

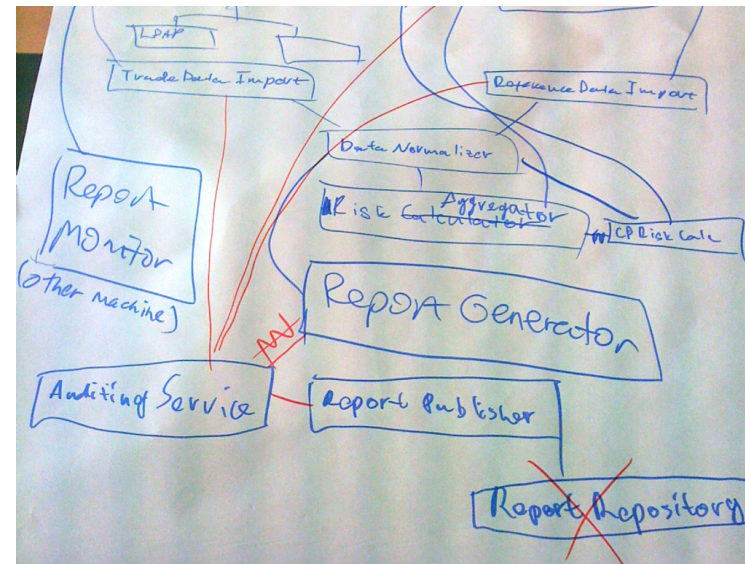
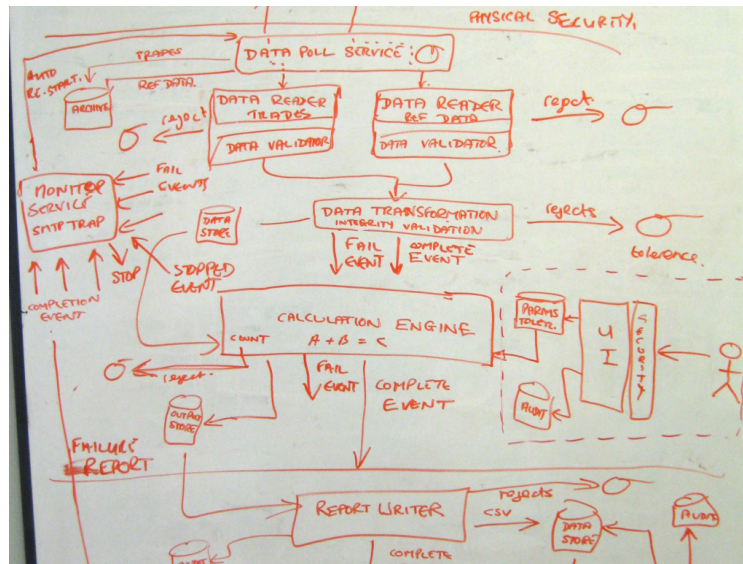
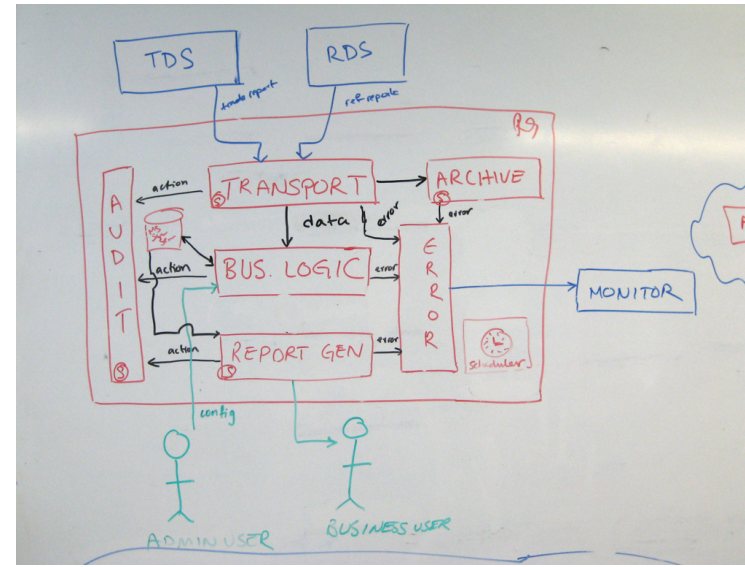
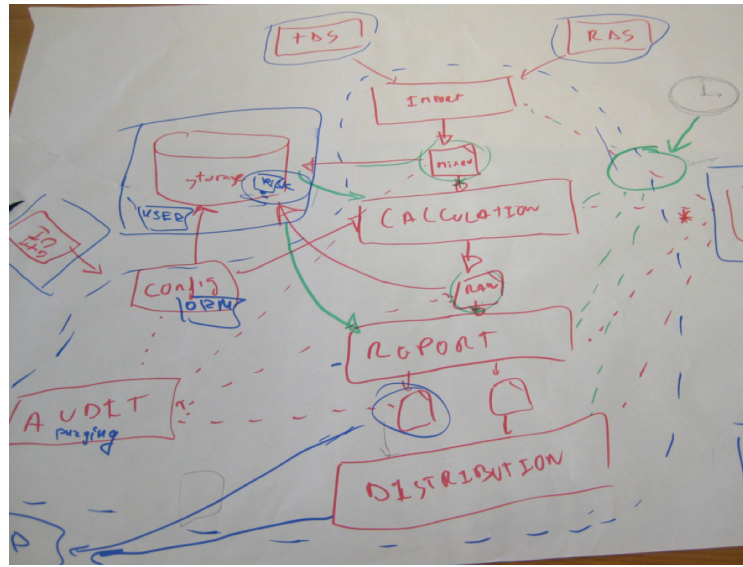
Diagram as Code and ADRs

