

Paweł Rajba

pawel@cs.uni.wroc.pl

<http://pawel.ii.uni.wroc.pl/>

Domain Driven Design

Wprowadzenie

- Domain Driven Design
 - What is it?
 - Two main parts: Strategic & Tactical

Strategic DDD

- Domain, Subdomain
- Ubiquitous Language
- Bounded Context
- Context Maps

Tactical DDD

- Entities
- Value Objects
- Aggregates
- Modules
- Repositories
- Factories
- Services
- Domain Events

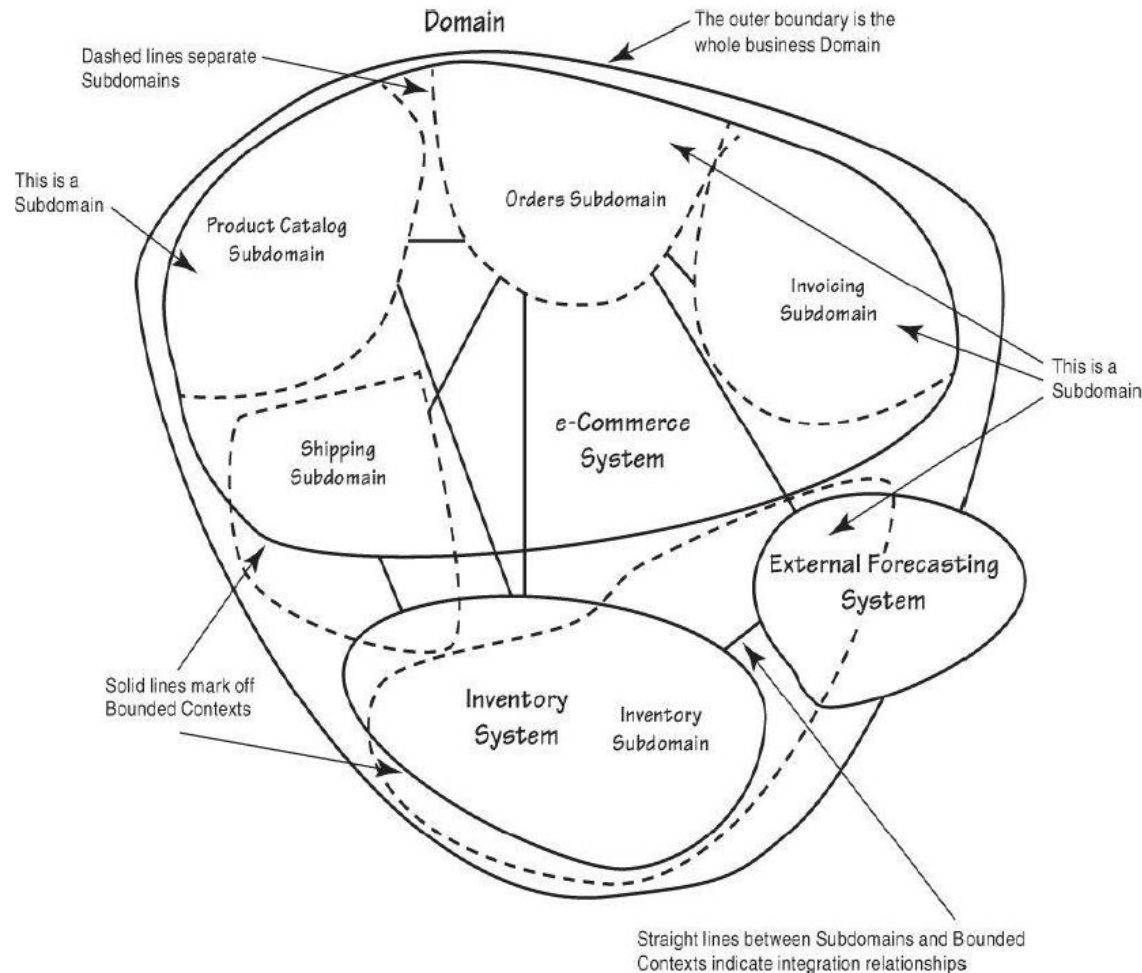
Domains, Subdomains

Bounded contexts

- Domain
 - A sphere of knowledge, influence, or activity
 - The subject area to which the user applies a program is the domain of the software.
- Subdomain
 - We can divide domain into smaller areas
- Bounded Context
 - Each bounded context is the context for its own self-contained domain model
 - Has its own ubiquitous language.
 - You can also view a bounded context as an autonomous business component defining clear consistency boundaries:
 - One bounded context typically communicates with another bounded context by raising events.
(<https://msdn.microsoft.com/en-us/library/jj591575.aspx>)

Domains, Subdomains

Bounded contexts



Domains, Subdomains

Bounded contexts

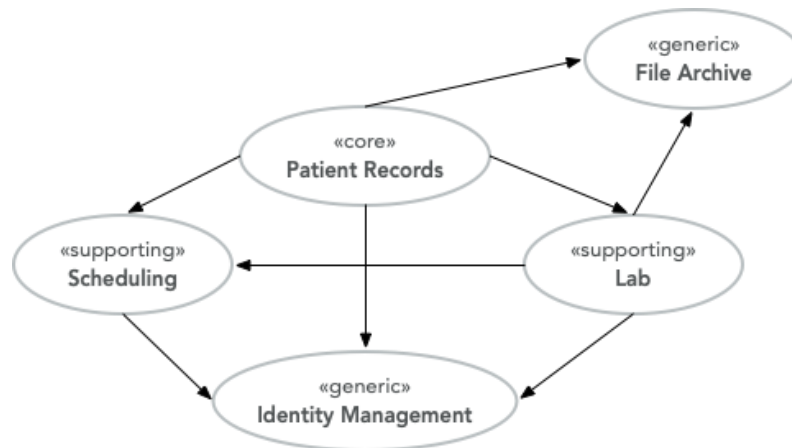
- Core Domain
 - One, implementing main processes
- Supporting Domains
 - Important, but not core
- Generic Domains
 - Not related to business, but required to the solution

