Domain-Driven Design

A Deep Dive

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

About Me

I have more than 24 years of experience in software development, being involved in various projects covering desktop, client/server, web, cloud, and mobile applications using mainly Microsoft tools & technologies, but also Apple and open-source.

I'm a Certified Scrum Master and a certified APMG Agile Project Manager. Since 2003, I'm holding a Ph.D. degree in Industrial Engineering.

You can find more about me on LinkedIn: https://www.linkedin.com/in/eduardghergu/

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Domain-Driven Design: A Deep Dive

1. Short Recap from previous session
2. Advanced Topics
3. Software architecture patterns
4. Conclusions. Q&A

Eduard Ghergu, Eng., PhD.

www.professional-programmer.com

Short Recap from previous session

We have talked about the main DDD building blocks...

- Ubiquitous Language
- Bounded Context / Context Maps
- Domain
- Model

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Short Recap from previous session - cont.

… and Patterns

- Entity
- Value Object
- Aggregate
- Aggregate Root

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Short Recap from previous session - cont.

… and Patterns - cont.

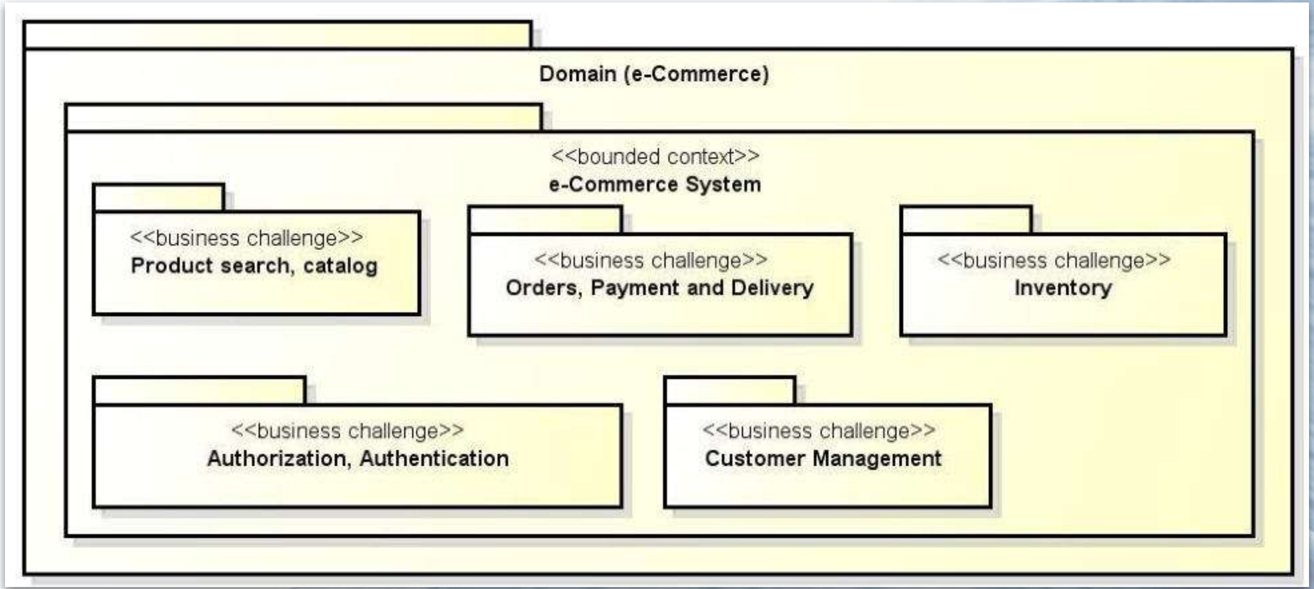
- Domain Event
- Domain Service
- Repository
- Factory

Eduard Ghergu, Eng., PhD.

www.professional-programmer.com

Advanced Topics

Let’s start with sample e-commerce domain, before applying DDD [12]...



Domain (e-Commerce)

<<bounded context>>
e-Commerce System

<<business challenge>>
Product search, catalog

<<business challenge>>
Orders, Payment and Delivery

<<business challenge>>
Inventory

<<business challenge>>
Authorization, Authentication

<<business challenge>>
Customer Management

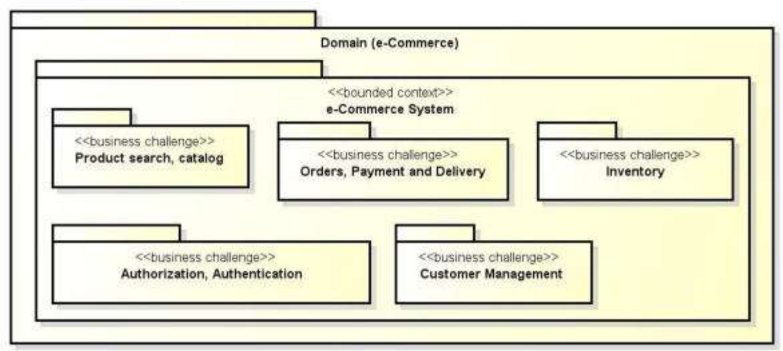
Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Advanced Topics

Let's start applying some DDD to this problem…

1. **Identify the possible Subdomains inside a Domain** (the "problem" to solve). A Domain has its own strategic challenges which can be seen as Subdomains that can be classified as Core, Support and Generic.

