Advanced Refactor Using DDD

https://github.com/BrewUp/DDD-Europe-2025

Not THE solution, but A solution

Disclaimer

We're sharing what worked for us, not a universal approach

Our Context

Solutions adapt to team size, tech stack, and business domain

Your Mileage

Take principles, adapt implementation to your needs

What define a legacy system?

Are you sure it is not just old?

Redefining "Legacy"

Not Just Age

A 20-year system can be healthy

A 2-year system can be legacy

It's About Health

Measures adaptability, not birthdate

Reflects organizational knowledge

Key Indicators

Fear of changes

Unknown dependencies

Tribal knowledge

Why do we touch it anyway?

Because duct tape and prayers stopped working

Why do we touch it anyway?

It’s business-critical
Because if it goes down, so does half the company…

It’s blocking change
Because trying to build around it feels like coding in a minefield

It’s too risky to ignore
It’s not a ticking time bomb — it’s already ticking *and* smoking

It’s costly to maintain
Because Karen from finance won’t stop emailing us

It holds valuable data or logic
Because it knows things… ancient, forbidden things no one documented

Modernization effort
We called it “phase one” in 2019. We still do…

New features
Because the shiny new microservice can’t do anything without this dinosaur.

Legacy is not when it is old, but when no one dares to change it anymore.

Awful monolith

Show me the code

Exercise

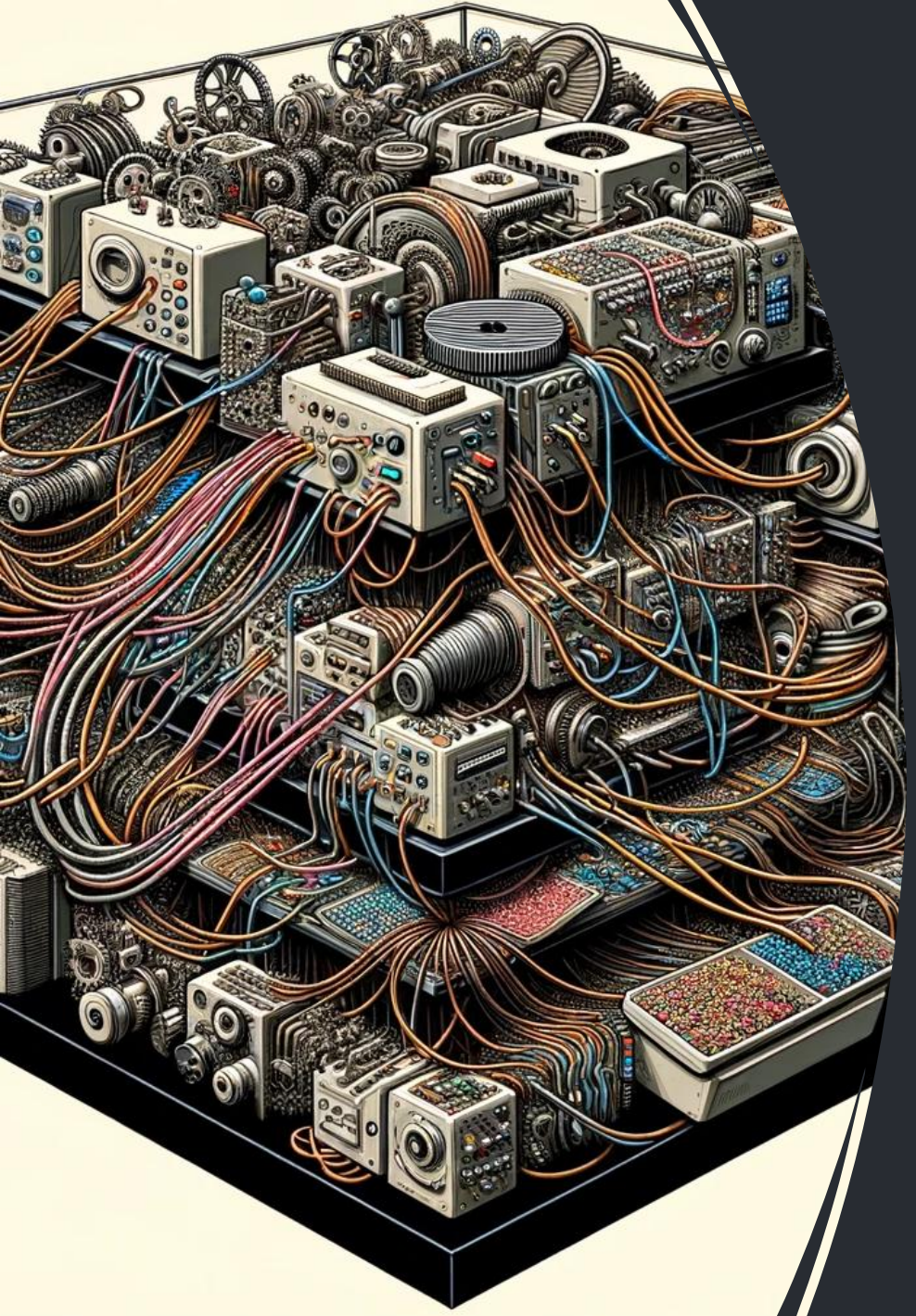
What is wrong with branch-01?

