Developing MVC Apps with DDD

About me

Trayan Iliev

– CEO of **IPT – Intellectual Products & Technologies**
http://www.iproduct.org

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6+, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer: Spring, Java EE, Node.js, Express, GraphQL, SOA, REST, DDD & Reactive Microservices

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

MVC Comes in Different Flavors

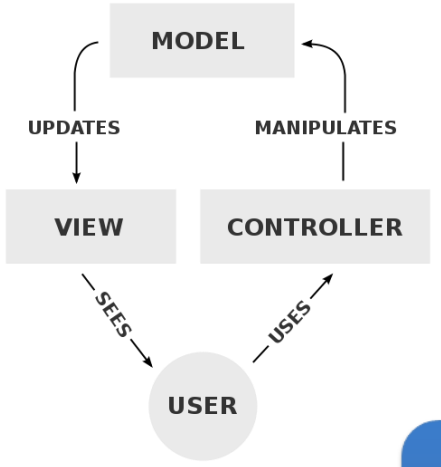
What is the difference between following patterns:

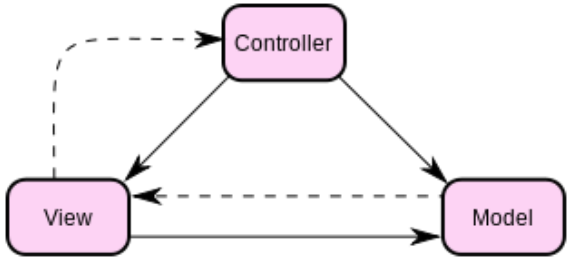
- Model-View-Controller (MVC)
- Model-View-ViewModel (MVVM)
- Model-View-Presenter (MVP)

