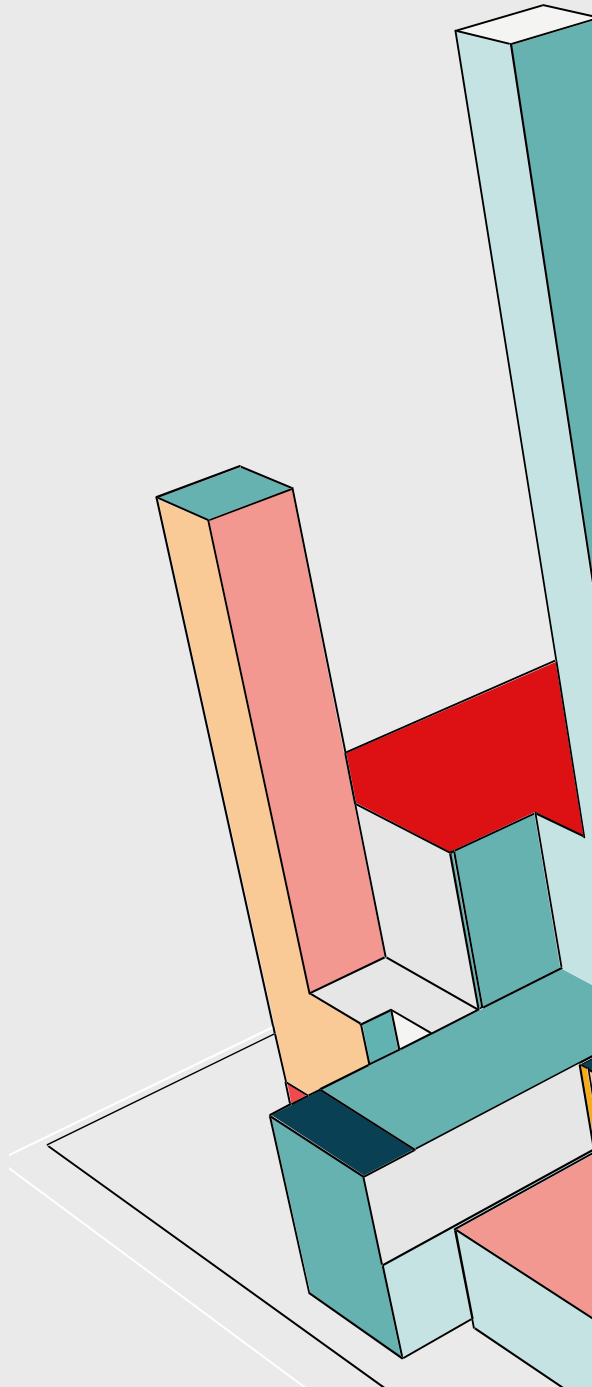
КОНСТРУИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ПЛАН ЛЕКЦИИ № 7

Самое важное в DDD

1. Основные принципы
2. Разница между анемичной и богатой моделью
3. Изоляция доменной модели
4. DDD трилемма

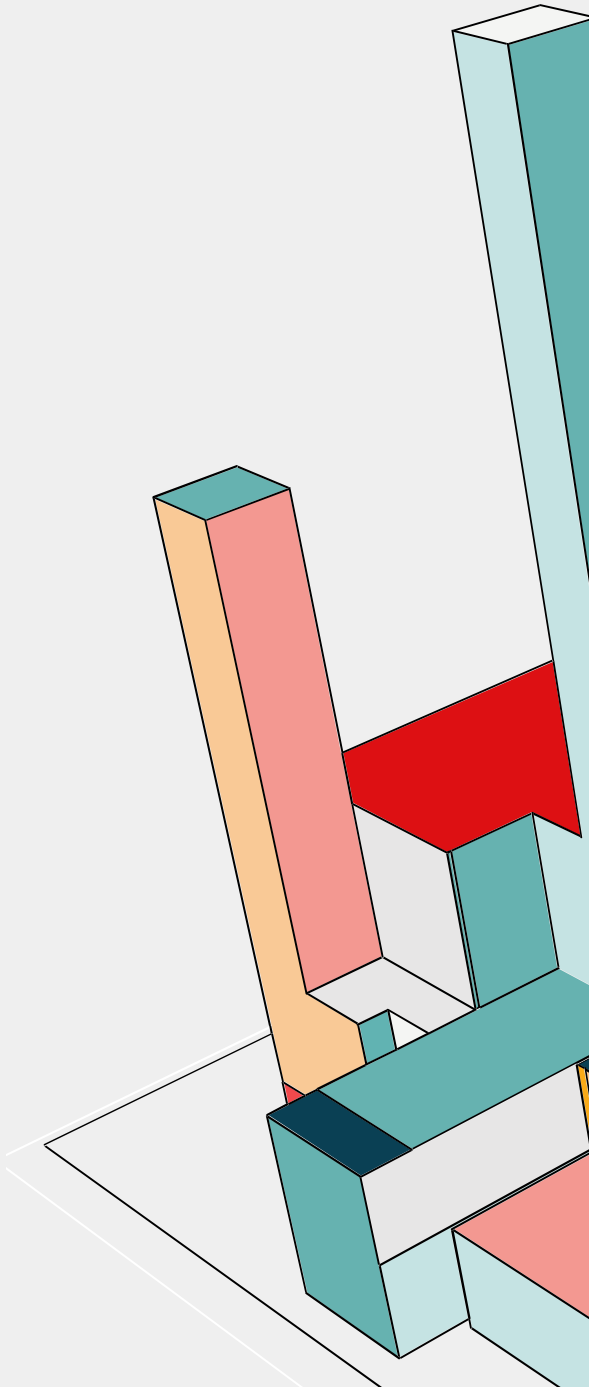


ЗАЧЕМ ИЗУЧАТЬ ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ?

Ради холивора:
DDD vs Clean Architecture...

Ради маркетинга:
DDD это исчерпывающее описание бизнес-процессов компании

Ради инжиниринга:
подход DDD не связан с технологиями



ЗАЧЕМ ИЗУЧАТЬ ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ?

Ради холивора

01

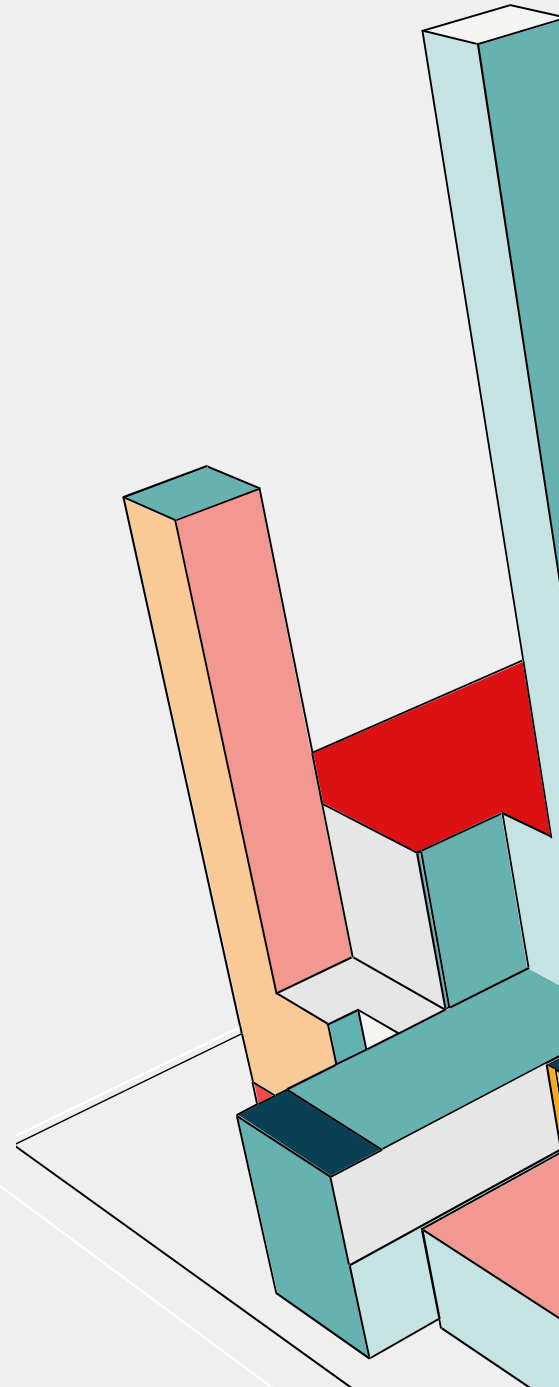
Это про то, как все писать на микросервисах/сервисах/лямбдах

02

Мы используем Clean Architecture/DRY/SOLID/... и у нас все хорошо

03

Мы живем на SOAP/CORBA/REST



ЗАЧЕМ ИЗУЧАТЬ ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ?

