



Únete a las mujeres que desarrollan software grandioso

Visualizing Software Architecture

Perla Velasco-Elizondo

November 2021

What is software architecture?



Software Architecture Defined

*“The software architecture of a system is the **set of structures** needed to **reason about the system**, which comprise **software elements**, **relations among them**, and **properties of both**.”*

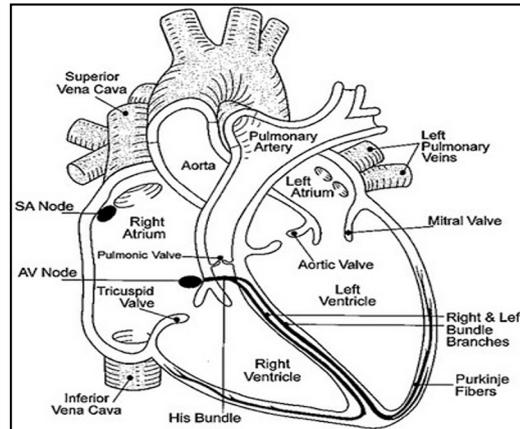
Len Bass, Paul Clements, and Rick Kazman.
Software Architecture in Practice (3rd ed.). Addison-Wesley Professional, 2012.

Structures ...

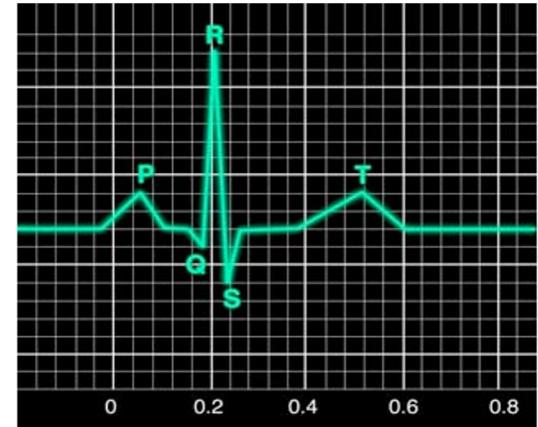
Let's revise this analogy ...



Human body
comprised of multiple
real structures

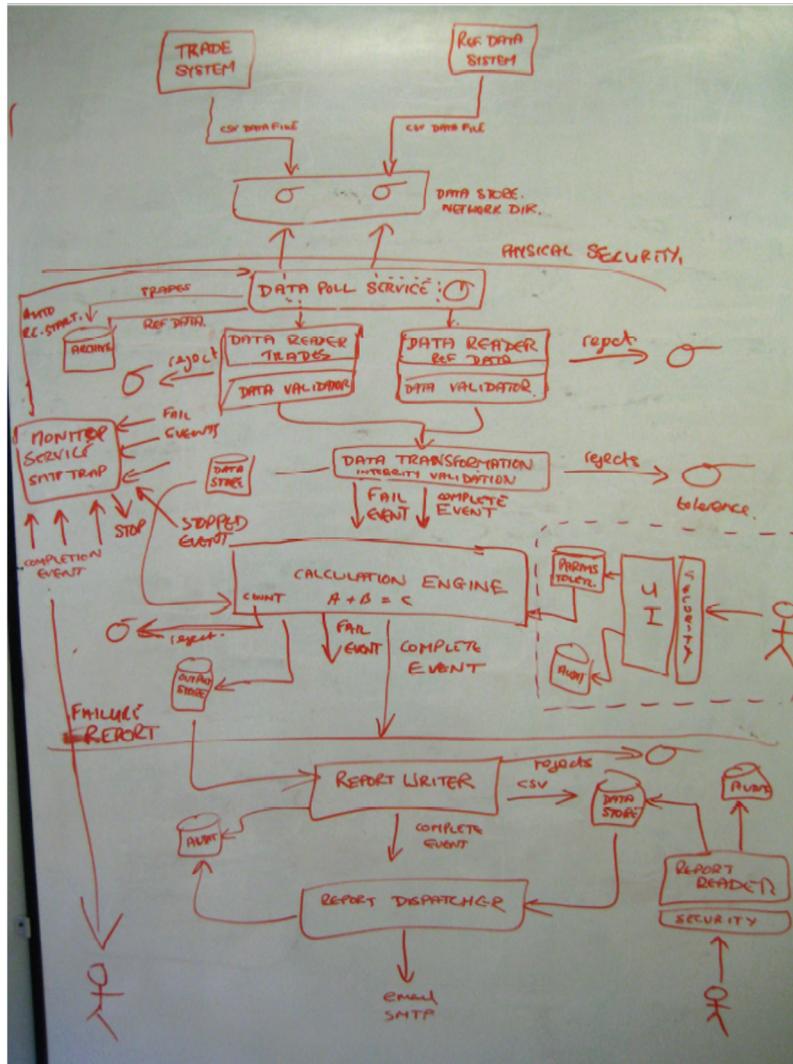


A **static view** of one
human structure



A **dynamic view** of
the same structure

It's usually difficult to show **all structures**, from **all perspectives**, on a single representation