Documenting the 'What' and 'Why'

Ioannis Skitsas

Patras Tech Talk 2026.01 | January 20, 2026



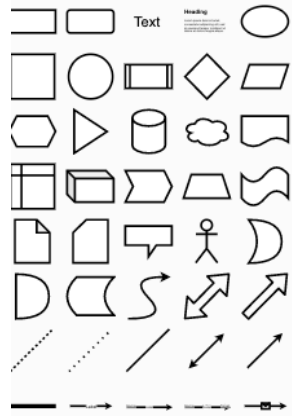


Q

? + ✎ ✕

Drag elements here

General



Misc

Advanced

Basic

Arrows

Flowchart

Entity Relation

UML

BPMN General

BPMN Gateways

BPMN Events

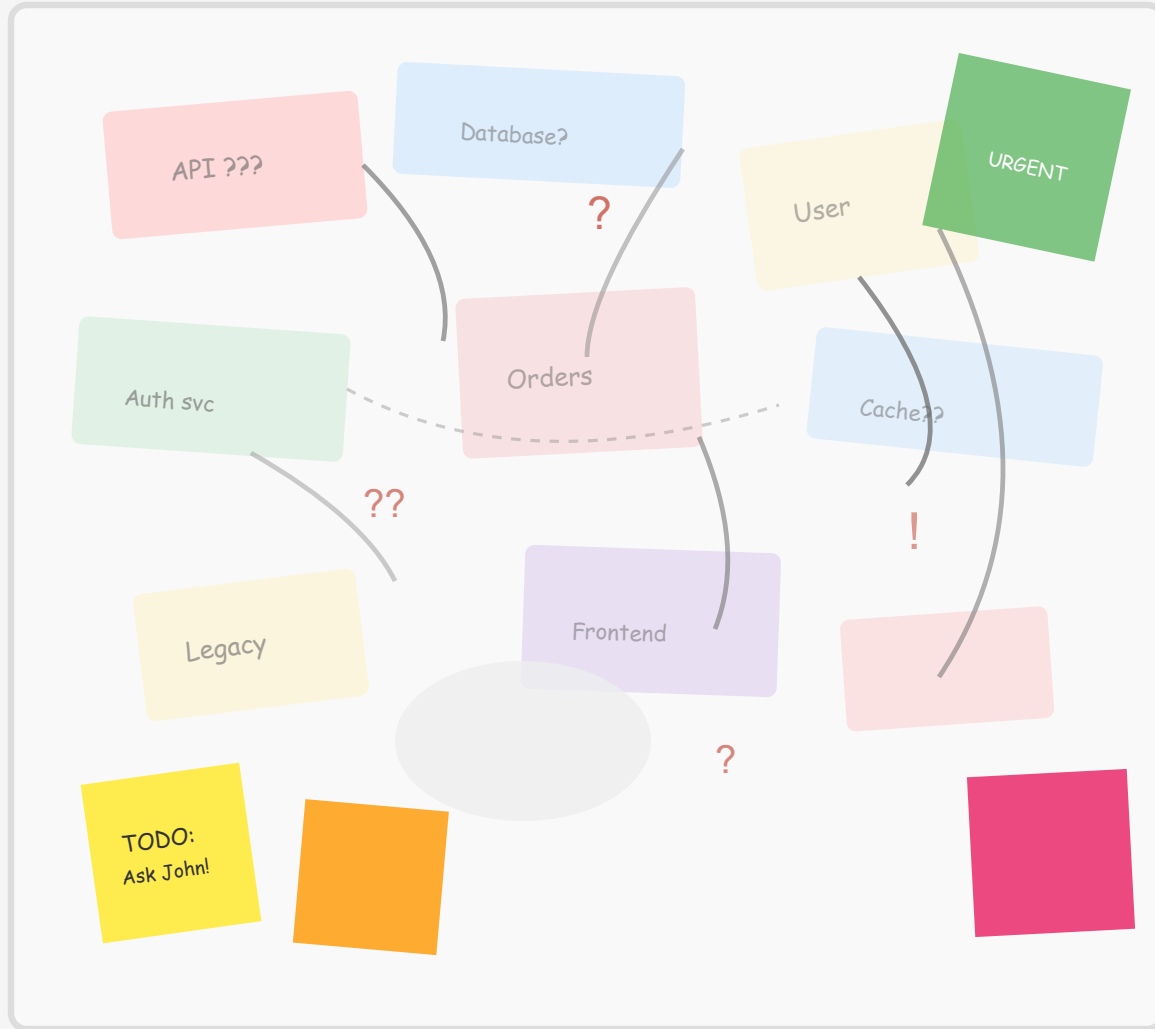
I STARTED THE DAY WITH
LOTS OF PROBLEMS.