“Developers are drawn to complexity, like moths to a flame, often with the same outcome”

Neal Ford

Big ball of mud

Lack of a clear architecture, leading to a system that is haphazardly structured and difficult to understand or modify.

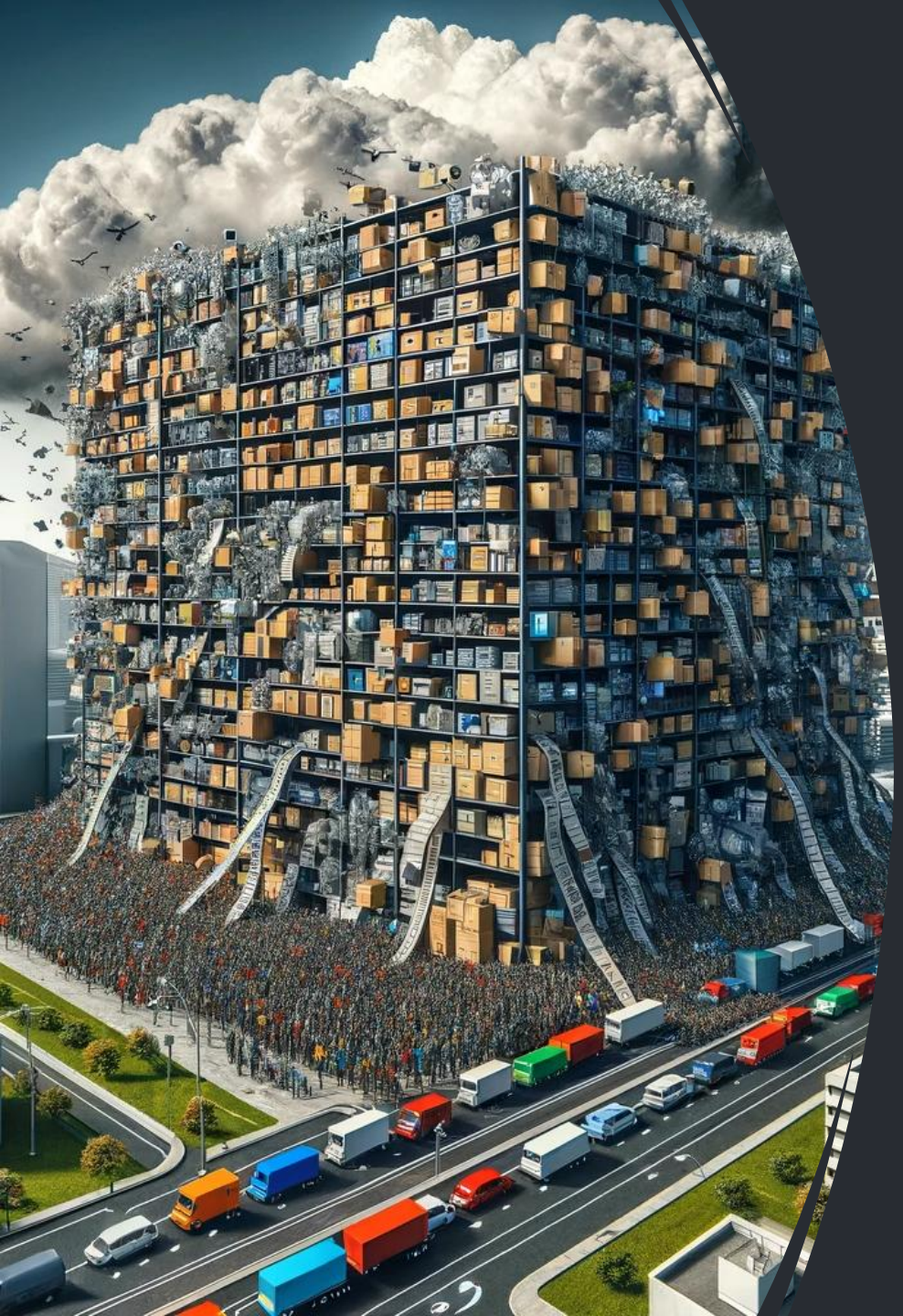


Lack of modularity

