On Great Software Architecture Design

CodeArtify

Hi, my Name is Olly

Backend- and Frontend Software Engineer (mainly Spring Boot/Angular, framework/language agnostic)

Software Craftsman

Technical Agile Coach @ Swisscom

Technical Trainer @ CodeArtify, Letsboot.ch, Swisscom

Tech Host & Event Organizer @ techexcellence.io

eLearning and Online Content Creator @ Swisscom, O'Reilly

Professional Ranter @ linkedin.com

Problems in Architecting Software

In your entire career, what problems have you encountered with software architecture, structure, and design?

… what’s the cause of these problems?

Detrimental Problems

- Splitting an Application according to
  - frameworks and technologies
  - high-level layers (FE, BE, DB, …)
  - silo boundaries (Org A vs. Org B)

… and assigning a number of teams to each partition accordingly.

What are causes?

- Hierarchical, top-down, waterfall, cmd&ctrl style management without team autonomy.
- Missing business involvement and unclear business requirements leading to aimless development.
- Emphasis on hiring devs for framework proficiency over fundamental software engineering skills.
- Lack of design and refactoring skills and interest among developers.
- These same developers become ivory tower architects later.
- …

Result

- Software that fails to meet business needs effectively.
- Increased costs and delayed delivery times.
- Big ball of mud software that cannot be changed anymore.
  - With Microservices: distributed BBoM

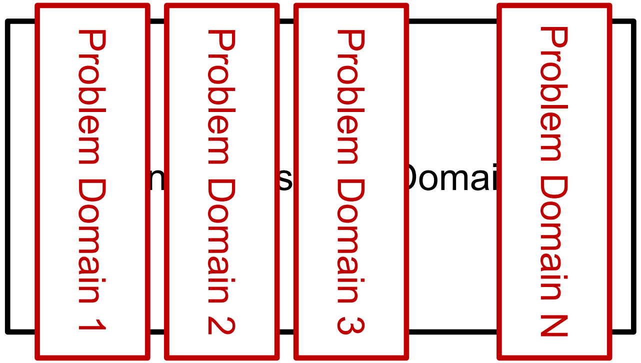
Seems like we need a different approach to software engineering…

How?

Close Collaboration!

Business & Dev Teams work closely together!

- Collaborative Tools like EventStorming can help
- Foster a shared mental model!
- Devs become quasi domain experts
- From that shared mental model, business use cases can be derived to achieve a desired business outcome!
- Dev team and business representatives become a **vertical business problem solving entity** (no feature factory anymore!)



Focus on **business domain problems**, not technical details!

From shared mental model to user stories

- Each **user story** delivers a tiny bit of value or outcome
    - As a **student**, I want to **enroll for courses**, so that I can **receive my degree**
    - Value focused: In order to **receive my degree**, as a **student**, I need to be able **to enroll for courses**
- **Vertical Slice through Business Domain!**
    - No split into BE, FE, etc. anymore
- After every story: user needs to be able to do something they were not able to do before!
- Small, incremental, iterative
- Business always gets something for their money and can change their mind
- Agile: responding to chang(-ing requirements)

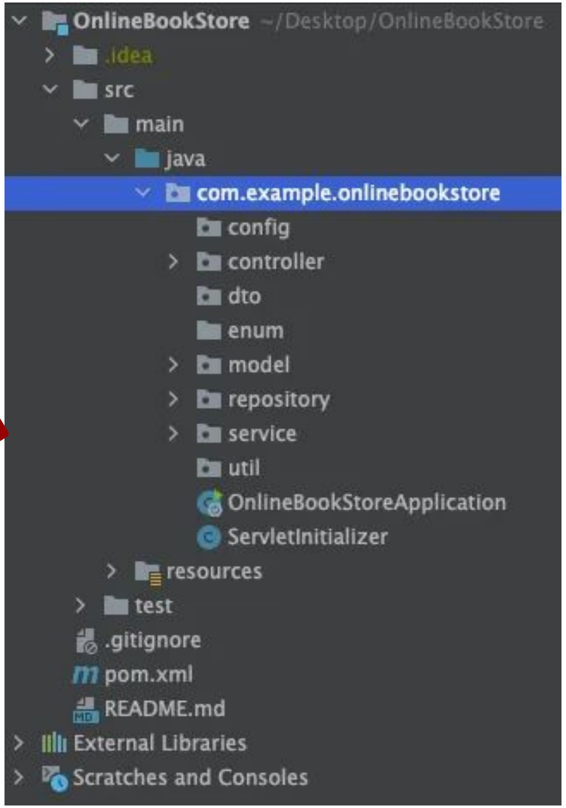
How to Architect Software then?