Architectural View

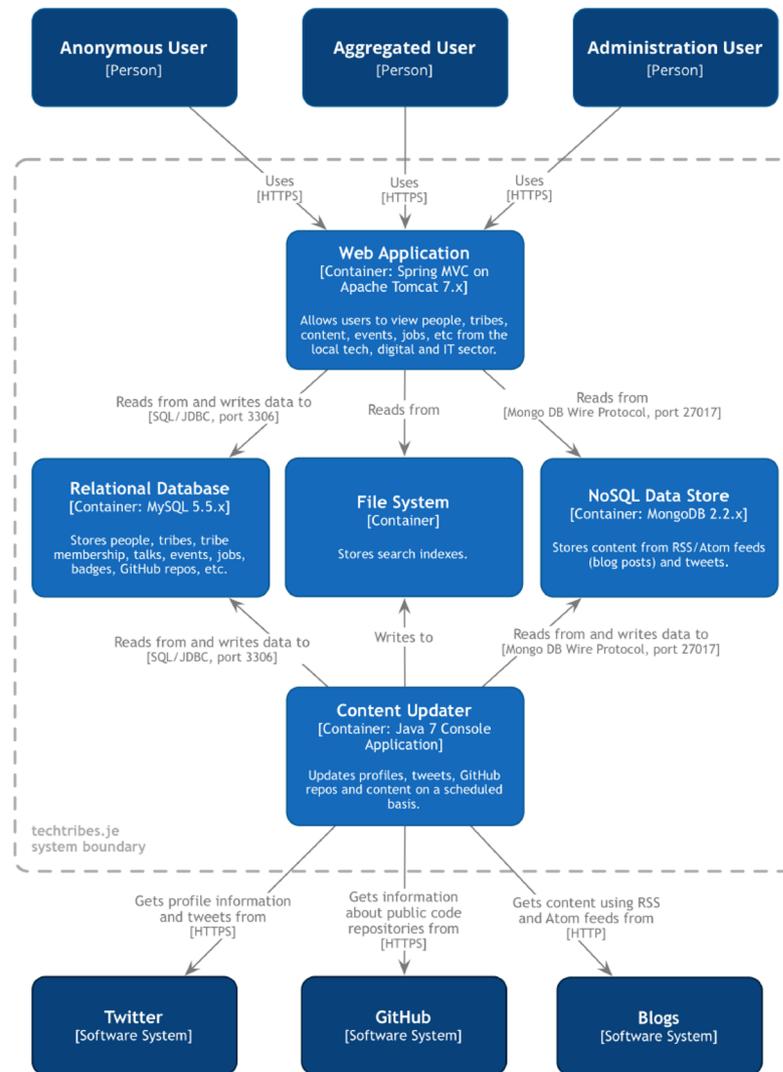
- A view is a **diagram**.
- A view is a **abstract** representation of **one** or **more structures** of an architecture that illustrates how the architecture addresses **one** or **more architectural requirements**.

Some well-known views are:

1. Static
2. Dynamic
3. Physical

Static View

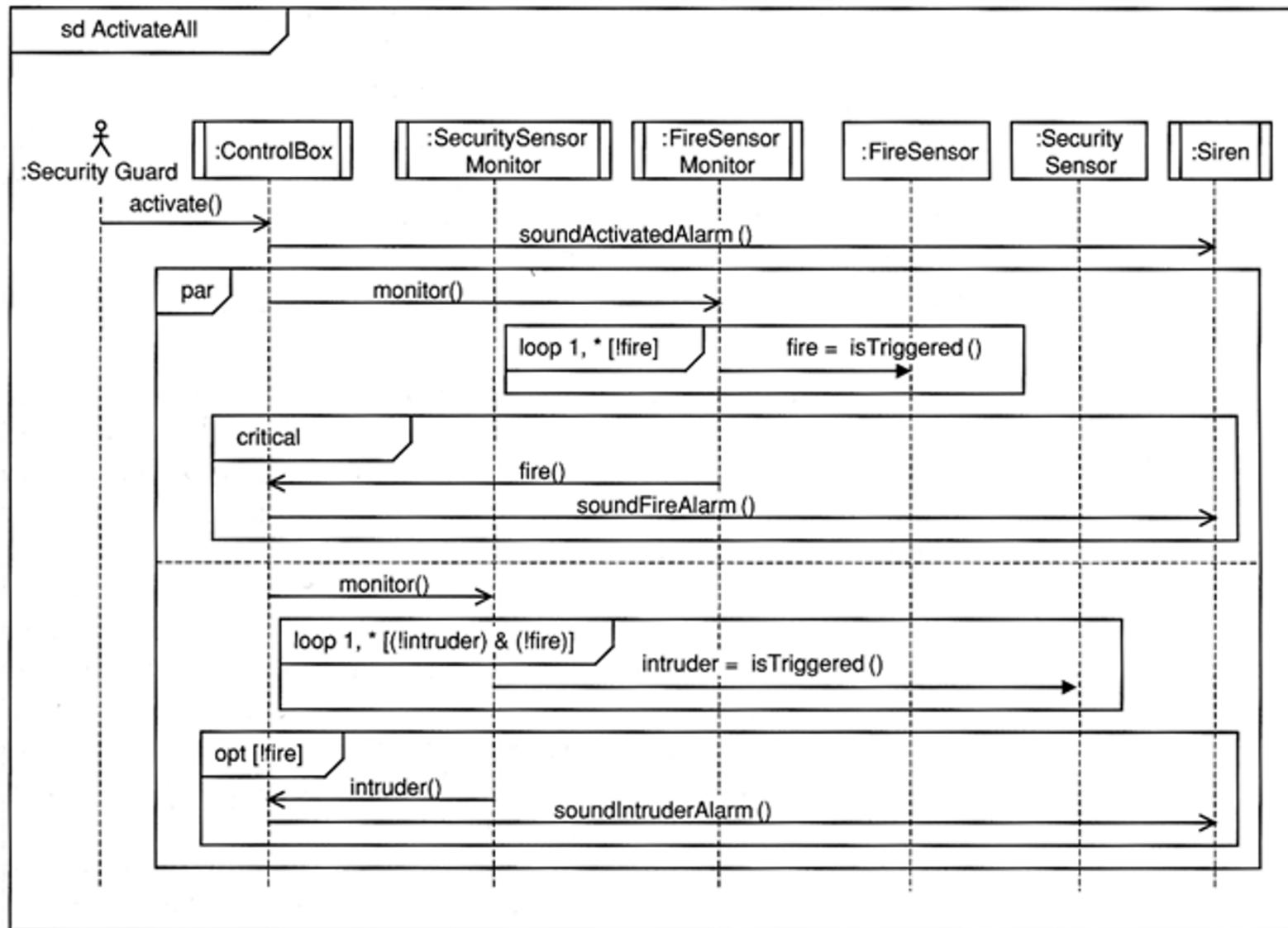
- A.k.a. Module, Logic View.
- Shows structures with elements and relationships that exist in development and correspond to **implementation units**.
- Shows the system **partitioning** and **assignment of responsibilities**.



Source: Simon Brown. 2016.
 Software Architecture for
 Developers - Volume 2.
 Leanpub.

Dynamic View

- A.k.a. Components & Connectors View
- Shows structures with elements and relationships that exist in **execution time**.



Physical View

- A.k.a Allocation View
- Shows structures with elements and relationships that exist in **deployment time**.



Deployment diagram for Internet Banking System - Live

An example live deployment scenario for the Internet Banking System.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)