Failure to properly encapsulate different functionalities, leading to a system where changes in one module ripple through others.

Shared data model

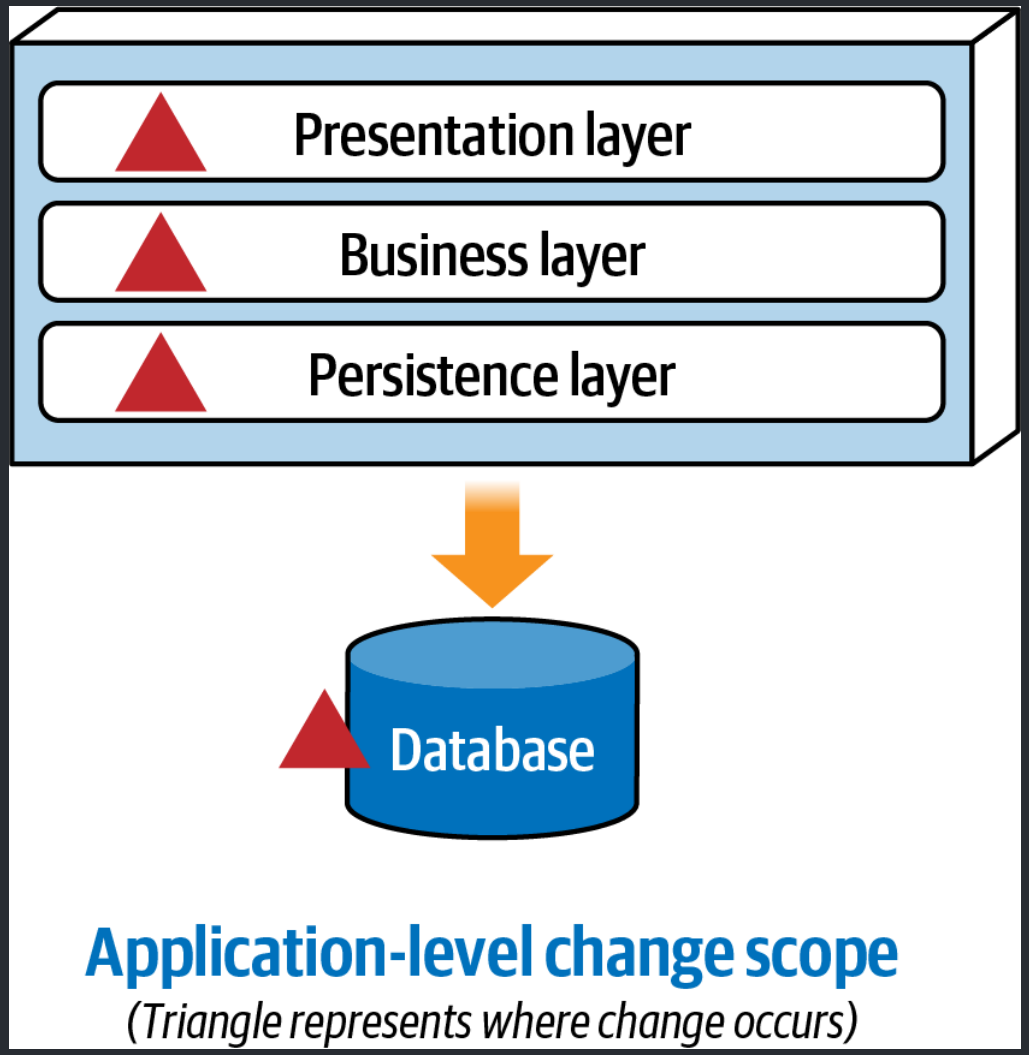
Different modules or functionalities directly accessing and modifying a shared data model, leading to high risk of data corruption and conflicts.