Ради маркетинга

01

Код и бизнес должны говорить на одном языке, формализуем этот язык

02

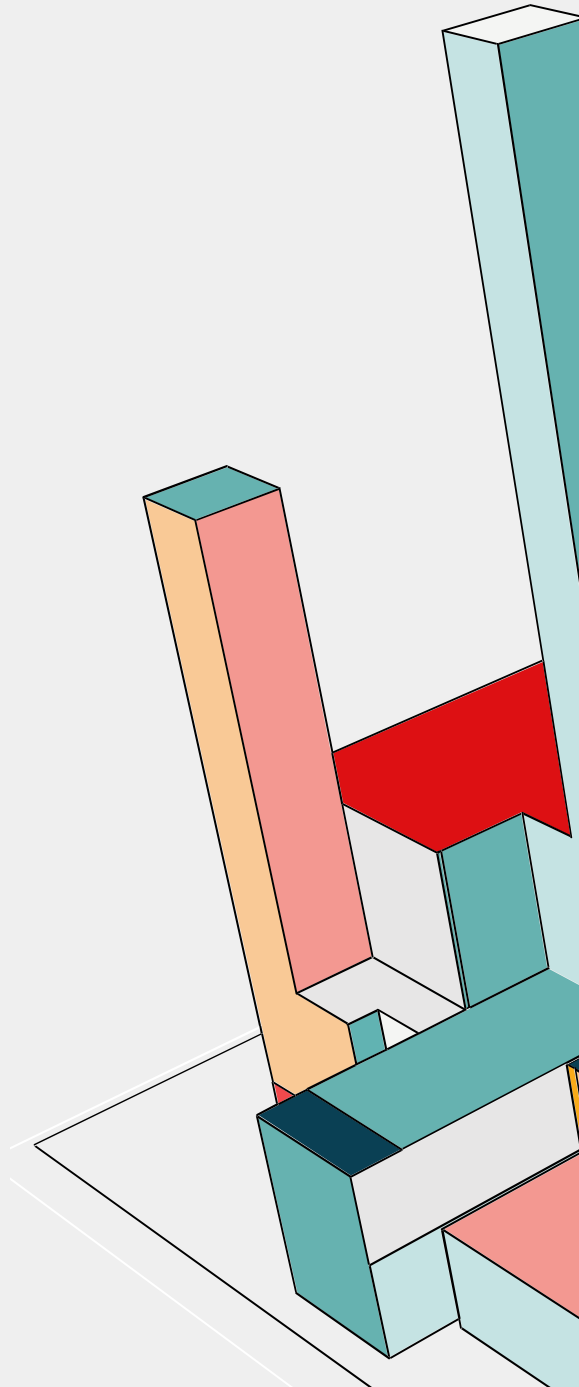
Каждый участник влияет на бизнес

03

Внедрить технологию ИКС

04

Обеспечивает основу для стратегического и тактического моделирования и планирования



ЗАЧЕМ ИЗУЧАТЬ ПРЕДМЕТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ?

Ради инжиниринга

01

Микросервисы или полновесные сервисы или ...

02

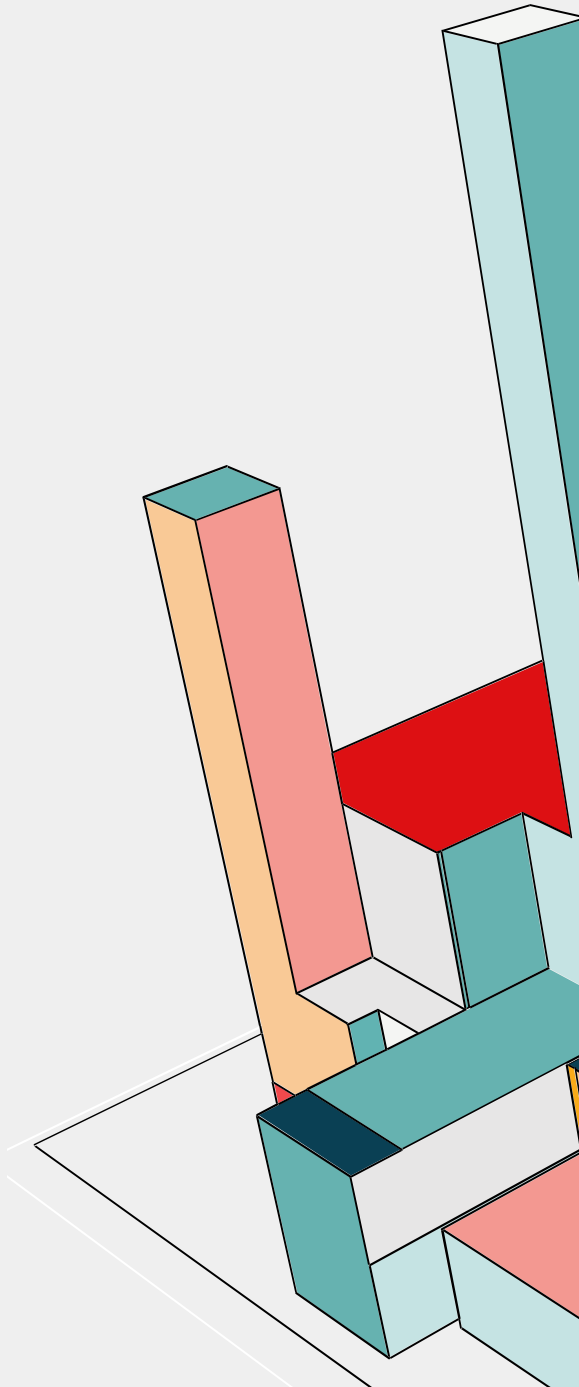
Развитие знаний о бизнесе, декларативно описать все сценарии бизнес-логики, понижаем порог входа