Abstractions

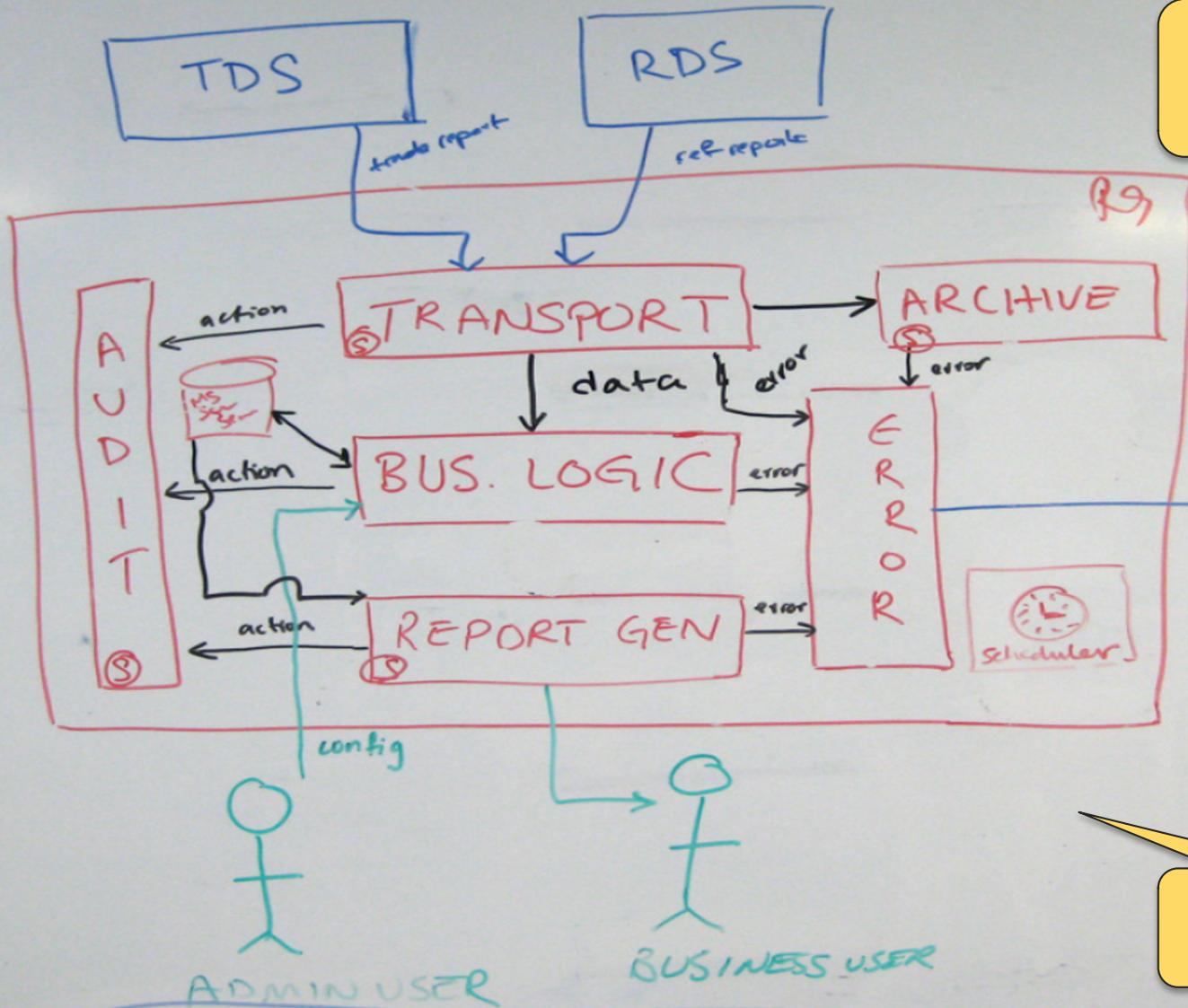
- Abstraction is about **reducing detail**; important characteristics are made more visible by leaving details aside.
- Abstractions help us **reason** about a big and/or complex software system

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

Simon Brown

Notations

- There are different notations for documenting architectural views
- Each notation has its benefits and shortcomings
- **However**, when using a notation for documenting a view, **you must remember** some important aspects ...



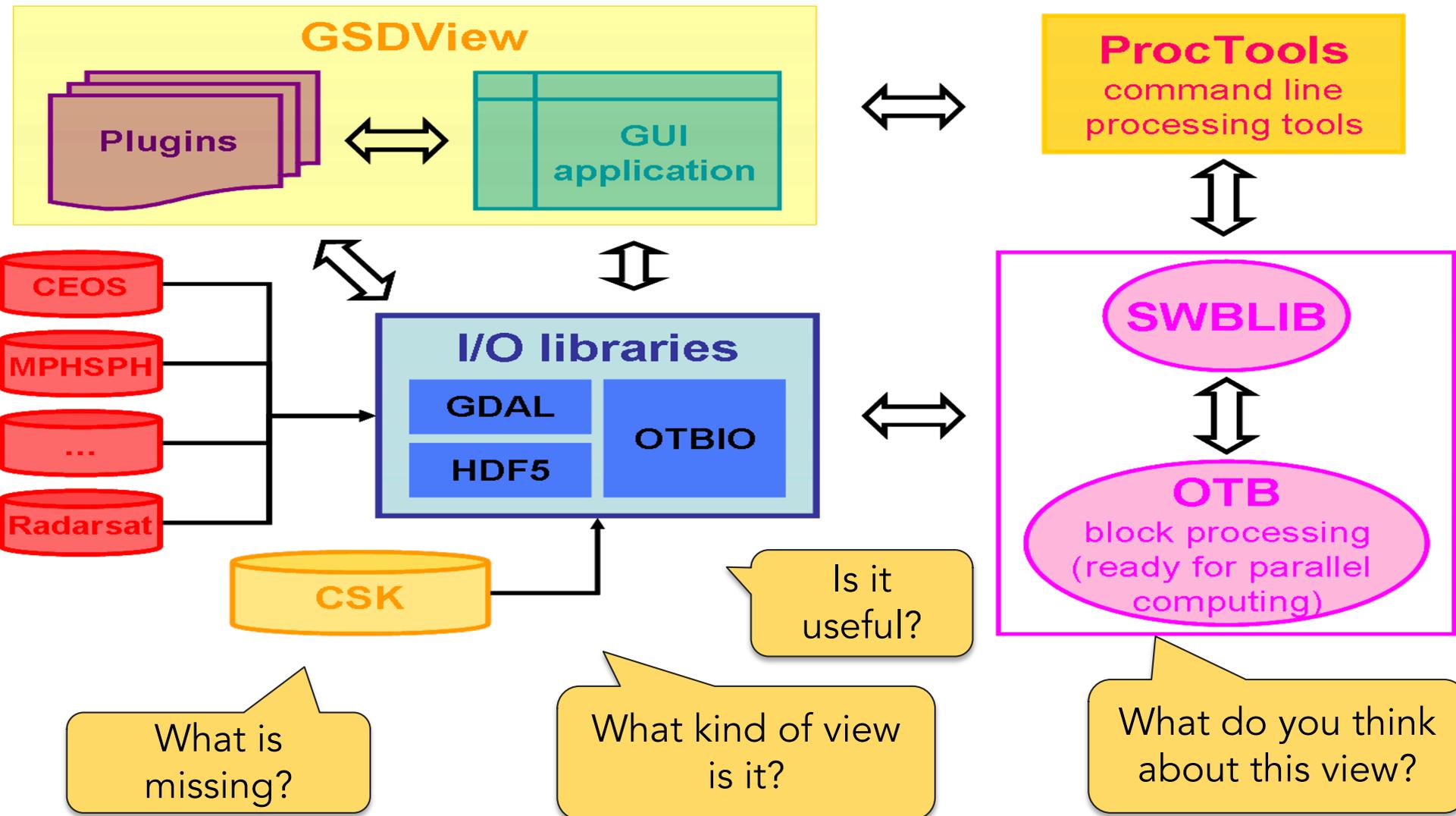
What do you think about this view?

What kind of view is it?

MONITOR

Is it useful?

What is missing?



Software Architecture Defined

*“The software architecture of a system is the **set of structures** needed to **reason about the system**, which comprise **software elements**, **relations among them**, and **properties of both**.”*

Len Bass, Paul Clements, and Rick Kazman.
Software Architecture in Practice (3rd ed.). Addison-Wesley Professional, 2012.

Aspects to remember

General

- The view has a title
- It is clear the type of the view
- It is clear the scope of the view
- The view has a key/legend

Aspects to remember

Elements

- Every element has a name
- It is clear the type of every element? (e.g. software system, component, etc)
- It is clear what the element does
- Where applicable, it is clear the technology choices associated with every element
- It is clear the meaning of all acronyms and abbreviations
- It is clear the meaning of all shapes/icons
- It is clear the meaning of all border styles (e.g. solid, dashed, etc)
- It is clear the meaning of all element sizes (e.g. small vs large boxes)

Aspects to remember

Relationships

- Every line has a label describing the intent of that relationship
- Where applicable, it is clear the technology choices associated with every relationship? (e.g. protocols for communication)
- It is clear the meaning of all acronyms and abbreviations used
- It is clear the meaning of all colours used
- It is clear the meaning of all arrowheads used
- It is clear the meaning of all line styles (e.g. solid, dashed, etc)

Aspects to remember

Other

- ❑ Each view fits on **one page**
- ❑ **Do not mix** runtime and static elements in the same diagram
- ❑ There is **more than one view** to describe the whole system
- ❑ Keep structural and semantic **consistency** across diagrams

Notations

There are three main notations:

1. Informal
2. Semi-formal
3. Formal

The C4 model for visualising software architecture

Context, Containers, Components, and Code

🕒 In a hurry? Read the [Wikipedia page](#) and the 5 minute introduction to the C4 model at InfoQ

[The C4 model for software architecture](#)

[O modelo C4 de documentação para Arquitetura de Software](#)

[用于软件架构的C4模型](#)

[ソフトウェアアーキテクチャのためのC4モデル](#)

👍 Uses and benefits

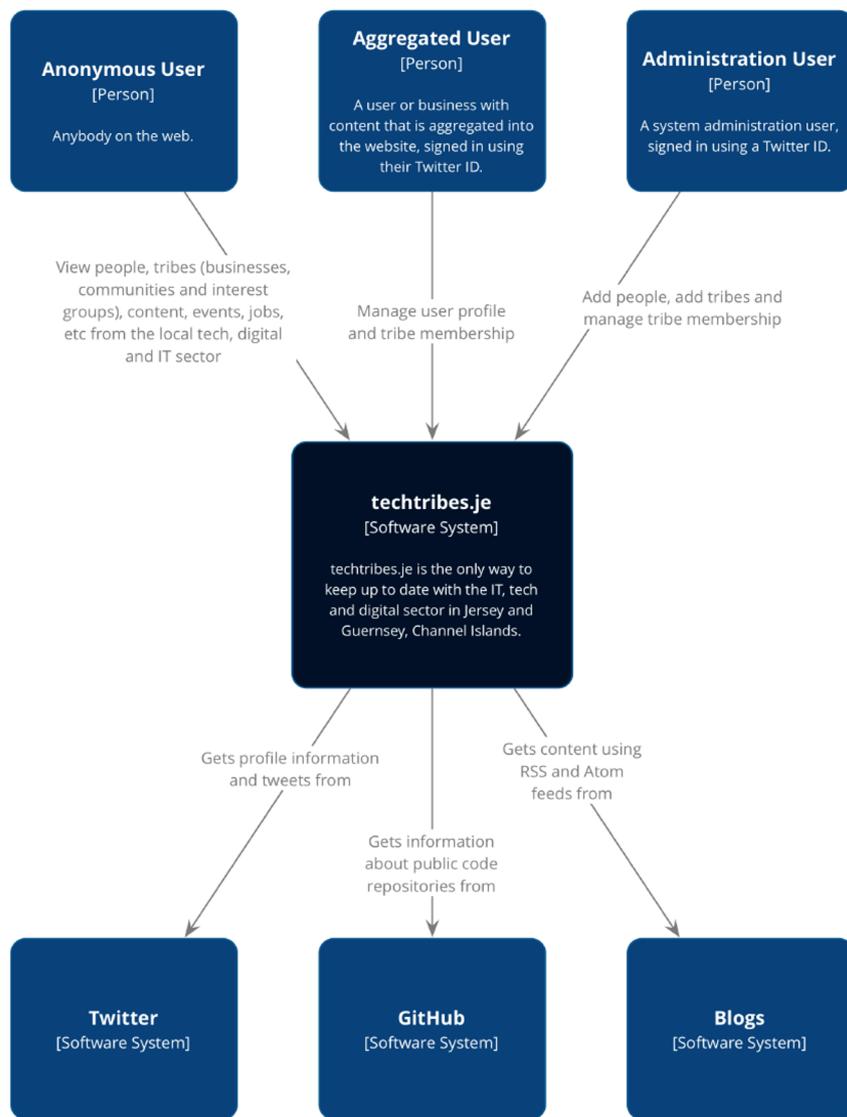
The C4 model is an easy to learn, developer friendly approach to software architecture diagramming. Good software architecture diagrams assist with communication inside/outside of software development/product teams, efficient onboarding of new staff, architecture reviews/evaluations, risk identification (e.g. [risk-storming](#)), threat modelling (e.g. [STRIDE/LINDDUN](#)), etc.

🕒 Or watch "Visualising software architecture with the C4 model"



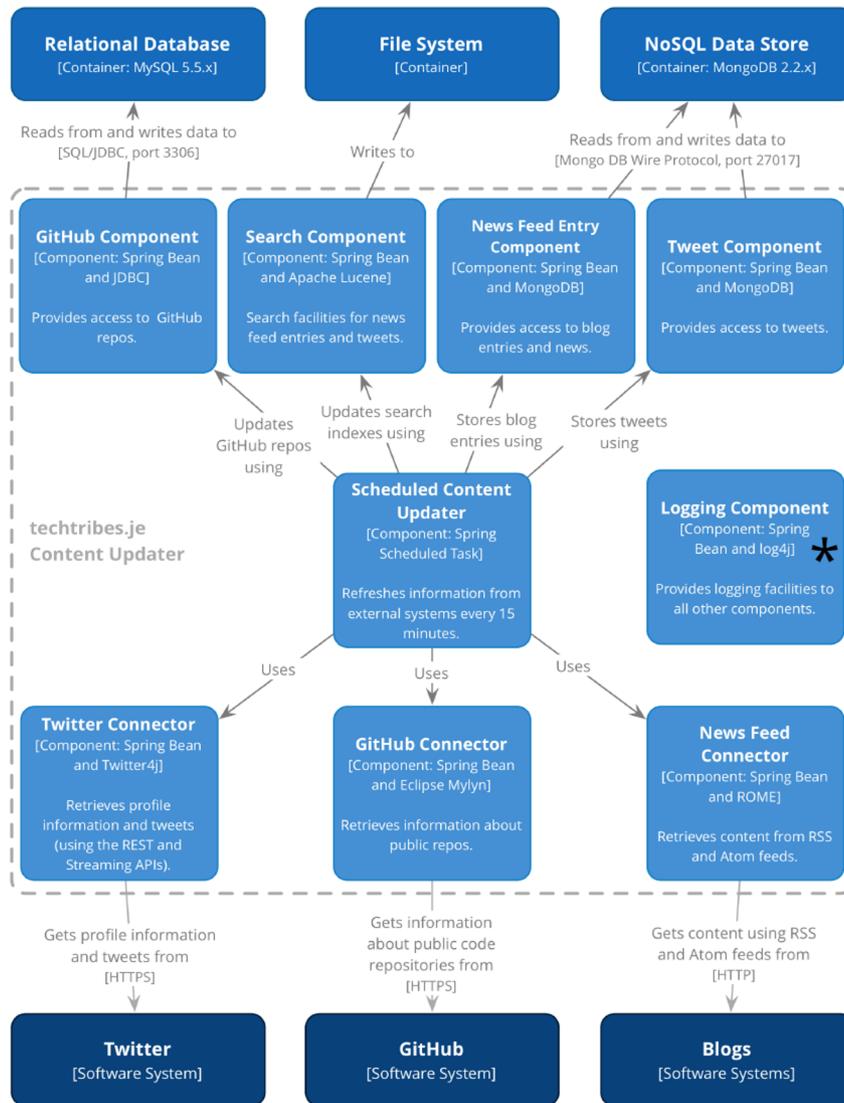
C4 Model

- **Context**: A high-level diagram that sets the scene; including key system dependencies and actors.
- **Container**: A container diagram shows the high-level technology choices, how responsibilities are distributed across them and how the containers communicate.
- **Component**: For each container, a component diagram lets you see the key logical components and their relationships.
- **Classes**: This is an optional level of detail and I will draw a small number of high-level UML class diagrams if I want to explain how a particular pattern or component will be (or has been) implemented.



Context

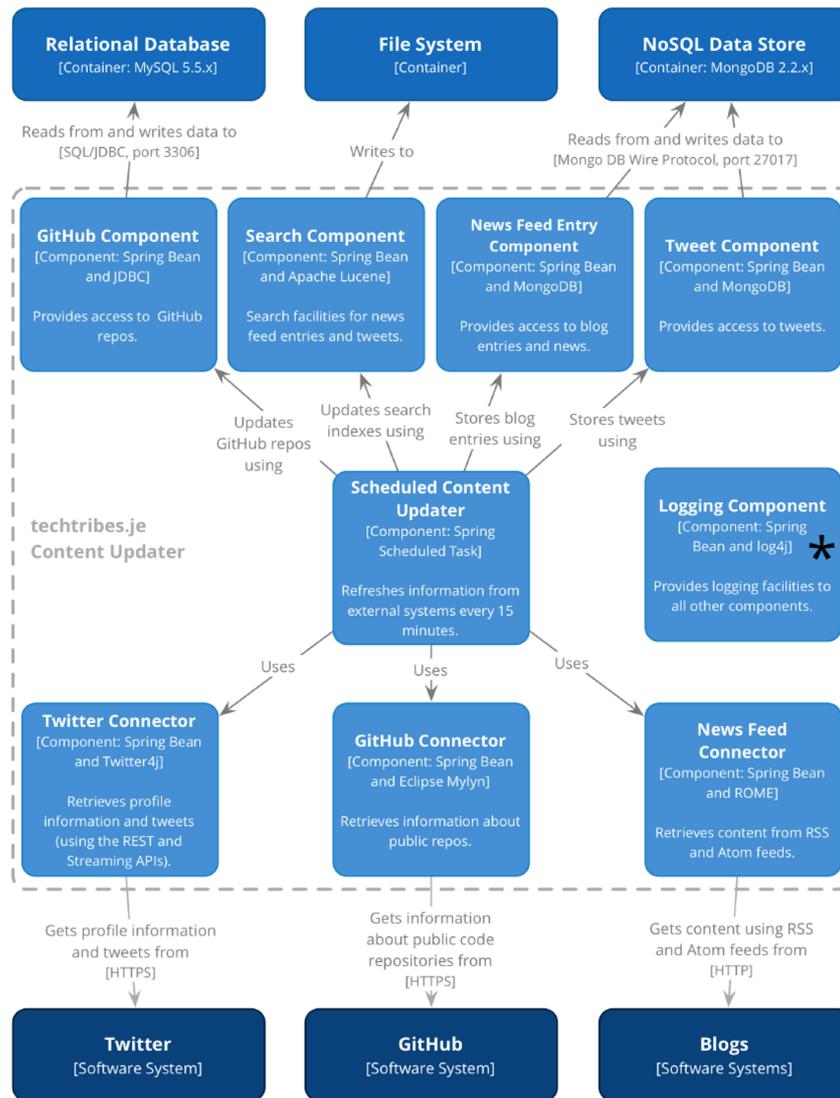
Source: Simon Brown. 2016.
Software Architecture for
Developers - Volume 2. Leanpub.



Containers

Source: Simon Brown. 2016.
Software Architecture for
Developers - Volume 2. Leanpub.

Components



Source: Simon Brown. 2016.
Software Architecture for
Developers - Volume 2. Leanpub.

{name}

[Person]

{description}

{name}

[Software System]

{description}

{name}

[Container: {technology}]

{description}

{name}

[Component: {technology}]

{description}

Tooling

For design sessions, you might find a whiteboard or flip chart paper better for collaboration, and iterating quickly. For long-lived documentation, the following modelling and diagramming tools can help create software architecture diagrams based upon the C4 model.

Modelling tools (recommended)

Structurizr

Structurizr is a collection of tooling to create software architecture diagrams and documentation based upon the C4 model. Structurizr was started in 2014 by Simon Brown (creator of the C4 model), and has grown into a community of tooling, much of which is open source.

Structurizr is unique in that it supports diagrams as code (Java, Clojure, .NET, TypeScript, PHP, Python, Go) or text (DSL or YAML) via a number of different authoring methods, with it being possible to render diagrams using a number of different tools (Structurizr cloud service/on-premises installation, PlantUML, Mermaid, WebSequenceDiagrams, etc).

➤ Text-based GUI-based

☁ Cloud 🏠 On-premises

💰 Free and paid

Archi

Archi provides a way for you to create C4 model diagrams with ArchiMate. See C4 Model, Architecture Viewpoint and Archi 4.7 for more details...

GUI-based

🏠 On-premises

💰 Free

Sparx Enterprise Architect

LieberLieber Software has built an extension for the C4 model, based upon the MDG Technology built into Sparx Enterprise Architect.

GUI-based

🏠 On-premises

💰 Paid

IcePanel

IcePanel is a structured diagramming tool that supports the C4 model.

GUI-based

☁ Cloud

💰 Free and paid

Gaphor

Gaphor has built-in support for the C4 model.

GUI-based

🏠 On-premises

💰 Free

MooD

MooD has support for the C4 model via a set of blueprints.

GUI-based

☁ Cloud 🏠 On-premises

💰 Paid

Astah

Astah has support for the C4 model via a C4 model plugin.

GUI-based

🏠 On-premises

💰 Paid

Diagramming tools

PlantUML

There are a number of extensions for PlantUML to assist in the creation of C4 model diagrams:

C4-PlantUML by Ricardo Niepel
C4-PlantumlSkin by Savvas Kleanthous
c4builder by Victor Lupu
plantuml-libs by Thibault Morin

You can also create C4-PlantUML diagrams using C# code via the C4Sharp library.

➤ Text-based

☁ Cloud 🏠 On-premises

💰 Free

diagrams.net

diagrams.net includes support for the C4 model, and there are also a number of plugins that allow you to create diagrams using pre-built shapes:

c4-draw.io by Chris Kaminski
c4-draw.io by Tobias Hochgürtel
EasyC4 by Maciek Sliwinski

GUI-based

☁ Cloud 🏠 On-premises

💰 Free

OmniGraffle

Dennis Laumen has created a C4 model stencil for OmniGraffle, that allows you to create diagrams using pre-built shapes.

GUI-based

🏠 On-premises

💰 Paid

Microsoft Visio

"pihavel" has created a C4 model template for Microsoft Visio, that allows you to create diagrams using pre-built shapes.

GUI-based

☁ Cloud 🏠 On-premises

💰 Paid

Visual Paradigm

Visual Paradigm supports the C4 model via some pre-built shapes.

GUI-based

🏠 On-premises

💰 Paid

yEd

Ferhat Kalinci has created some C4 model shapes for yEd.

GUI-based

🏠 On-premises

💰 Free

“

Communication
works
for those
who work at it ”

Speakers Dev Day 4 Women



Perla Velasco Elizondo

Senior Product Manager @ Kueski |
Profesora @ Universidad Autónoma
de Zacatecas

Visualización de la arquitectura del software