2. **Split the Domain in Subdomains**. It is a good practice to set a Bounded Context for each Subdomain.

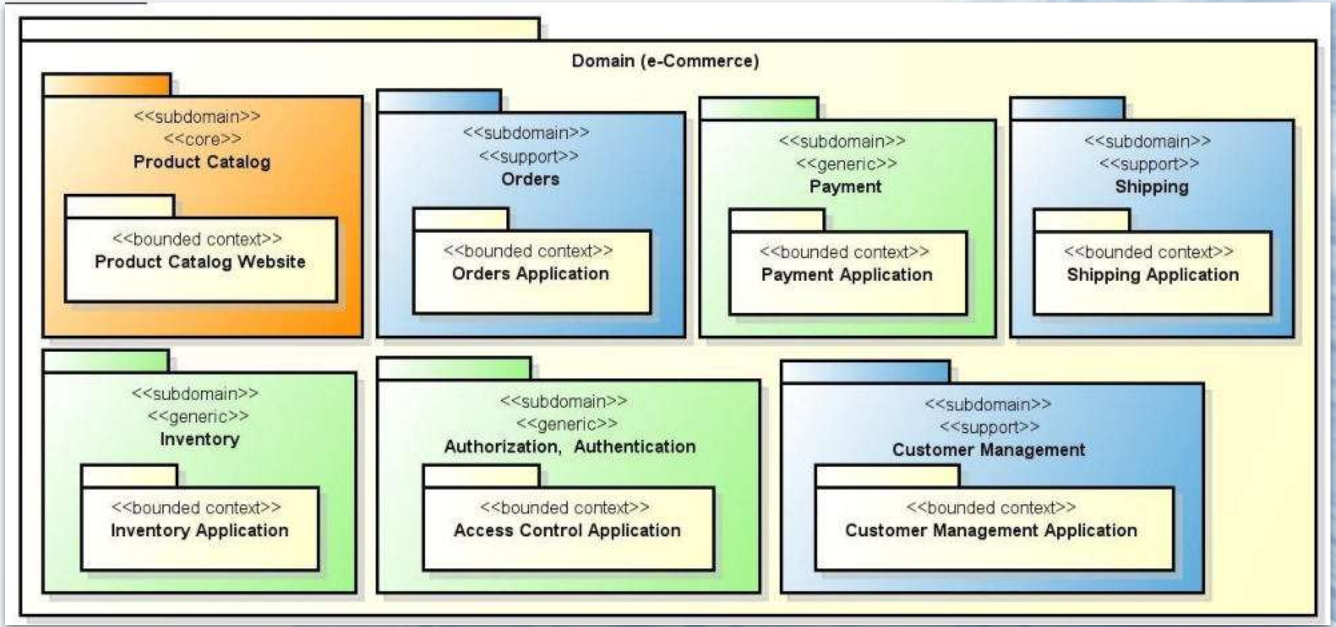


Eduard Ghergu, Eng., PhD.

www.professional-programmer.com

Advanced Topics

The result [12]:



Domain (e-Commerce)