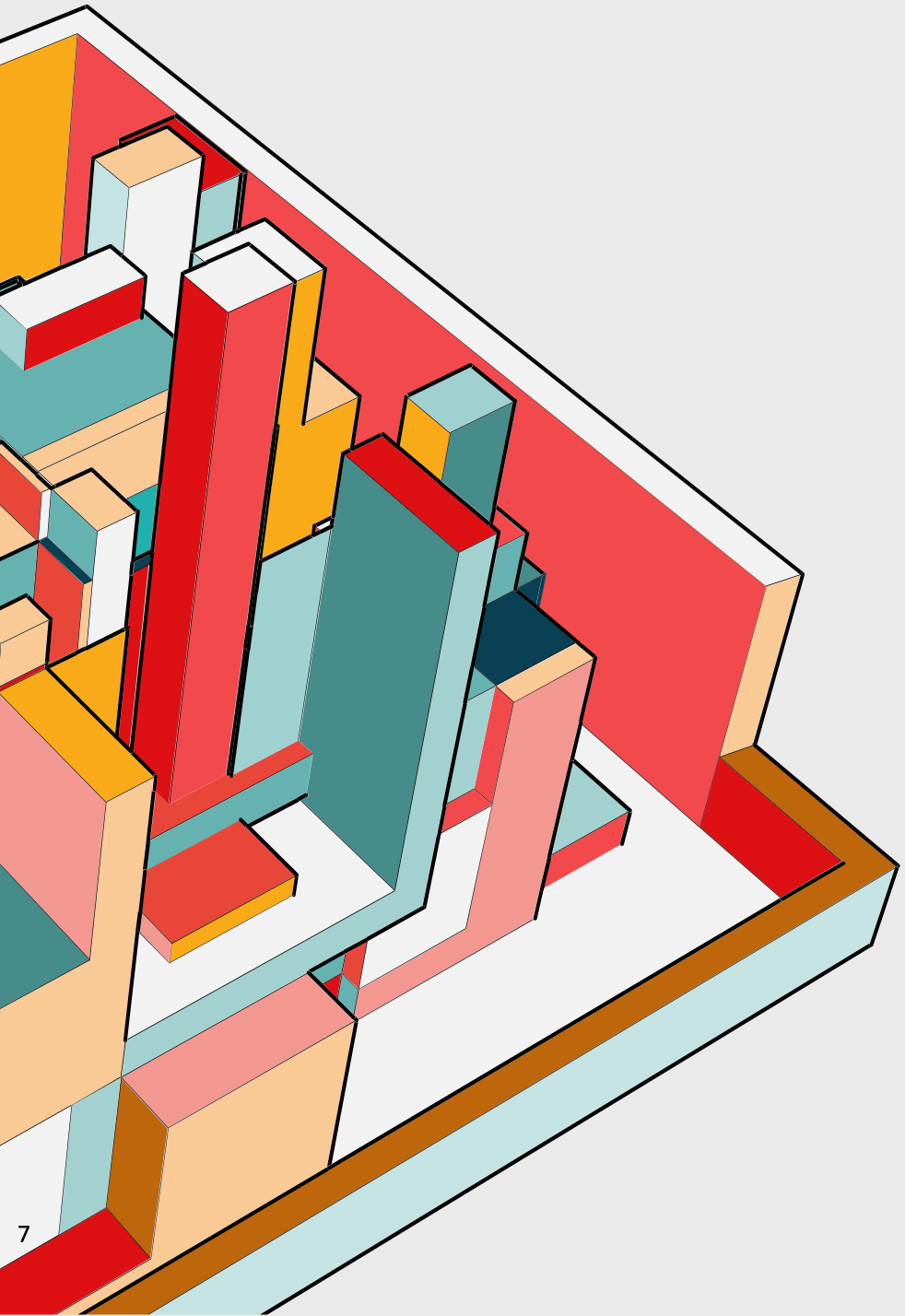
03

Использование технологий для обеспечения ценности

04

Улучшение наблюдаемость бизнес-процесса





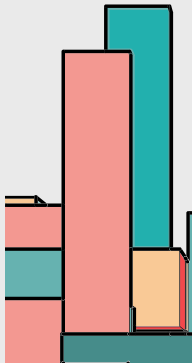
ОСНОВНЫЕ ПРИНЦИПЫ

ОСНОВНЫЕ ПРИНЦИПЫ

Ограниченные контексты (bounded contexts)

Единый язык (ubiquitous language)