Monolithic database design

A single database for all application needs, creating a bottleneck and complicating any attempt to scale or separate concerns.

What is the impact of changes?



Exercise

Let’s start refactoring (branch-02)

Other things we can do?

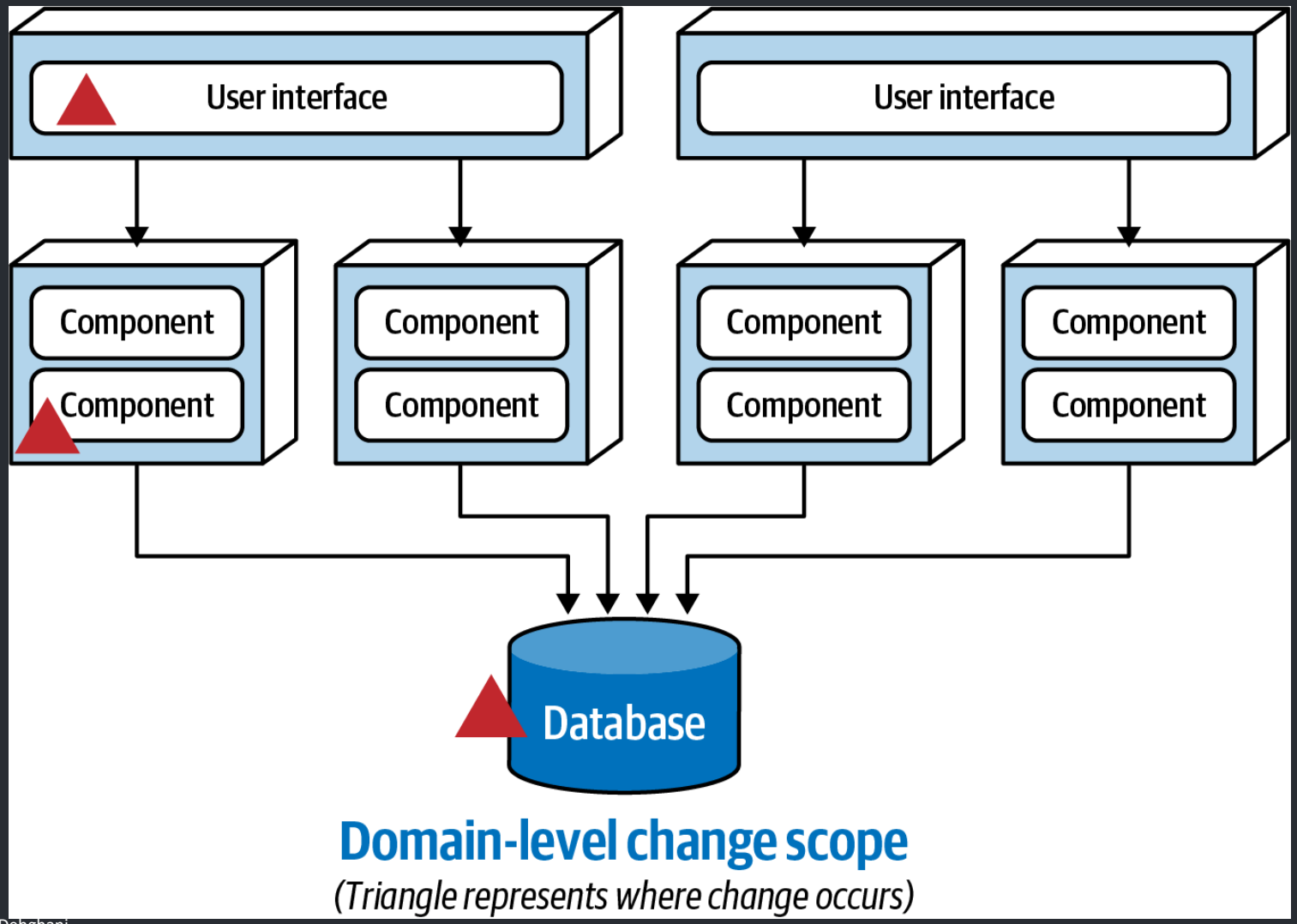
- A DB for every module: Autonomy vs Consistency
- Mediator pattern: Decoupling vs Verbosity