Ready for Change from a Business Perspective

Typical Spring Boot Application Structure

what is this application about?

echnical Structure Missing Business Focus

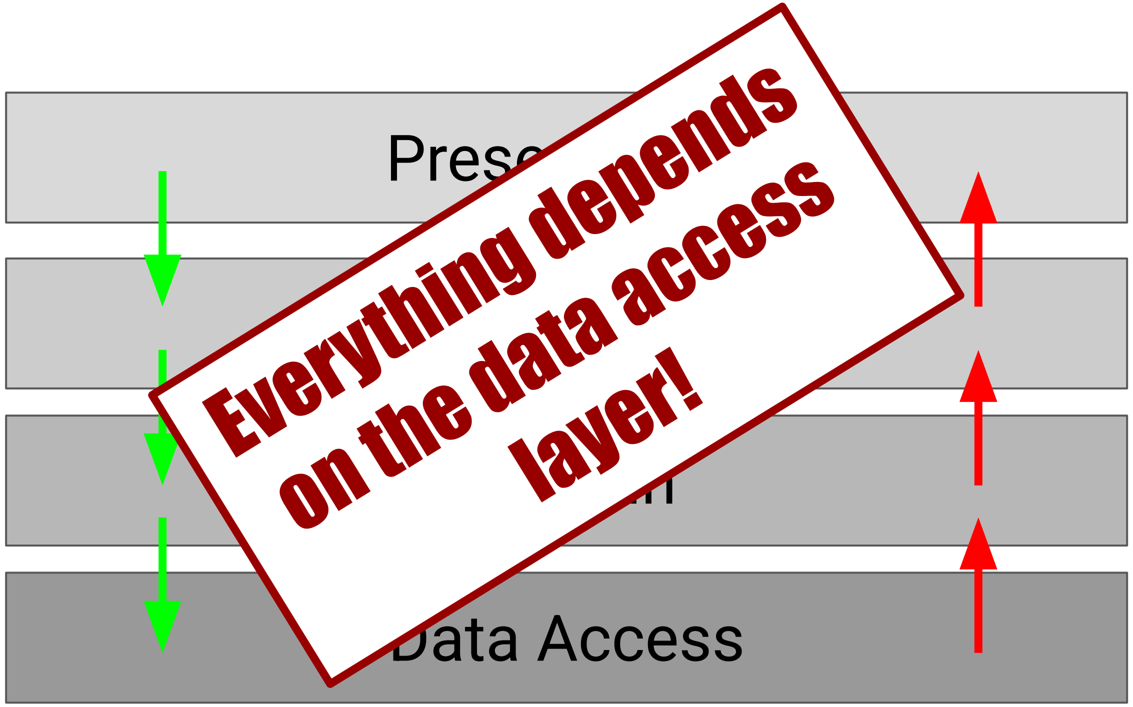


```
OnlineBookStore ~/Desktop/OnlineBookStore
  .idea
  src
    main
      java
        com.example.onlinebookstore
          config
          controller
          dto
          enum
          model
          repository
          service
          util
          OnlineBookStoreApplication
          ServletInitializer
      resources
    test
  .gitignore
  pom.xml
  README.md
External Libraries
Scratches and Consoles
```

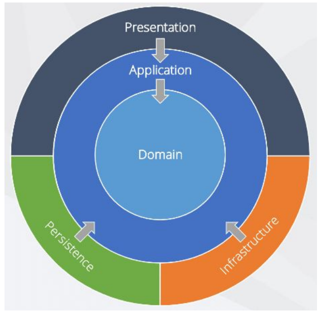
what are it’s capabilities?

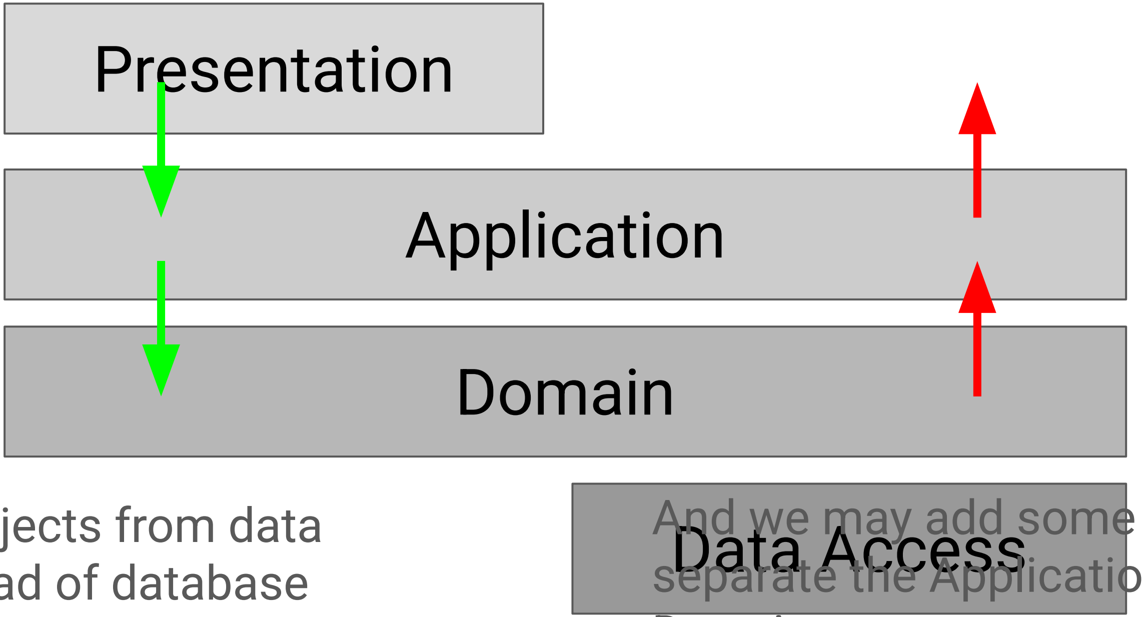
Where do I have to make changes to the “order book” use case?

Domain-Focused Layered Architecture



Inverted Dependency: Onion Architecture

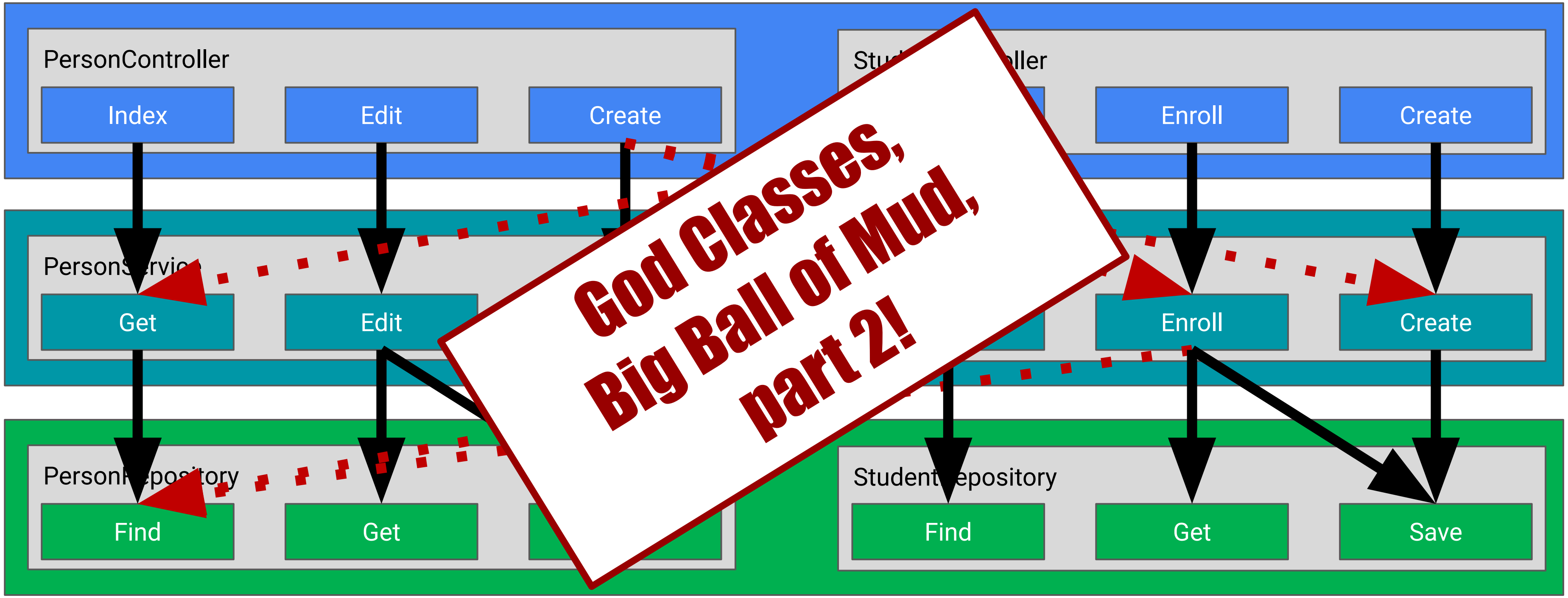




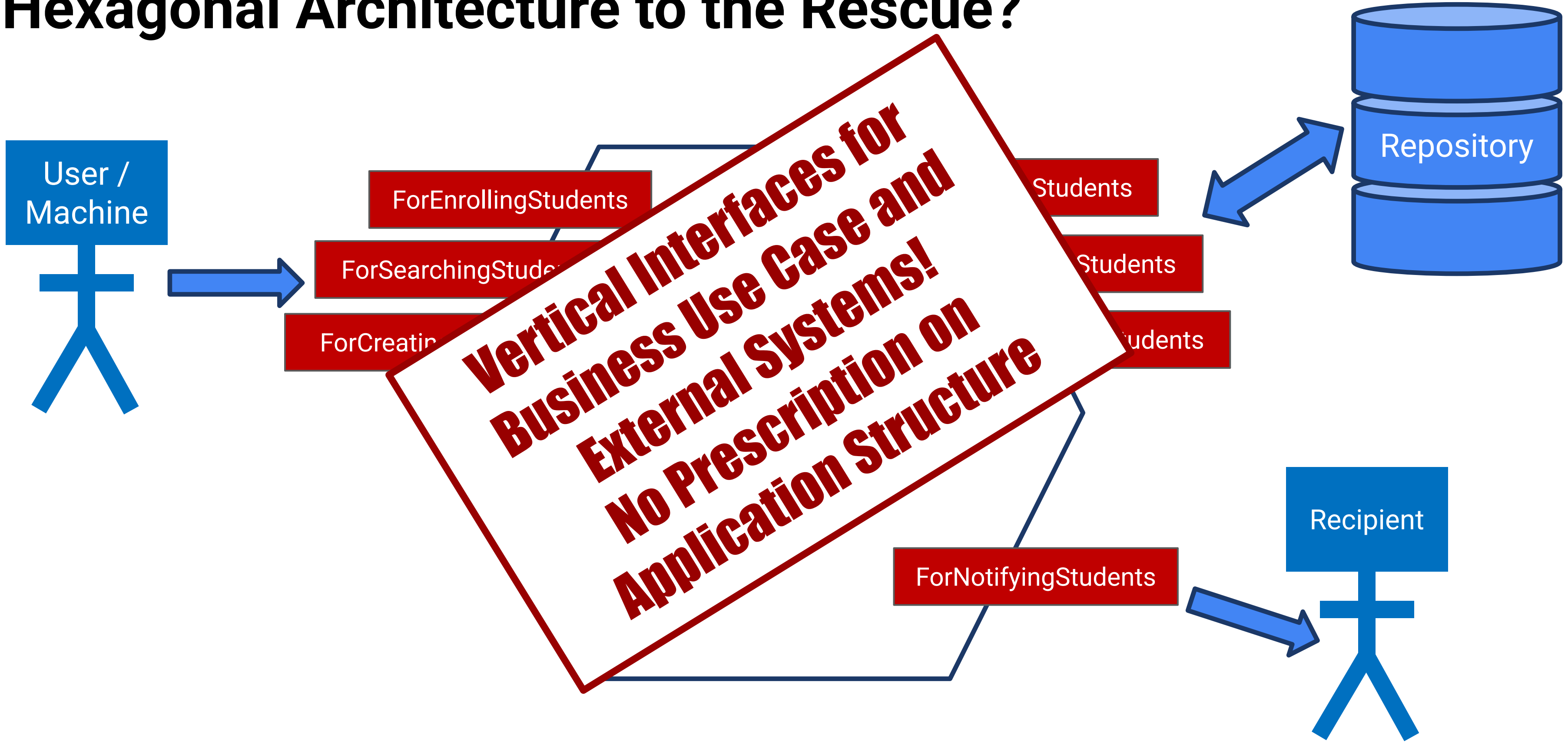
Return domain objects from data repositories instead of database row entities.

And we may add some interfaces to separate the Application from the Data Access even more

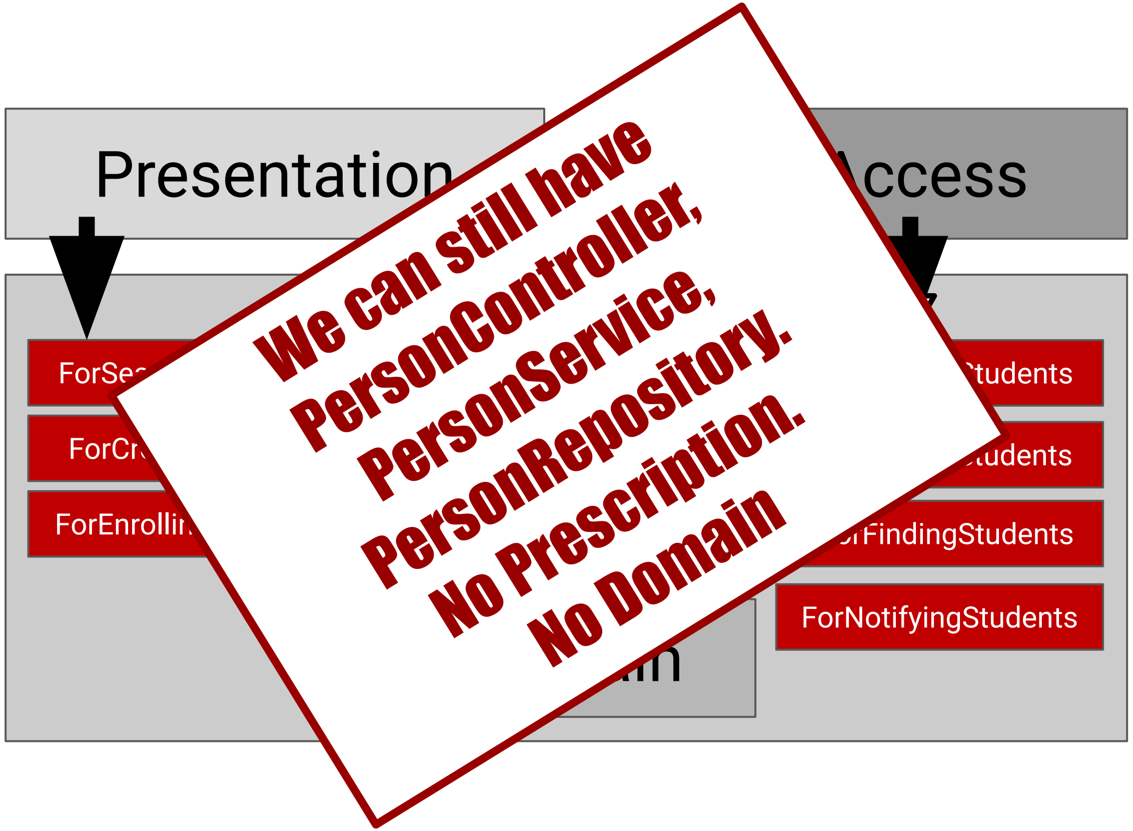
Problems with Onion Architecture



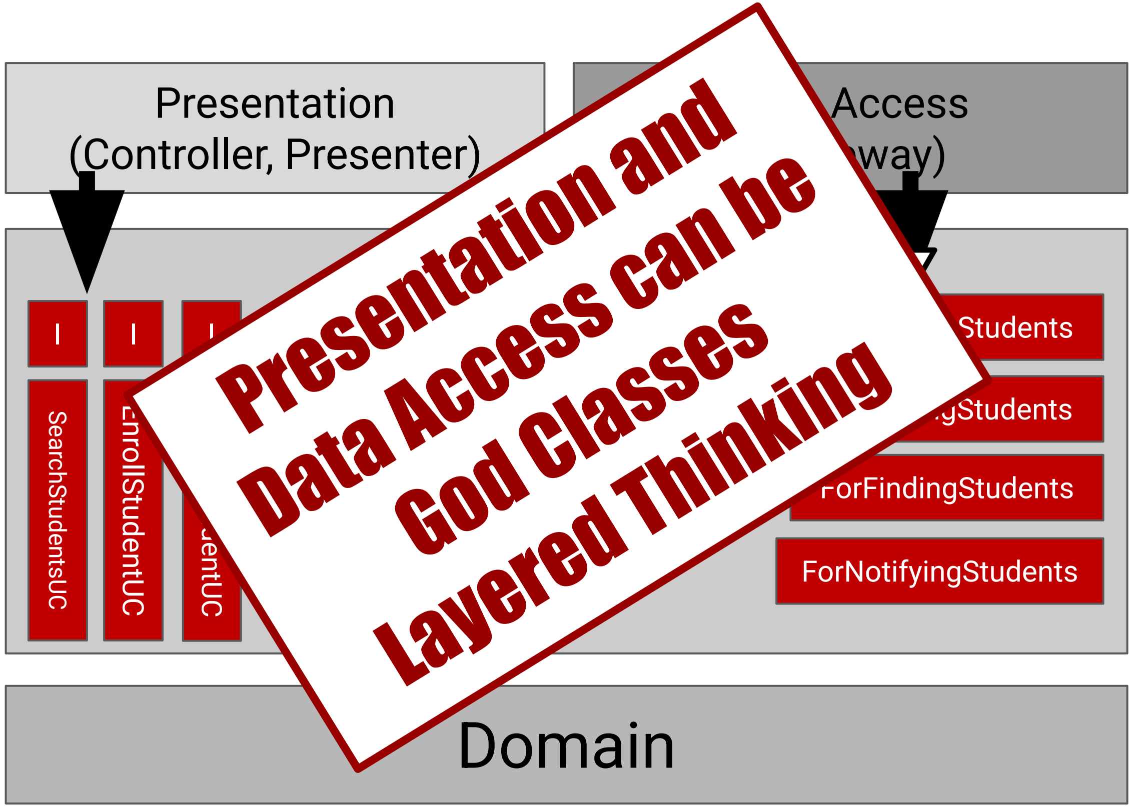
Hexagonal Architecture to the Rescue?



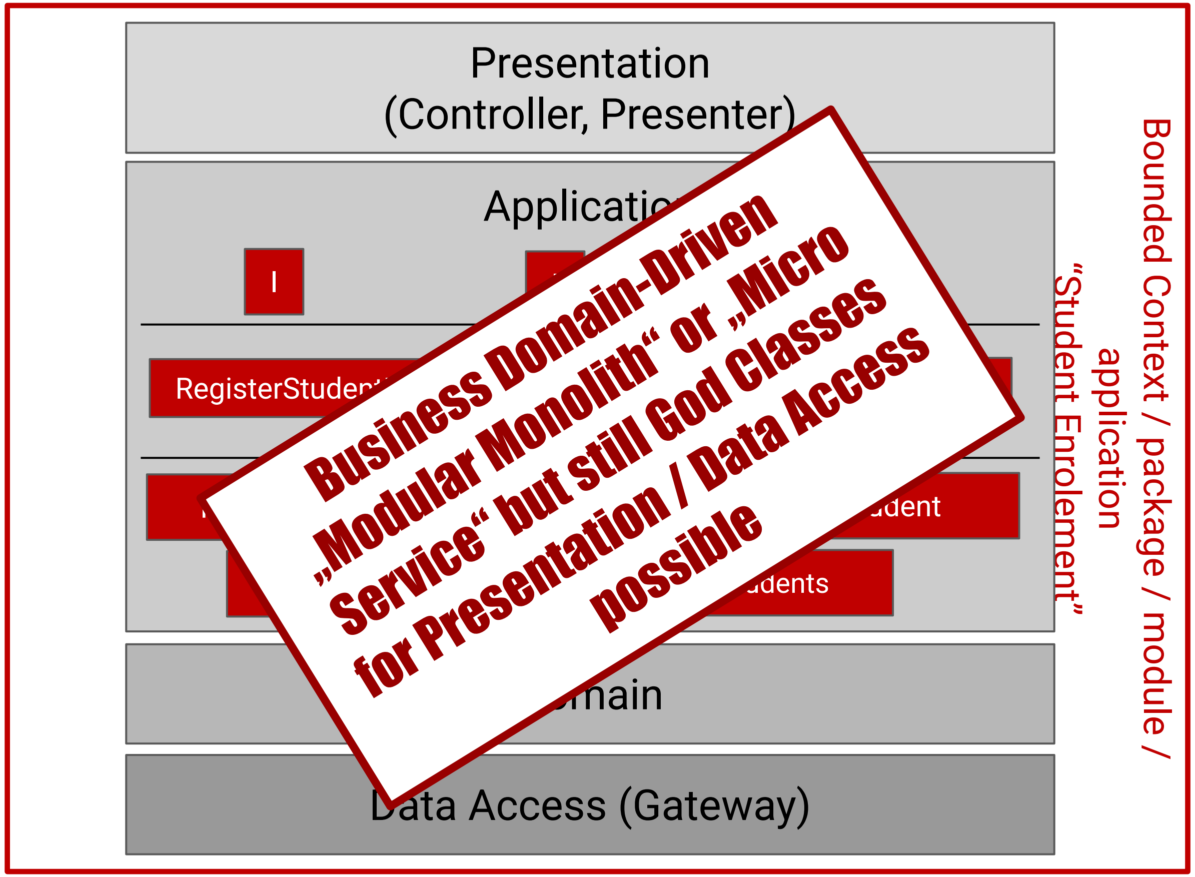
Hexagonal / Ports and Adapters as Layered Architecture



Extending Ports & Adapters: Clean Architecture



Clean Architecture & DDD Bounded Contexts

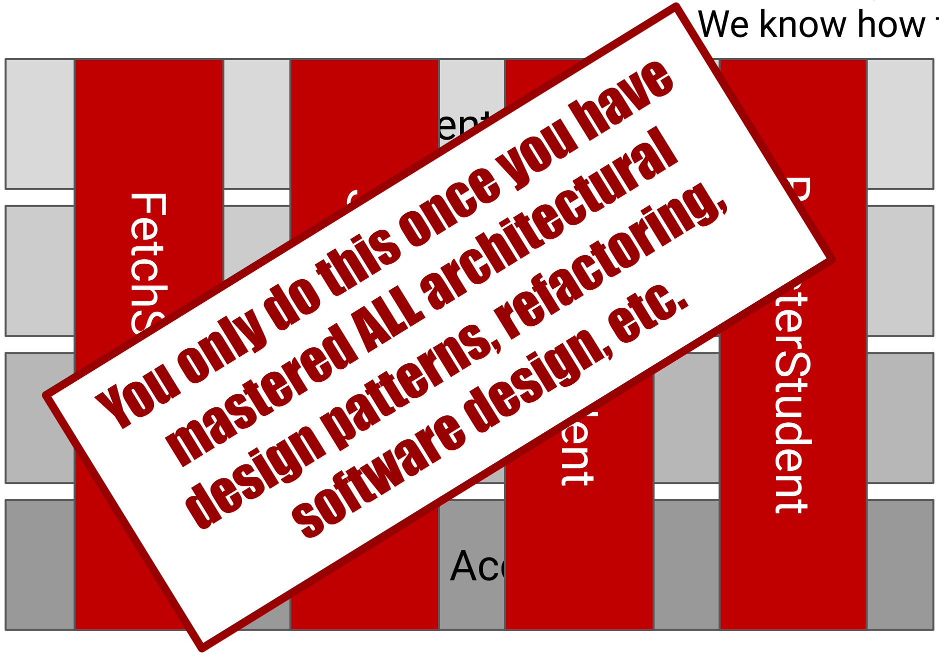


Business Domain-Driven „Modular Monolith" or „Micro Service" but still God Classes for Presentation / Data Access possible

„Real“ Vertical Slice Architecture

Remove layering: every use case on its own

Emerging Software Design!
- We know all Code Smells
- We are very quick in refactoring
- We know how to refactor towards domain



ent
FetchS
Acc
ent
terStudent
You only do this once you have mastered ALL architectural design patterns, refactoring, software design, etc.

CQRS
- GETs = simple
- POSTs = complex

Conclusions

To become more agile, we need to align software architecture with business reality

CodeArtify

Conclusions

Domain-Driven Design aims to align business and code to decrease the effort needed to respond to change (agile)

CodeArtify

Conclusions

Modular Monoliths with business-domain focused architectures allow us to more quickly react to changes and undo unwise slice decisions compared to Micro Services

CodeArtify

Conclusions

However: the biggest levers to better software are business understanding, team autonomy, improved social interactions and trust, and organizational change!

CodeArtify