ADCenter

Agile Delivery Center

Capgemini

A brief introduction to

Software Design and Abstractions

What *is* Software Architecture

https://www.youtube.com/watch?v=DngAZyWMGR0

Ralph Johnson – as paraphrased by Martin Fowler:

"….Expert developers shared understanding of the system design

…

very much a social activity

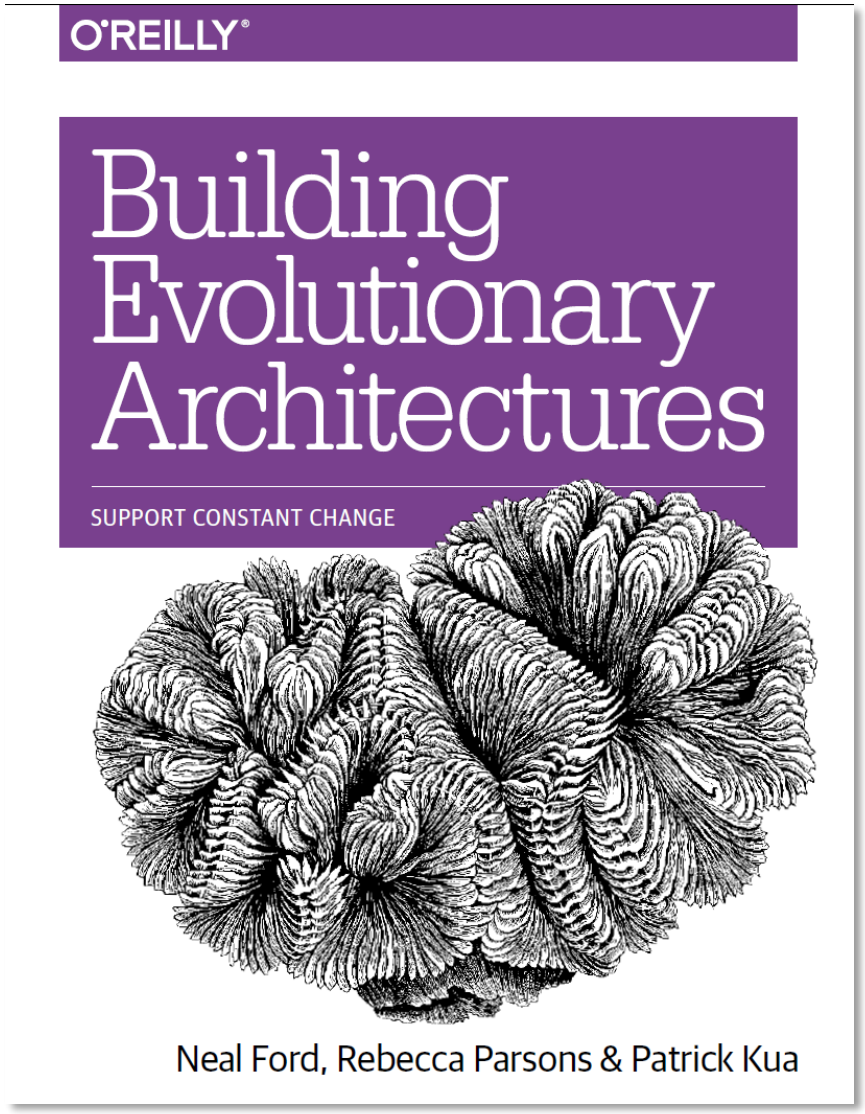
…

The set of design decisions that are hard to change

…

Which boils down to …."

Software Architecture is

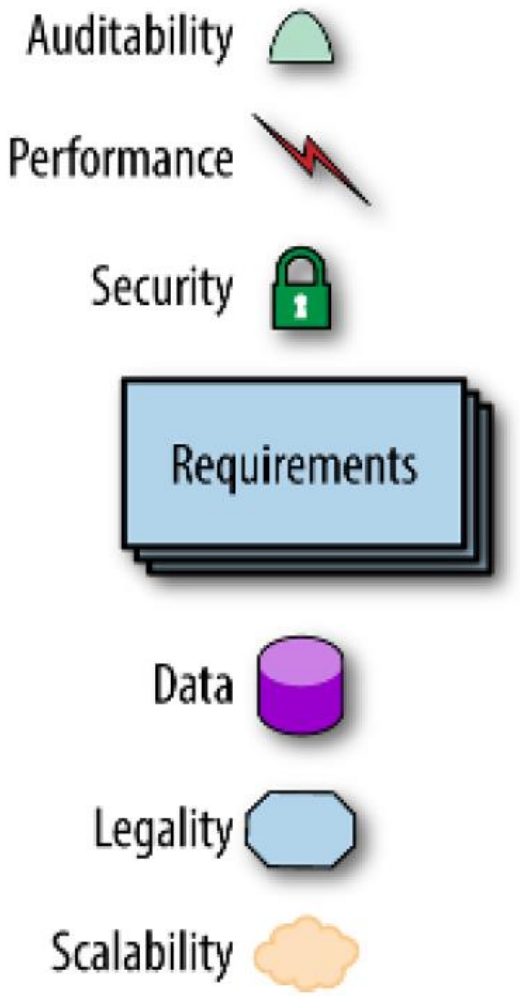


"……**the important stuff**

(whatever that is)….."

The architect's job is to understand and balance all of those important things (whatever they are)

The important stuff



accessibility accountability accuracy adaptability administrability affordability agility auditability autonomy availability compatibility composability configurability correctness credibility customizability debugability degradability determinability demonstrability dependability deployability discoverability distributability durability effectiveness efficiency usability extensibility failure transparency fault tolerance fidelity flexibility inspectability installability integrity interoperability learnability maintainability manageability mobility modifiability modularity operability orthogonality portability precision predictability process capabilities producibility provability recoverability relevance reliability repeatability reproducibility resilience responsiveness reusability robustness safety scalability seamlessness self-sustainability serviceability securability simplicity stability standards compliance survivability sustainability tailorability testability timeliness traceability

Architecture and Software Design

"....Architecture is the bigger picture: the choice of frameworks, languages, scope, goals, and high-level methodologies (Rational, waterfall, agile, etc.).

Design is the smaller picture: the plan for how code will be organized; how the contracts between different parts of the system will look; the ongoing implementation of the project's methodologies and goals. Specification are written during this stage...."

Architecture => Structure

Design => Structure and Meaning

The central question

How do we **define** and **communicate** this **structure** and **meaning** of the system (to be) build

So what is (a) Software Design

"...Software design is both a process and a model..."

- The design as a model is the "plan" of what to build
- It should define the structure of the code
- It should describe, make explicit, the functionality
- **Using abstractions to arrange and reduce complexity**

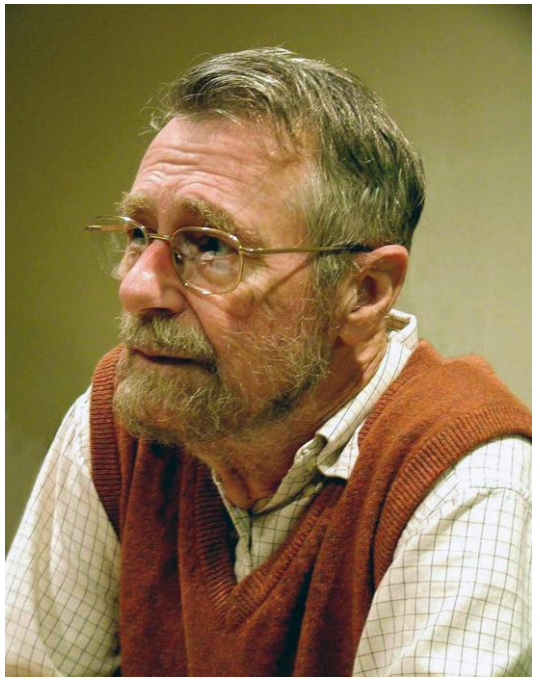
“...It's too abstract...”

- The imperative mind
- “...It's too vague...” or ”.. too abstract..”
- Mistake low-level imperative operations for exactness
- Every programming language, even assembler, is composed of high-level abstractions

Abstraction

"...Being abstract is something profoundly different from being vague … The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise…."

Edsger Dijkstra



How to express Abstractions

Depending the language

Types

Classes

Methods

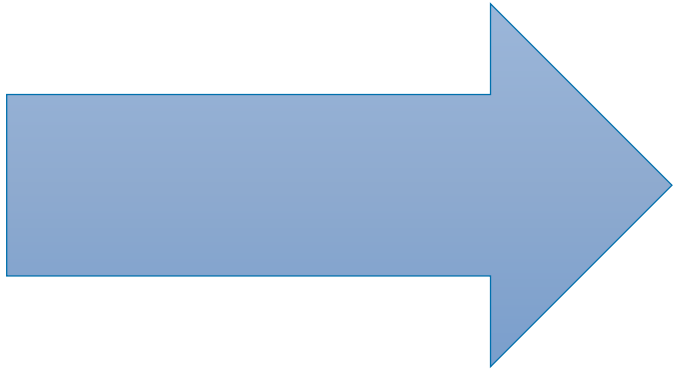
Functions

Decorators

Attributes

Modules

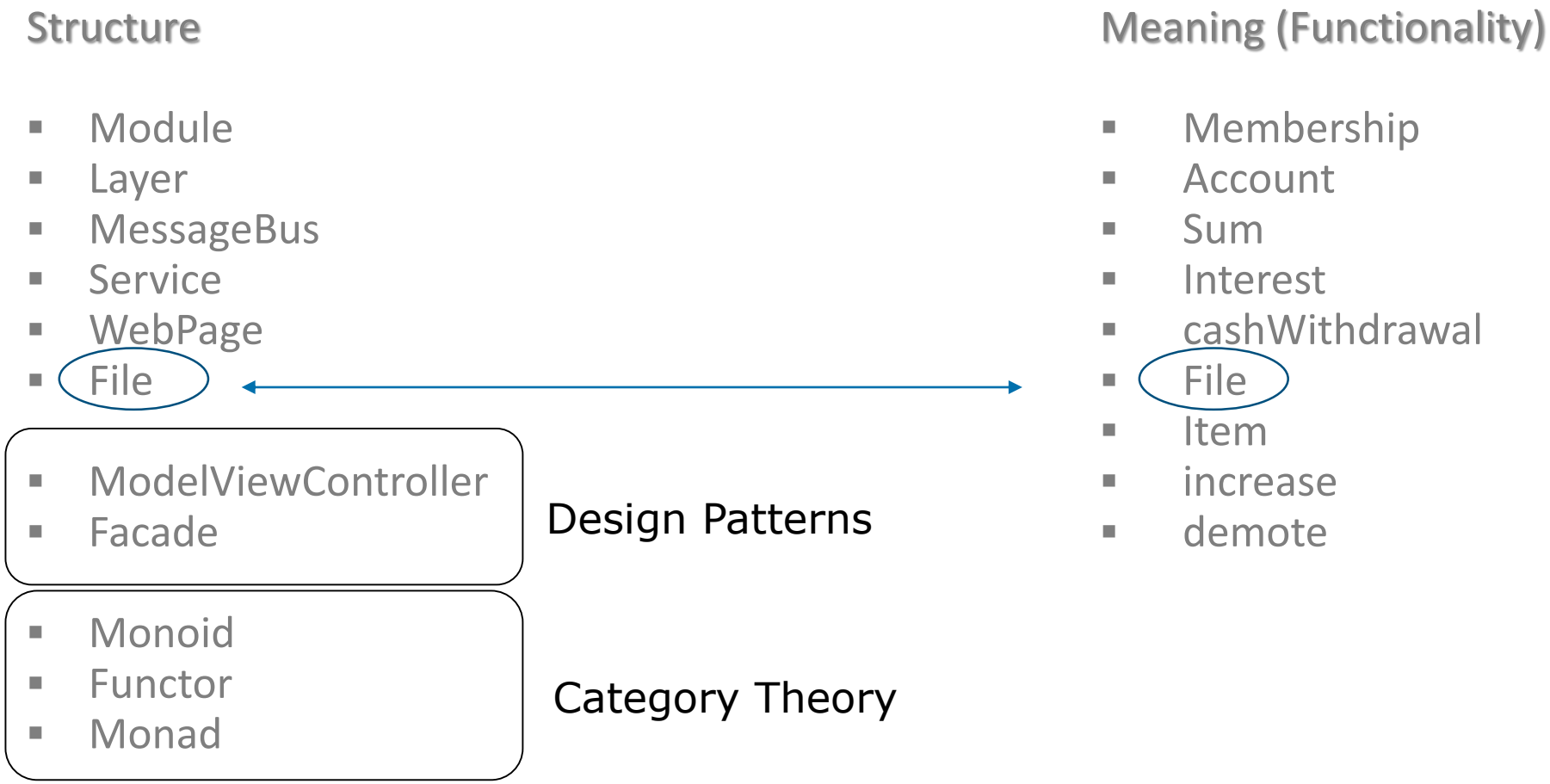
Macros



Nouns, verbs and adjectives

(and ….Spells, Magical incantaciones, through macros and embedded languages or DSL. But that is for another presentation)