<<subdomain>>
<<core>>
Product Catalog

<<bounded context>>
Product Catalog Website

<<subdomain>>
<<support>>
Orders

<<bounded context>>
Orders Application

<<subdomain>>
<<generic>>
Payment

<<bounded context>>
Payment Application

<<subdomain>>
<<support>>
Shipping

<<bounded context>>
Shipping Application

<<subdomain>>
<<generic>>
Inventory

<<bounded context>>
Inventory Application

<<subdomain>>
<<generic>>
Authorization, Authentication

<<bounded context>>
Access Control Application

<<subdomain>>
<<support>>
Customer Management

<<bounded context>>
Customer Management Application

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Advanced Topics

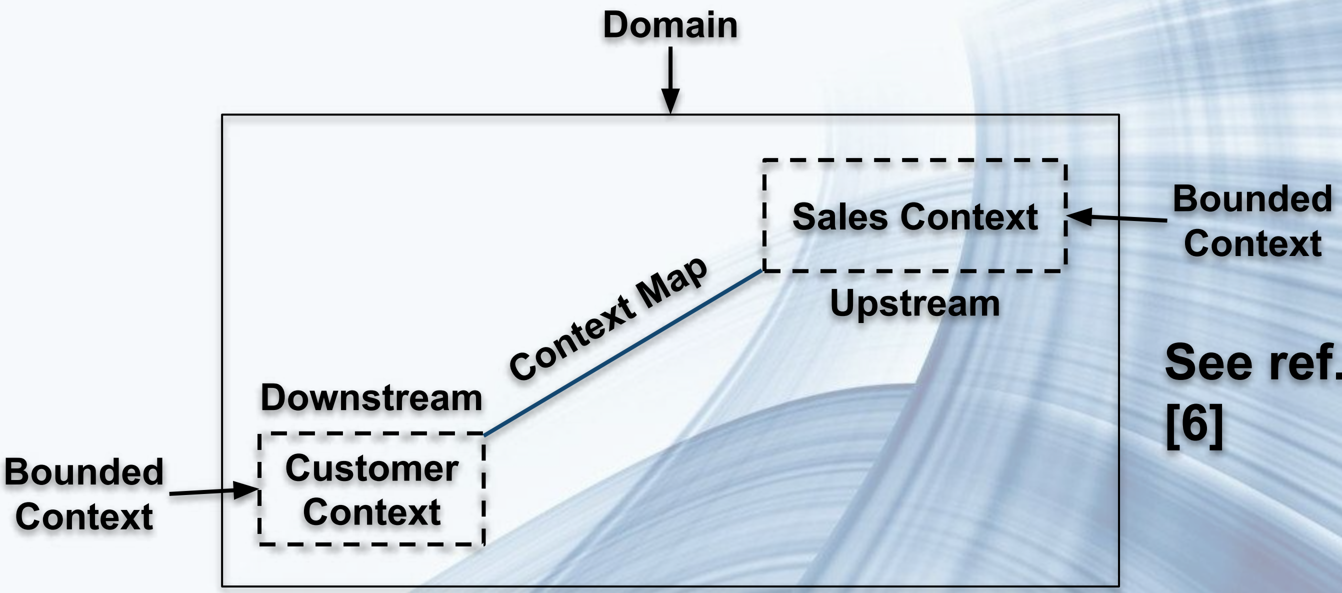
Interaction between identified bounded contexts:

- Context Maps
- Domain Events
- Anti Corruption Layer
- Shared Kernel

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Advanced Topics

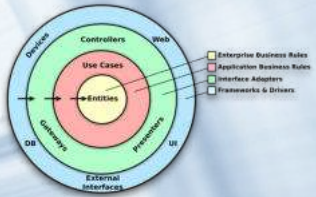
Interaction between identified bounded contexts:



See ref. [6]

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Software Architecture Patterns: Clean Architecture



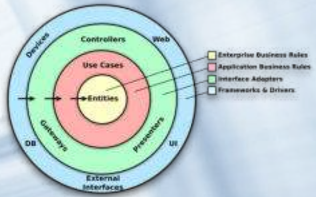
- ...is a software architectural pattern coined by Robert “Uncle Bob” Martin in his book called, “Clean Architecture: A Craftsman's Guide to Software Structure and Design”

- ...evolved over time from several other architectures including Hexagonal Architecture, and Onion Architecture

- ...has a number of principles that I’ll summarize here:
  ✓ Independent of Frameworks
  ✓ Testable
  ✓ Independent of UI
  ✓ Independent of Database
  ✓ Independent of any external agency

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

The Clean Architecture Principles [10]



- **Independent of Frameworks**. The architecture does not depend on the existence of some library. This allows you to use such frameworks as tools, rather than having to cram your system into their limited constraints.

- **Testable**. The business rules can be tested without the UI, Database, Web Server, or any other external element.

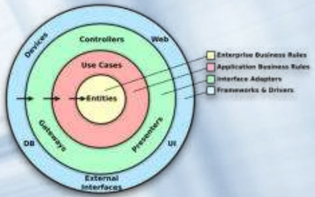
- **Independent of UI**. The UI can change easily, without changing the rest of the system. A Web UI could be replaced with a console UI, for example, without changing the business rules.