BUT NOW, AFTER HOURS
AND HOURS OF WORK,

I HAVE LOTS OF PROBLEMS
IN A DIAGRAM

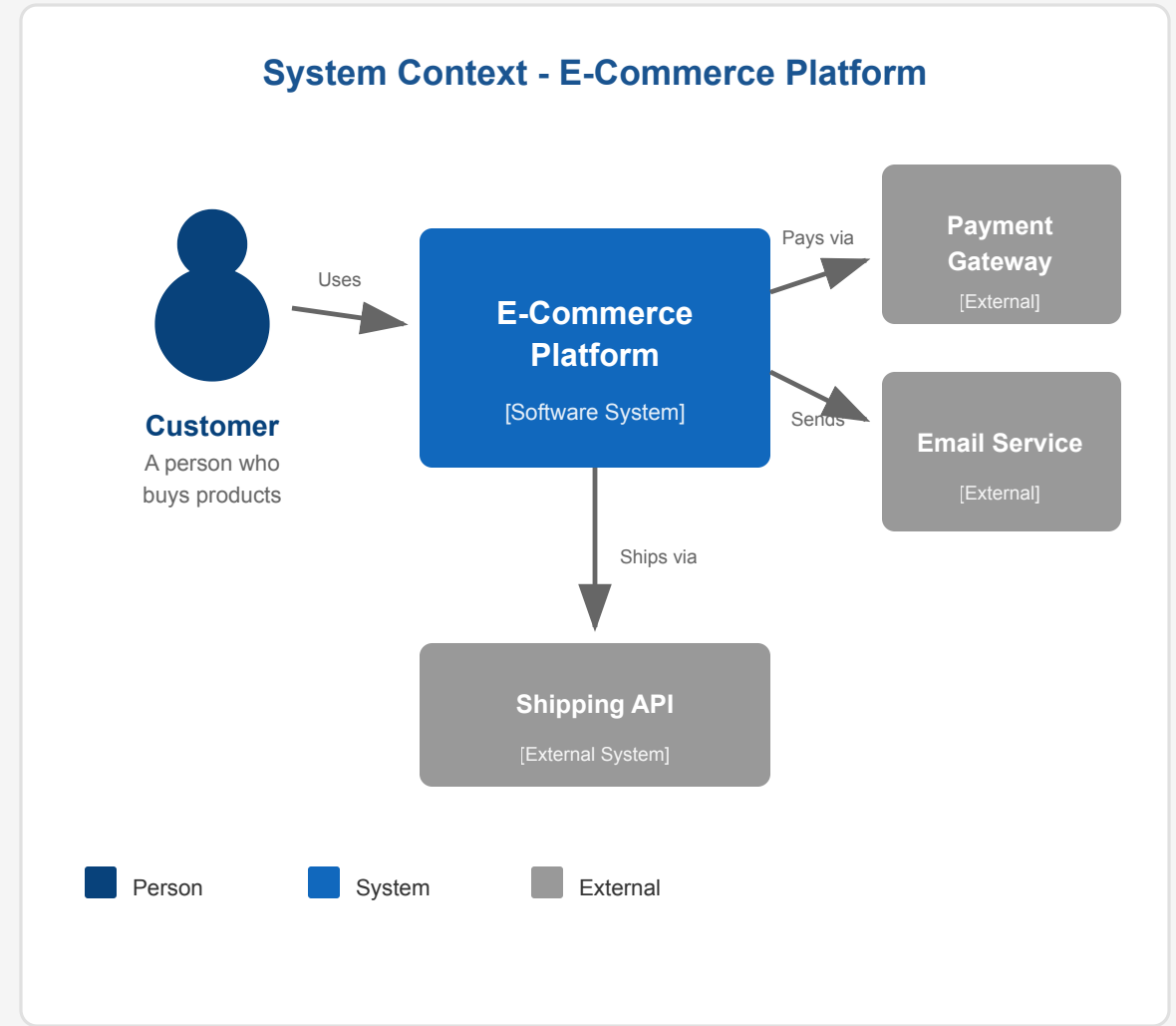


BEFORE



VS

AFTER



Why Diagrams Matter

Communication

- **Developers** need technical details
- **Ops** needs deployment architecture
- **Management** needs the big picture
- **New hires** need onboarding

The Problem

- Outdated the moment they're drawn
- Lost in Confluence/SharePoint
- Multiple versions, no source of truth
- Created in tools outside dev workflow

"A map gets outdated as roads change — what if your map updated itself?"