Disclaimer

Every solution has a price.

Don't pair up with the first CQRS that comes along.

What is the impact of changes?



Exercise

How do you protect yourself?

Testing strategy: E2E

- Validate external behavior
- Same payloads, same outcomes

Testing Strategy: Fitness Functions

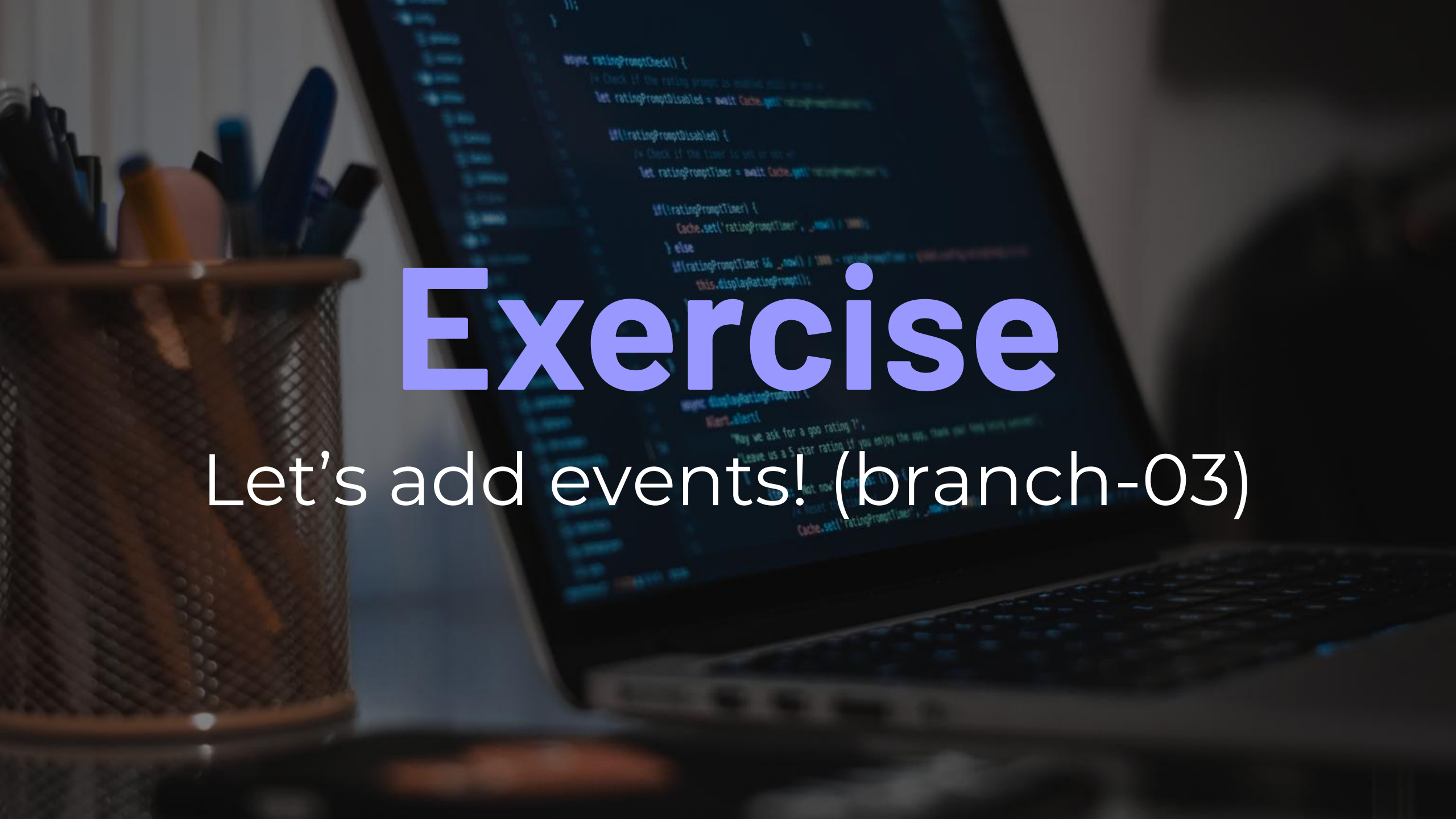
- Assert architectural rules
- Protect module boundaries

“Any mechanism that performs an objective integrity assessment of some architecture characteristics or combination of architecture characteristics.”
Building Evolutionary Architectures