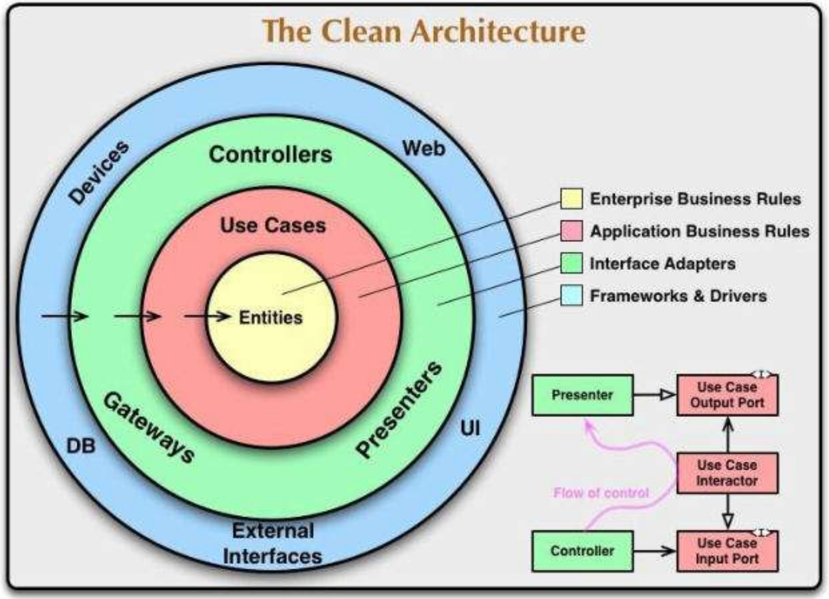
- **Independent of Database**. You can swap out Oracle or SQL Server, for Mongo, BigTable, CouchDB, or something else. Your business rules are not bound to the database.

- **Independent of any external agency**. In fact your business rules simply don't know anything at all about the outside world.

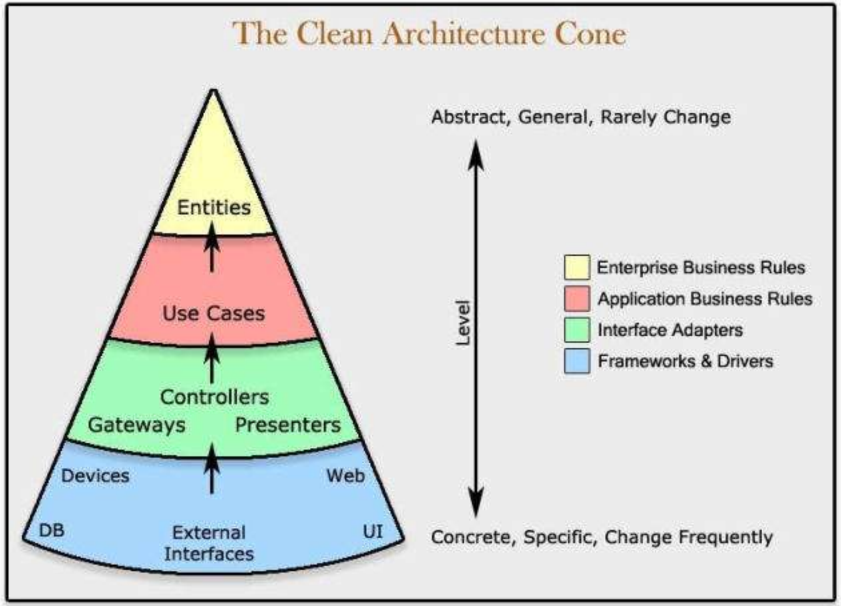
Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Clean Architecture Building Blocks[10]





Original view [11]



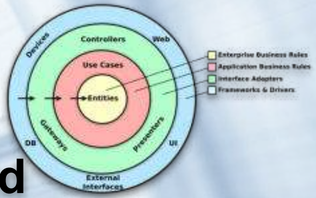
The Clean Architecture Cone
Entities
Use Cases
Controllers
Gateways
Presenters
Devices
Web
DB
External Interfaces
UI
Abstract, General, Rarely Change
Level
Concrete, Specific, Change Frequently
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers

Alternative view [12]

Eduard Ghergu, Eng., PhD.

www.professional-programmer.com

Clean Architecture Building Blocks[10]



- **Entities** encapsulate enterprise wide domain entities that could be used by many different applications in the enterprise

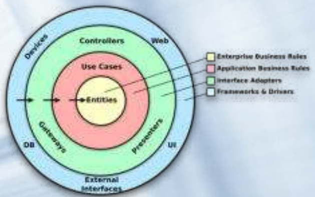
- **Use Cases** contain application specific business rules. They orchestrate the flow of data to and from the **Entities**

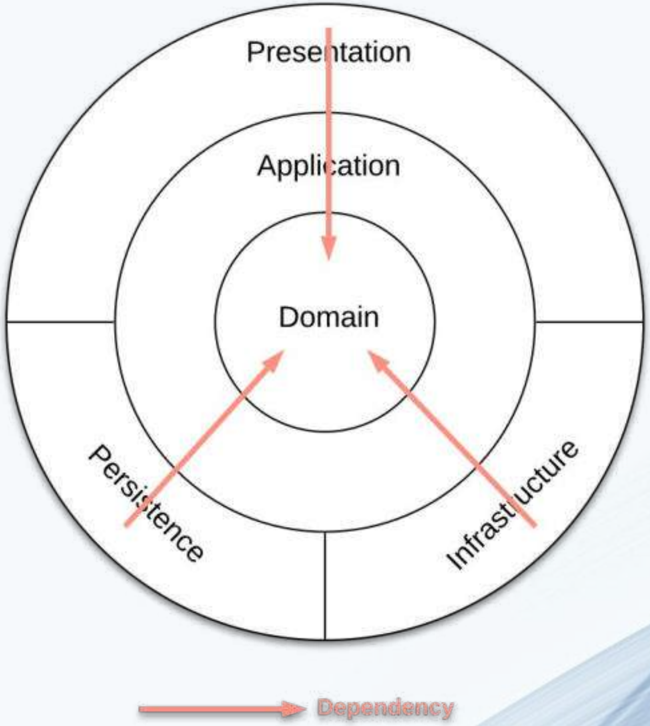
- **Interface Adapters** convert data from the format most convenient for the **Use Cases** and **Entities**, to the format most convenient for some external agency such as the Database or the Web