Domains, Subdomains

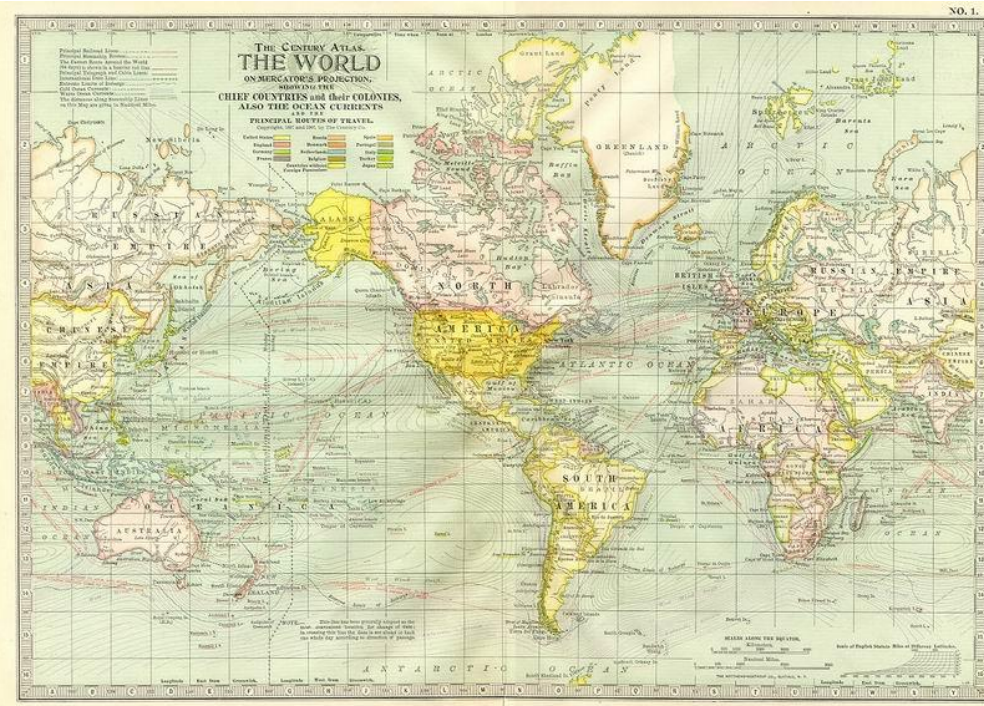
Bounded contexts

■ Core vs. Supporting vs. Generic Domains

1. **Patient Records** for managing patient medical records (personal information, medical history, etc.).
2. **Lab** for ordering lab tests and managing test results.
3. **Scheduling** for scheduling appointments.
4. **File Archive** for storing and managing files that are attached to the patient records (such as different documents, X-ray pictures, scanned paper documents).
5. **Identity Management** for making sure the right people have access to the right information.



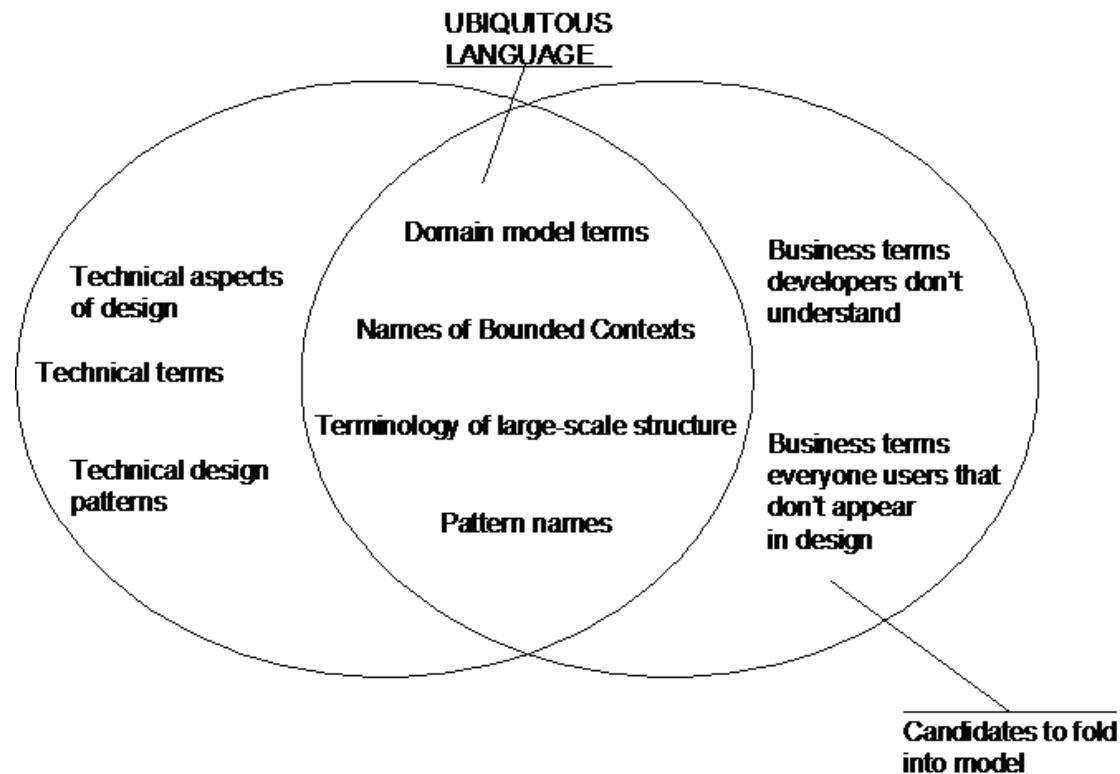
Domain Model



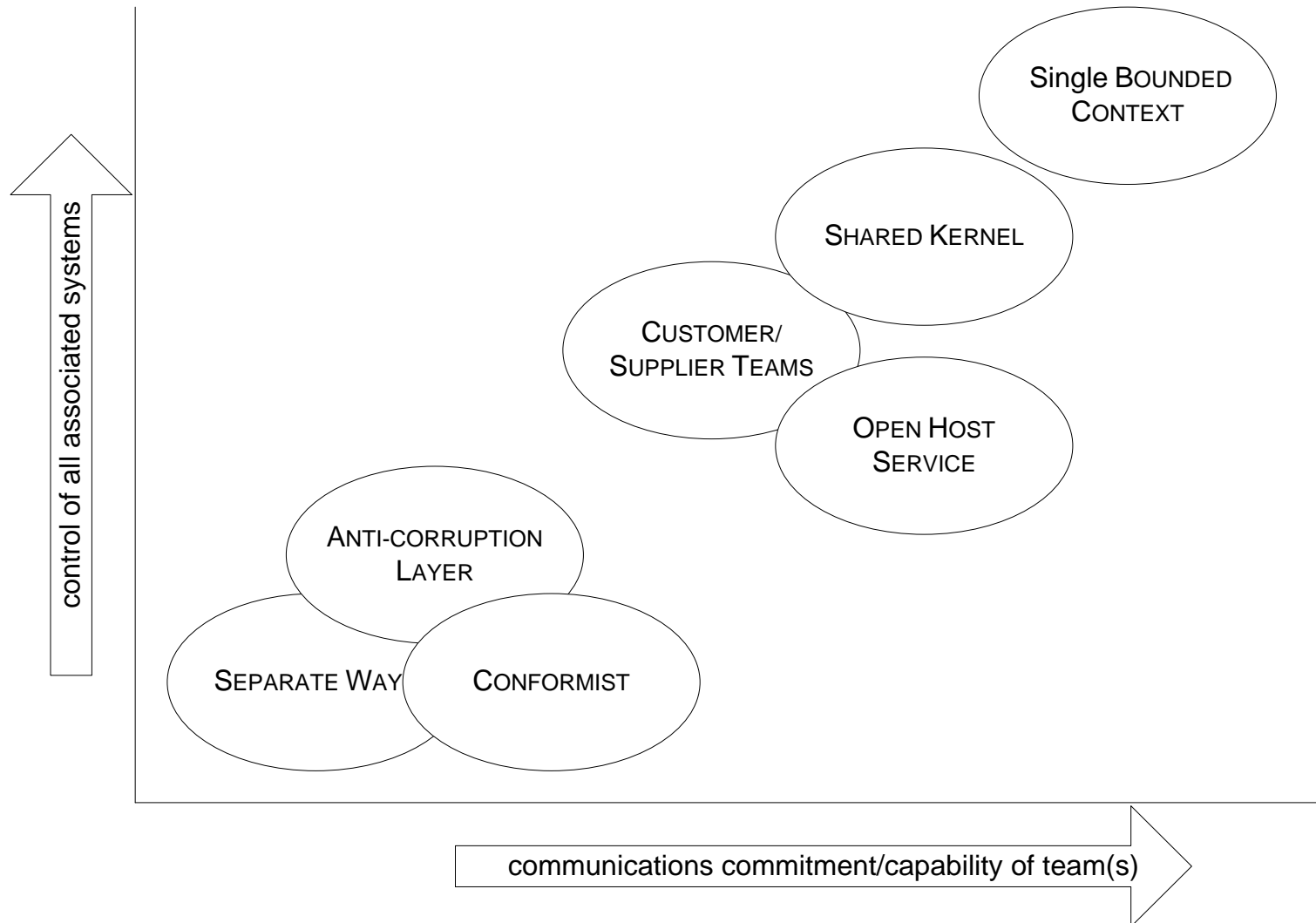
Domain Model

- A *system of abstractions* that describes *selected* aspects of a domain and can be *used* to solve problems related to that domain.
 - Not “as realistic as possible”
 - Useful relative to specific set of domain scenarios
 - Pojęcie modelu anemicznego
- Jak opisujemy model?
 - User stories: as a [role], I want [feature] so that [benefit]
 - Use cases
 - UML, Word, Prototypes, Drawings, ...