The Architecture Amnesia Problem

"Has anyone joined a project and spent days trying to figure out how things connect?"

- Scattered wiki pages
- Outdated Visio diagrams
- "Ask John, he knows"
- Tribal knowledge

The Solution

- Version-controlled diagrams
- Always up-to-date
- Self-documenting
- Searchable decisions

Diagram Tools Landscape

Category	Tools	Pros	Cons
Manual	Visio, Draw.io, Lucidchart, Miro	Easy, flexible	No version control
As Code	PlantUML, Mermaid, D2, Structurizr	Version controlled	Learning curve

Why "as Code"?

- **Version control** → Track changes, blame, history
- **Review process** → PRs for architecture changes
- **Automation** → Generate from code, CI/CD integration
- **Consistency** → Same notation everywhere

Introducing C4

Created by **Simon Brown**

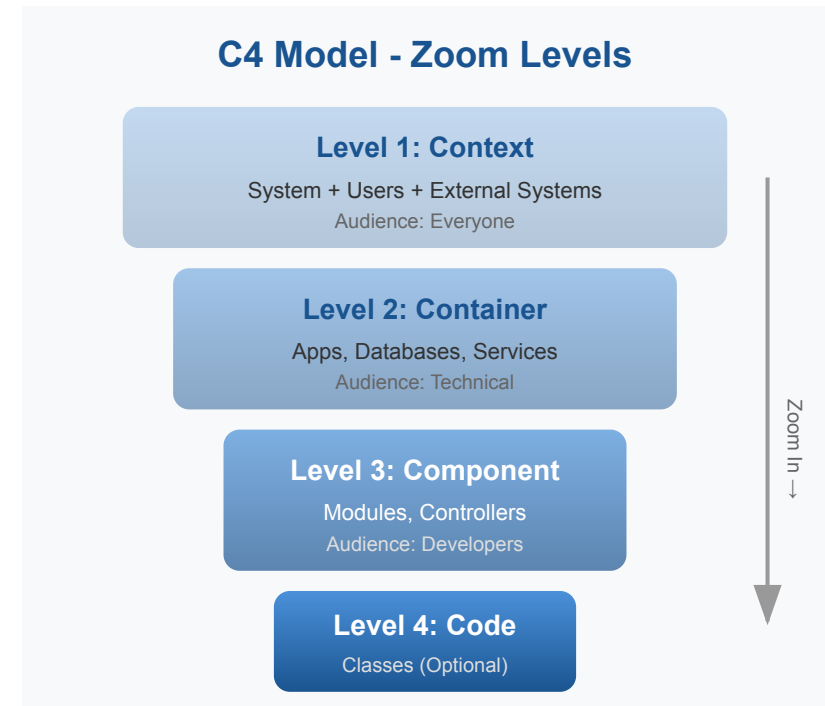
The 4 Levels

1. **C**ontext
2. **C**ontainer
3. **C**omponent
4. **C**ode

Think Google Maps

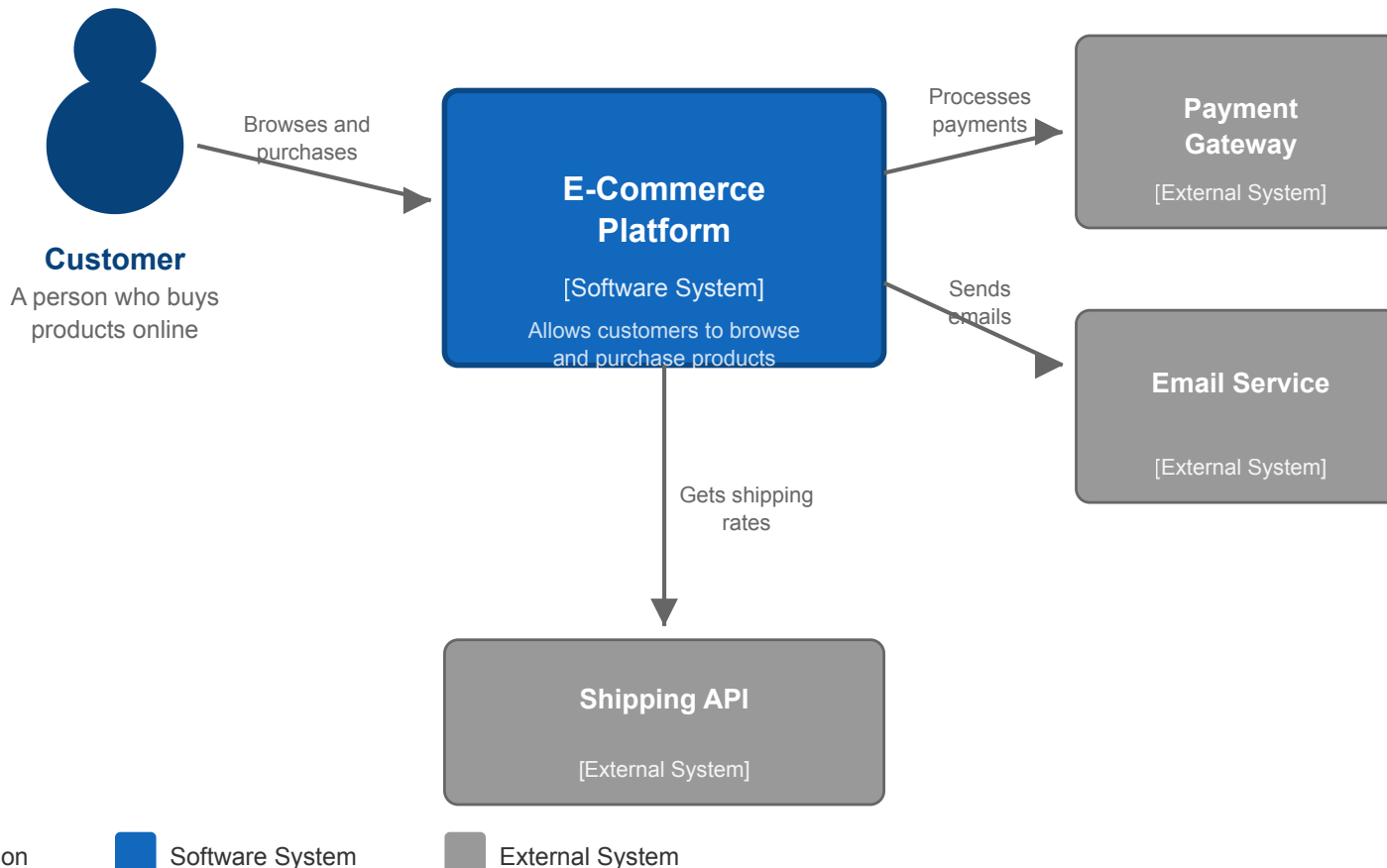
- Zoom out → Country level
- Zoom in → Street level

Same for architecture!



Level 1: System Context

System Context Diagram - E-Commerce



Audience:

Everyone (including non-technical)

Shows:

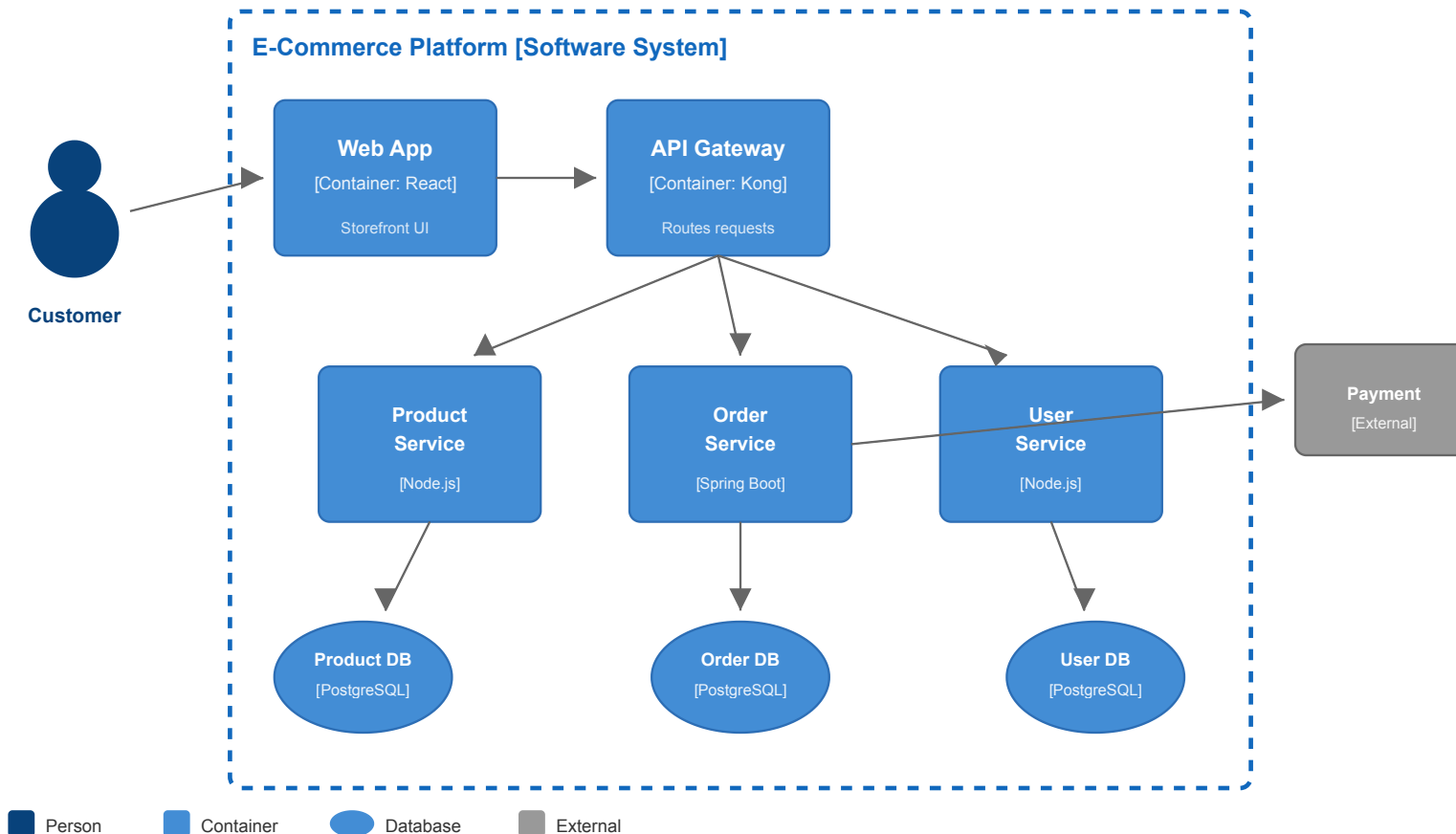
System as a black box + users + external systems

Answers:

"What is this and who uses it?"

Level 2: Container Diagram

Container Diagram - E-Commerce Platform



Audience:

Technical people (devs, architects, ops)

Shows:

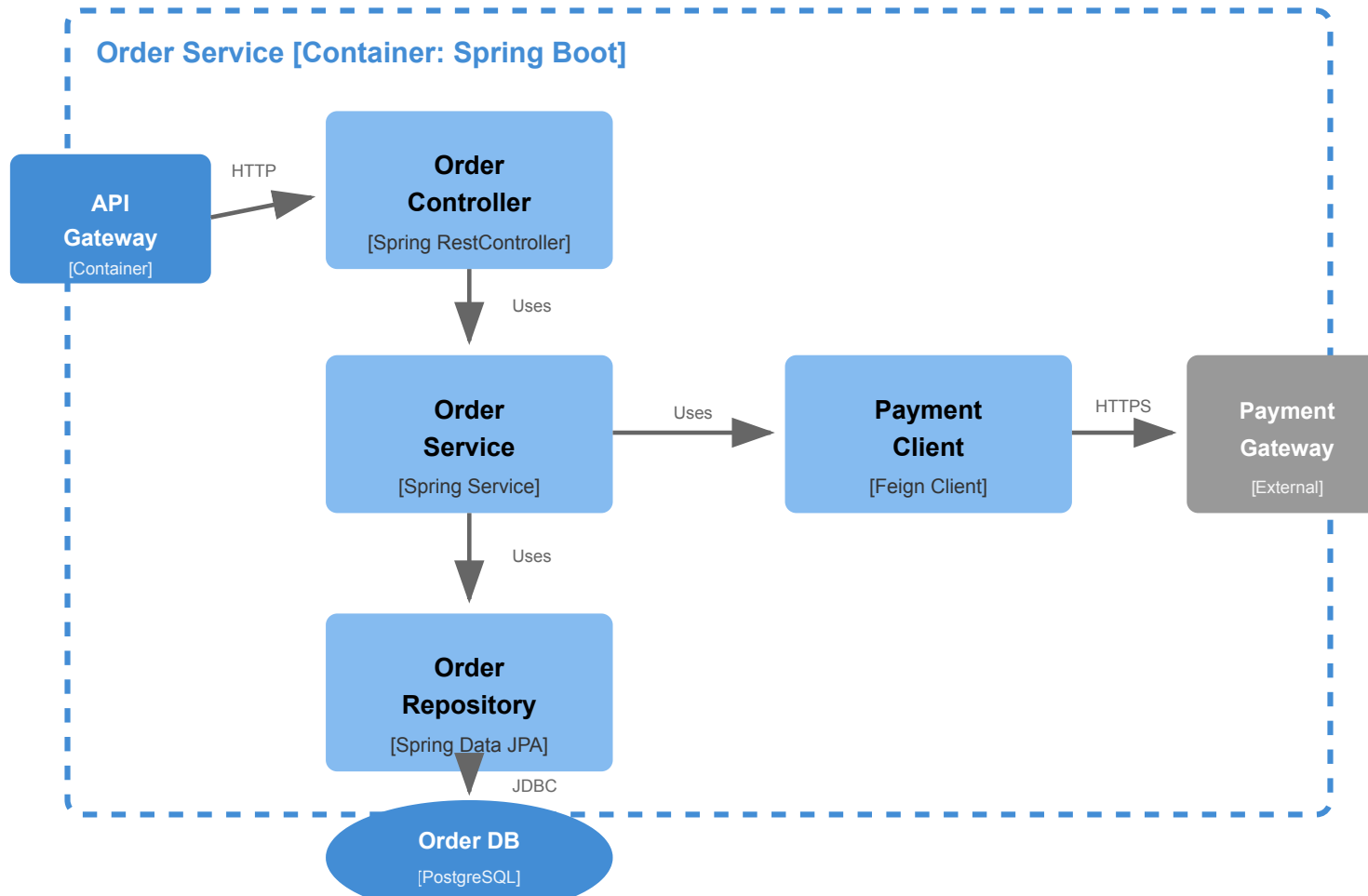
Applications, databases, message queues

Answers:

"What are the high-level technology decisions?"

Level 3: Component Diagram

Component Diagram - Order Service



Audience:

Developers working on that container

Shows:

Internal modules, services, controllers

Answers:

"How is this container organized?"

Level 4: Code (Optional)

Usually auto-generated from code

- UML class diagrams
- Entity-relationship diagrams
- Often skipped — code IS the truth at this level

C4 Key Principles

1. **Hierarchical abstraction** — Zoom in for detail
2. **Simple notation** — Boxes and arrows, not UML symbols
3. **Self-describing** — Every element has name + description
4. **Technology agnostic** — Focus on concepts

Documentation Maturity Model [1]

Software architecture diagramming maturity model

Level 1	Level 2	Level 3	Level 4	Level 5
Initial No software architecture diagrams.	Ad hoc Software architecture diagrams with ad hoc abstractions and notation, in a general purpose diagramming tool.	Defined Software architecture diagrams with defined abstractions and notation, in a general purpose diagramming tool.	Modelled Software architecture diagrams with defined abstractions and notation, in a modelling tool, authored manually.	Optimising <ul style="list-style-type: none">- Model elements are shared between teams across the organisation.- Models are used as queryable datasets.- Automatic generation of model elements from source code, deployment environment, logs, etc.- etc

..... C4 model

What is Structurizr?

Created by **Simon Brown** (same person who created C4)

Features

- DSL for defining architecture
- **Model once, view many**
- Multiple output formats
- Version controllable

Options

- **Structurizr Lite** (free, Docker)