Фокус на доменной модели

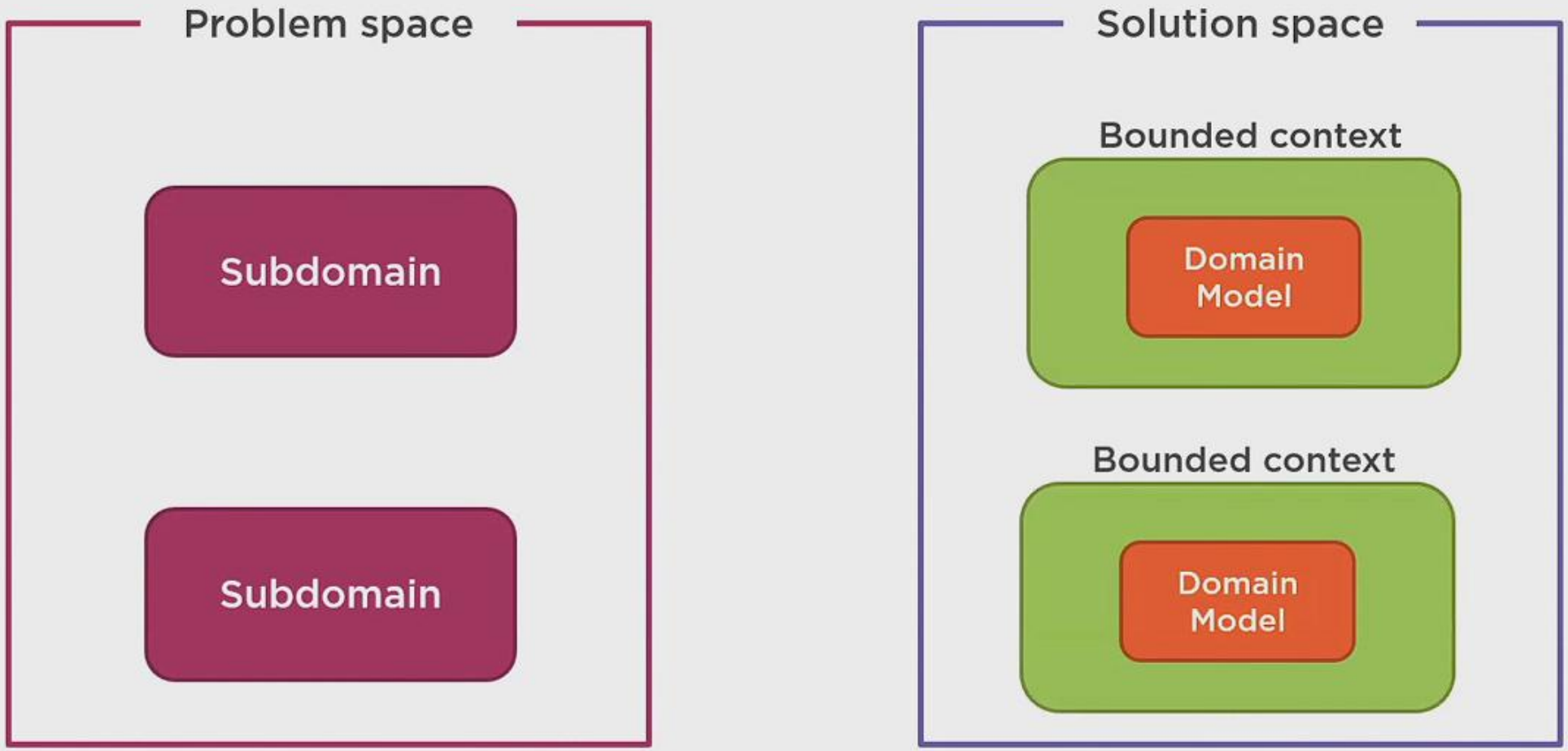


ОГРАНИЧЕННЫЕ КОНТЕКСТЫ

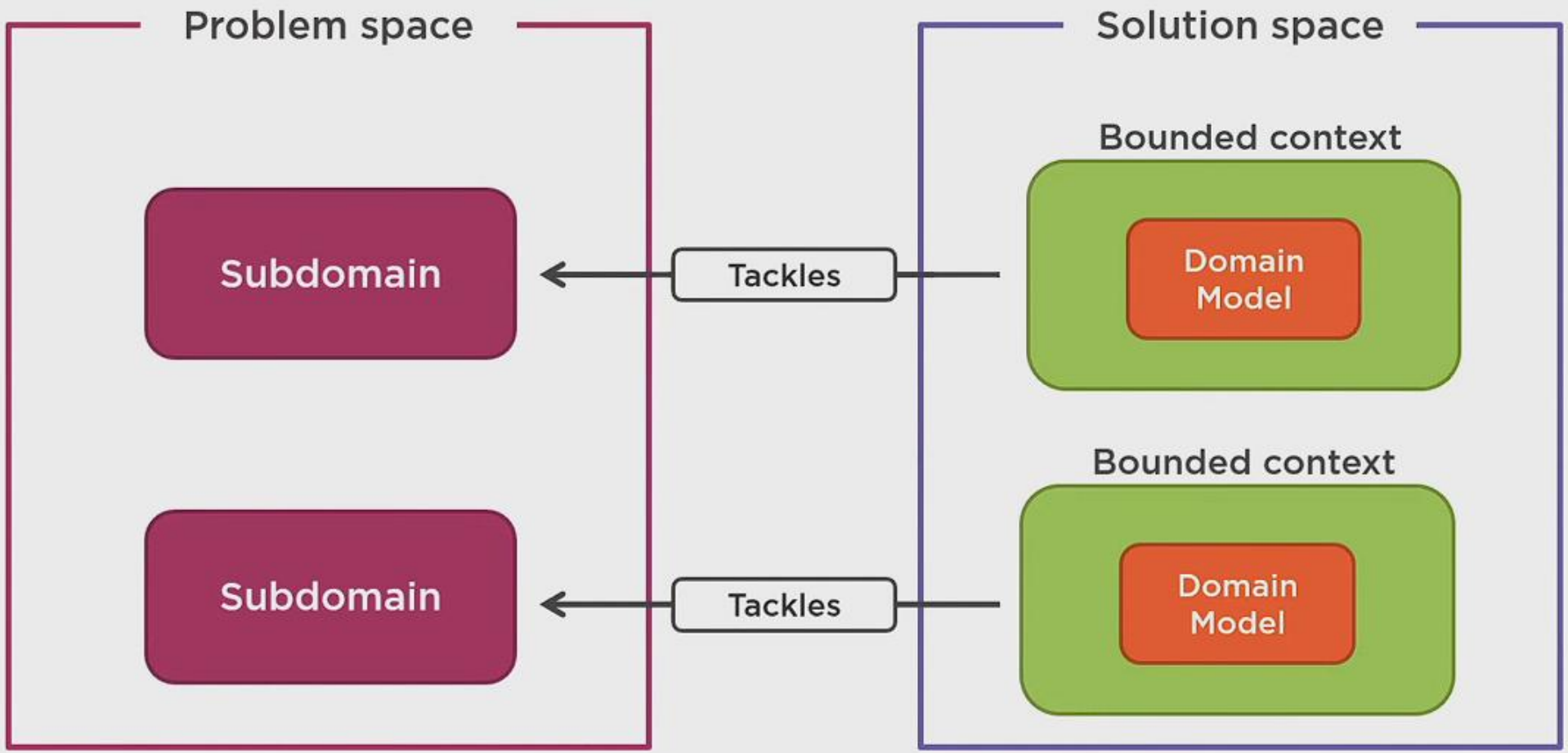
Problem space

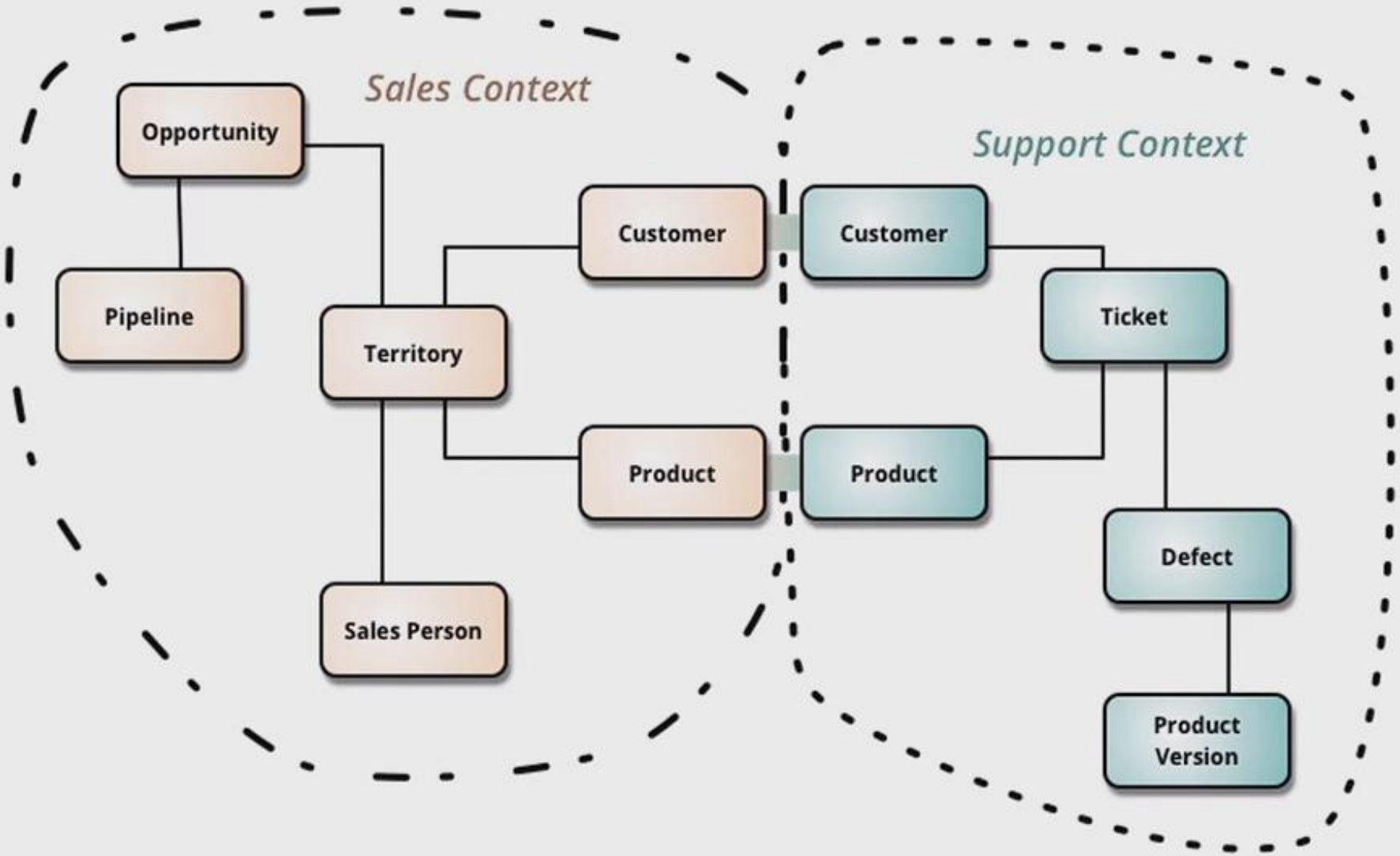
Solution space

ОГРАНИЧЕННЫЕ КОНТЕКСТЫ

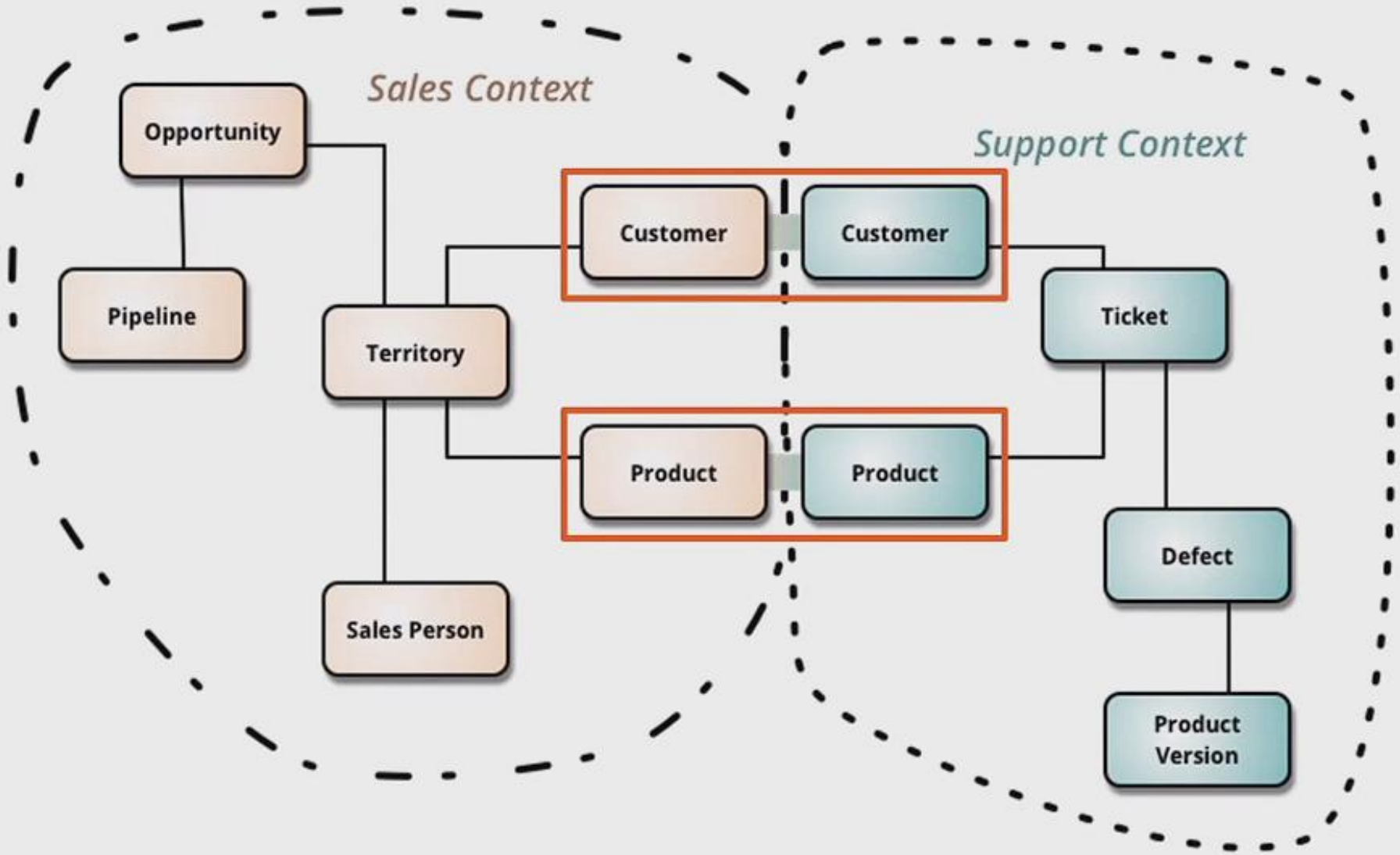


ОГРАНИЧЕННЫЕ КОНТЕКСТЫ





Source: https://martinfowler.com/bliki/BoundedContext.html



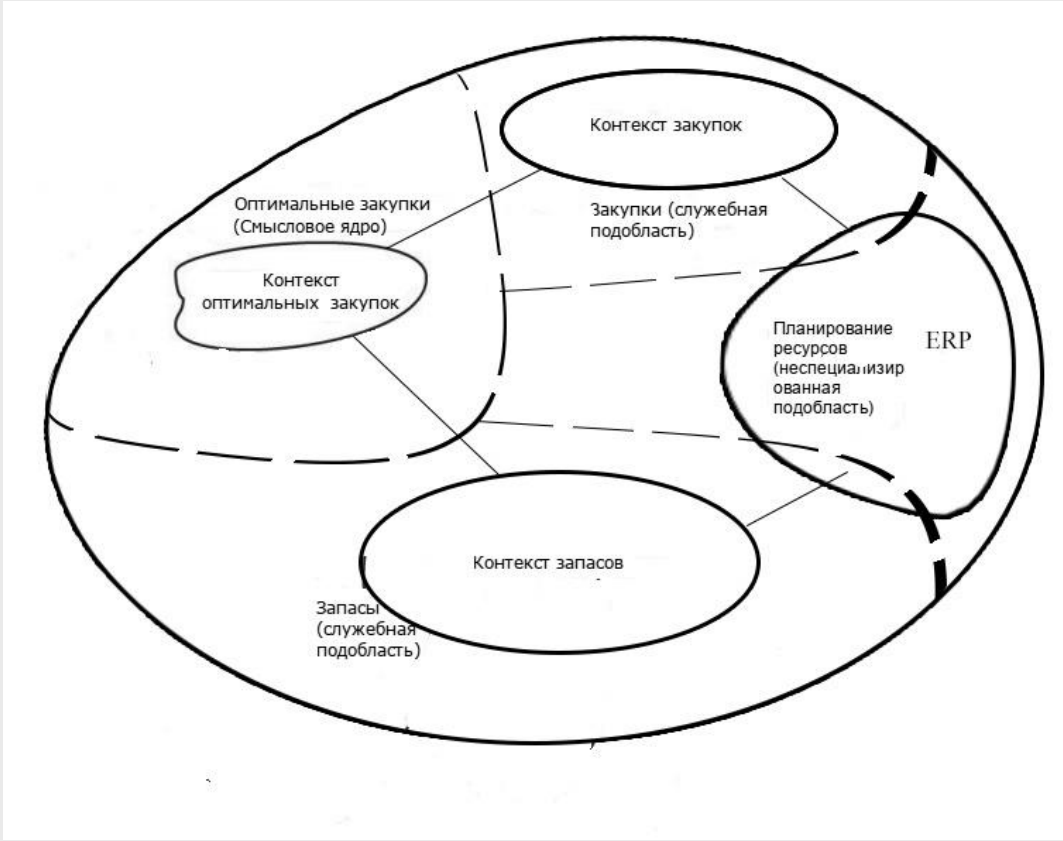
Source: https://martinfowler.com/bliki/BoundedContext.html

СМЫСЛОВОЕ ЯДРО

Смысловое ядро делает организацию гораздо более конкурентоспособной, способной быстро идентифицировать выгодные сделки и гарантировать необходимый уровень запасов.

Для оценки пространства задач в первую очередь необходимо:

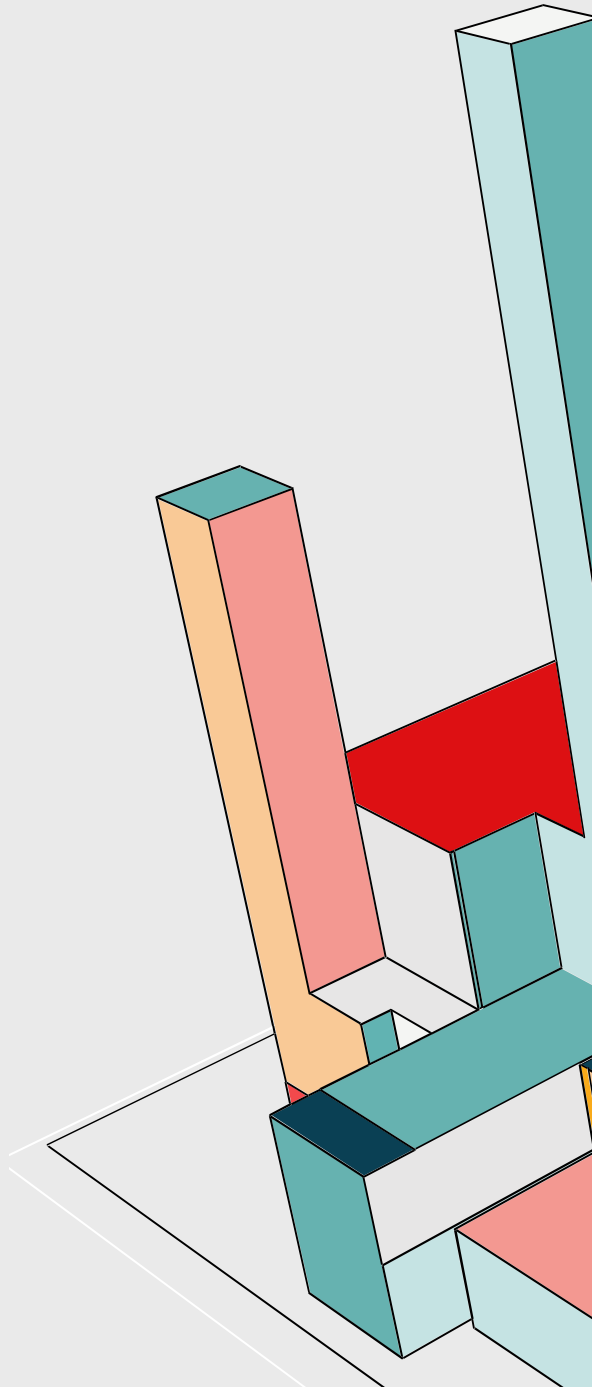
1. Определить, как выглядит стратегическое смысловое ядро
2. Какие концепции должны рассматриваться как часть смыслового ядра
3. Перечислить служебные и неспециализиро ванные подобласти



ЕДИНЫЙ ЯЗЫК

ЕДИНЫЙ ЯЗЫК

1. Создание диаграмм физической и концептуальной предметной областей и нанесение на них меток, обозначающих имена и действия.
2. Создание глоссария с простыми определениями, а также альтернативными терминами с оценкой их перспективности. Таким образом разрабатываются устойчивые словосочетания единого языка.
3. Обсуждение готовых фраз с остальными членами команды, которые не могут сразу освоить глоссарий или другие письменные документы.