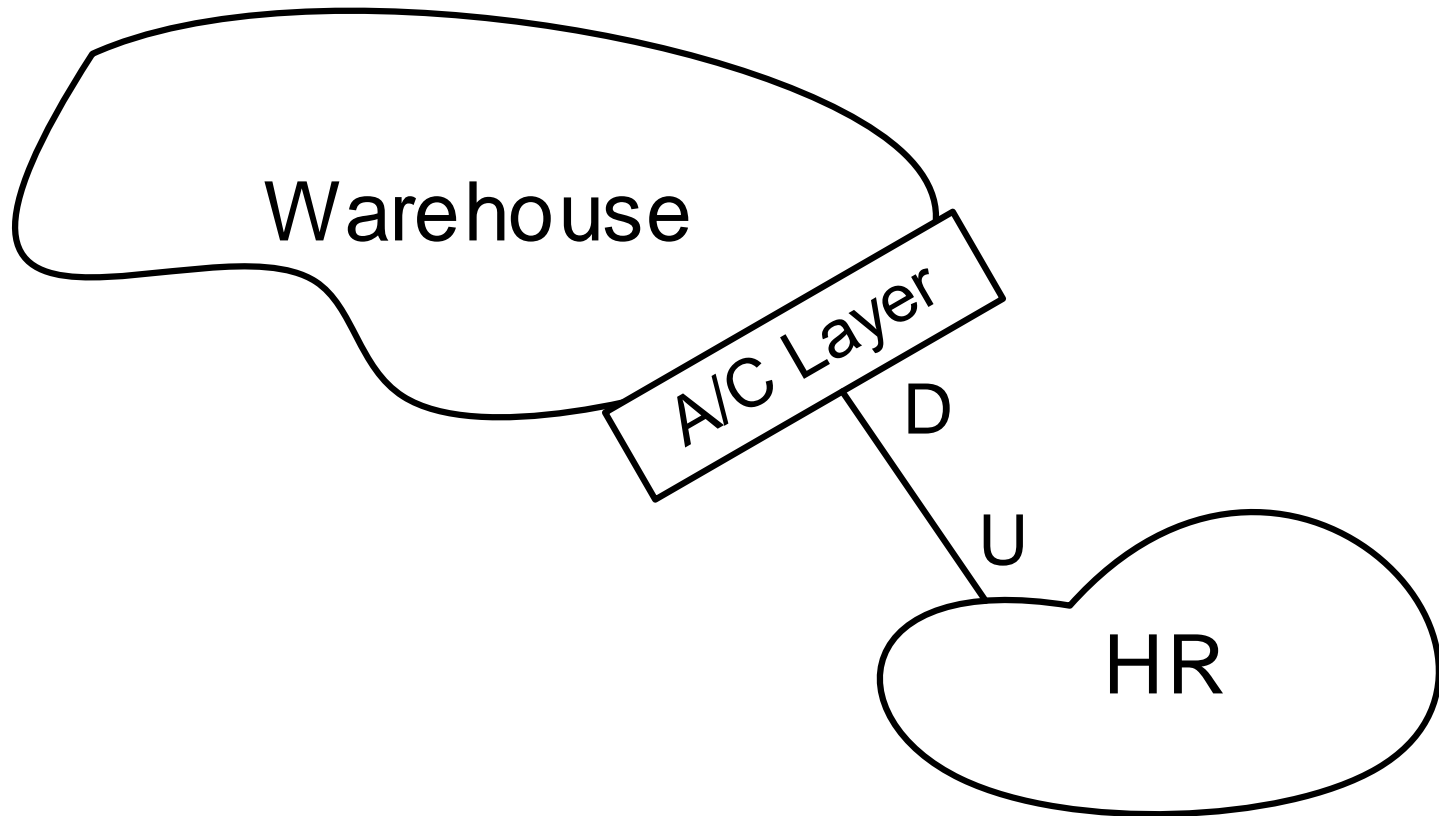
Ubiquitous language



Context Maps



Context Maps



Architecture & DDD

- DDD doesn't require any specific architecture
- The goal is to use right architecture together with right architecture patterns what comes from quality requirements
 - avoid patterns' overuse
- **Architecture is not a coolness factor**

Architecture & DDD

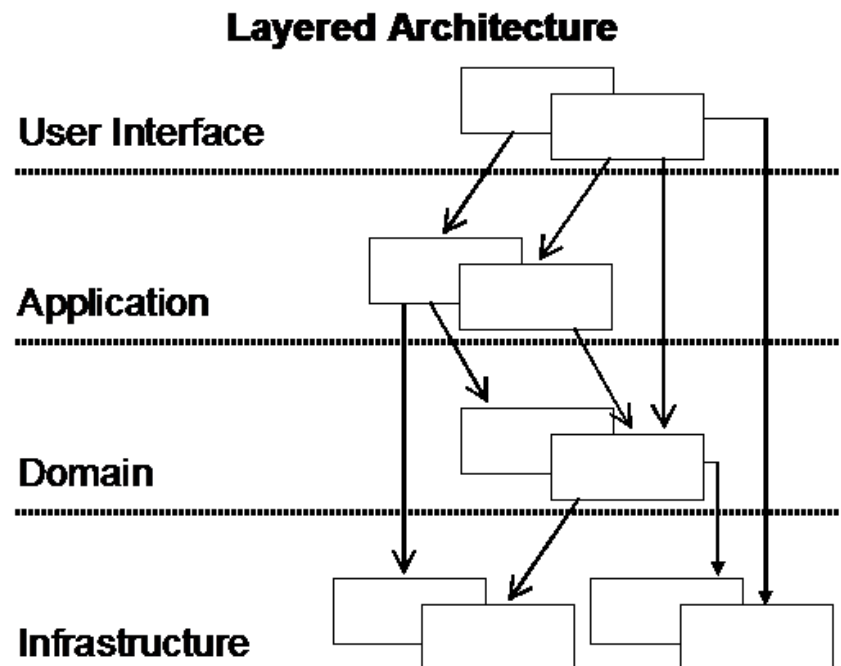
- Why Layers?
 - separate different responsibilities across layers and reduce dependences in a solution
- Very common and well adopted pattern in many types of applications web, enterprise, desktop
- Example layers:
 - business logic, UI, data access

Architecture & DDD

- Layered architecture can be:
 - Strict
dependency only to layer below
 - Relaxed
dependency possible to all layers below

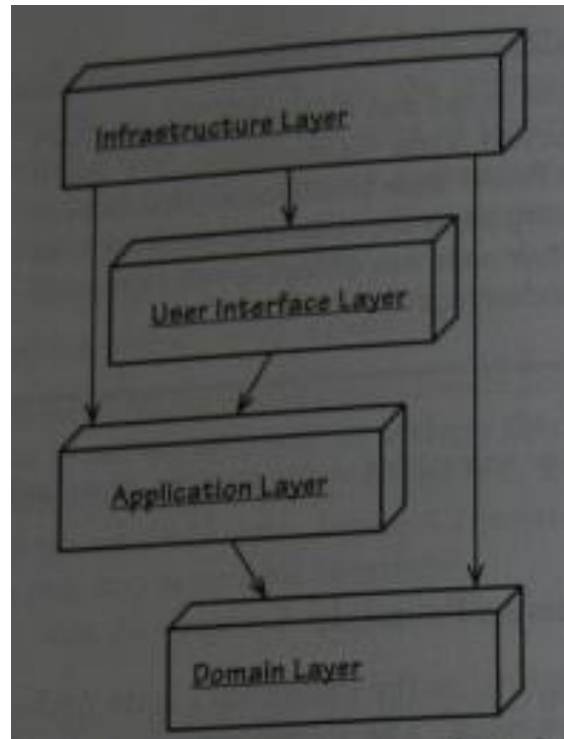
Architecture & DDD

- Architektura warstwowa (tradycyjnie)
 - Presentation Layer (User Interface)
 - Application Layer
 - Domain Layer
 - Infrastructure Layer

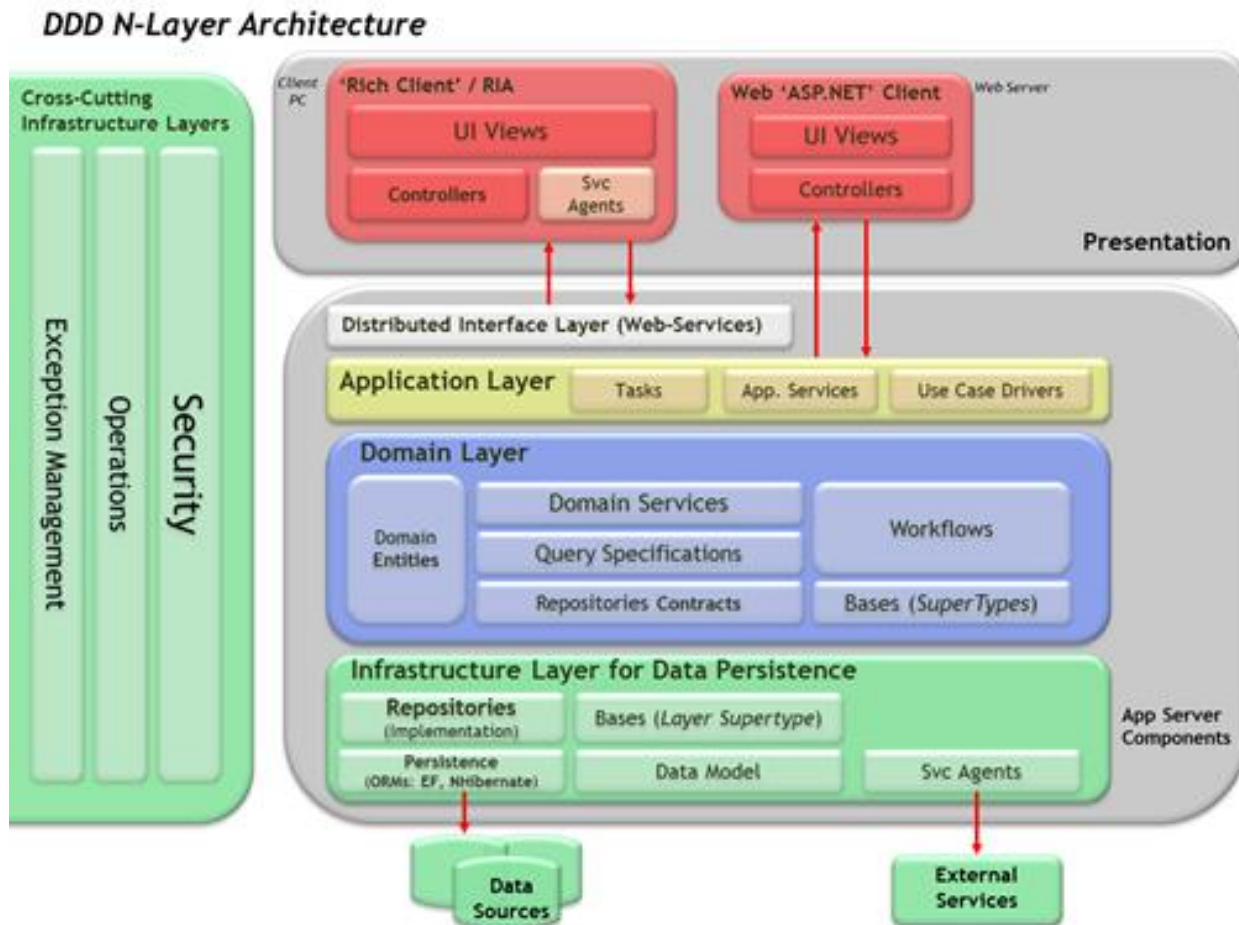


Architecture & DDD

- Architektura warstwowa (raz jeszcze)



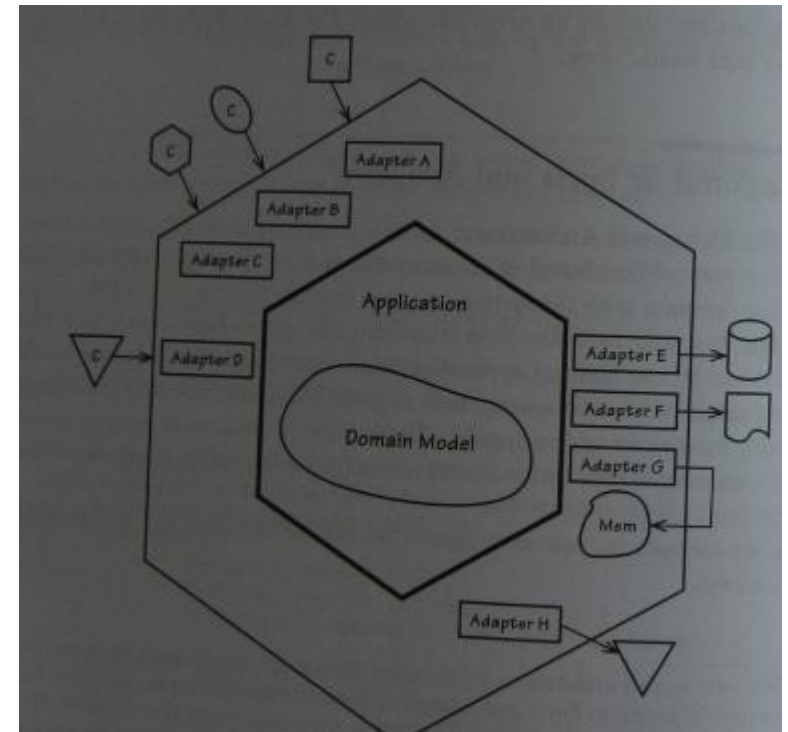
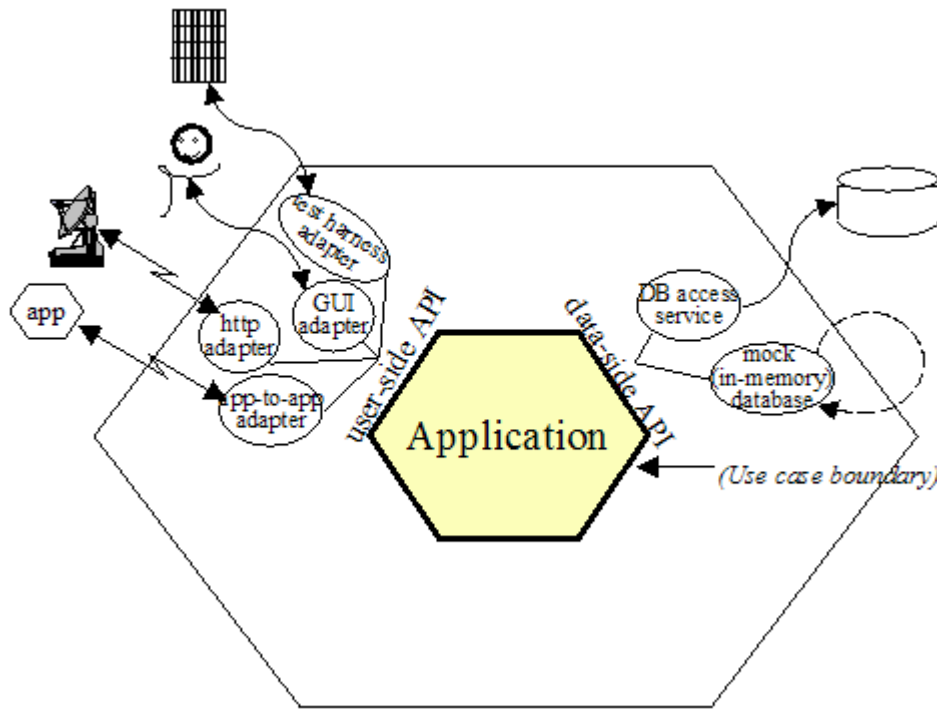
Architecture & DDD



Źródło: <http://blogs.msdn.com/b/cesardelatorre/archive/2010/03/26/our-brand-new-ddd-n-layer-net-4-o-architecture-guide-book-and-sample-app-in-codeplex.aspx>

Architecture & DDD

- Hexagonal or Ports and Adapters



Architecture & DDD

- Port
 - a place where application communicates with external parties
 - The protocol for a port is given by the purpose of the conversation
 - The protocol takes the form of an API
- Adapter
 - is to translate external party to the API and vice-versa
- Internal hexagon
 - Core logic based on functional requirements, use cases, user stories
 - Application Boundary

Składowe modelu

■ Entities

- pojedyncze „rzeczy”
- mają identyfikator biznesowy i cykl życia
- zawierają logikę operującą na encji
- wartości atrybutów mogą się zmieniać
- Przykładowe identyfikatory biznesowe
 - Nr faktury, Nr VIN, Nr rejestracyjny (?), PESEL (?)
- Elementy związane z identyfikatorami
 - Późne vs. wczesne ich nadawanie
 - Surrogate identity

Składowe modelu

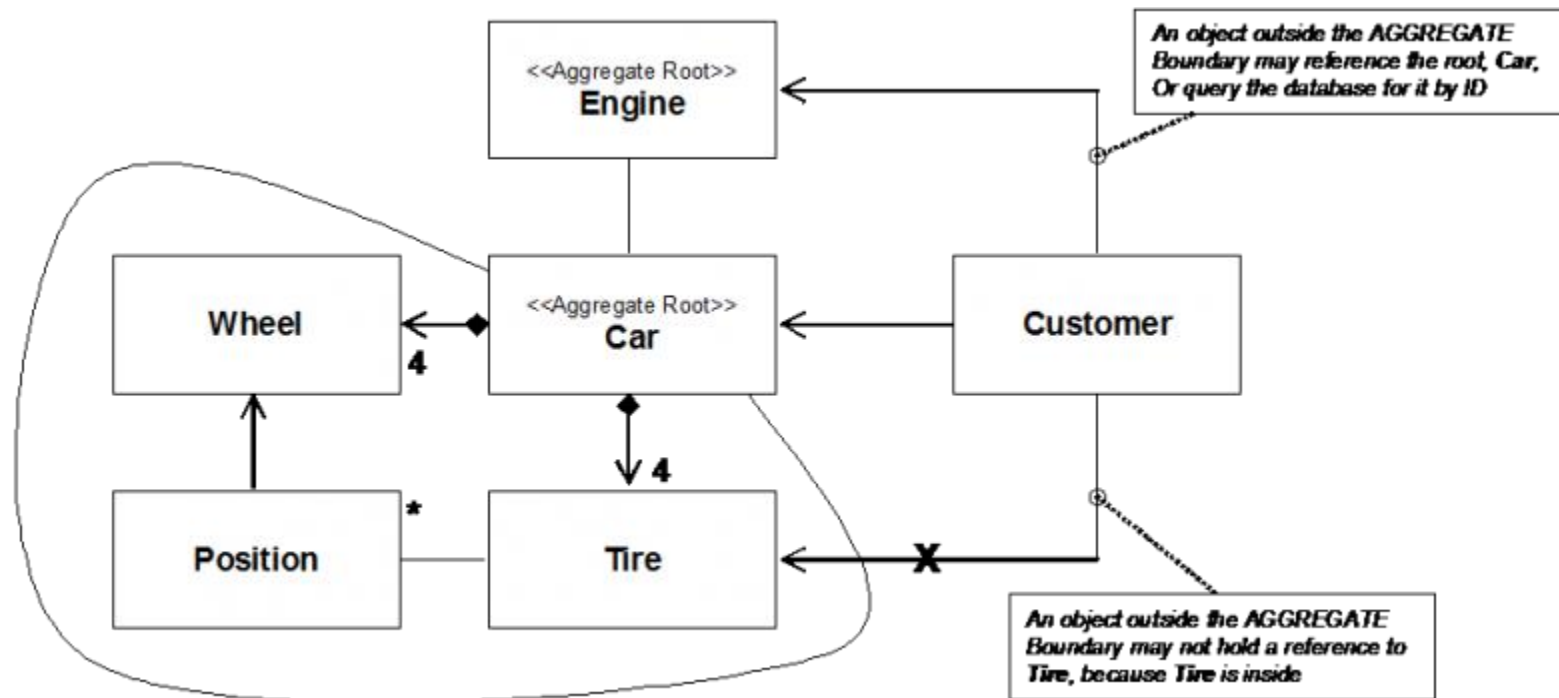
- Value objects
 - nie mają ID globalnego (biznesowego)
 - cykl życia zależy do cyklu życia encji
 - powiązane z encjami, opisują je
 - wartości atrybutów nie mogą się zmieniać
 - porównywanie powinny być przez sprawdzenie wszystkich wartości

Składowe modelu

- Aggregates
 - Zbiór entities i value objects
 - Rozpatrywana jako „single business unit”
 - Dostęp tylko przez „aggregate root” (który jest entity)
 - Ale wewnątrz mogą być referencje do innych agregatów
 - Operacje
 - Zapytania: tylko w kontekście „aggregate root”
 - Update: spójność w zakresie jednego agregatu
 - Delete: usunięcie aggregate root implikuje usunięcie całego agregatu

Składowe modelu

- Aggregates, przykład



Składowe modelu

- Aggregates
 - Transactional consistency vs. eventual consistency
 - Problemy
 - Concurrency (transactions, locks)
 - Performance
 - Scalability
 - Jak wygląda realizacja powyższych wymagań w przypadku dużych agregatów?

Składowe modelu

- Jak zatem należy tworzyć agregaty?
 - Jak najmniejsze
 - Referencje pomiędzy agregatami powinny być przez ID, a nie przez referencje w kodzie
 - W ramach agregatu – transactional consistency, pomiędzy agregatami – eventual consistency
 - Dobrze zdefiniować reguły spójności wewnątrz agregatu
 - Jedna operacja/transakcja powinna modyfikować tylko jeden agregat

Składowe modelu

■ Services

■ Application Services

- Wyrażają use-cases lub user stories
- Koordynacja operacji wykonywanych na modelu
- Dostarczają danych i funkcjonalności do UI
- Obsługują: persistence, security, event-based notifications, composing e-mails for users
- Brak logiki biznesowej

Składowe modelu

■ Services

■ Domain Services

- Bezstanowe operacje operujące na modelu
 - Jeśli jakaś operacja nie należy do żadnej encji lub value object, przenosimy ją do domain service
- Element modelu zawierający logikę biznesową

■ Infrastructure Services

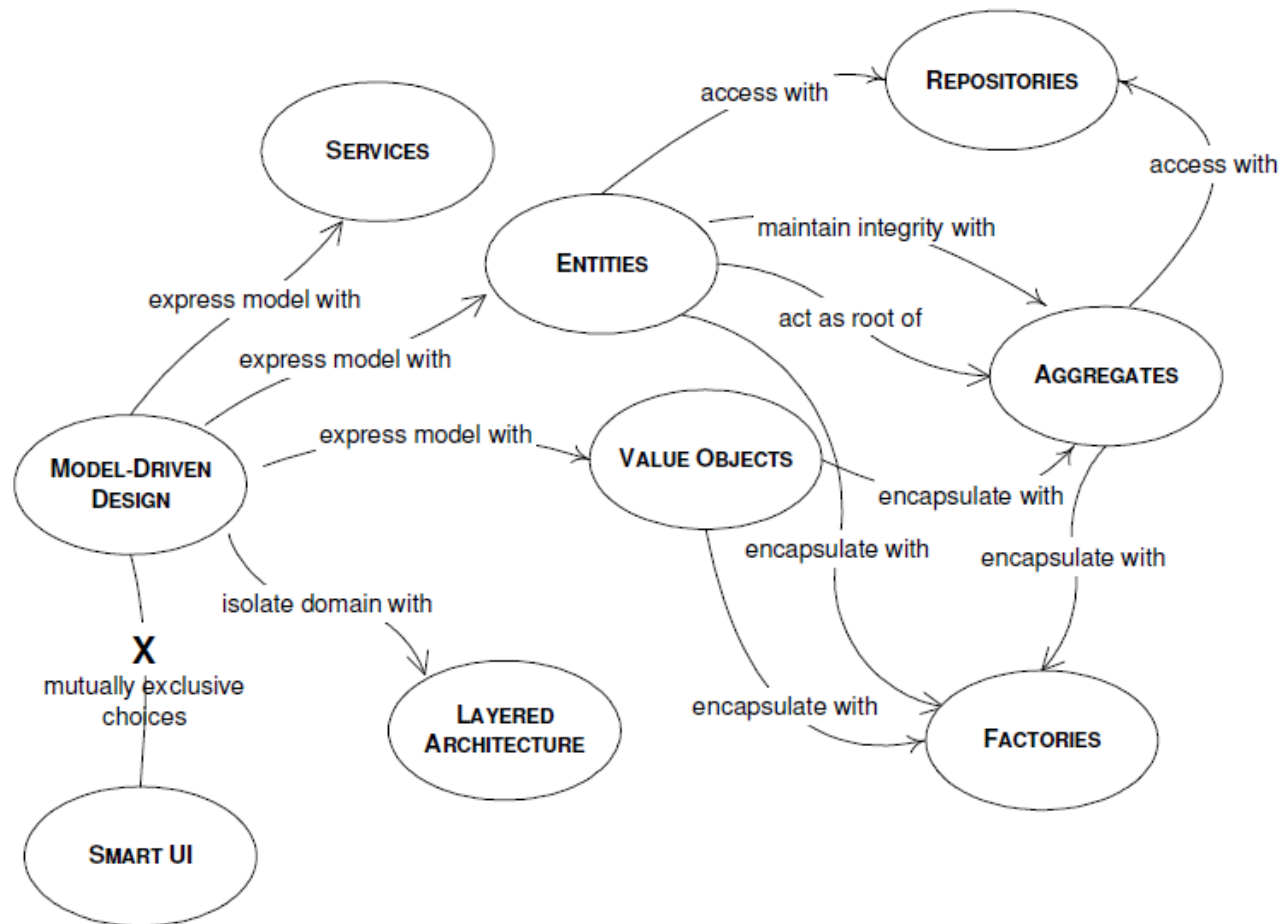
- Implementują współpracę z zewnętrznymi zasobami, np.
 - persistence,
 - Messaging

Składowe modelu

- Repositories
 - odpowiedzialne za utrwalanie agregatów
 - wprowadzają separację modelu od utrwalania
 - Myślimy o repozytorium jako kolekcji agregatów
 - Każdy agregat ma co najmniej jedno repozytorium
 - I budujemy repozytoria tylko dla agregatów
- Factories
 - tworzą encje i agregaty

Składowe modelu

■ Podsumowanie



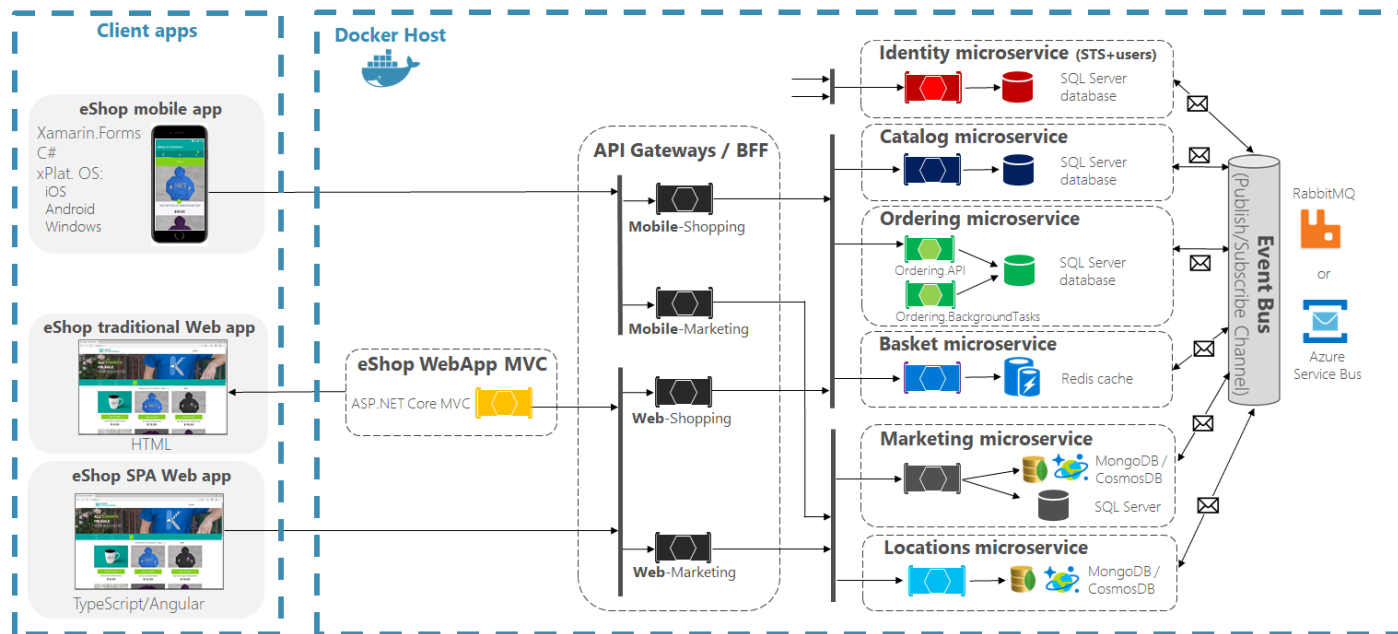
Domain Driven Design

- Dodatkowe elementy
 - Dbłość o spójność modelu
 - Refaktoryzacja
 - Testowalność, utrzymywanie testów
 - jednostkowe
 - integracyjne
 - Continuous integration

DDD in .NET architecture

- <https://docs.microsoft.com/en-US/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/>
- <https://github.com/dotnet-architecture/eShopOnContainers>

eShopOnContainers reference application
(Development environment architecture)



Ćwiczenie

- Zamówienie nr 1/2/3, data
 - Kurtka, 150 zł, 3 szt., 5XL
 - Sweter, 100 zł, 2 szt. 3XL
 - Spodnie 500 zł, 1 szt. 5XL

Ćwiczenie

Obiekt 1 (E/V)

Obiekt 1 (E/V)

Obiekt 1 (E/V)

Zamówienie nr 2, data

Kurtka, 150 zł, 1 szt., 5XL

Sweter, 100 zł, 2 szt. 3XL

Spodnie 500 zł, 1 szt. 5XL

Domain Driven Design

■ Literatura

■ Wprowadzenie w DDD

- <http://msdn.microsoft.com/en-us/magazine/dd419654.aspx>
<http://www.infoq.com/articles/ddd-in-practice>
- <http://www.infoq.com/minibooks/domain-driven-design-quickly>
- https://qconsf.com/sf2007/dl/QConSF2007/slides/public/EricEvans_StrategicDesign.ppt?path=/QConSF2007/slides/public/EricEvans_StrategicDesign.ppt

■ Services

- <http://gorodinski.com/blog/2012/04/14/services-in-domain-driven-design-ddd/>
- <http://lostechies.com/jimmybogard/2008/08/21/services-in-domain-driven-design/>