http://csl.ensm-douai.fr/noury/uploads/20/ModelViewController.mp3

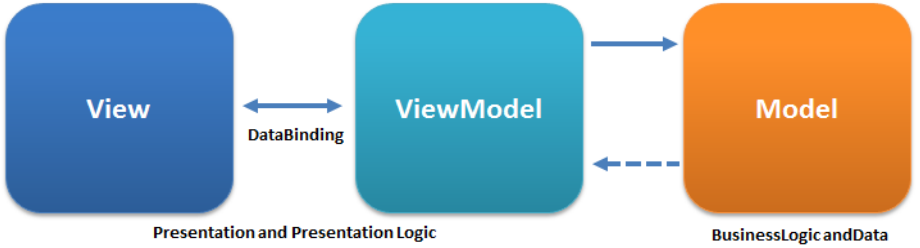
MVC Comes in Different Flavors - II

- MVC

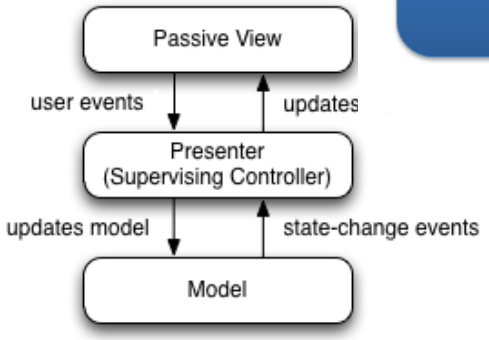




- MVVM

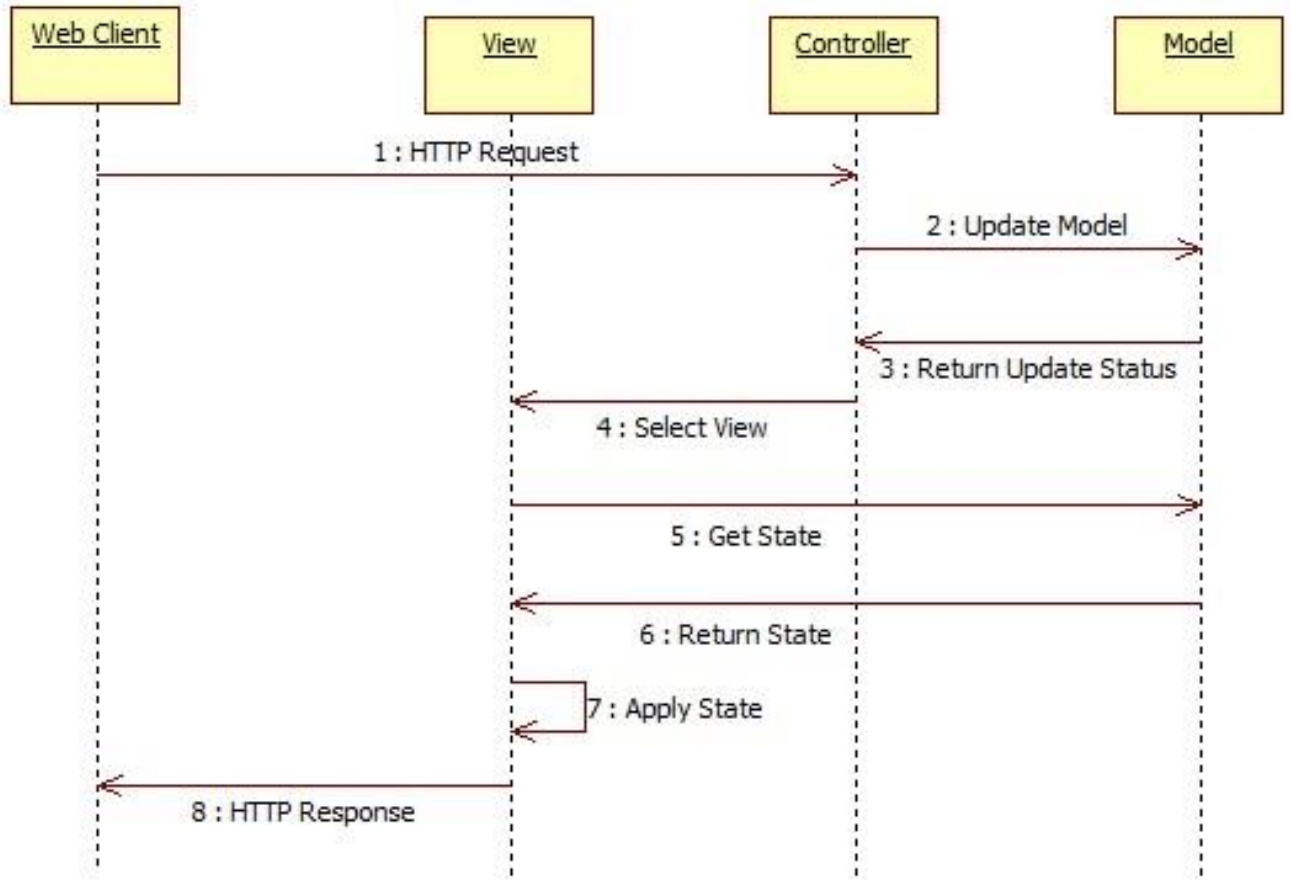


- MVP



Sources:https://en.wikipedia.org/wiki/Model_View_ViewModel#/media/File:MVVMPattern.png,
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter#/media/File:Model_View_Presenter_GUI_Design_Pattern.png
License: CC BY-SA 3.0, Authors:Ugaya40, Daniel Cardenas

Web MVC Interactions Sequence Diagram



SOLID Design Principles of OOP

- Single responsibility principle - a class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class.
- Open–closed principle - software entities should be open for extension, but closed for modification.
- Liskov substitution principle - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
- Interface segregation principle - Many client-specific interfaces are better than one general-purpose interface.
- Dependency inversion principle - depend upon abstractions, not concretions.

Domain Driven Design (DDD)

We need tools to cope with all that complexity inherent in robotics and IoT domains.

Simple solutions are needed – cope with problems through divide and concur on different levels of abstraction:

Domain Driven Design (DDD) – back to basics:
domain objects, data and logic.

Described by Eric Evans in his book:
Domain Driven Design: Tackling Complexity in the Heart of Software, 2004

7

Domain Driven Design (DDD)

Main concepts:

- ❖ Entities, value objects and modules
- ❖ Aggregates and Aggregate Roots [Haywood]:

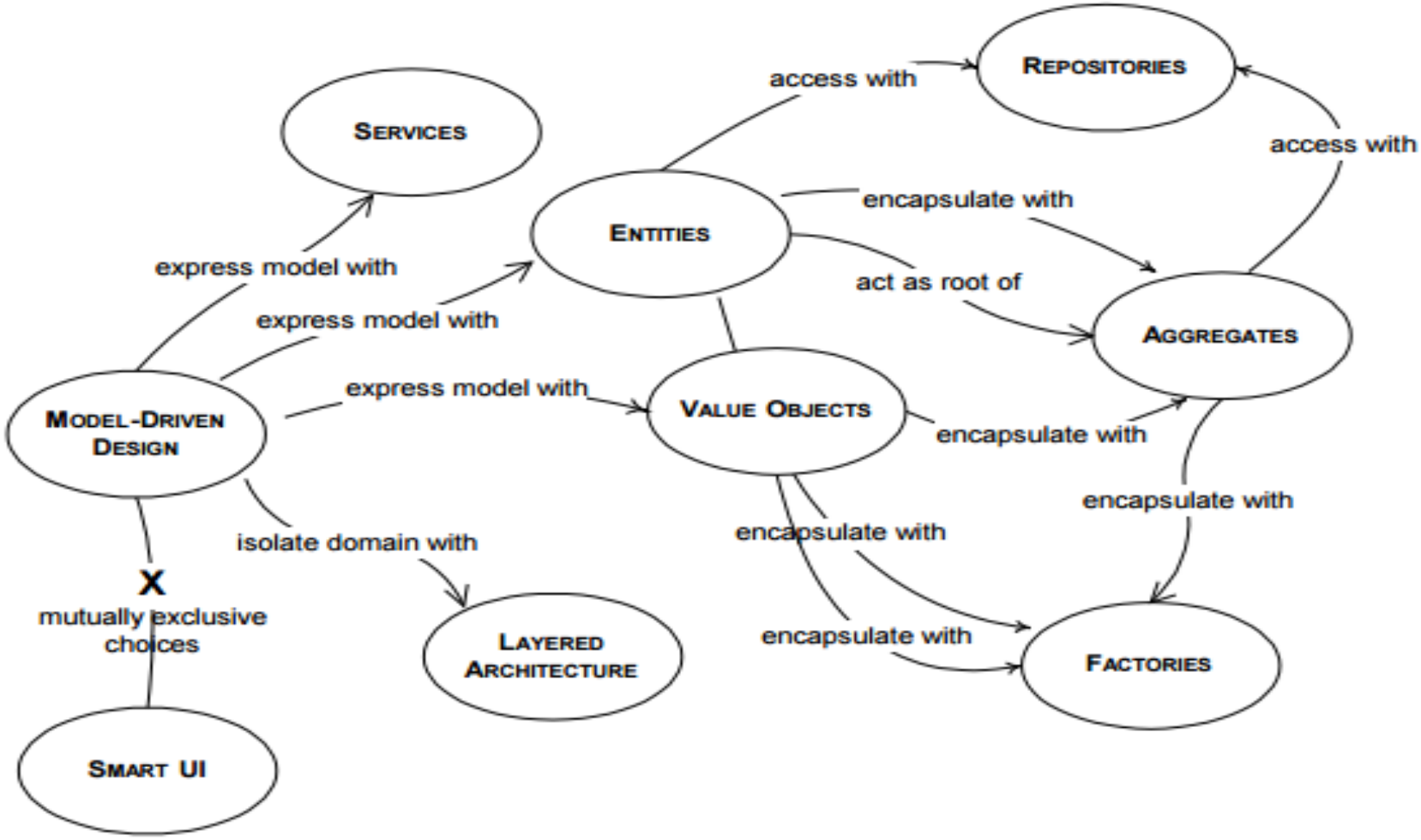
value < entity < aggregate < module < BC

- ❖ Repositories, Factories and Services:

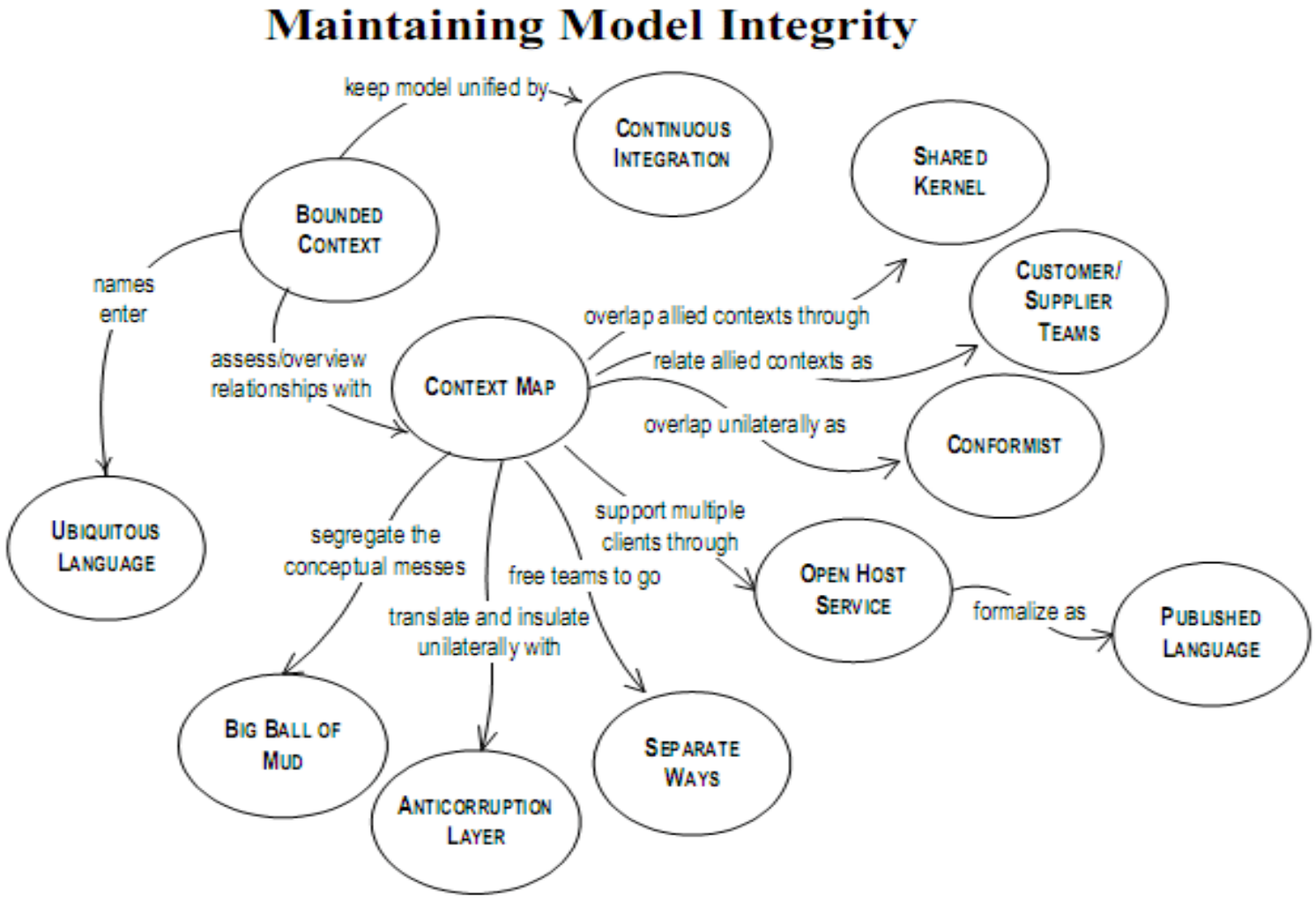
application services <-> **domain services**

- ❖ Separating interface from implementation

Domain Driven Design (DDD)



Domain Driven Design (DDD)



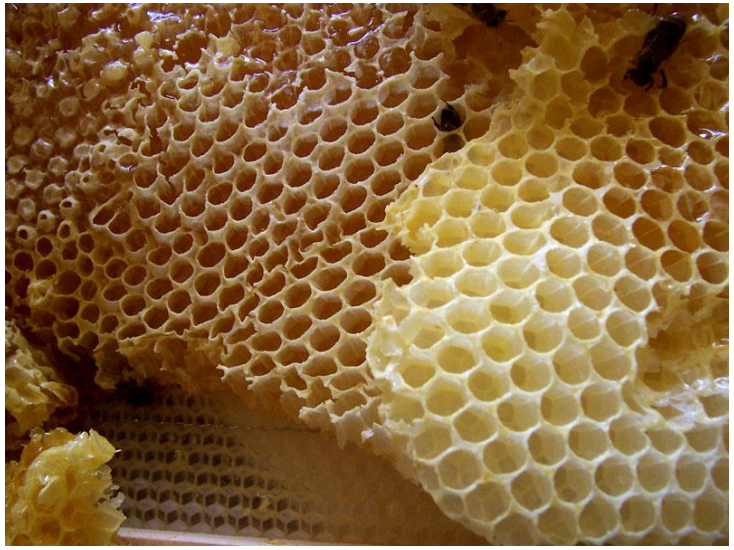
Domain Driven Design (DDD)

- Ubiquitous language and Bounded Contexts
- DDD Application Layers:
- Infrastructure, Domain, Application, Presentation
- Hexagonal architecture :