- **Structurizr Cloud** (SaaS)
- **On-premises** (enterprise)

"Instead of drawing boxes, you describe relationships"

Structurizr DSL Example

```
workspace {  
  model {  
    user = person "User" "A customer"  
  
    system = softwareSystem "My System" {  
      webApp = container "Web App" "Frontend" "React"  
      api = container "API" "Backend" "Spring Boot"  
      db = container "Database" "Storage" "PostgreSQL"  
    }  
  
    user -> webApp "Uses"  
    webApp -> api "Calls" "REST"  
    api -> db "Reads/Writes"  
  }  
  
  views {  
    systemContext system "Context" { include * }  
    container system "Containers" { include * }  
  }  
}
```

Live Demo

Let's see Structurizr in action!

```
docker run -it --rm -p 8080:8080 \
  -v $(pwd):/usr/local/structurizr \
  structurizr/lite
```

<http://localhost:8080>

The Missing Piece

C4 and Structurizr tell us **WHAT** we built...

But there's a crucial question they don't answer:

WHY did we make those decisions?

- *"Why didn't we use Kafka instead of RabbitMQ?"*
- *"Why is this a monolith and not microservices?"*
- *"Who decided to use React?"*

Introducing ADRs

Architectural **D**ecision **R**ecords

What

- Short, focused documents
- One decision per document
- Immutable once accepted
- Stored with your code

Why

- Prevent re-debating decisions
- Help future team members
- Create accountability
- Document trade-offs

"An ADR is like a git commit message for architecture"

ADR Template

ADR-001: Use PostgreSQL for primary database

Status

Accepted

Context



We need a relational database for our e-commerce platform. We evaluated PostgreSQL, MySQL, and MongoDB.

Decision

We will use PostgreSQL because:

- Strong ACID compliance for financial transactions
- Better JSON support than MySQL
- Team has existing expertise

Consequences

-  Reliable transactions for payments
-  Team needs training on advanced features

When to Write an ADR?