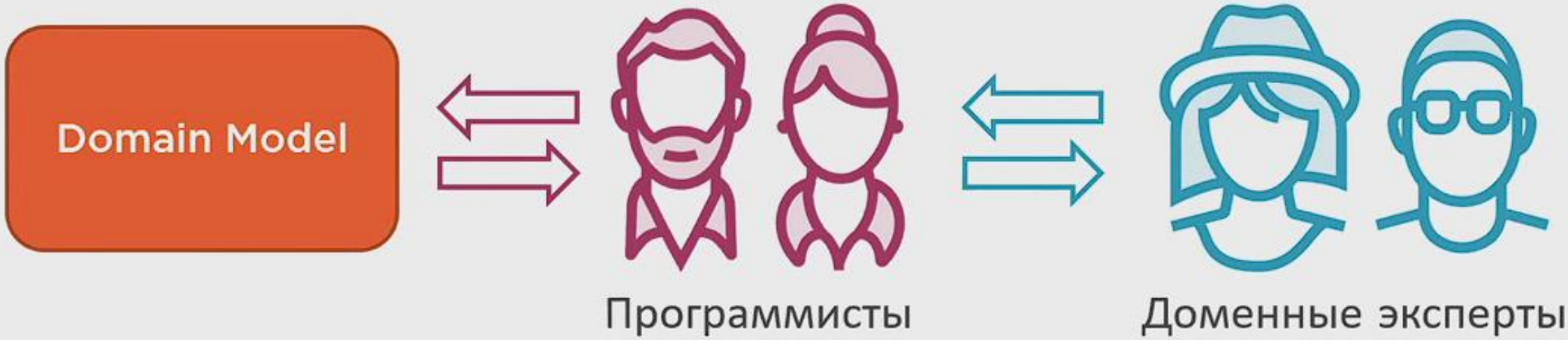
ПРИМЕР

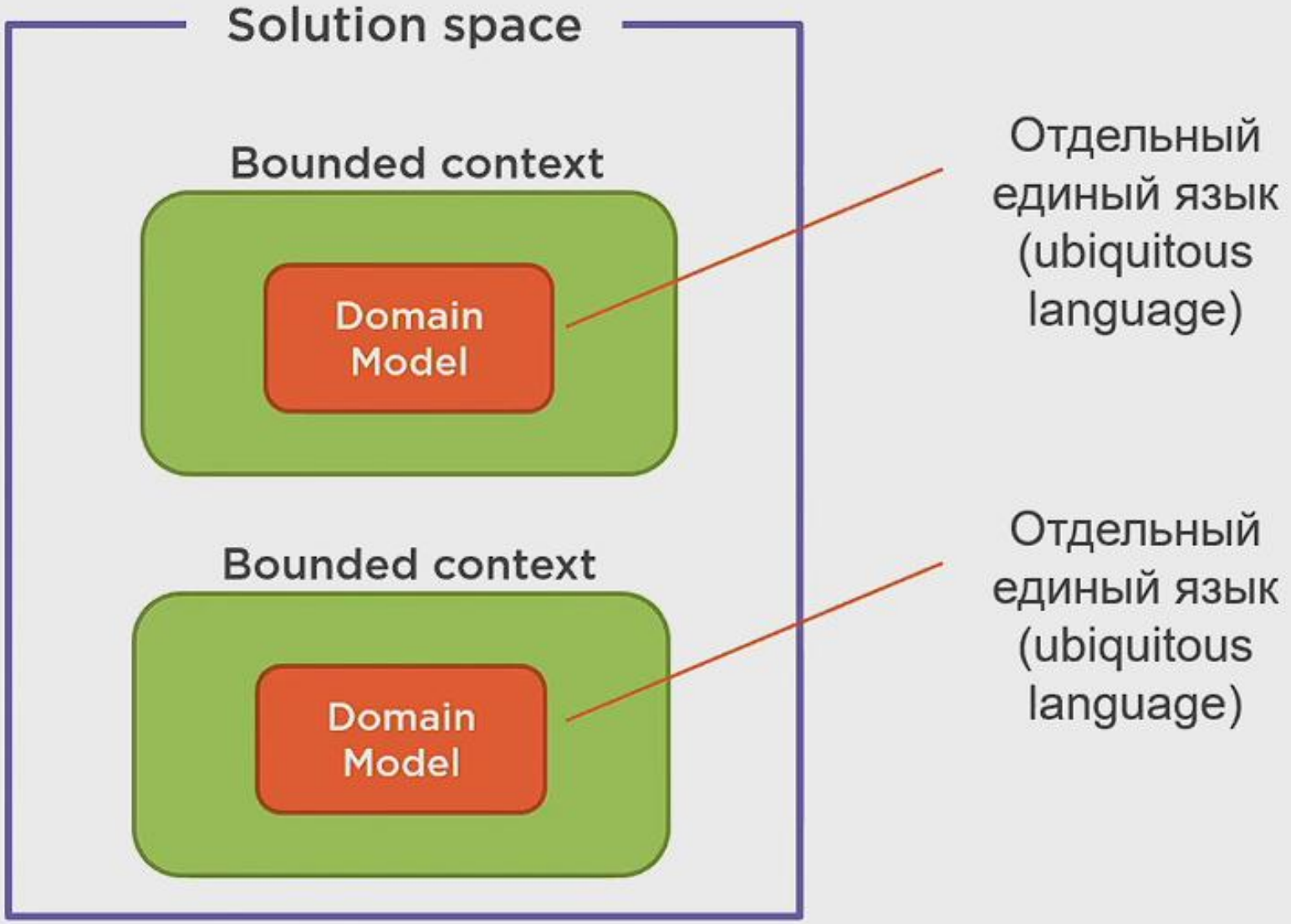
В команде, обсуждая модель применения вакцины от гриппа в виде кода, **произносят фразу**:

«Медсестры назначают вакцины от гриппа в стандартных дозах».

Возможные точки зрения	Итоговый Код
«Кого это волнует? Просто программируйте»	Patient.setShotType(ShotTypes.TYPE_FLU); patient.setDose(dose); patient.setNurse(nurse);
«Мы делаем пациентам прививку от гриппа»	patient.giveFluShot();
«Медсестры назначают вакцины от гриппа в стандартных дозах»	Vaccine vaccine = vaccines.standartAdultFluDose(); nurse.administerFluVaccine(patient,vaccine);

Единый язык требует постоянной поддержки и рефакторинга


Domain Model
Программисты
Доменные эксперты



Отдельный единый язык (ubiquitous language)

Отдельный единый язык (ubiquitous language)

Ubiquitous language

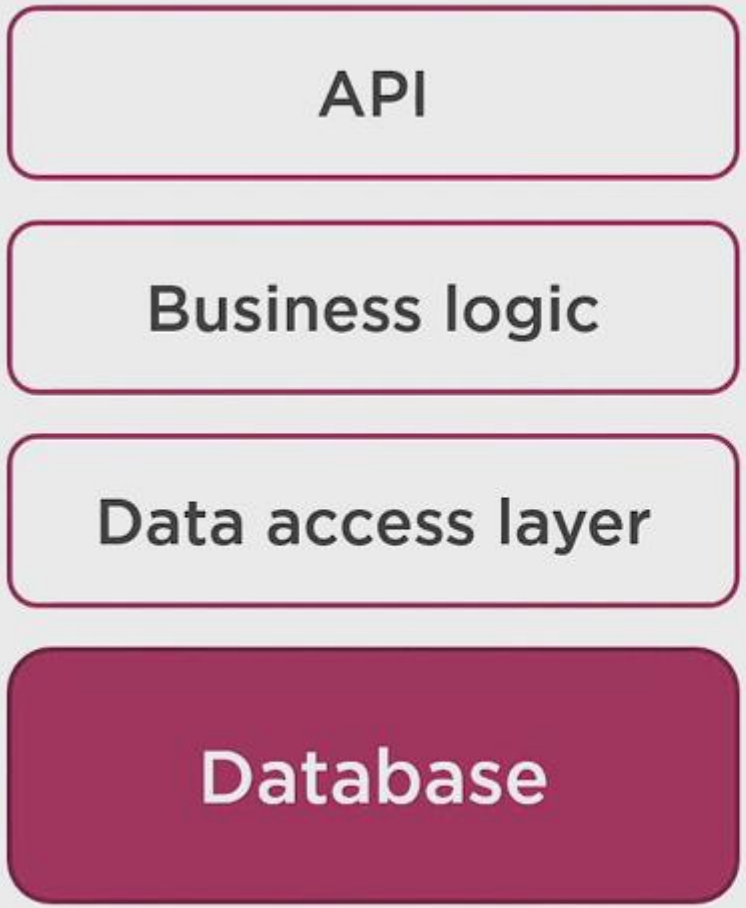
vs.

DSL

ФОКУС НА ДОМЕННОЙ МОДЕЛИ

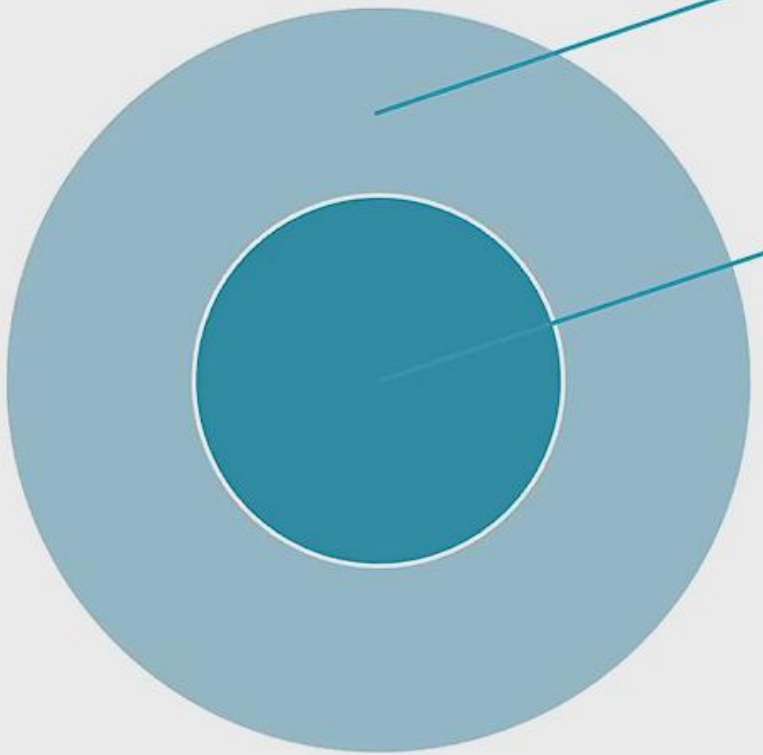
Доменная модель –
краеугольный камень
всего приложения

ТИПИЧНАЯ РАЗРАБОТКА НЕ ПО DDD



Inside-out development

А ЭТО ПО DDD



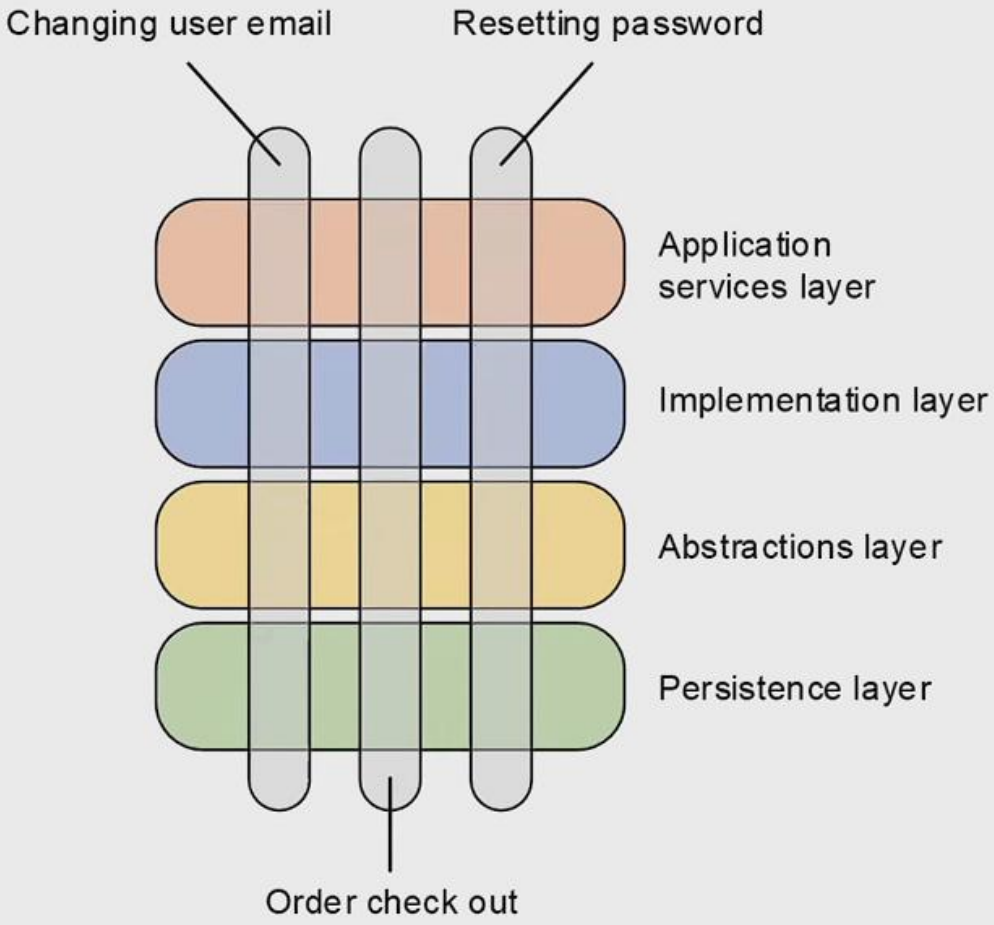
Application
Services

Infrastructure

Business logic
(domain model)