OUTSIDE <-> transformer <->

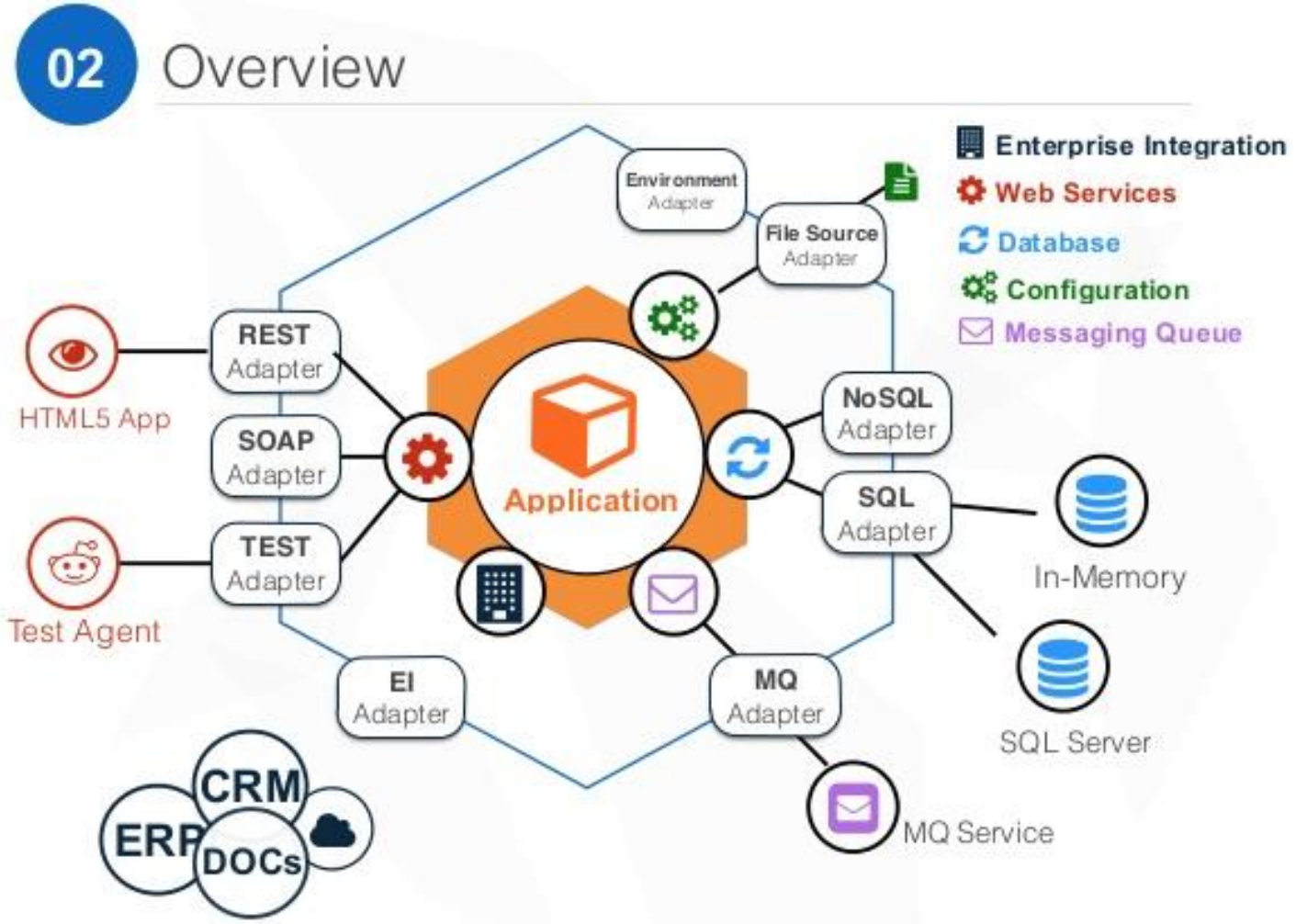
(application <-> domain)

[A. Cockburn]



11

Hexagonal Architecture



Source: https://www.slideshare.net/fabricioepa/hexagonal-architecture-for-java-applications

12

Hexagonal Architecture Principles

- Allows an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases.
- As events arrive from the outside world at a port, a technology-specific adapter converts it into a procedure call or message and passes it to the application
- Application sends messages through ports to adapters, which signal data to the receiver (human or automated)
- The application has a semantically sound interaction with all the adapters, without actually knowing the nature of the things on the other side of the adapters

Source: http://alistair.cockburn.us/Hexagonal+architecture

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD