SOLID Principles. Domain Driven Design

Where to Find The Code and Materials?

https://github.com/iproduct/intro-python

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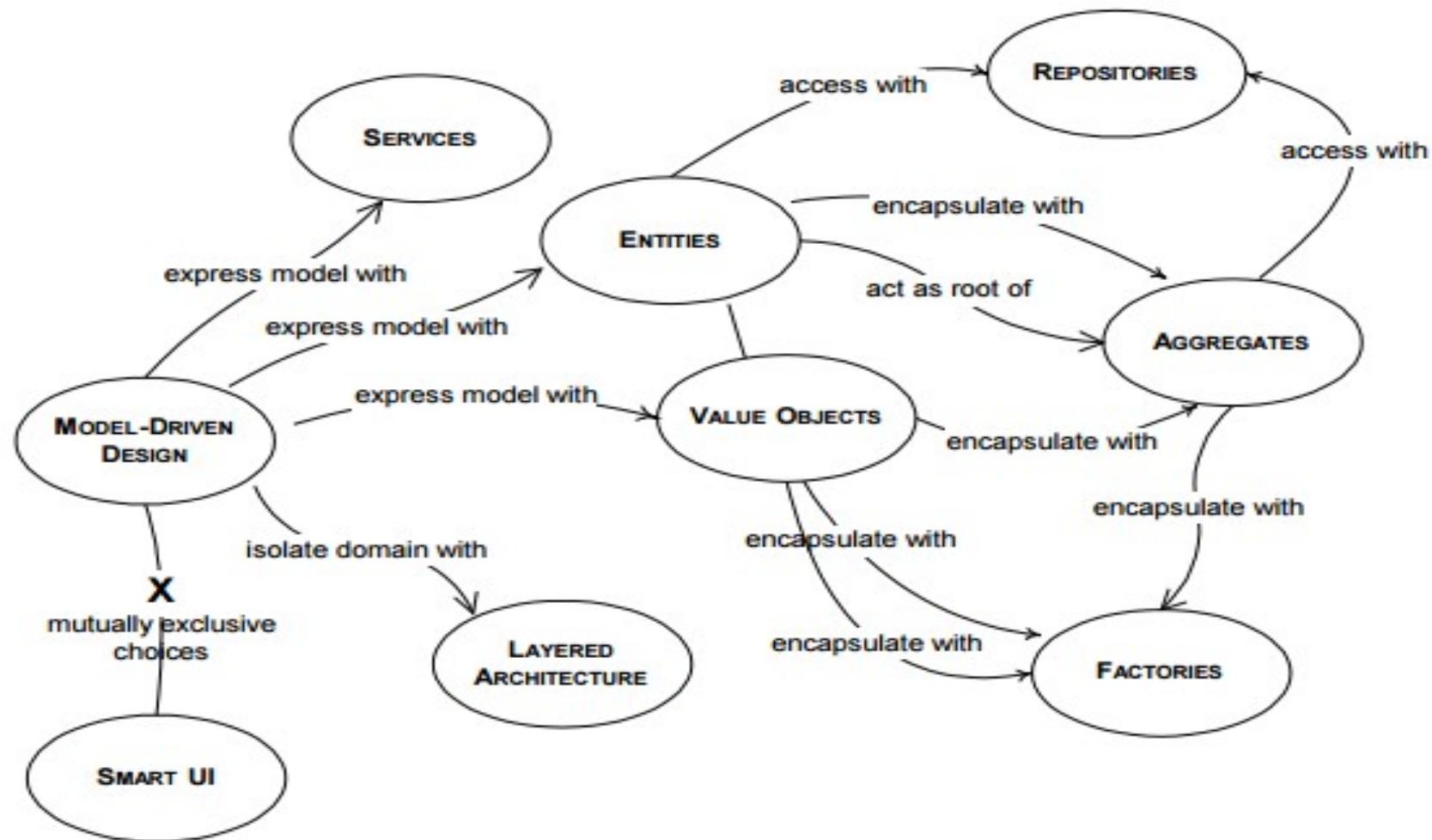
Described by Eric Evans in his book:
Domain Driven Design: Tackling Complexity in the Heart of Software, 2004

Domain Driven Design (DDD)

Main concepts:
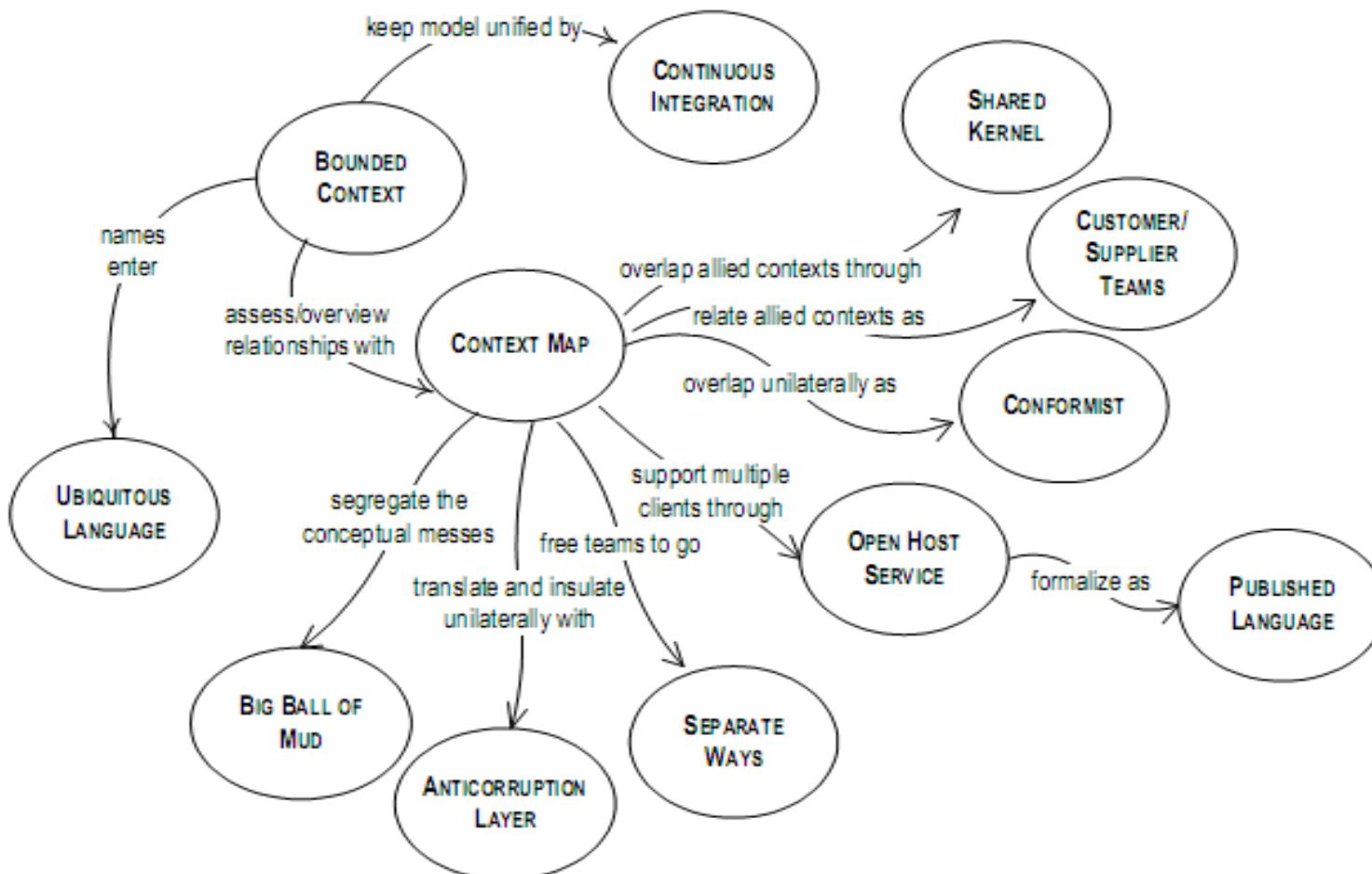
❖ Entities, value objects and modules

❖ 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)
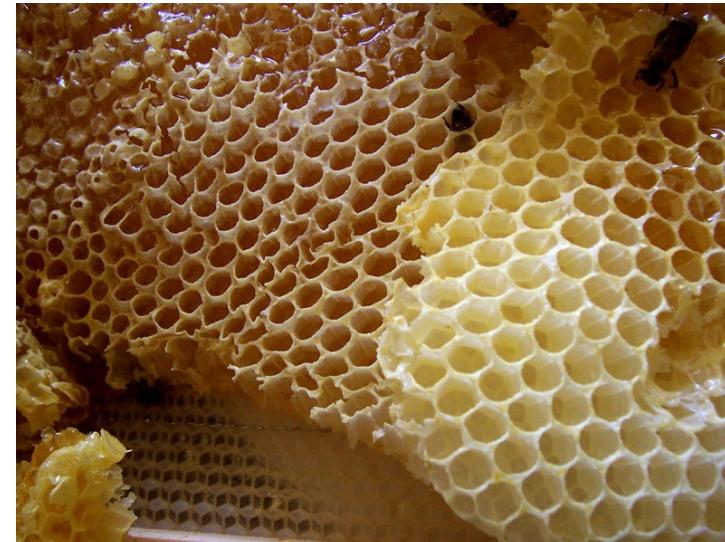
Domain Driven Design (DDD)