Example abstractions



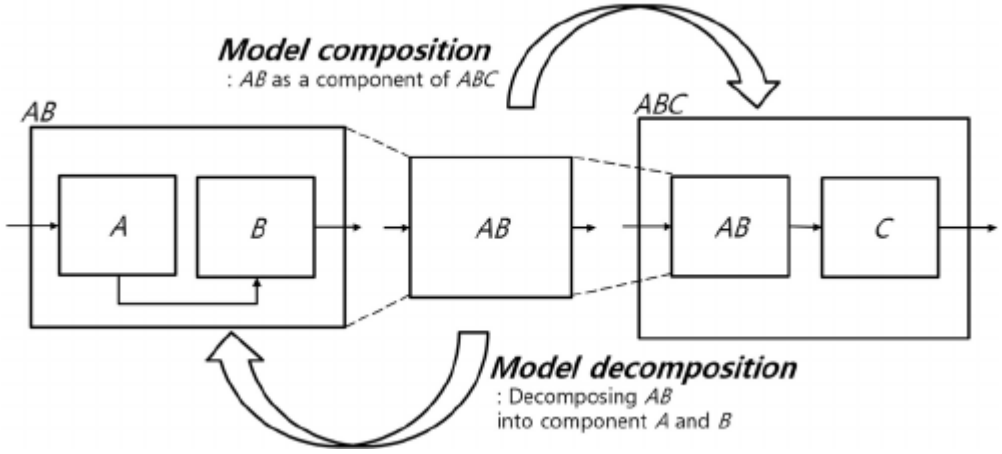
Software Design as Story Telling

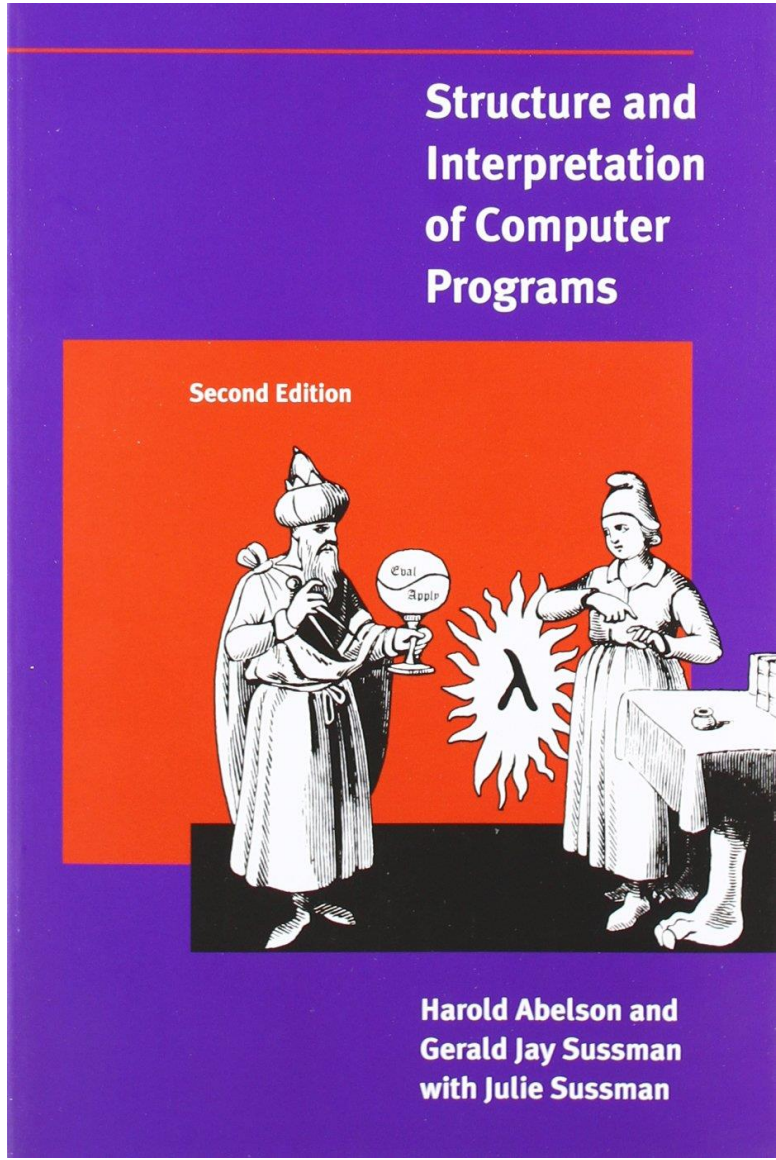
An abstraction can be created and combined with others to create a larger abstraction (**composition**). Once an abstraction needs to be detailed and specified it needs to be broken up in smaller components (**decomposition**).

This is the essence of programming….

This can be seen as a technical process. However, due to the intangible nature of software and modelling, it can actually be a linguistic process. Abstractions can consist of metaphors, similes, stories.

Software Design becomes a linguistic process. It becomes akin to telling a story.





So ……..

How do we **define** and **communicate** this **structure** and **meaning** of the system (to be) build

By Story Telling.

Software Design is

Story Telling with Abstractions

Code should tell the story, reflect the Design

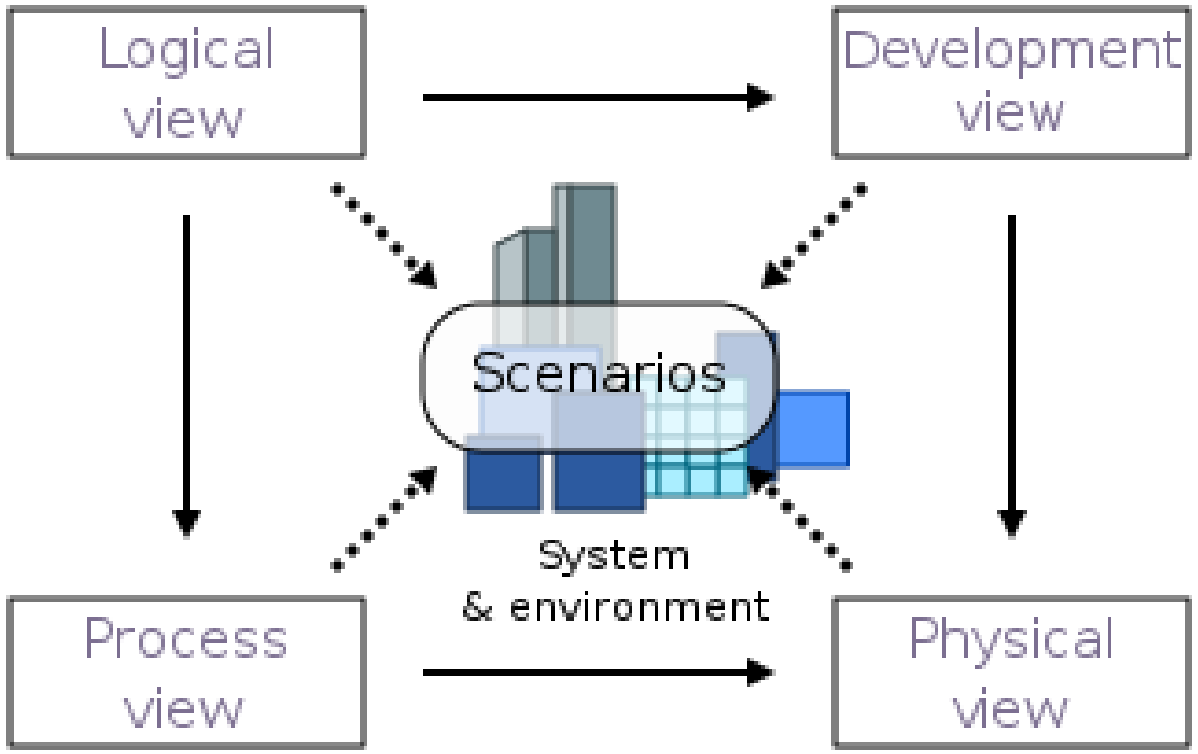
src ▸ system.clj

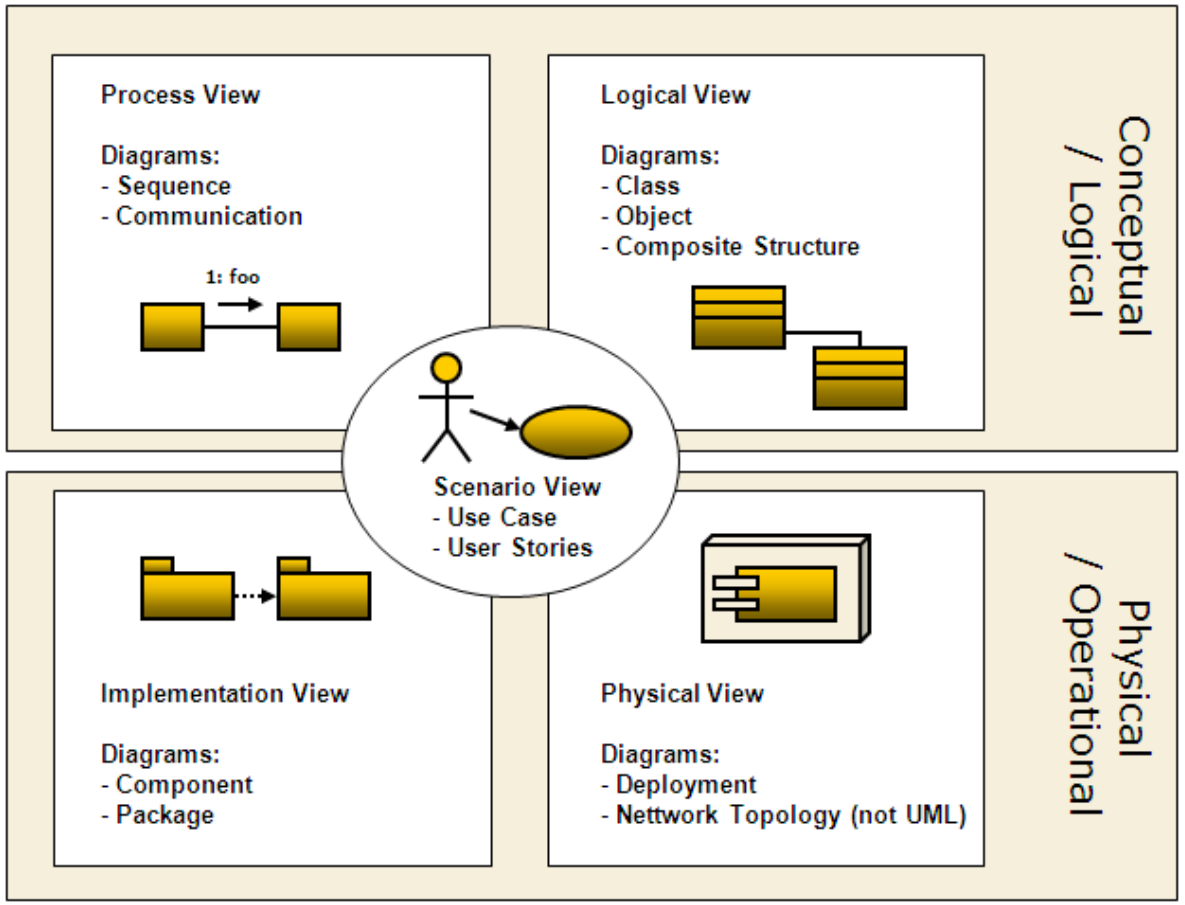
```
1
2   (system social-security-benefit-upload
3       (component
4           (security :config "/config/security.conf" :cert-store "/config/cert.cer"))
5       (component
6           (view upload-service)
7           (logic upload)
8           (data item-store-dao log-data))
9       (component
10          (view report-service)
11          (logic reporting)
12          (data jasper-reports item-store-dao log-data)))
```

But….