Testing Strategy: Unit Tests

- Pyramid structure
- Fast feedback for domain logic

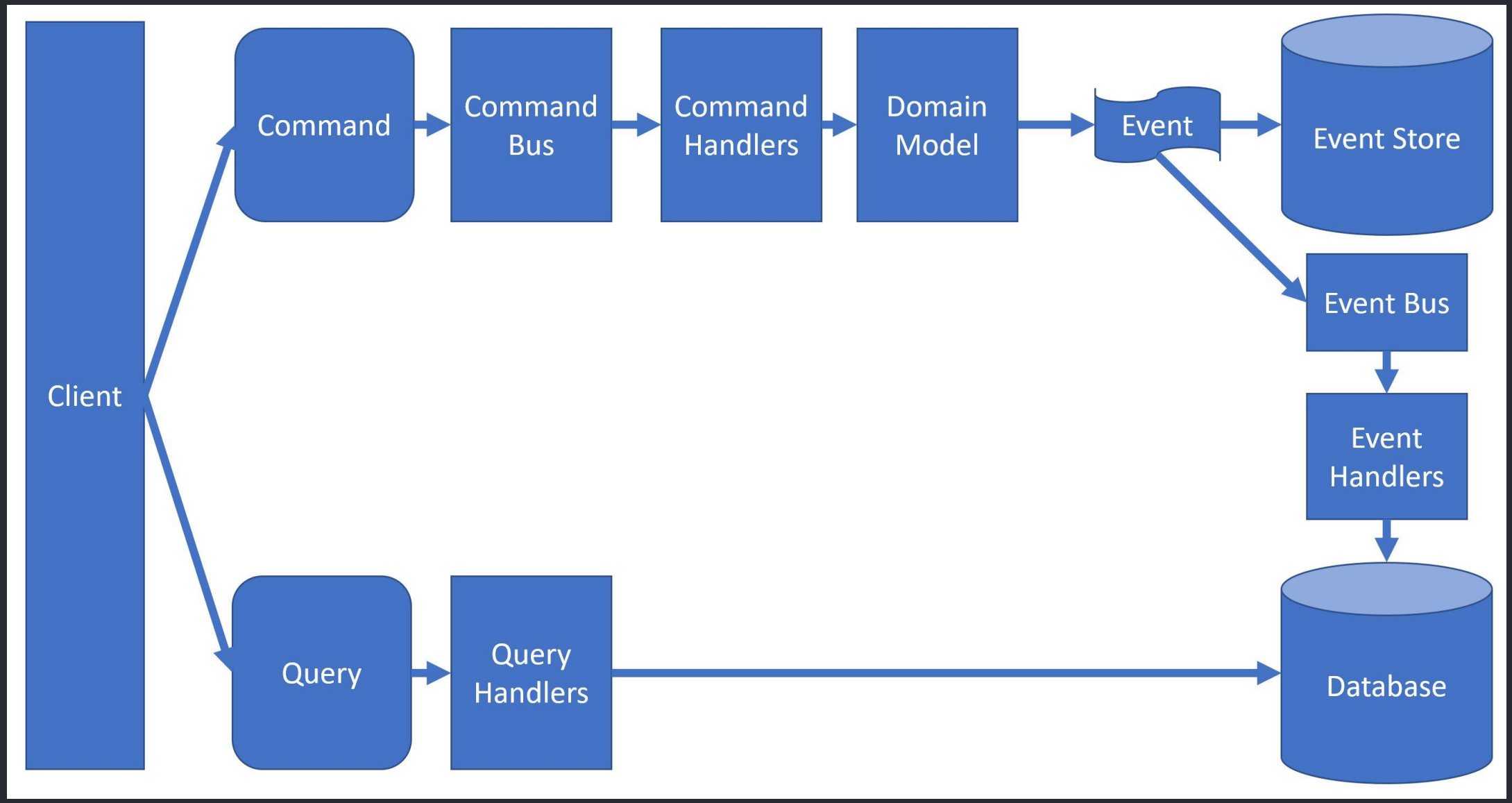
Need More Scalability? How do you achieve that without microservices?



Exercise

Let’s add events! (branch-03)

CQRS+ES


Client
Command
Command Bus
Command Handlers
Domain Model
Event
Event Store
Event Bus
Event Handlers
Query
Query Handlers
Database

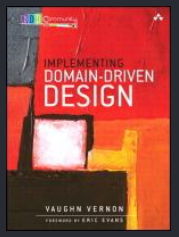
Given, When, Then

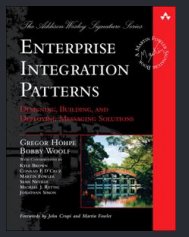
- The essential idea is to divide a scenario into three sections:
- **Given**: describes the state of the aggregate before sending the command. A kind of prerequisite of the test.
- **When**: represents the command to be sent to the aggregate.
- **Then/Expect**: describes the state changes expected as a result of the command.

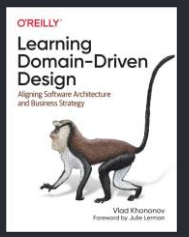
“Software architecture it's abstract by nature and we must ground it with some implementation details to make it concrete”

Bibliography

- Implementing Domain-Driven Design
- The Software Architect Elevator
- Learning Domain-Driven Design: Aligning Software Architecture and Business Strategy
- Hands-On Domain-Driven Design with .NET Core
- Software Architecture: The Hard Parts

