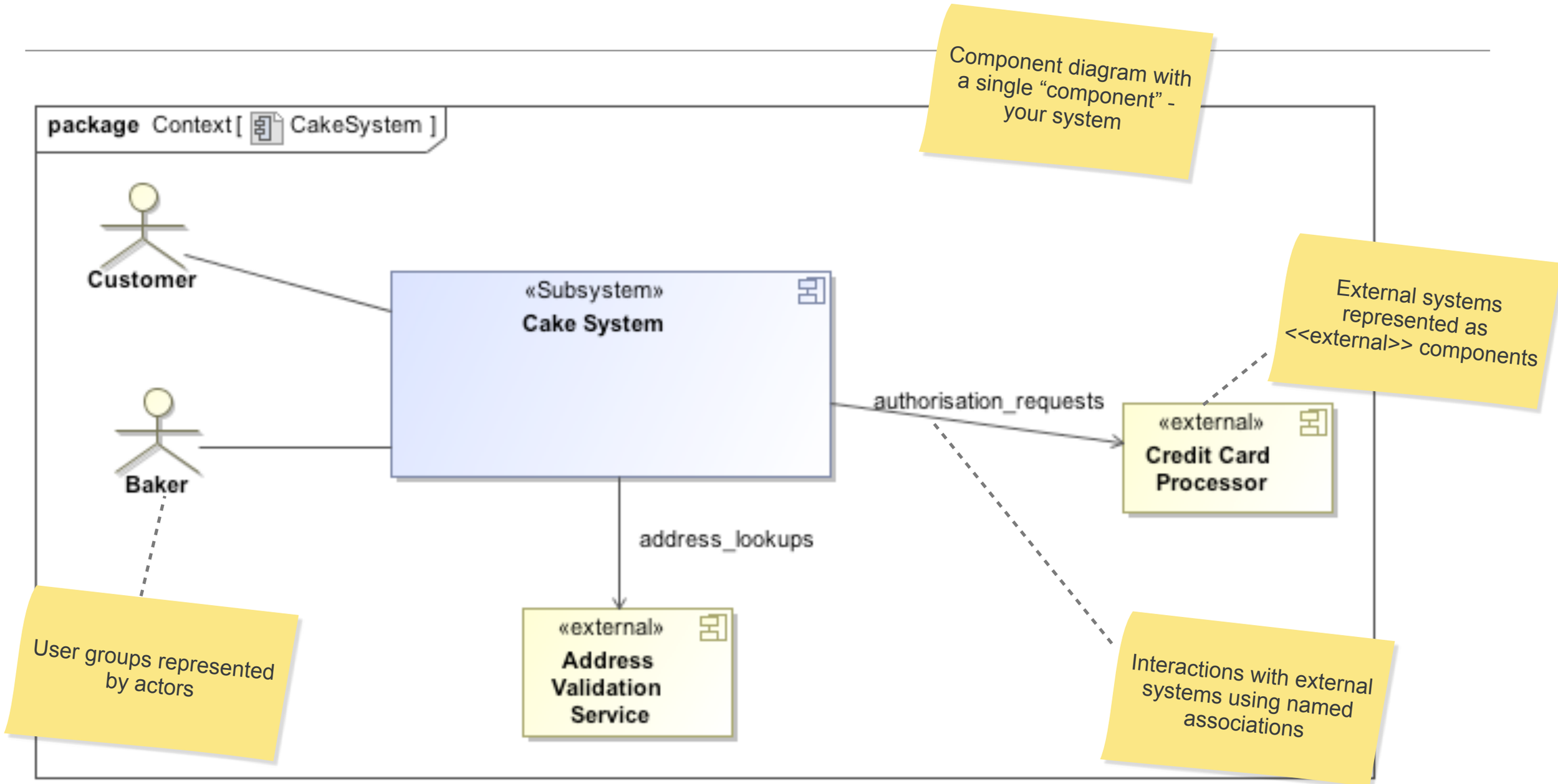
Concurrency View
(processes and threads)

Development View
(code structures)

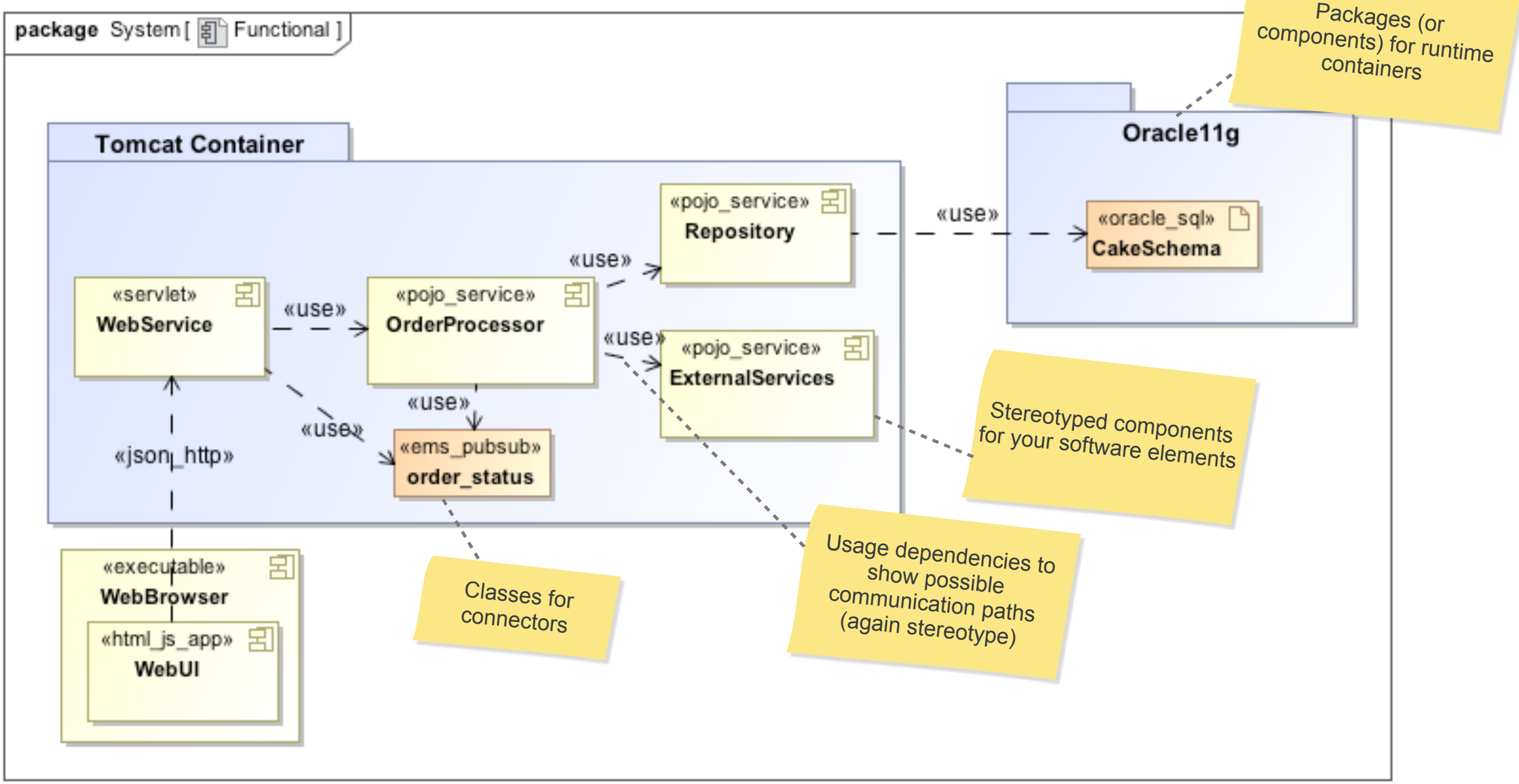
Deployment View
(system meets infra)

Operational View
(keeping it running)

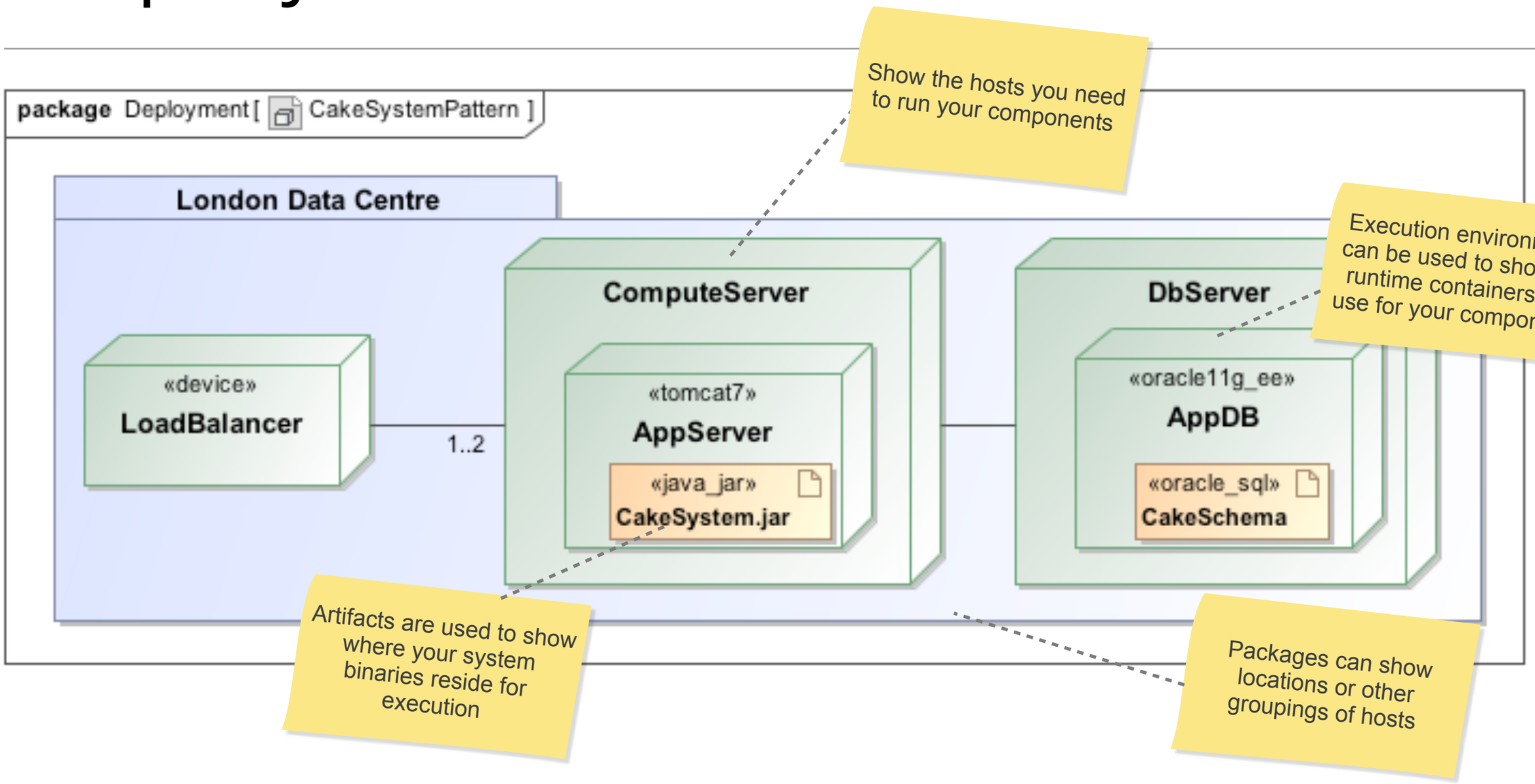
Context View



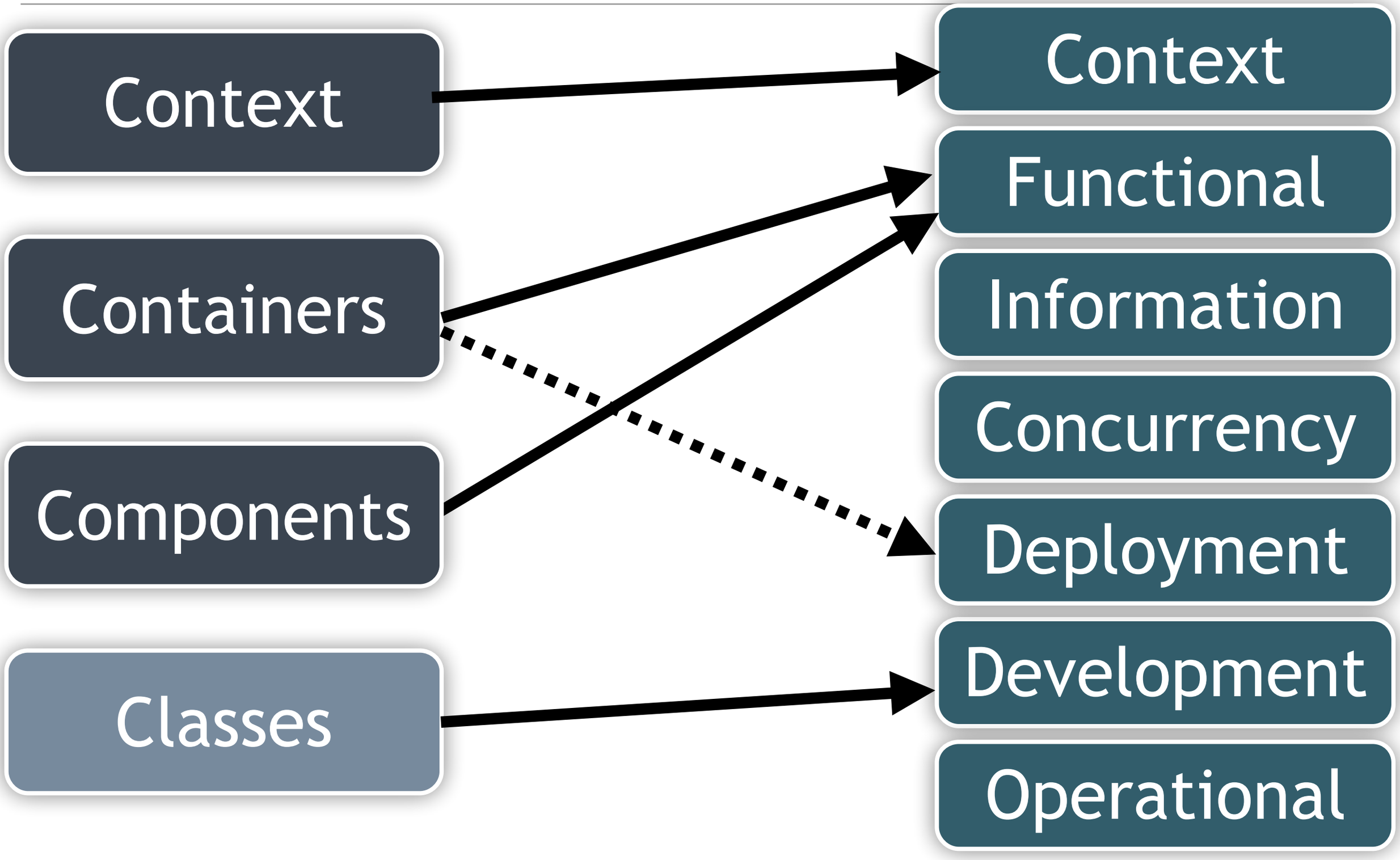
Functional View



Deployment View



C4 and V&P compared



Some Questions and Answers

Q1. Modelling - Why Bother?

- A **model** allows you to see the **big picture**
  - what you have and what you need
- A model aids **communication**
  - inside and outside of the team
  - ubiquitous language with which to describe software.
- You build **private** mental models anyway
- **Code** is a **model**, we often ignore this

Q2. Modelling and Agility

- Good **communication** helps you move **fast**.
- A **model** provides **long-lived** documentation.
- A **model** is a basis for **structure**, **vision** & **risks**

- **No** fundamental **conflict** with agility
  - "*model with a purpose*" (Daniels)
  - working software *over* comprehensive documentation
- **Agility** should be for the **long haul**, not this sprint
  - do you know all the feed dependencies from your system?

Q3. How to Do It?

- **Start small**, start with a definite **purpose**
  - start with the big picture, and work into the detail.
  - whiteboard, napkin or A4 sheet
- **Stop** when you reach "**sufficient**" detail
- Skip Visio … get a tool, get a model
- Include technology choices!

Q4. UML - Is It Worth the Hassle?