Write ADR for:

- Technology choices
- Architectural patterns
- Standards that affect architecture
- Hard-to-reverse decisions

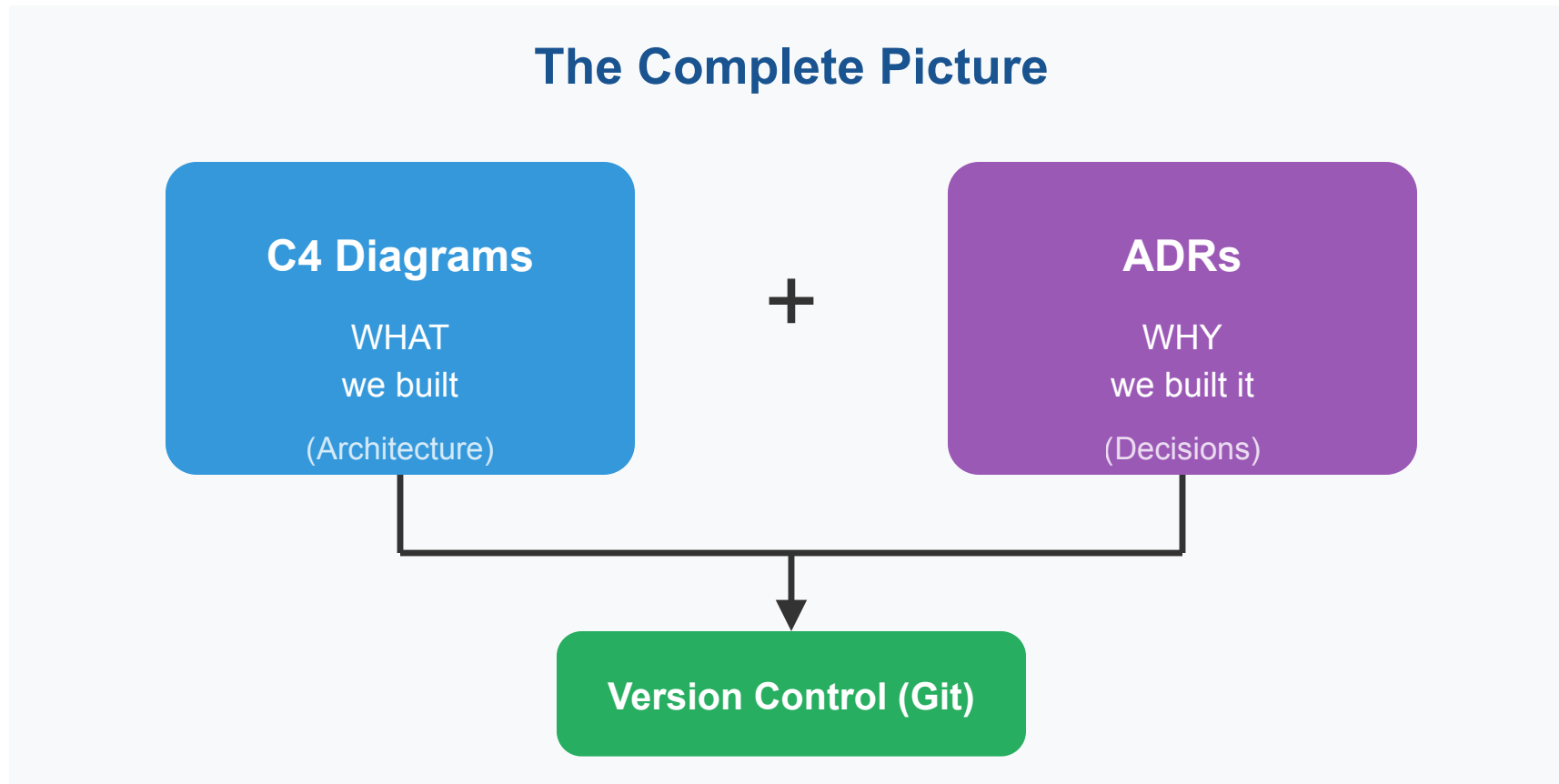
Skip ADR for:

- Easily reversible choices
- Implementation details
- Personal preferences
- One-time decisions

Tools

- **adr-tools** — CLI for managing ADRs
- **Log4brains** — Web UI for browsing
- **Just markdown files** — Simple and effective!

Bringing It Together



The Workflow: PR includes updated DSL + new ADR → Team reviews both → Merge
= documented forever

Start Small

Today, you can:

1. Create **ONE** context diagram of your main system
2. Write **ONE** ADR for a recent decision
3. Put both in your repo

"You don't need to boil the ocean"

Resources

Topic	Link
C4 Model	c4model.com
Structurizr	structurizr.com
ADR GitHub	adr.github.io
This presentation	github.com/patras-tech-talk/presentations

Software Architecture for Developers by Simon Brown

Behind the Scenes

This presentation was created with Claude (AI)

What I did:

- Set up this `presentations` repo
- Created folder structure for all 24 meetups
- Generated this entire slide deck

Total time: ~2 hours (prompting, reviewing, fixing)

Tools used:

- **Claude Code** (CLI) - AI pair programming
- **Structurizr** - C4 diagrams
- **Marp** - Presentations as code!

The prompt:

"I want to do a 30-40 min presentation about diagrams as code, C4, Structurizr, and ADRs..."

Questions?

Thank you!

Ioannis Skitsas

Patras Tech Talk 2026.01

Slides and examples available on [GitHub](#)