- Not all people find textual representation readable for all aspects of the design (especially relationships)
- Most programming languages cannot adequately express the whole software design satisfactorily
- Their syntax is not sufficiently expressive, flexible or extensible enough to viably communicate the wider architecture concerns
- We therefore reach for models and diagrams, representing the code

Instead we will model using paradigms like the 4+1 architectural view model



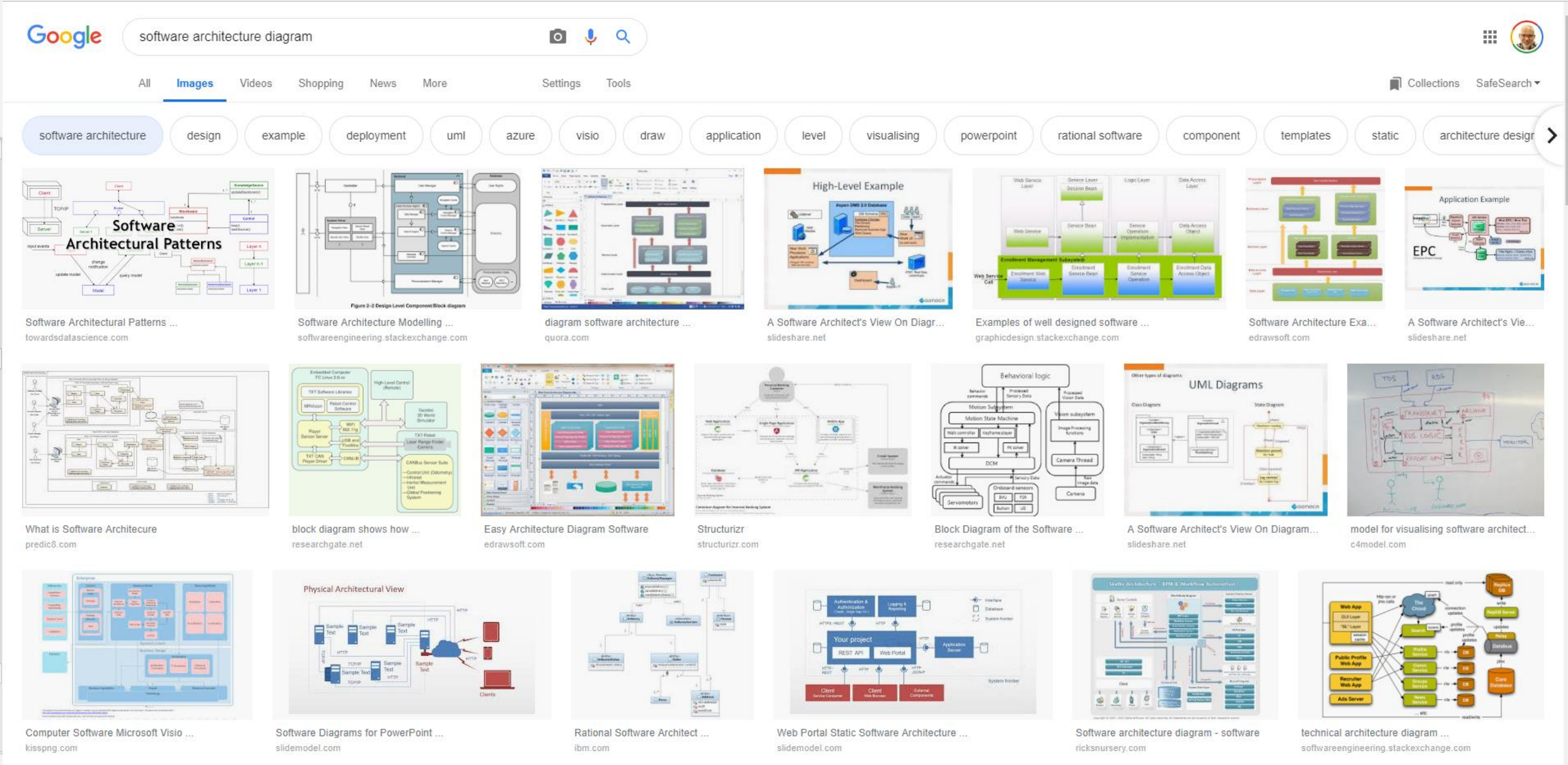


And tools like UML

- Standard notation
- Great tool support (we use PlantUML)

- Low level
- Limited expressiveness (needs OCL)
- Too complex
- Not very suitable for programming paradigms which are not “OOP”
- “Nobody uses UML”
  - Associated with “Big Design up-front”, “Waterfall”
  - Not being taught at University anymore

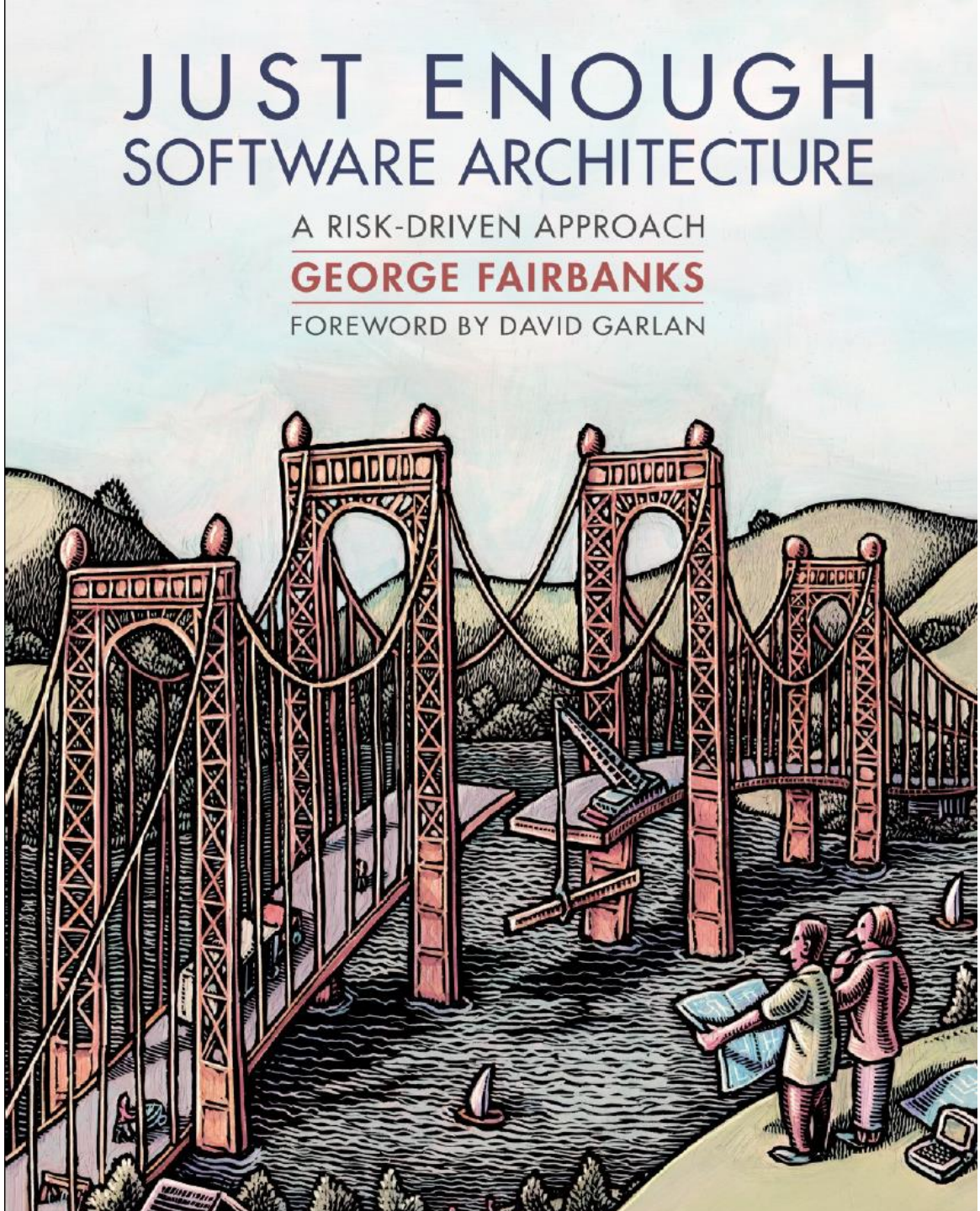
As many diagram types as there are designs



Model-code gap

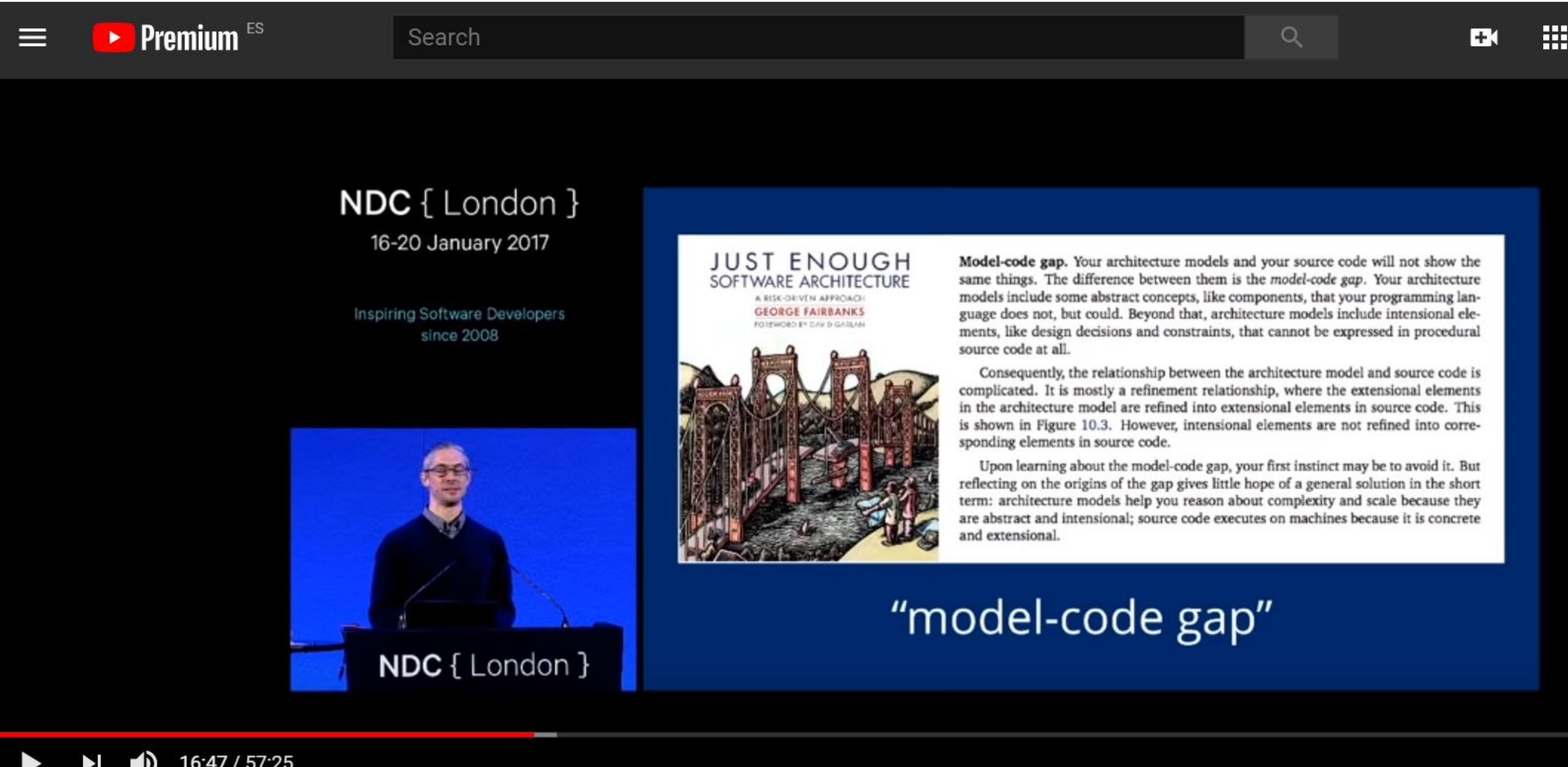
"…**Your architecture models and your source code will not show the same things.** The difference between them is the ***model-code gap***. Your architecture models include some abstract concepts, like components, that your programming language does not, but could. Beyond that, architecture models include intentional elements, like design decisions and constraints, that cannot be expressed in procedural source code at all.