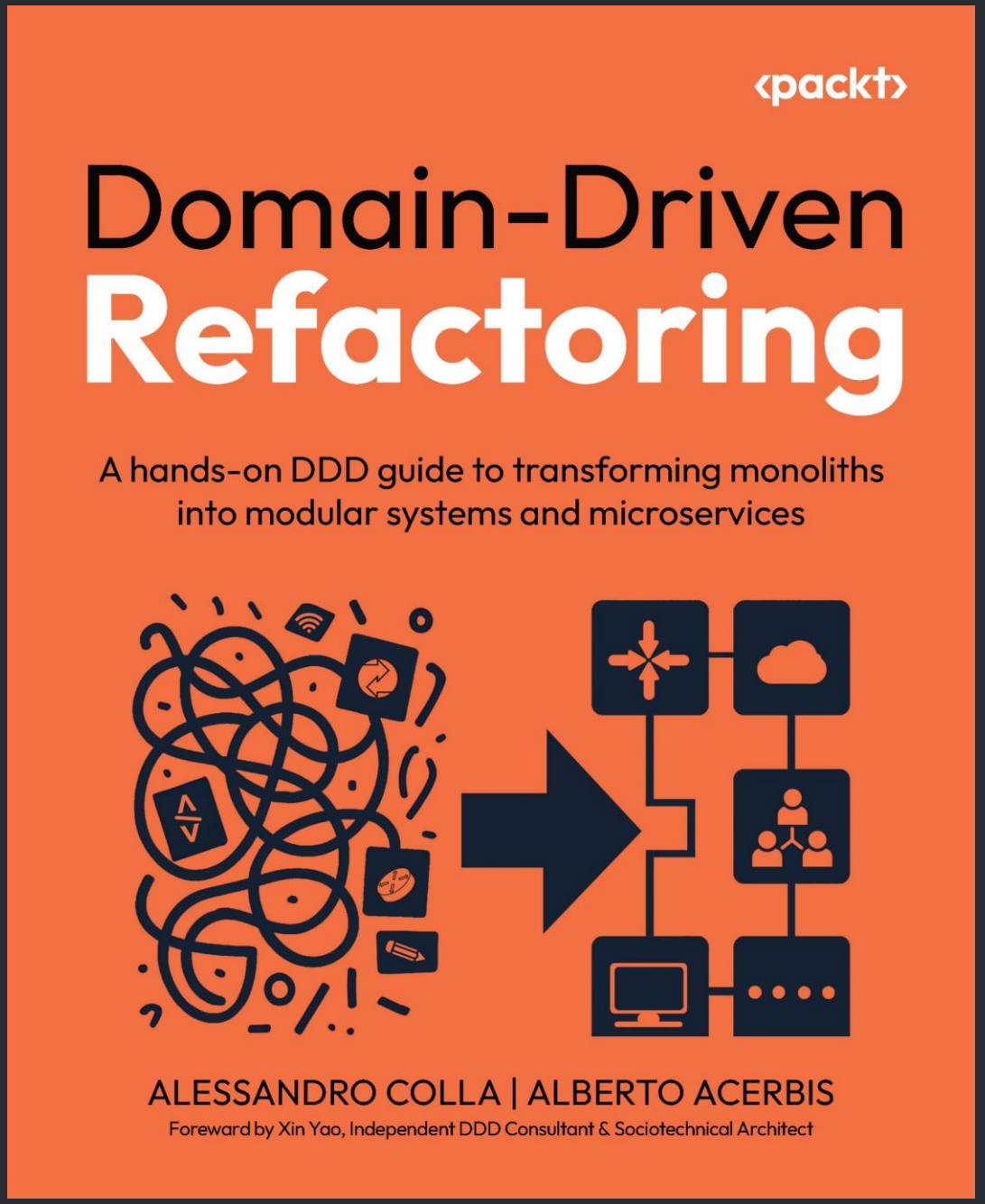






This workshop was based on the concepts explained in our recently released book

Use ***DDD20*** for a 20% discount during the conference days



Thank you!
We hope you learned something while having fun

linkedin.com/in/aacerbis/
alberto.acerbis@intre.it

linkedin.com/in/alessandrocolla
alessandro.colla@evoluzione.agency