- "**No**" (Simon)
- **Maybe** … depends what you need (Eoin)
- Is your model **useful**? Will it be so in the **future**?
  - do you care enough to keep it current?
- If you have **long lived models** and want to use the **data** then yes, *highly* ***tailored*** UML is worth the effort

"*UML is a pretty poor architectural description language but probably the best one we have*"

Q5. Modelling in the Large vs the Small

- **Sketches** will quickly become **out of date**
  - but reverse-engineering leads to cluttered diagrams
- Many **small diagrams** better than uber-diagrams
- A **large** system means you need to use a **computer** to understand it
- However large your model, the **code** is still “the truth”
- Modelling languages can **scale** like programming languages

Summary and Conclusions

What We Have Talked About

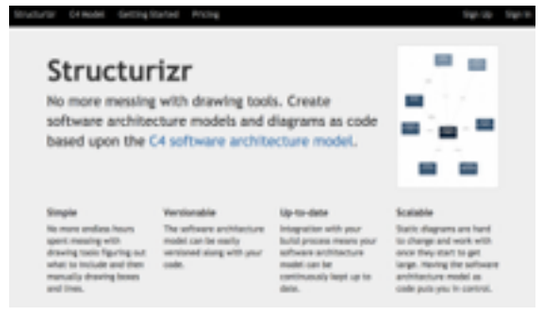
- **Modelling** is terrifically **useful**
  - communication
  - clarity
  - analysis
- **Many ways** of doing it
  - napkins to UML tools
- The key is to get **value** from what you do
  - don’t get stuck in “analysis paralysis”

Find Out More

C4 codingthearchitecture.com

V&P viewpoints-and-perspectives.info

Structurizr structurizr.com



Questions?

Eoin Woods

www.endava.com
www.eoinwoods.info
@eoinwoodz