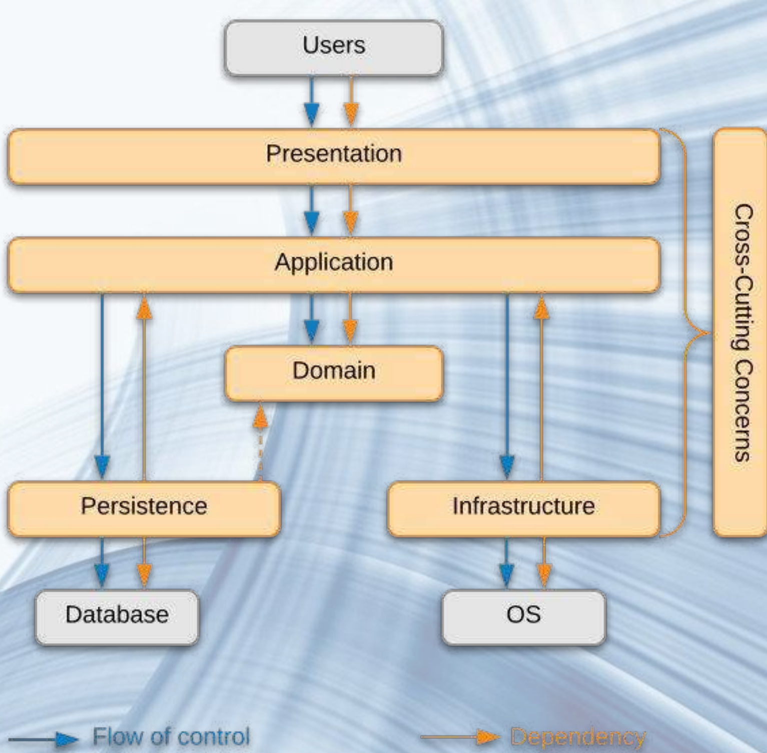
- **Frameworks and Drivers** - The outermost layer is generally composed of frameworks and tools such as the Database, the Web Framework, etc.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Clean Architecture - Down to Earth



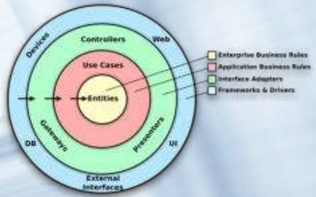




Developer view [11]

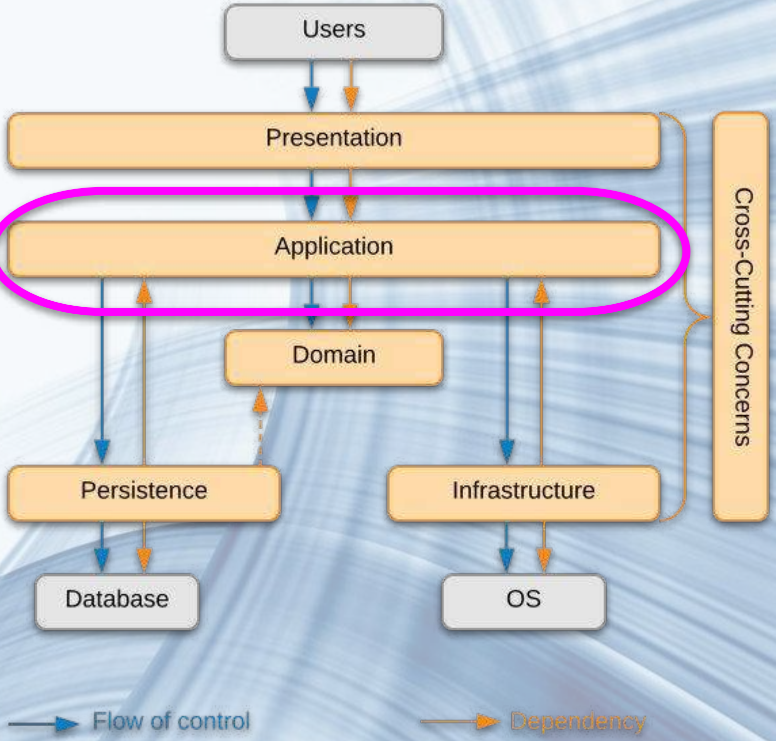
Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Clean Architecture - Down to Earth



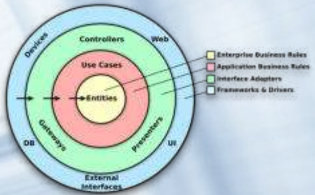
Application Layer

- Implements use cases
- High-level application logic
- Knows about domain, but no other layers
- Is implemented using the CQS pattern for simple scenarios; for complex ones, the CQRS pattern can be used



Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Clean Architecture - Down to Earth



Command-Query Separation (CQS)

- Software Design Pattern

Command:

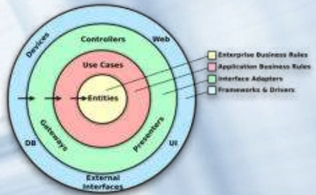
- Does something (execute a request)
- Should modify state
- Should not return a value (ideally; better, rise an event)

Query:

- Answers a question (respond to a request)
- Should not modify state
- Always returns a value

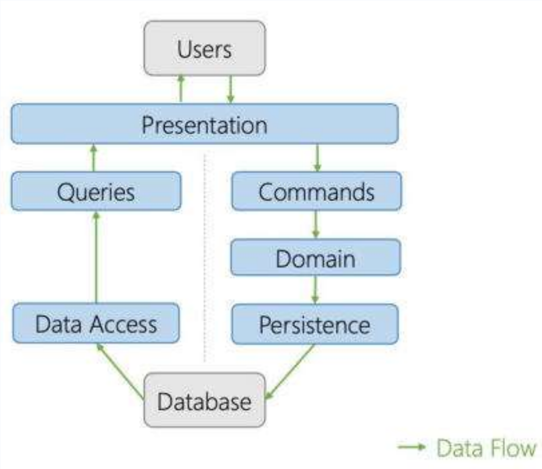
Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

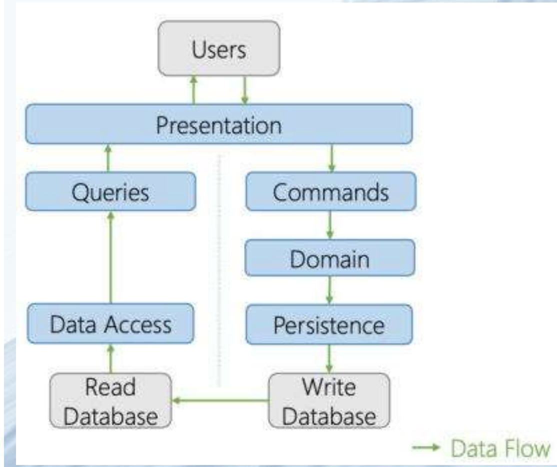
Clean Architecture - Down to Earth

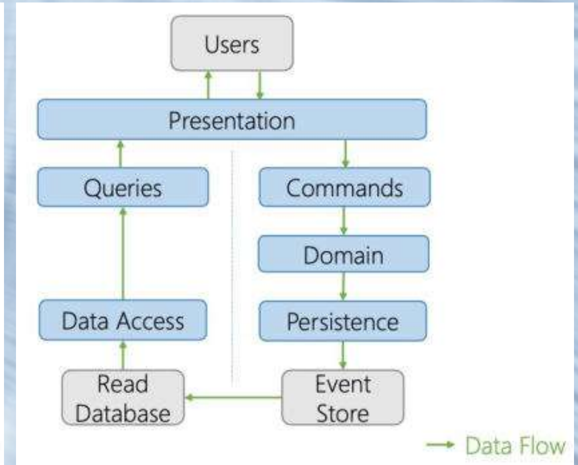


Command-Query Responsibility Segregation (CQRS)

- **Architecture Design Pattern**

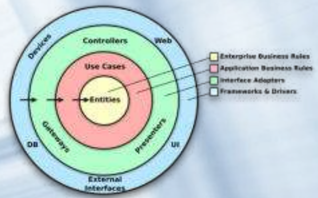






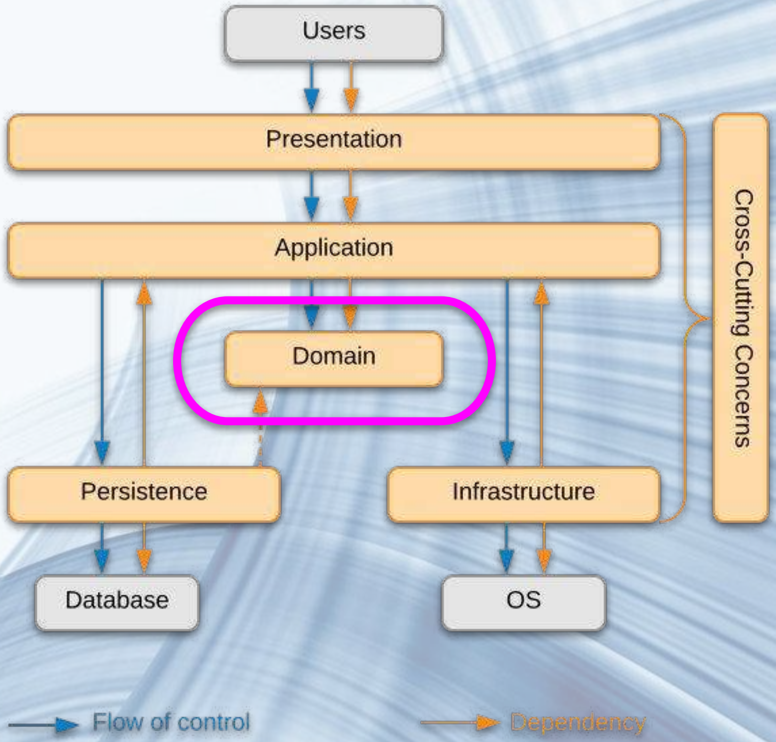
Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Clean Architecture - Down to Earth



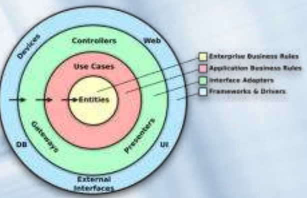
Domain Layer

- Contains the Domain core Model
- No knowledge of other layers
- Independent of frameworks (no attributes or decorations)
- **Beware of Anemic Domain Model antipattern**



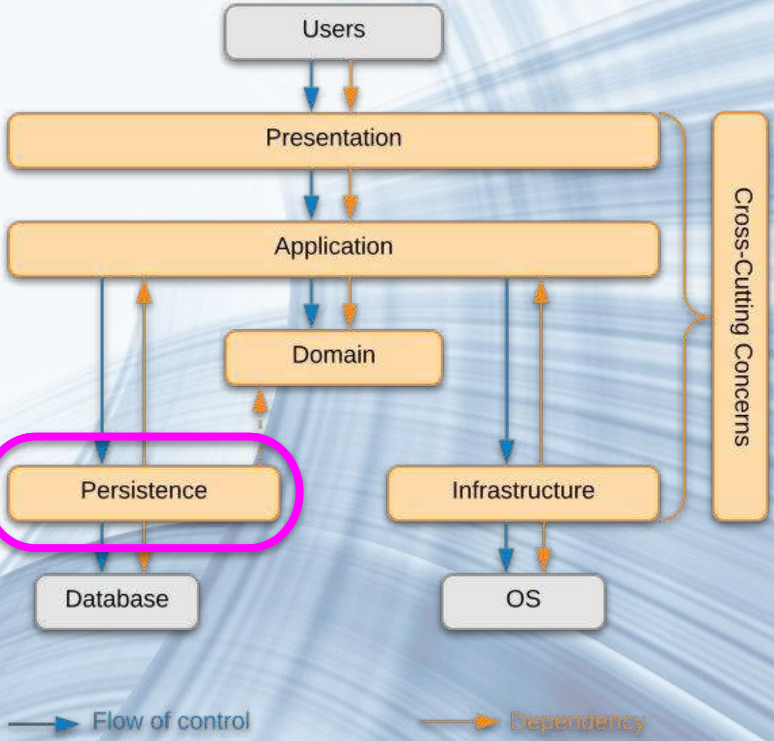
Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Clean Architecture - Down to Earth



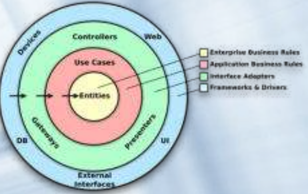
Persistence Layer

- Refers to Repository pattern incarnations; the data source can be anything from a relational database to a REST API
- Depends on Domain and is invoked by Application only
- Acts as a Mediator between Domain and storage-specific entities



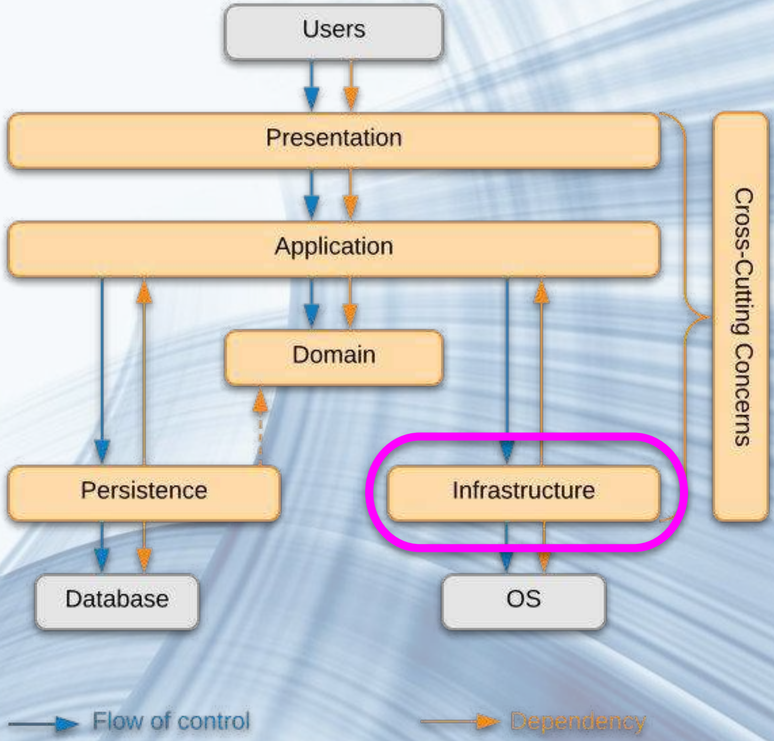
Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Clean Architecture - Down to Earth



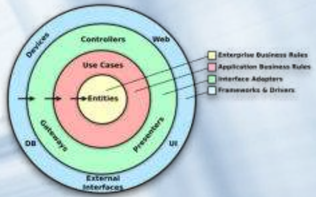
Infrastructure Layer

- Facilitates the interaction with external systems and/or services
- Is invoked by Application only
- It can have a dependency on Domain, and can act as a Mediator between Domain and infrastructure specific entities



Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

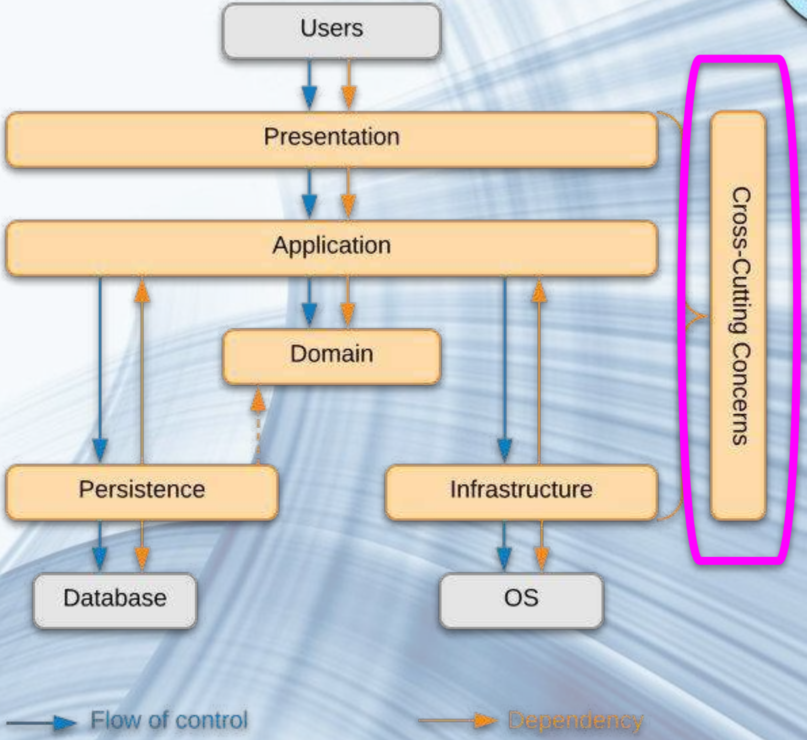
Clean Architecture - Down to Earth



Cross-Cutting Concerns

- Contains functionality used by more than one layer like Logging, Exception Handling, Services Events, etc.

- No knowledge about any layer



Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Q & A

For any inquiries or requests:
eduard.ghergu@professional-programmer.com

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Thank you!
See you at the next webinar!

Eduard Ghergu, Eng., PhD.

www.professional-programmer.com

References

1. https://www.infoq.com/minibooks/domain-driven-design-quickly/
2. https://vladikk.com/2016/04/05/tackling-complexity-ddd/
3. https://en.wikipedia.org/wiki/Domain-driven_design
4. https://www.jamesmichaelhickey.com/clean-architecture/
5. https://blog.knoldus.com/is-shifting-to-domain-driven-design-worth-your-efforts/
6. https://thedomaindrivendesign.io/
7. https://www.culttt.com/2014/11/12/domain-model-domain-driven-design/
8. https://martinfowler.com/bliki/BoundedContext.html
9. https://github.com/zkavtaskin/Domain-Driven-Design-Example

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

References - cont.

10. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

11. https://www.freecodecamp.org/news/a-quick-introduction-to-clean-architecture-990c014448d2/

12. http://www.fabriciosuarte.com/2016/02/domain-driven-design-hands-on-example.html

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com