Maintaining Model Integrity

Domain Driven Design (DDD)
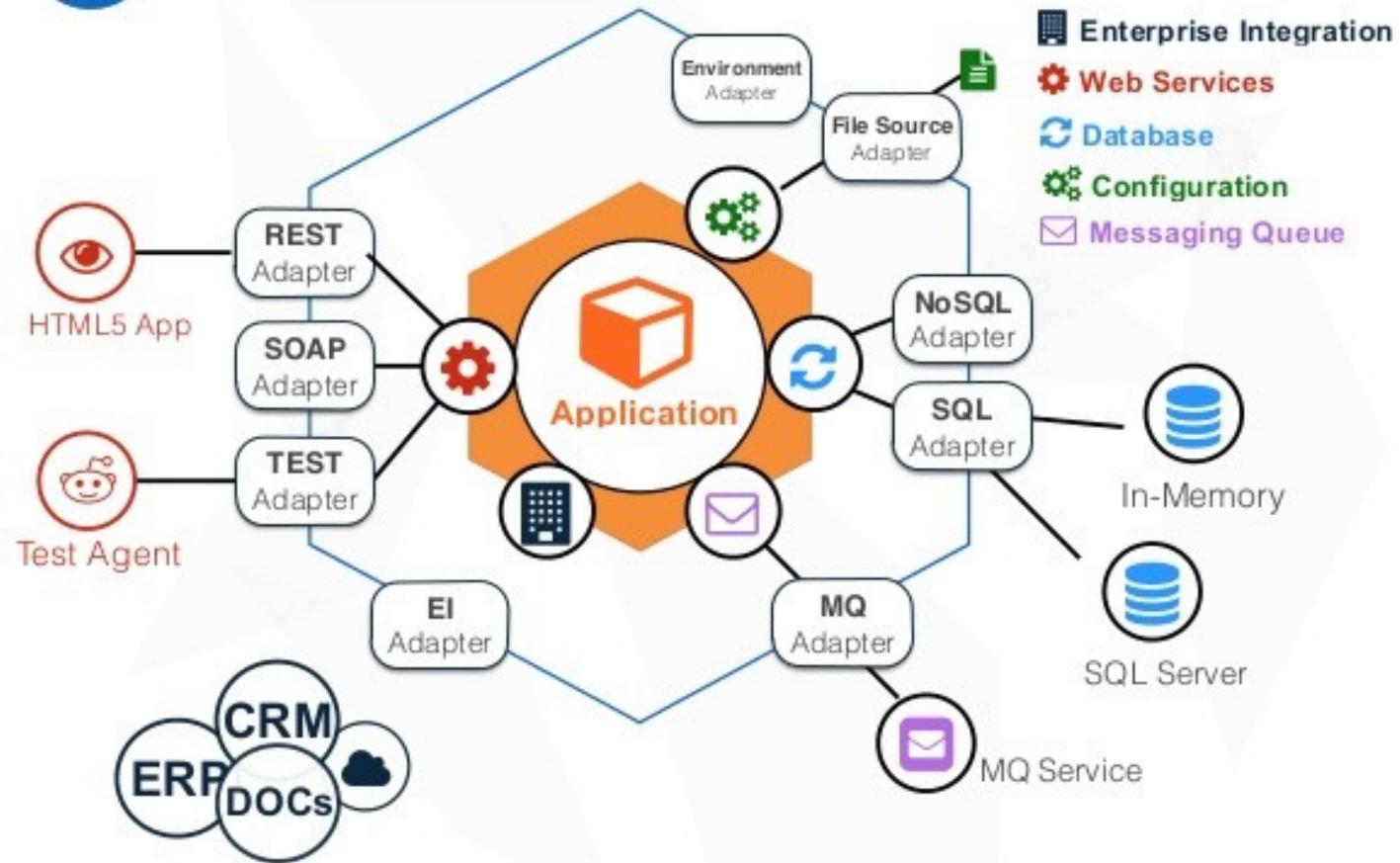
❖Ubiquitous language and Bounded Contexts

❖DDD Application Layers:

❖Infrastructure, Domain, Application, Presentation

❖ Hexagonal architecture :

OUTSIDE <-> transformer <->

(application <-> domain)

[A. Cockburn]

Hexagonal Architecture

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

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