Consequently, the relationship between the architecture model and source code is complicated



https://www.youtube.com/watch?v=Ym9nhVZs89o

Visualise, document and explore your software architecture - Simon Brown



https://c4model.com/

The C4 model for visualising software architecture

Context, Containers, Components and Code

In a hurry? Read 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

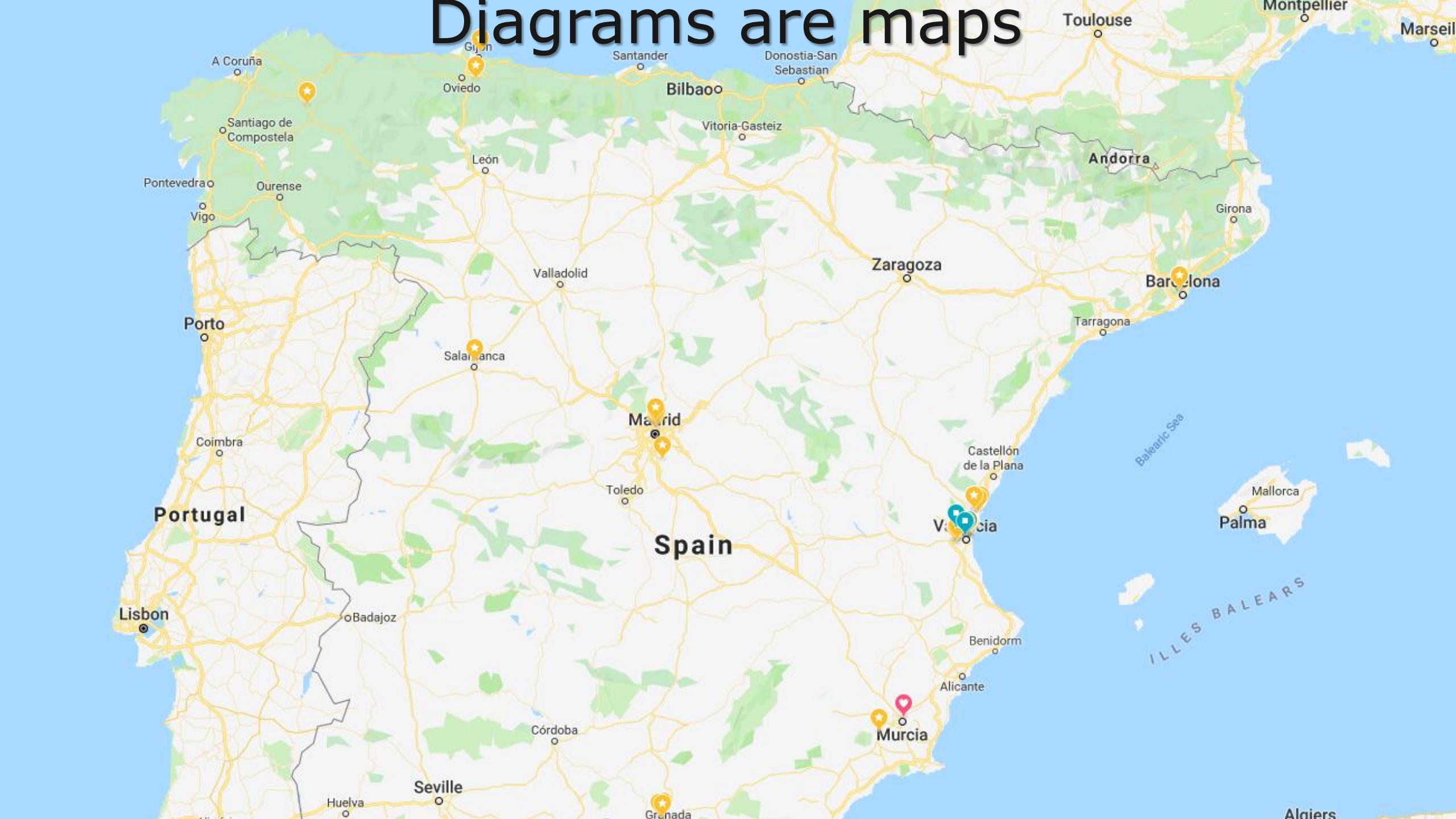
ソフトウェアアーキテクチャのためのC4モデル

Introduction

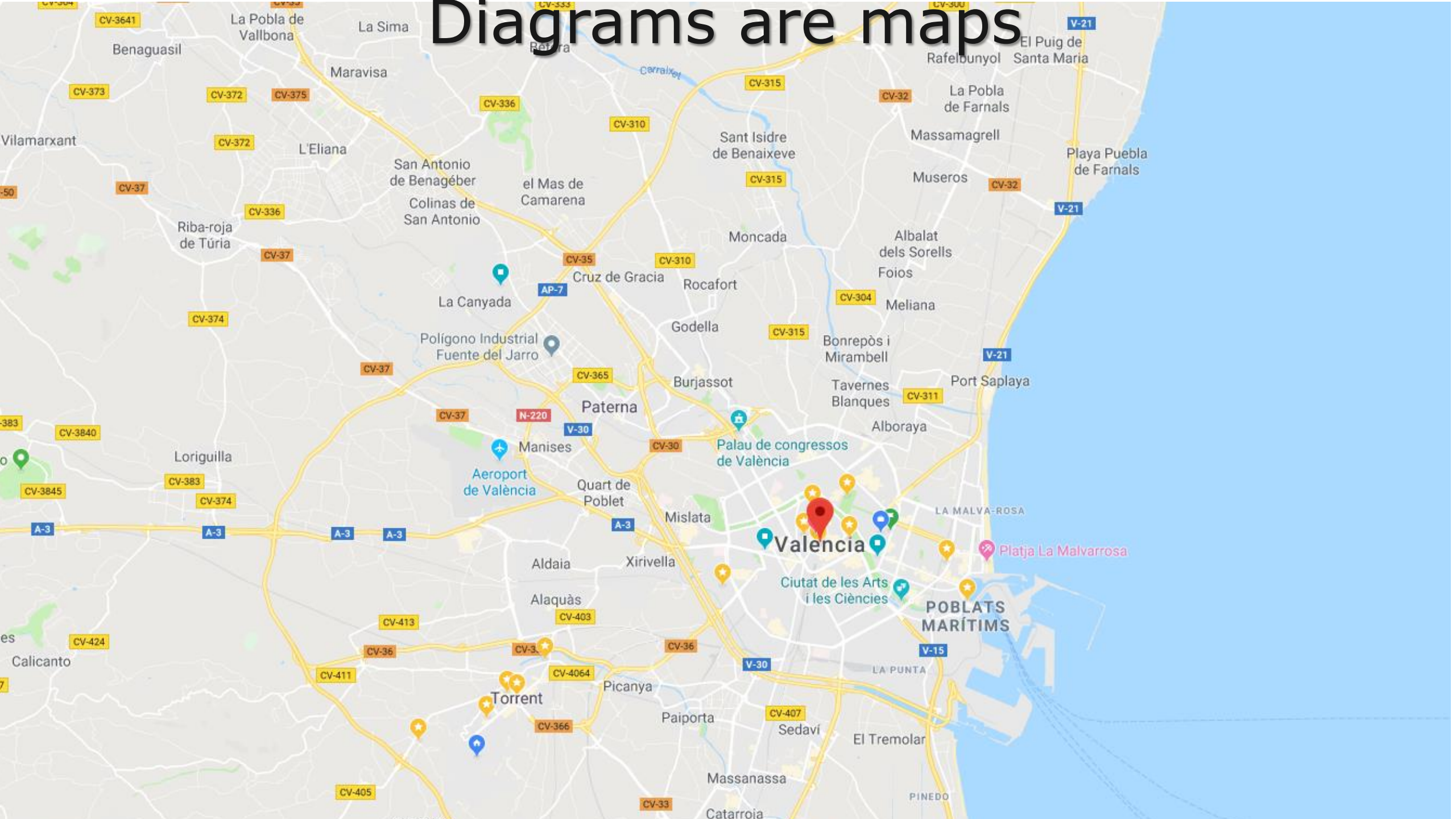
Ask somebody in the building industry to visually communicate the architecture of a building and you'll be presented with site plans, floor plans, elevation views, cross-section views and detail drawings. In contrast, ask a software developer to communicate the software architecture of a software system using diagrams and you'll likely get a confused mess of boxes and lines ... inconsistent notation (colour coding, shapes, line styles, etc), ambiguous naming, unlabelled relationships, generic terminology, missing technology choices, mixed abstractions, etc.



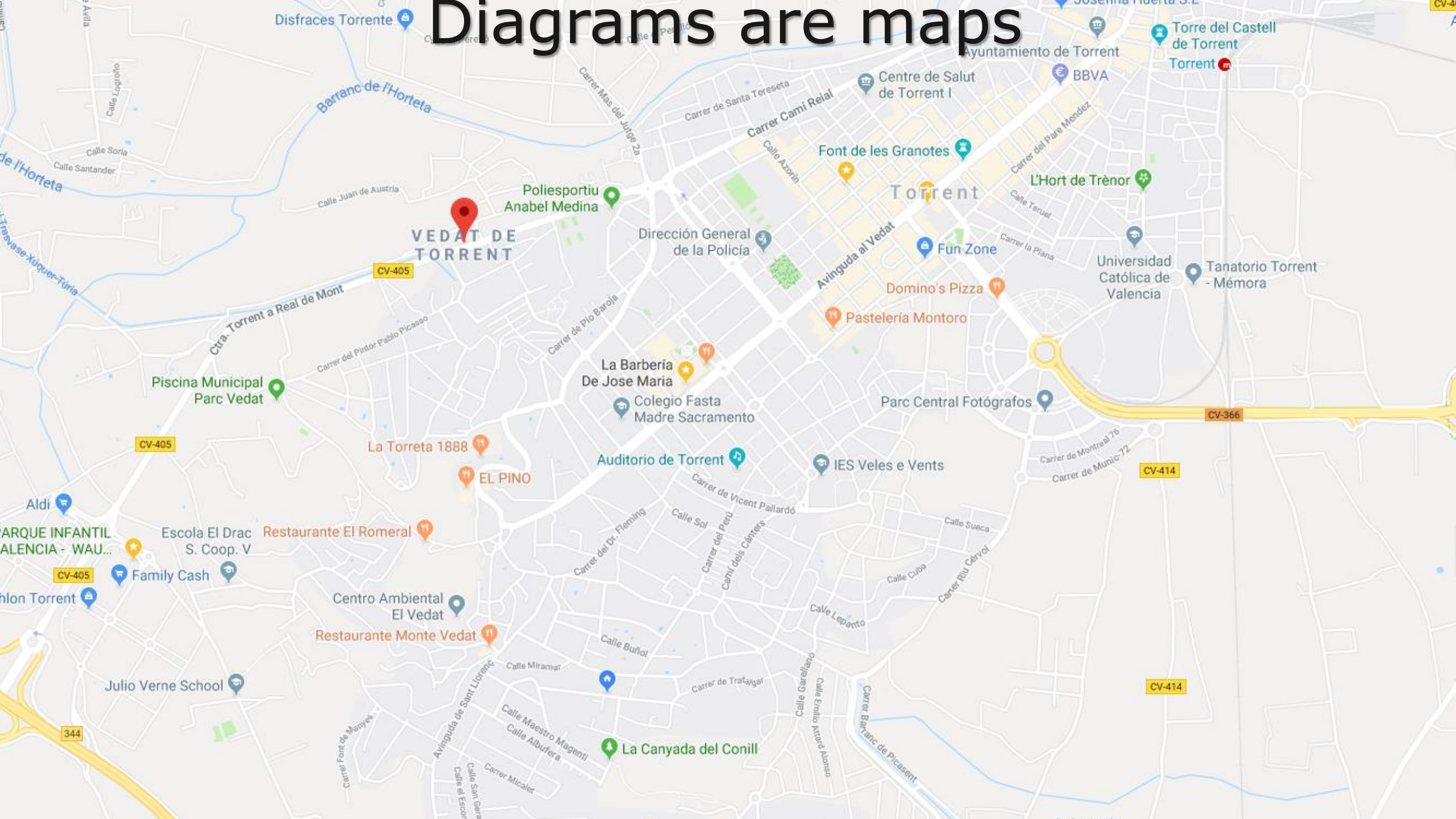

Diagrams are maps



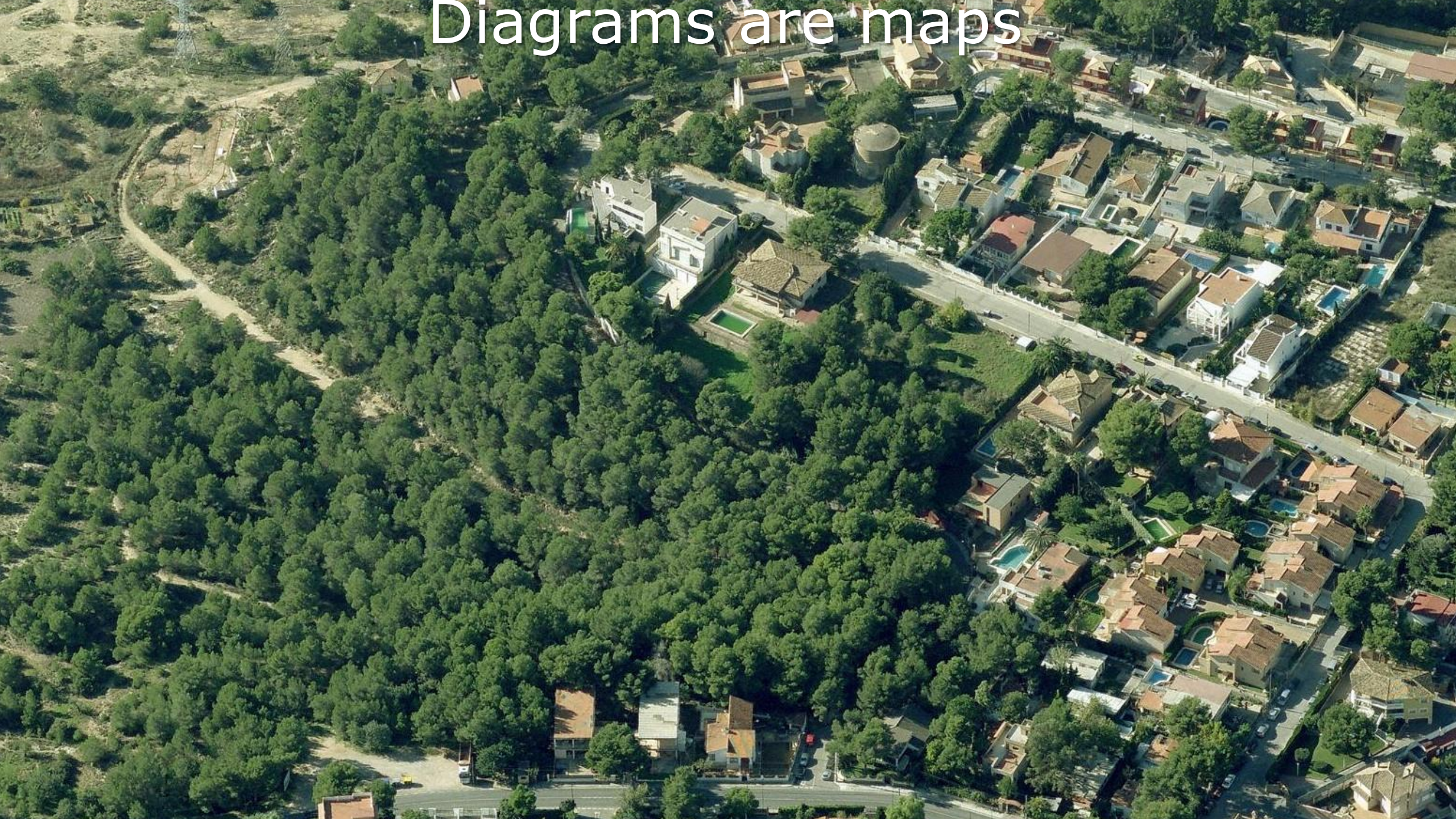
Diagrams are maps

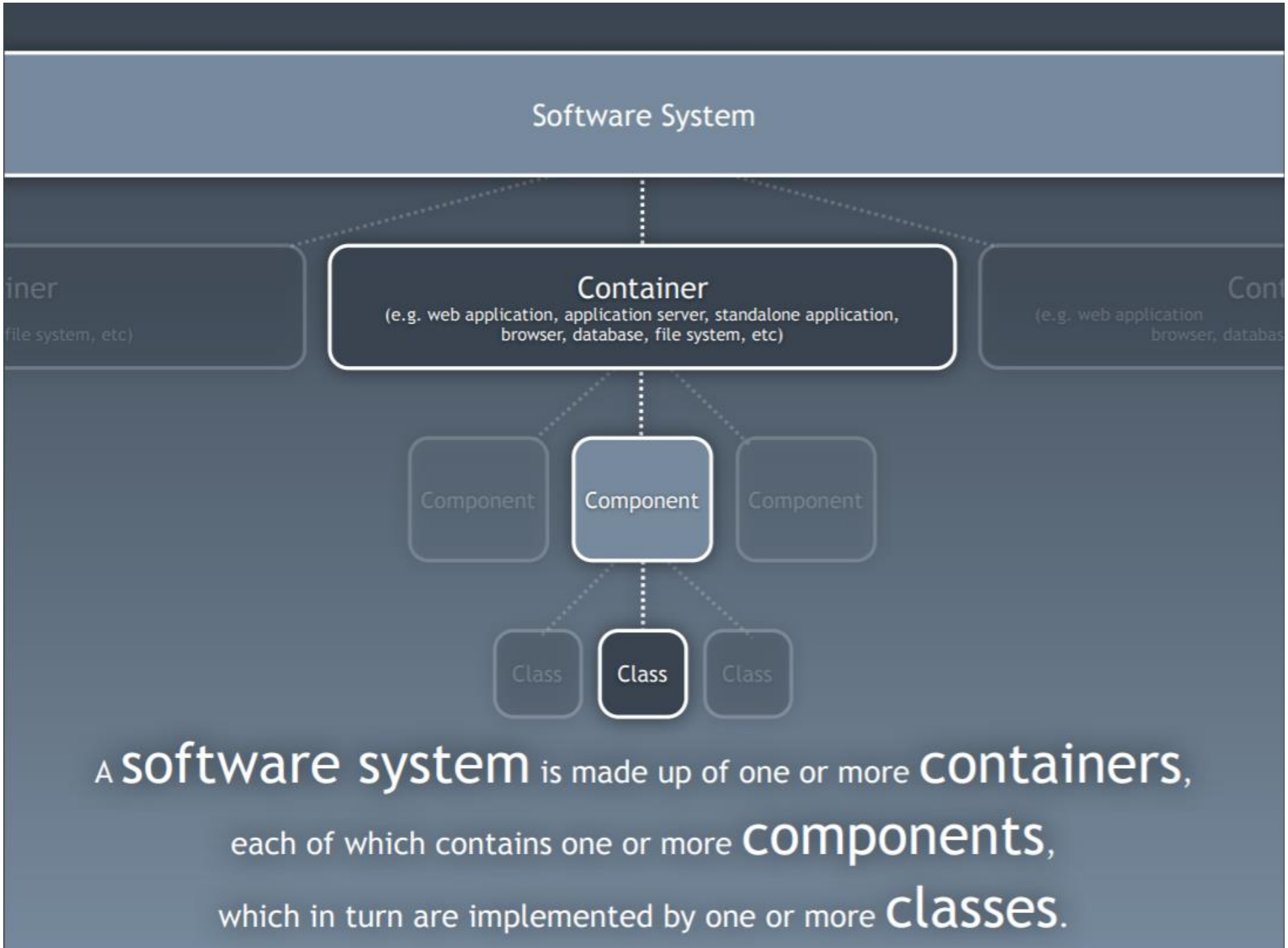


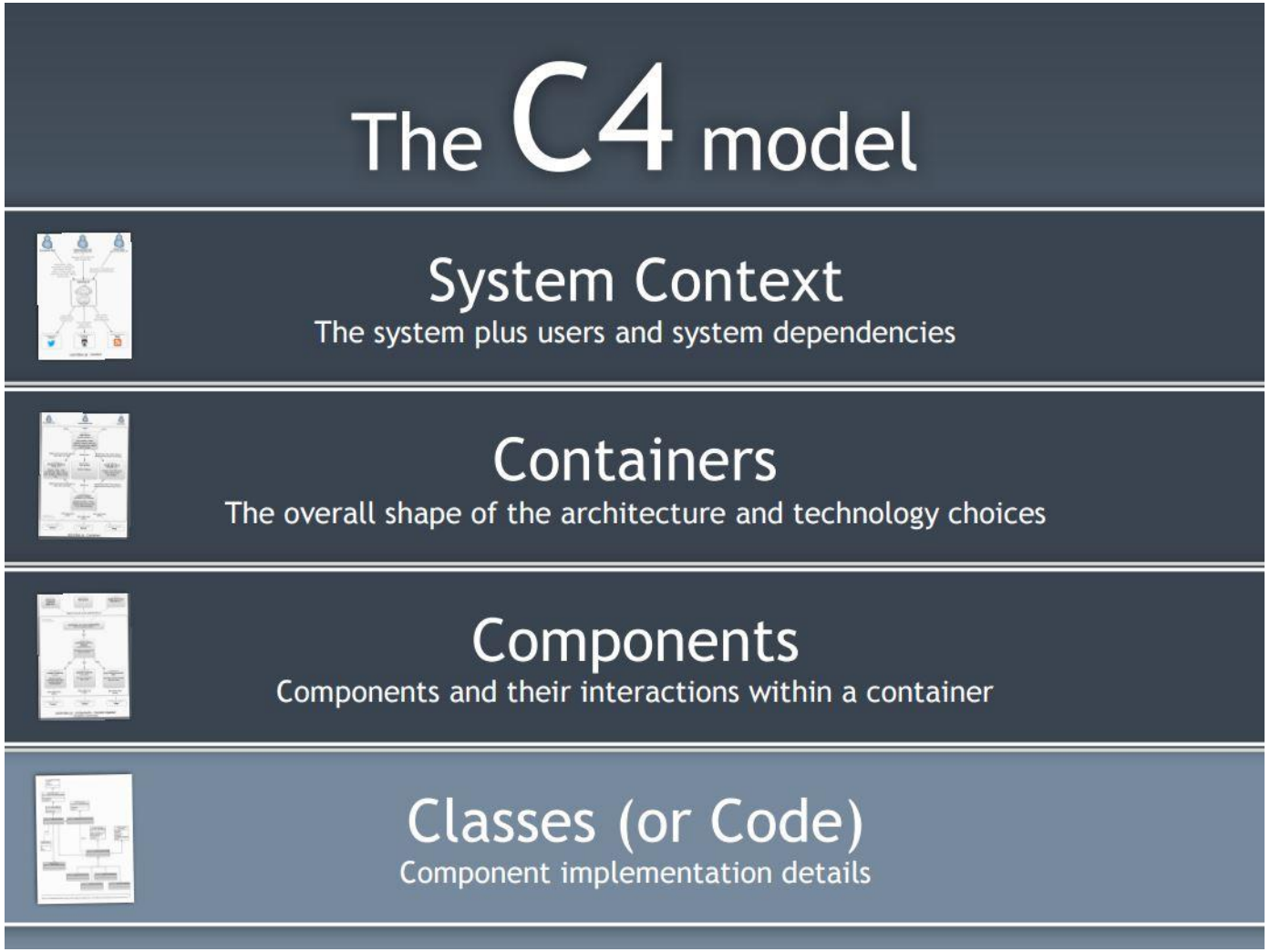
Diagrams are maps



Diagrams are maps




Software System
iner
file system, etc)
Container
(e.g. web application, application server, standalone application,
browser, database, file system, etc)
Cont
(e.g. web application
browser, databas
Component
Component
Component
Class
Class
Class
A software system is made up of one or more containers,
each of which contains one or more components,
which in turn are implemented by one or more classes.



Compose

Tell the story

Decompose

Social Security Example

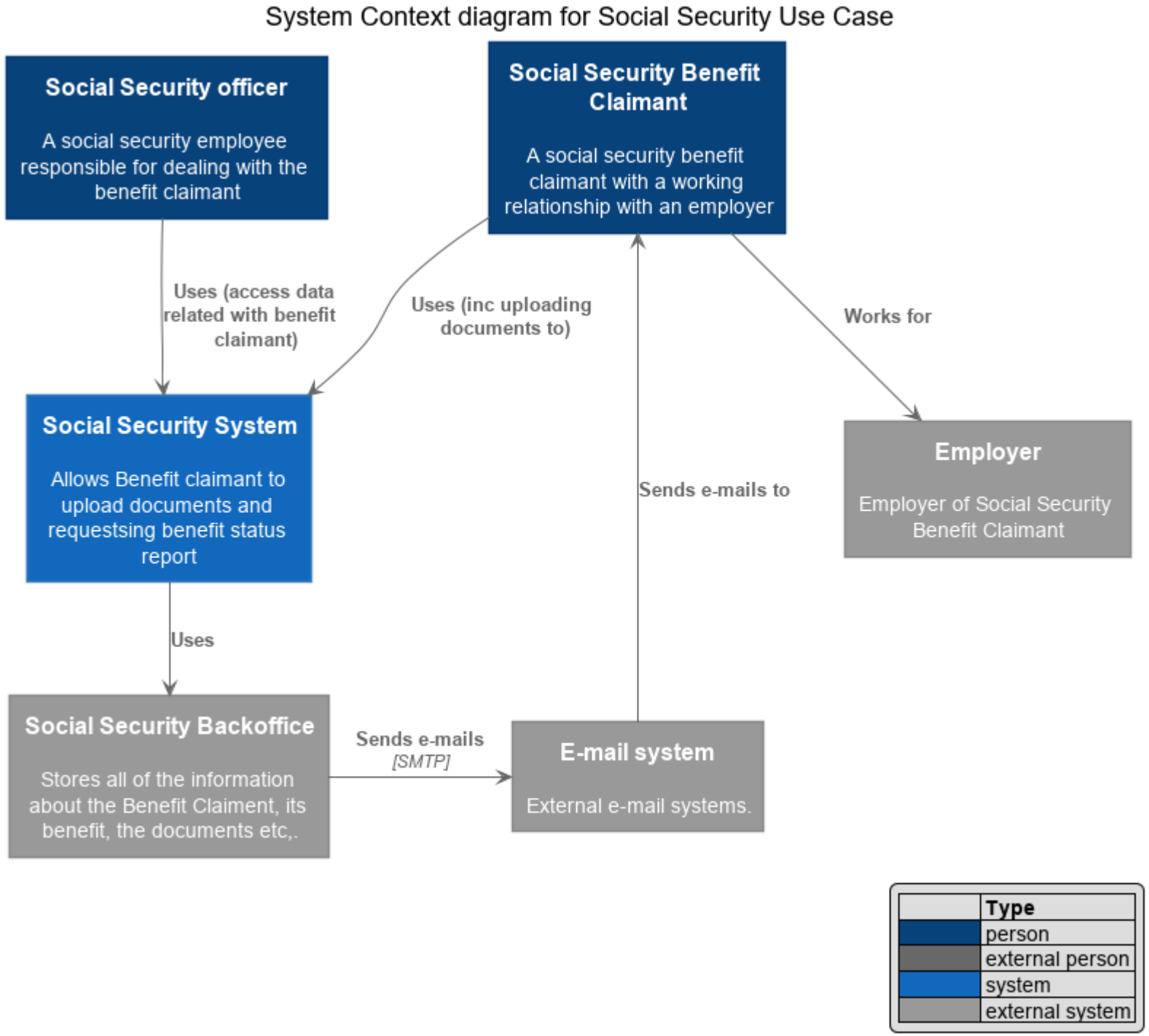
The application is used to give benefit recipients the option of uploading 'supporting documents' (documents such as pay slips) that prove a working relationship.

If a benefit claimant has a working relationship with an employer, this has an impact on the benefit. These supporting documents are checked and it is checked whether the data is in accordance with the data made available to the Social Security from the employers.

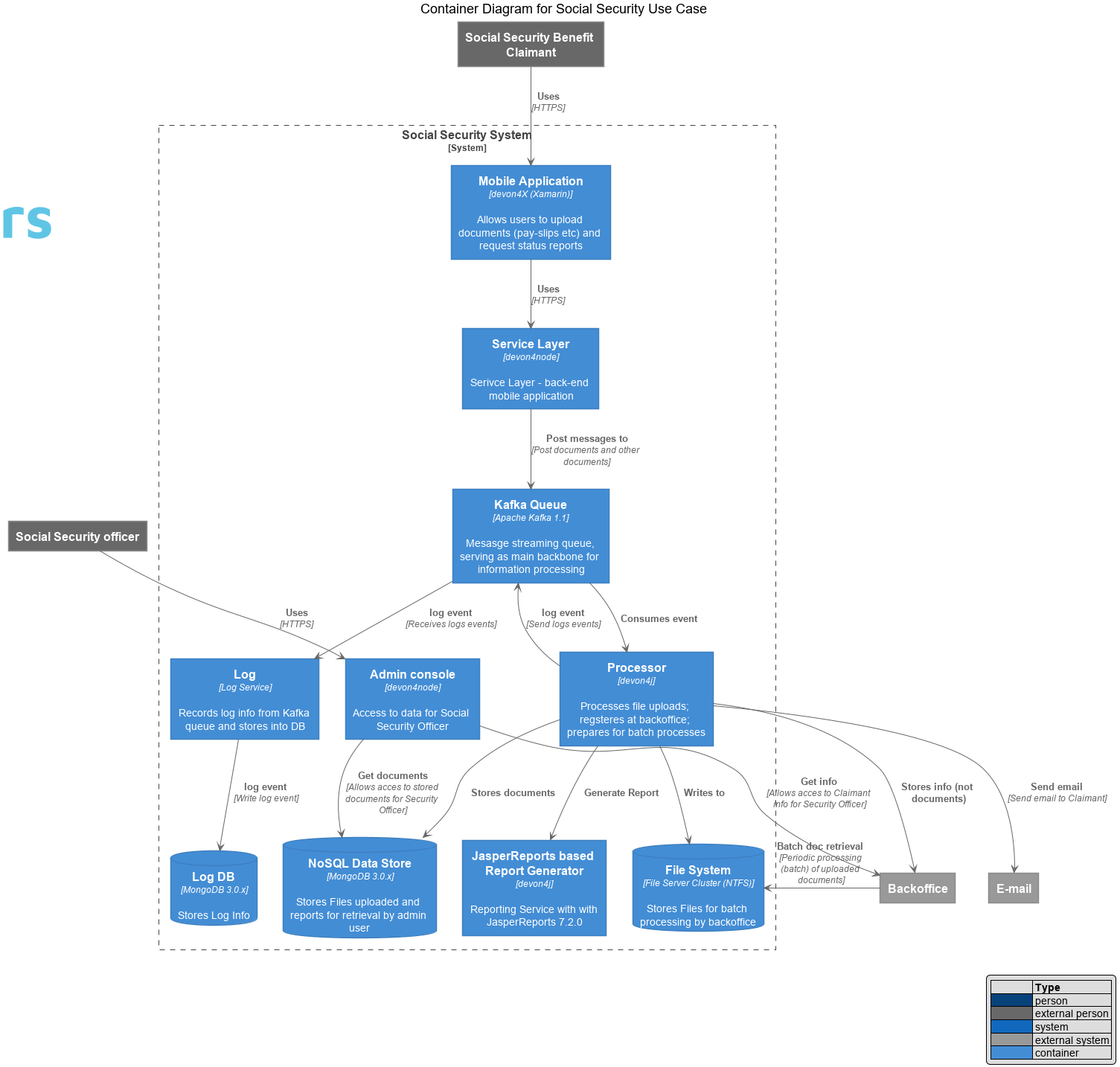
The application can also be used to obtain an official, signed, report about any change of the benefit requested by or granted to the benefit claimant.

The Social Security Officer responsible for the case of the claimant should be able to access the information through an internal application.

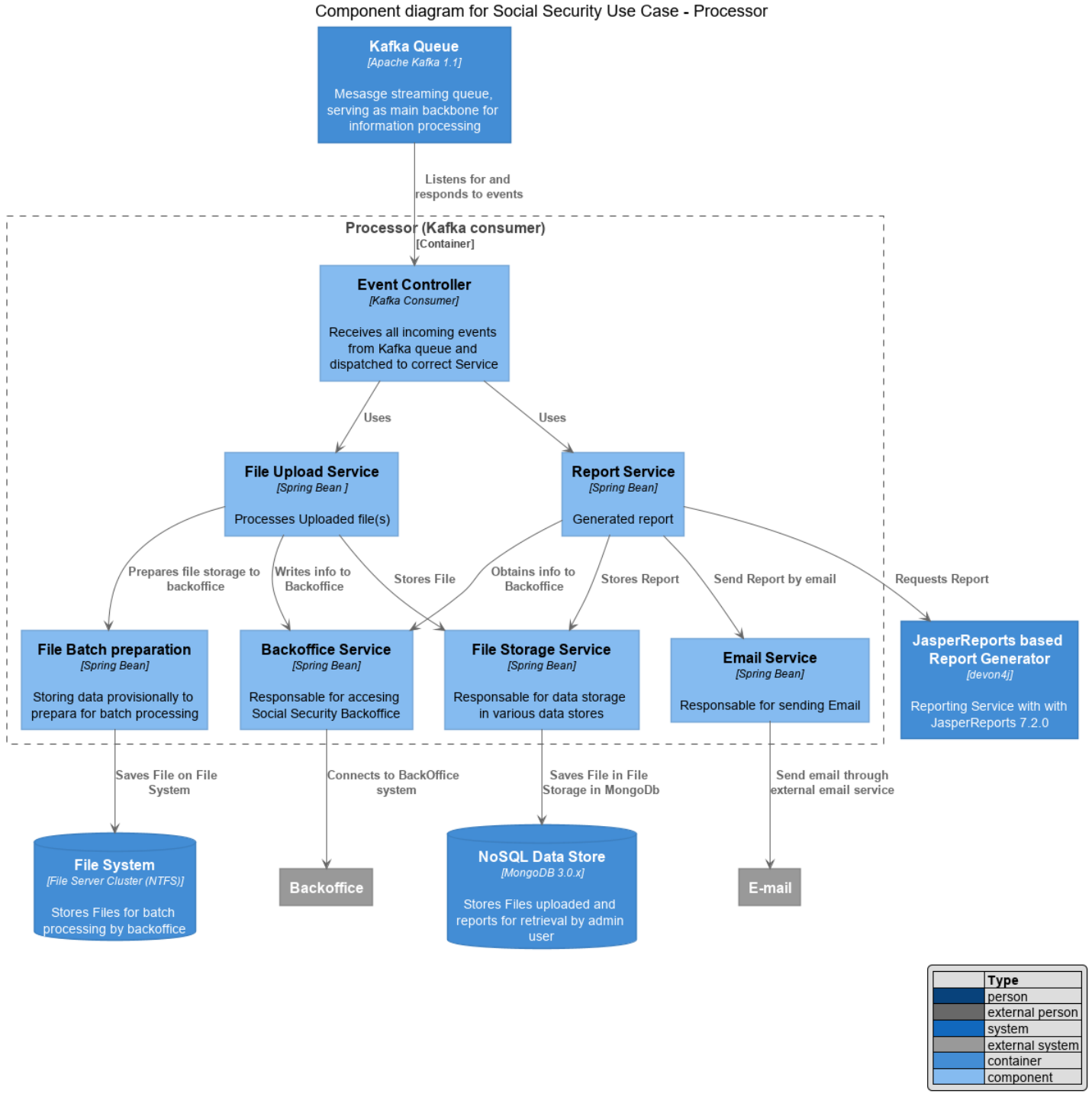
System Context



Containers


Container Diagram for Social Security Use Case
Social Security Benefit Claimant
Uses
[HTTPS]
Social Security System
[System]
Mobile Application
[devon4X (Xamarin)]
Allows users to upload documents (pay-slips etc) and request status reports
Uses
[HTTPS]
Service Layer
[devon4node]
Serivce Layer - back-end mobile application
Post messages to
[Post documents and other documents]
Kafka Queue
[Apache Kafka 1.1]
Mesasge streaming queue, serving as main backbone for information processing
Social Security officer
Uses
[HTTPS]
log event
[Receives logs events]
log event
[Send logs events]
Consumes event
Log
[Log Service]
Records log info from Kafka queue and stores into DB
Admin console
[devon4node]
Access to data for Social Security Officer
Processor
[devon4j]
Processes file uploads; regsteres at backoffice; prepares for batch processes
log event
[Write log event]
Get documents
[Allows acces to stored documents for Security Officer]
Stores documents
Generate Report
Writes to
Get info
[Allows acces to Claimant info for Security Officer]
Stores info (not documents)
Send email
[Send email to Claimant]
Batch doc retrieval
[Periodic processing (batch) of uploaded documents]
Log DB
[MongoDB 3.0.x]
Stores Log Info
NoSQL Data Store
[MongoDB 3.0.x]
Stores Files uploaded and reports for retrieval by admin user
JasperReports based Report Generator
[devon4j]
Reporting Service with with JasperReports 7.2.0
File System
[File Server Cluster (NTFS)]
Stores Files for batch processing by backoffice
Backoffice
E-mail
Type
person
external person
system
external system
container

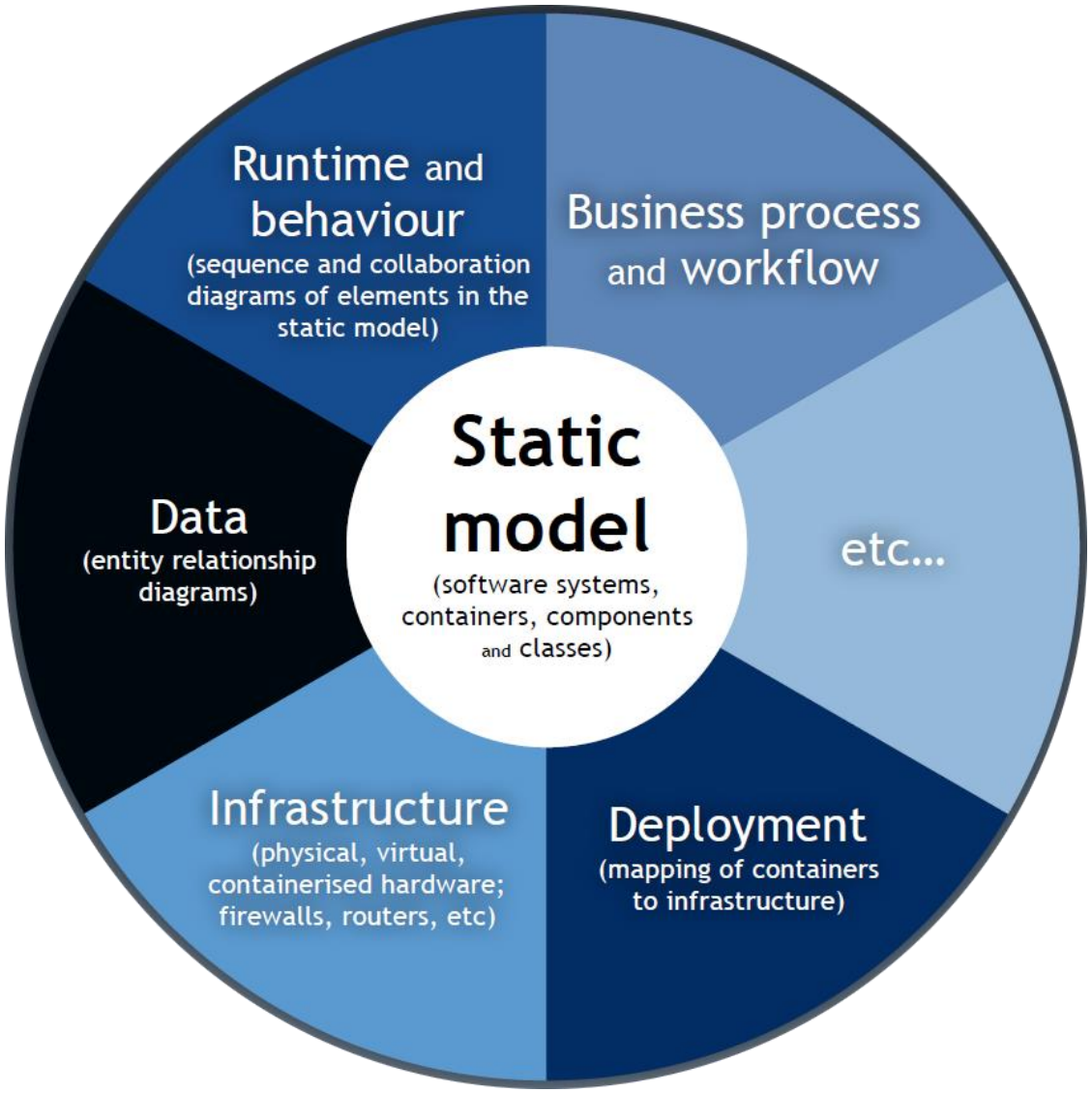
Components


Component diagram for Social Security Use Case - Processor
Kafka Queue
[Apache Kafka 1.1]
Mesasge streaming queue, serving as main backbone for information processing
Listens for and responds to events
Processor (Kafka consumer)
[Container]
Event Controller
[Kafka Consumer]
Receives all incoming events from Kafka queue and dispatched to correct Service
Uses
Uses
File Upload Service
[Spring Bean]
Processes Uploaded file(s)
Report Service
[Spring Bean]
Generated report
Prepares file storage to backoffice
Writes info to Backoffice
Stores File
Obtains info to Backoffice
Stores Report
Send Report by email
Requests Report
File Batch preparation
[Spring Bean]
Storing data provisionally to prepara for batch processing
Backoffice Service
[Spring Bean]
Responsable for accesing Social Security Backoffice
File Storage Service
[Spring Bean]
Responsable for data storage in various data stores
Email Service
[Spring Bean]
Responsable for sending Email
JasperReports based Report Generator
[devon4j]
Reporting Service with with JasperReports 7.2.0
Saves File on File System
Connects to BackOffice system
Saves File in File Storage in MongoDb
Send email through external email service
File System
[File Server Cluster (NTFS)]
Stores Files for batch processing by backoffice
Backoffice
NoSQL Data Store
[MongoDB 3.0.x]
Stores Files uploaded and reports for retrieval by admin user
E-mail
Type
person
external person
system
external system
container
component

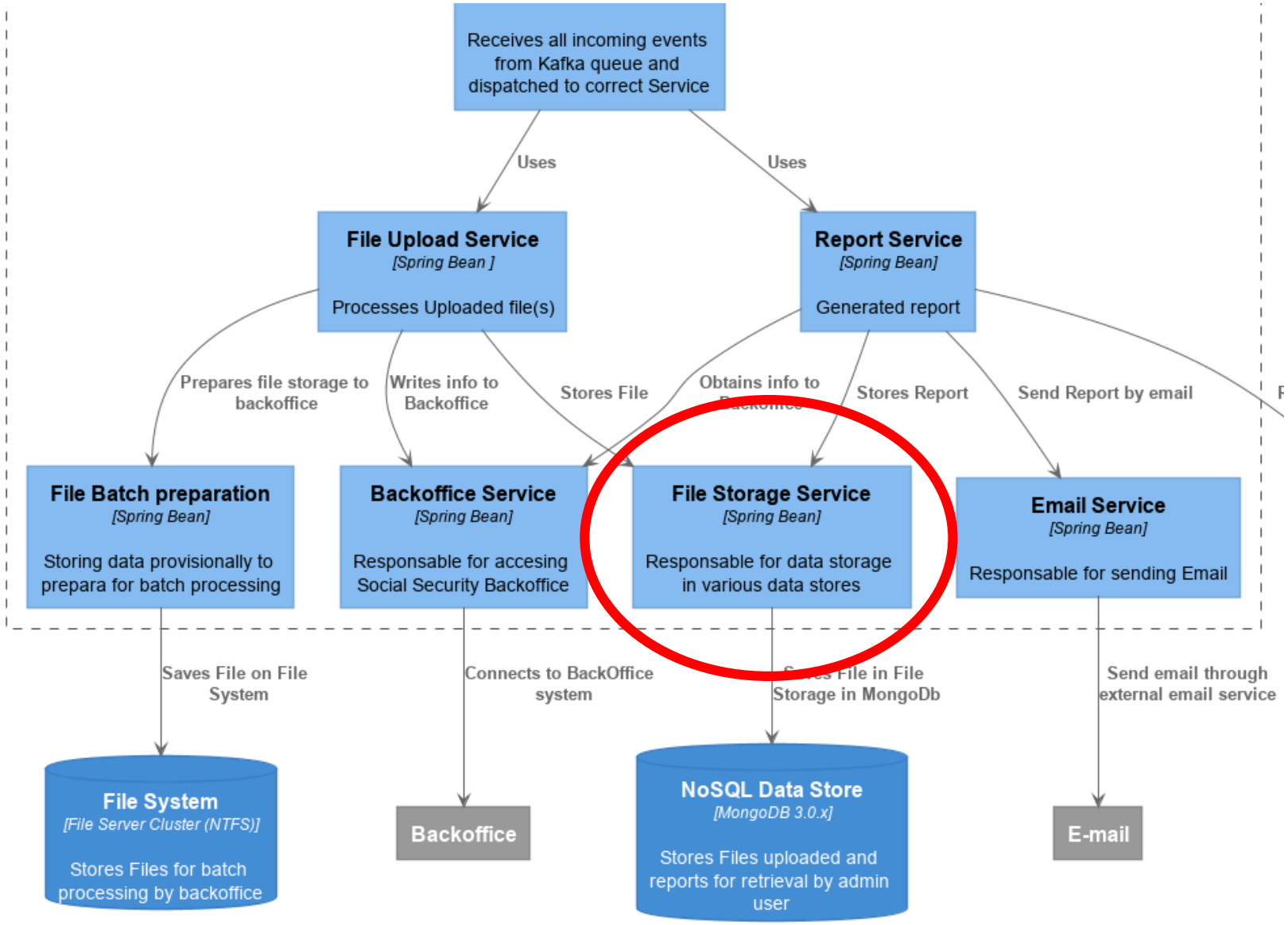
Extend the model

Based on the static model, the Design can be enriched with other diagram types.

UML and other (in)formal standards (ERD) can play a very useful role

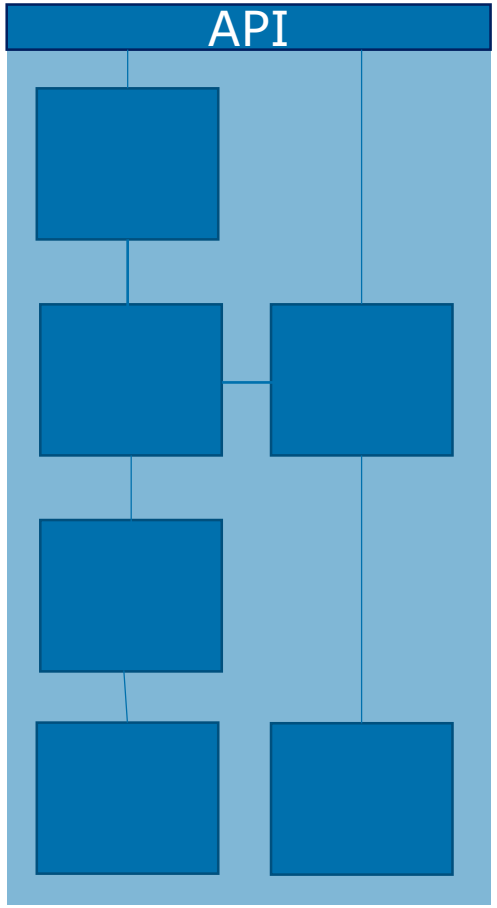


How to design a Component



Component

- Definition of a component in the C4 model
- Should be the smallest building block of the “Architecture”
- Is should contain a clearly defined API / **Interface**
- Will typically consist of multiple classes and functions
- A component should restrict access to its subcomponents (i.e.. Java packages, .NET Assemblies, ECMAScript/TypeScript modules etc)
- “Visible as” Angular Service, Spring Bean, Node module
- It should typically be a global singleton and not Support instantiation (multiple instances)
- Should have a Functional or Service Orientated Interface rather than OOP (no maintaining of state between invocations!)



Create abstractions to define meaning (functionality)

Apart from the structure of the system, the design also needs to describe the functionality of the system.

The basis for this form the Functional Requirements for the applications as contained in the Specifications like Use Cases or User Stories.

As with the “Structure” of the Architecture, there needs to be a clear link to defined functionality and Code.

And telling the story works here as well….

System Design as a story

As a user generating a **document**,

I need to be able to **store** the document,

so that it can be **retrieved** at any later moment by myself or other parties

“I can have a **storage unit** when I can **store documents.**

I or other parties can **retrieve** documents from the store

For this the stored document needs to be represented by a **unique ID**”

Postpone the “definitions” of these concepts

```
interface Storage {

    save(arg0: Store, arg1: Document) : ID;
    get(arg0: Store, arg1: ID): Document;

}
```

Elaborating the Story

“I can have **storage unit** when I can **store and retrieve documents of an undeterminable length and/or varying size. Basically I should consider them to be a stream of data. “**

“This stream of data can be either a **stream of bytes** or a stream of **utf-8 encoded text.** As the stream does not represent a “Document” I can accompany it with a corresponding set of **Document attributes”**

```
type Data = Text | Binary
interface Storage {

    save(arg0: Store, arg1: Stream<Data>, arg2: DocumentAttrs) : ID;
    get(arg0: Store, arg1: ID): Stream<Data>;
    getAttr(arg0: Store, arg1: ID): DocumentAttrs;
}
```

Refining the Story

“I can have **storage unit** when I can **store and retrieve documents of an undeterminable length and/or varying size without having to wait** for the completion of the operation “

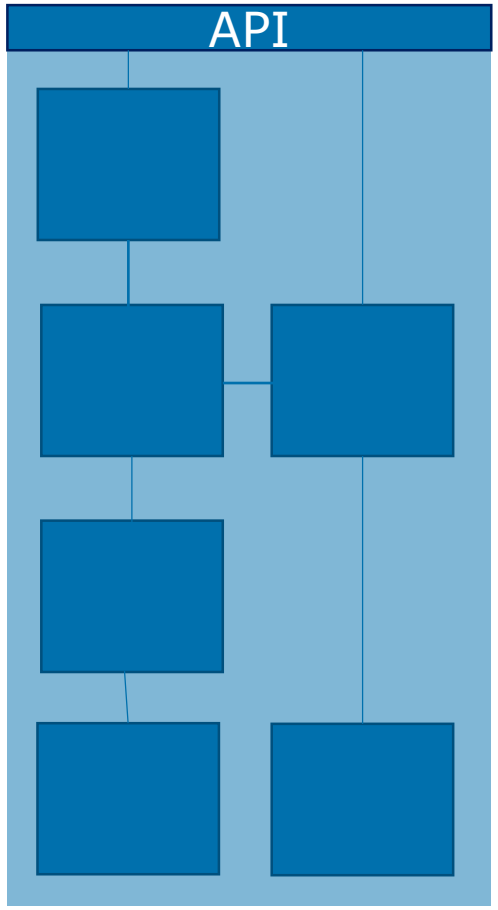
```
interface Storage {

    save(arg0: Store, arg1: Stream<Data>, arg2: DocumentAttrs) : Promise<ID>;
    get(arg0: Store, arg1: ID): Promise<Stream<Data>>;
    getAttr(arg0: Store, arg1: ID): Result<DocumentAttrs>;

    remove(arg0: ID): Result<void>
}
```

I need to be able to **initiate an asynchronous operation**. In case of both an asynchronous as well as a synchronous operation, I need to be **advised of the success** of the operation.

Finally, I should be able to **remove the stored document.**

“

The Storage Interface




```
interface Storage {

    save(arg0: Store, arg1: Stream<Data>, arg2: DocumentAttrs) : async<ID>;
    get(arg0: Store, arg1: ID): async<Stream<Data>>;
    getAttr(arg0: Store, arg1: ID): Result<DocumentAttrs>;

    remove(arg0: ID): Result<void>
}
```

By having begun as an abstraction, the component is assured to be fully decoupled of – i.e. have no dependencies on – any underlying storage mechanism. In that way:

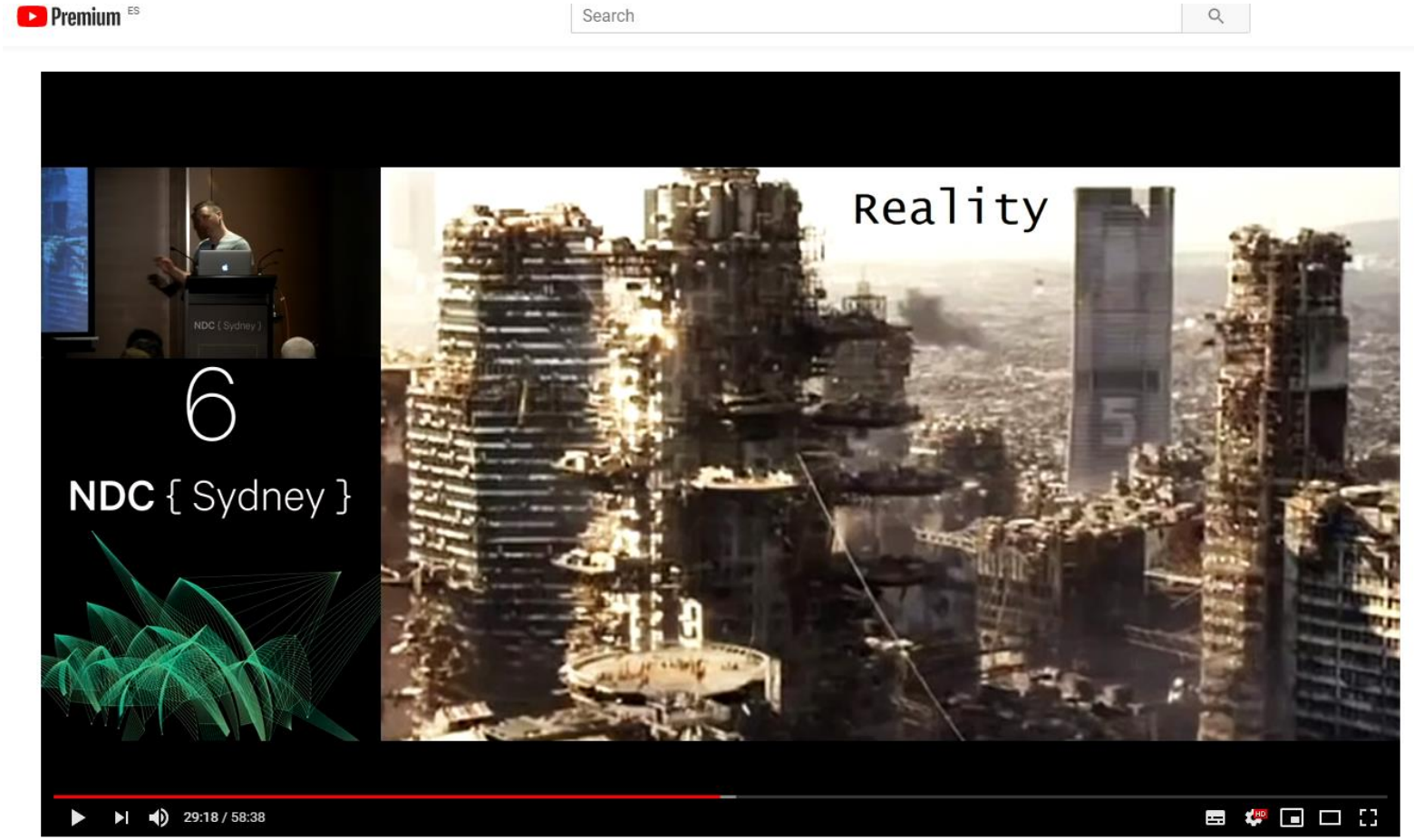
- It is generic enough to be used by different storage mechanism (File system, Sharepoint, Mongo, etc etc)
- It is testable
- And “pluggable”

Domain Driven Design

A established, relatively formal, methodology for such language driven design is **Domain Driven Design** as introduced by Eric Evans in the 2000´s

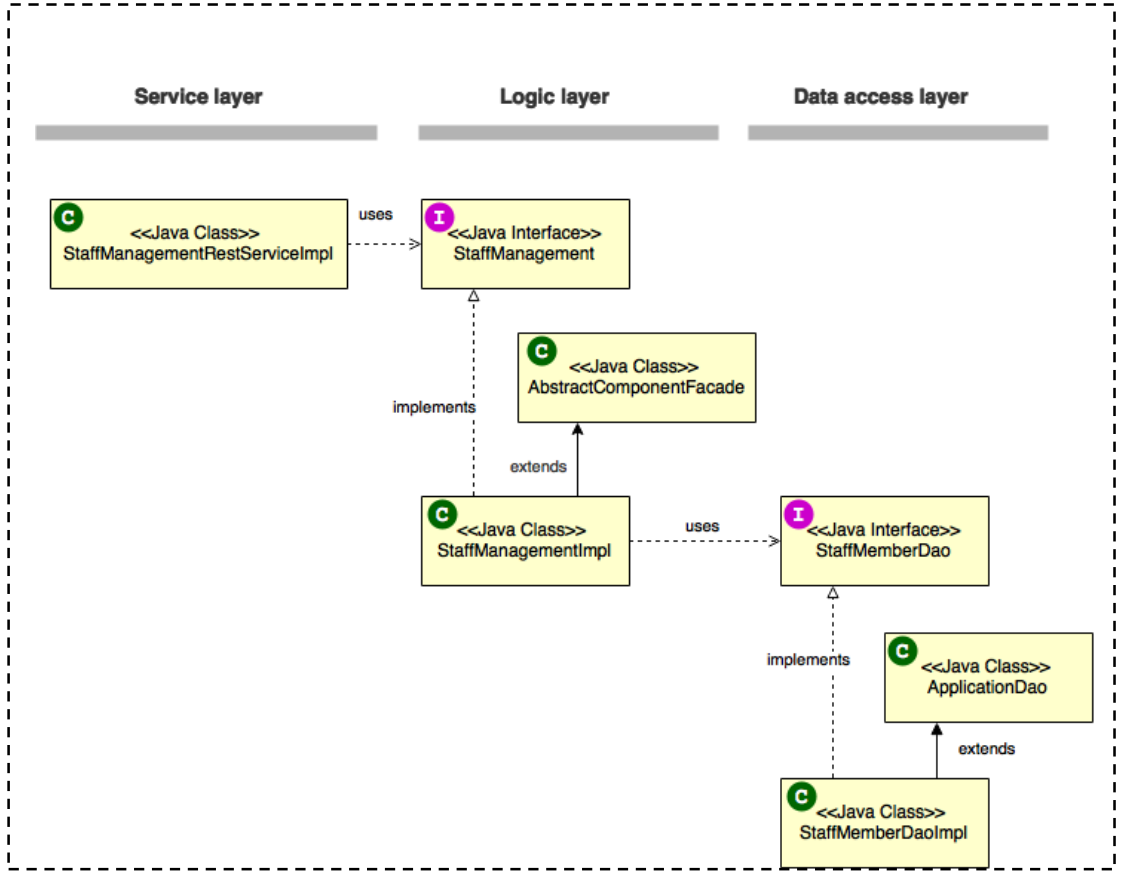
See: Domain Driven Desgin by Jimmy Bogard

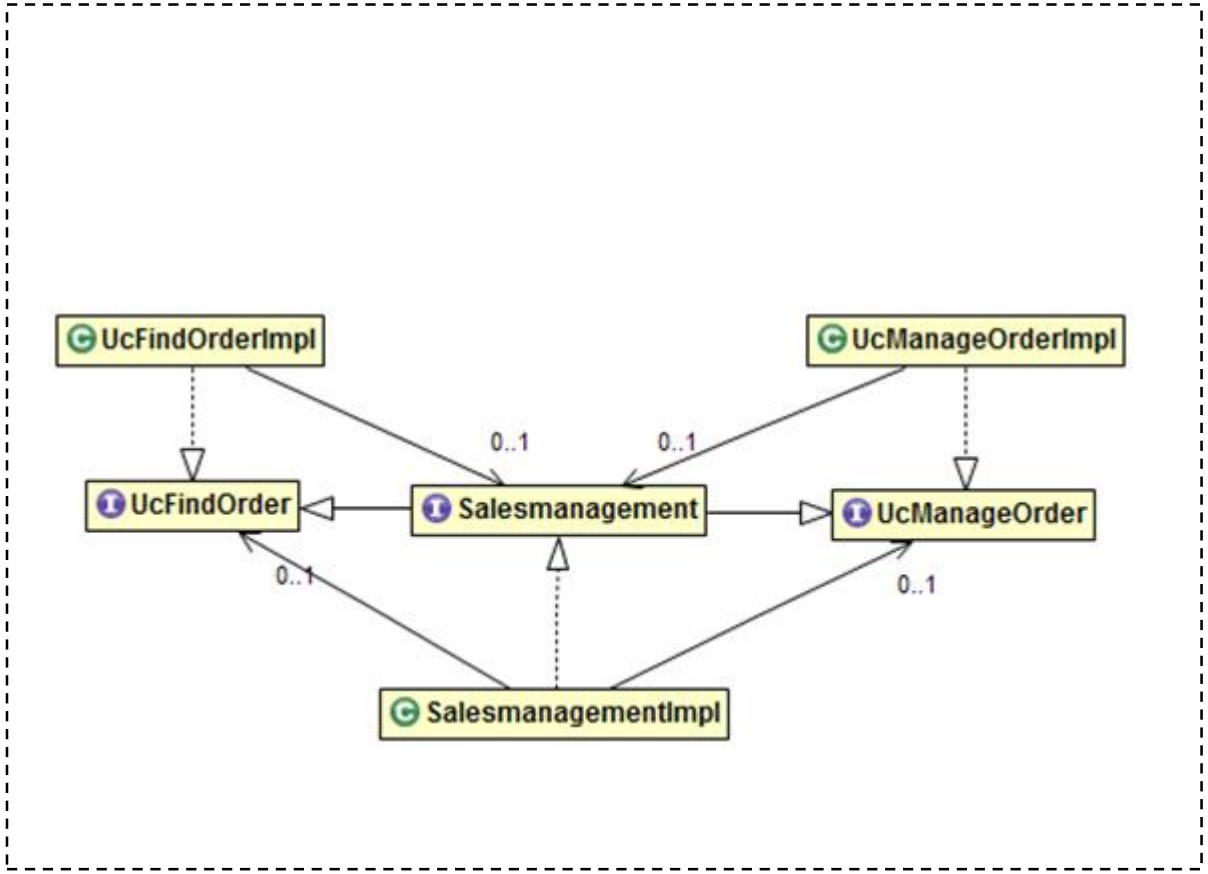
https://www.youtube.com/watch?v=U6CeaA-Phqo



Domain Driven Design: The Good Parts - Jimmy Bogard

devonfw as Code level architecture + for both Structure and functionality (DDD inspired)





https://github.com/devonfw/devon4j/wiki/

Next Steps for those in a hurry

https://www.infoq.com/articles/architecture-as-language-a-story/

Summary – Next Steps

Watch the videos

Read the **devonfw Architecture Guide**

Read the **C4 Model**

Install and play around with **PlantUML**

Try to model **Jump The Queue** with the **C4 Model**

Try to define the interfaces for the internal components

Rinse – Repeat

Read up on **UML** and try to play with it

Read up on DDD and try to play with it

Deeper…

The Language of the System – Rich Hickey https://www.youtube.com/watch?v=ROor6_NGIWU

Capgemini

People matter, results count.

About Capgemini

A global leader in consulting, technology services and digital transformation, Capgemini is at the forefront of innovation to address the entire breadth of clients' opportunities in the evolving world of cloud, digital and platforms. Building on its strong 50-year heritage and deep industry-specific expertise, Capgemini enables organizations to realize their business ambitions through an array of services from strategy to operations. Capgemini is driven by the conviction that the business value of technology comes from and through people. It is a multicultural company of 200,000 team members in over 40 countries. The Group reported 2016 global revenues of EUR 12.5 billion.

Learn more about us at

www.capgemini.com