Middle-out development

НЕ РАЗМАЗЫВАЙТЕ ДОМЕННУЮ ЛОГИКУ



Доменная логика должна иметь отдельное, четко обозначенное место в проекте

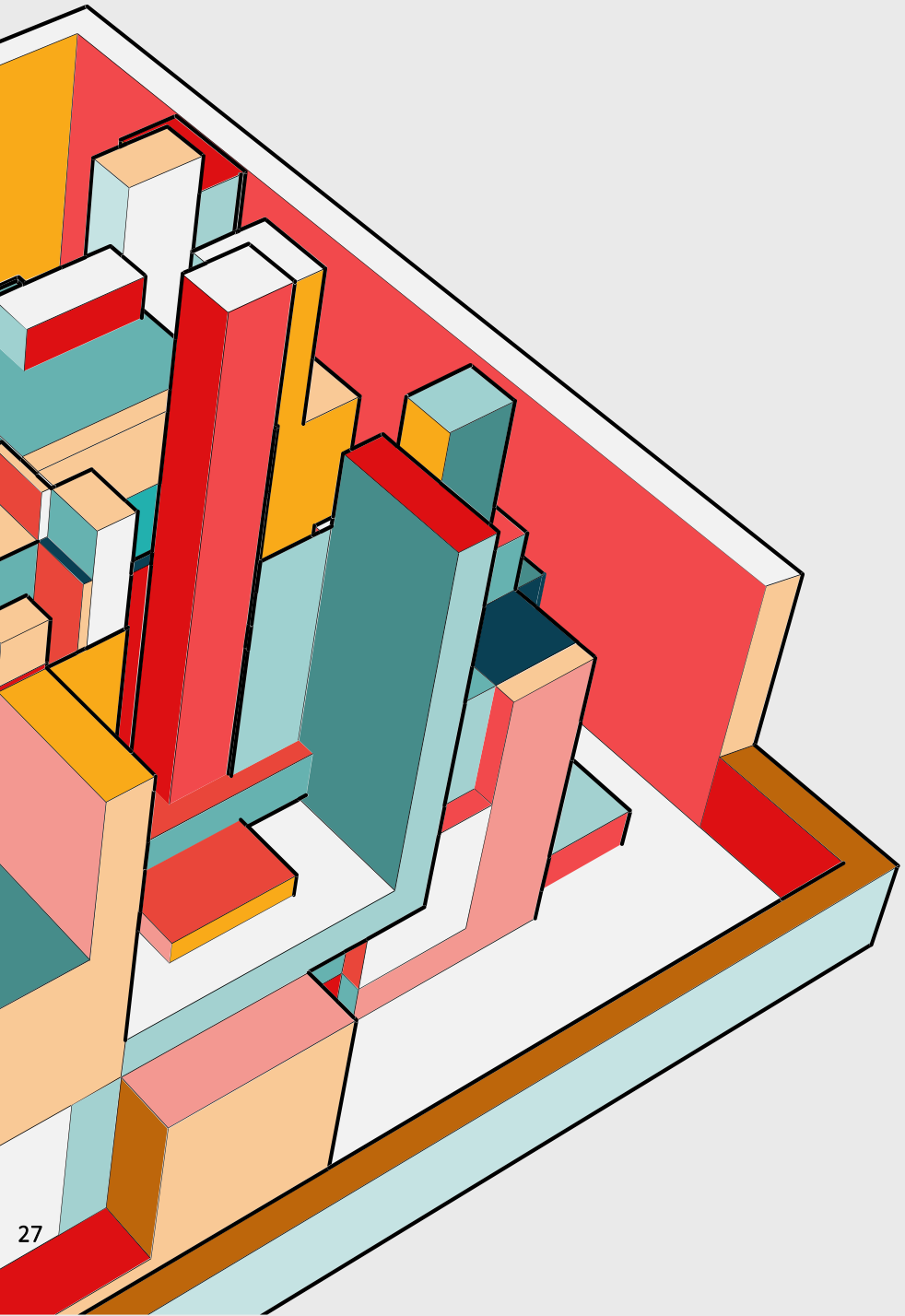
ЭТО НЕ ПО DDD

Microsoft Dynamics AX

SAP ERP

✓ Хороши для generic subdomains

✗ Плохи для core subdomains



РАЗНИЦА МЕЖДУ АНЕМИЧНОЙ И БОГАТОЙ МОДЕЛЬЮ

Богатая модель

Полная инкапсуляция
Domain model
Новая фича занимает 2 дня

Анемичная модель

Логика в хранимках
Transaction script
Новая фича занимает 2 месяца

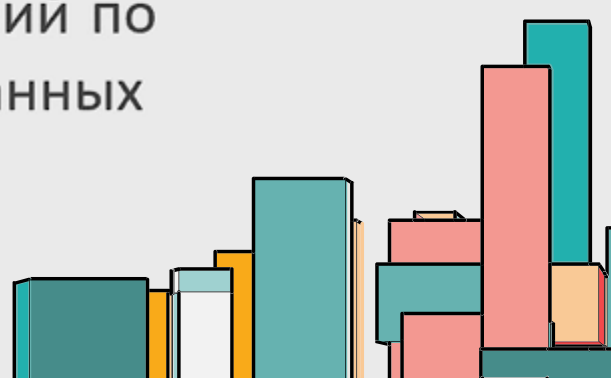
АНЕМИЧНАЯ МОДЕЛЬ


```
public class Person
{
    public string Name { get; set; }
    public string Email { get; set; }
    public bool IsEmployee { get; set; }
}
```

АНЕМИЧНАЯ МОДЕЛЬ


```
public class Person
{
    public string Name { get; set; }
    public string Email { get; set; }
    public bool IsEmployee { get; set; }
}
```

Нет ограничений по изменению данных



Богатая модель = Инкапсулированная модель

ЕЩЕ ОДНА АНЕМИЧНАЯ МОДЕЛЬ

```
public class Customer
{
    public string Name { get; set; }
    public string Email { get; set; }
    public string Status { get; set; }
    public List<Order> Orders { get; set; }
    public decimal CurrentDiscount { get; set; }
}
```

Инвариант – это то условие которое должно быть истинным на протяжении всей жизни модели

- Создает дополнительную когнитивную нагрузку
- Большой риск ошибок

АНЕМИЧНАЯ МОДЕЛЬ?

```
public class Square {
    public readonly double SideLength;

    public Square(double sideLength) {
        if (sideLength <= 0)
            throw new Exception("Invalid square side length: " + sideLength);

        SideLength = sideLength;
    }
}

public class Services {
    public static double CalculateArea(Square square) {
        return square.SideLength * square.SideLength;
    }
}
```

АНЕМИЧНАЯ МОДЕЛЬ?

“Объектно-ориентированное программирование делает код понятным через инкапсуляцию подвижных частей. Функциональное программирование делает код понятным через минимизацию подвижных частей.”

Michael Feathers

```
public class Square {
    public readonly double SideLength;

    public Square(double sideLength) {
        if (sideLength <= 0)
            throw new Exception("Invalid square side length: " + sideLength);

        SideLength = sideLength;
    }
}

public class Services {
    public static double CalculateArea(Square square) {
        return square.SideLength * square.SideLength;
    }
}
```

РЕФАКТОРИНГ АНЕМИЧНОЙ МОДЕЛИ

- Инкапсуляция
  - Строгая типизация
  - Уменьшение количества методов

РЕФАКТОРИНГ АНЕМИЧНОЙ МОДЕЛИ

Email != строка

Decimal != Discount

```
public class Customer
{
    public string Name { get; set; }
    public string Email { get; set; }
    public string Status { get; set; }
    public List<Order> Orders { get; set; }
    public decimal CurrentDiscount { get; set; }
}
```

String typing

Email

```
public string Email { get; set; }
```

```
public class Customer {
    public string Name { get; set; }
    public string Email { get; set; }
    public string Status { get; set; }
    public List<Order> Orders { get; set; }
    public decimal CurrentDiscount { get; set; }
}
```

```
public class Customer {
    public string Name { get; set; }
    public Email Email { get; set; }
    public Status Status { get; set; }
    public List<Order> Orders { get; set; }
    public Discount CurrentDiscount { get; set; }
}
```

Value Objects

УБЕРЕМ ЕЩЕ ОДНУ БРЕШЬ

```
public class Customer {
    public string Name { get; set; }
    public Email Email { get; set; }
    public Status Status { get; set; }
    public List<Order> Orders { get; set; }
    public Discount CurrentDiscount { get; set; }
}
```

```
public class Customer {
    public string Name { get; set; }
    public Email Email { get; set; }
    public Status Status { get; set; }
    public List<Order> Orders { get; set; }
    public Discount CurrentDiscount => Status.GetDiscount();
}
```

РЕФАКТОРИНГ АНЕМИЧНОЙ МОДЕЛИ

- Инкапсуляция
  - Строгая типизация
  - Уменьшение количества методов

Изменяемый список

Изменяемое свойство

```
public class Customer {
    public string Name { get; set; }
    public Email Email { get; set; }
    public Status Status { get; set; }
    public List<Order> Orders { get; set; }
    public Discount CurrentDiscount => CustomerStatus.GetDiscount();
}
```

```
public class CustomerService {
    public void AddOrder(Customer customer, Product product, int quantity)
    {
        customer.Orders.Add(new Order(product, quantity));

        if (customer.Orders.Count > 10)
        {
            customer.CustomerStatus = Status.Advanced;
        }
    }
}
```

```
public class Customer {
    public string Name { get; }
    public Email Email { get; }
    public Status Status { get; private set; }
    public Discount CurrentDiscount => Status.GetDiscount();

    private List<Order> _orders;
    public IReadOnlyList<Order> Orders => _orders;

    public void AddOrder(Product product, int quantity)
    {
        _orders.Add(new Order(product, quantity));

        if (_orders.Count > 10)
        {
            Status = Status.Advanced;
        }
    }
}
```

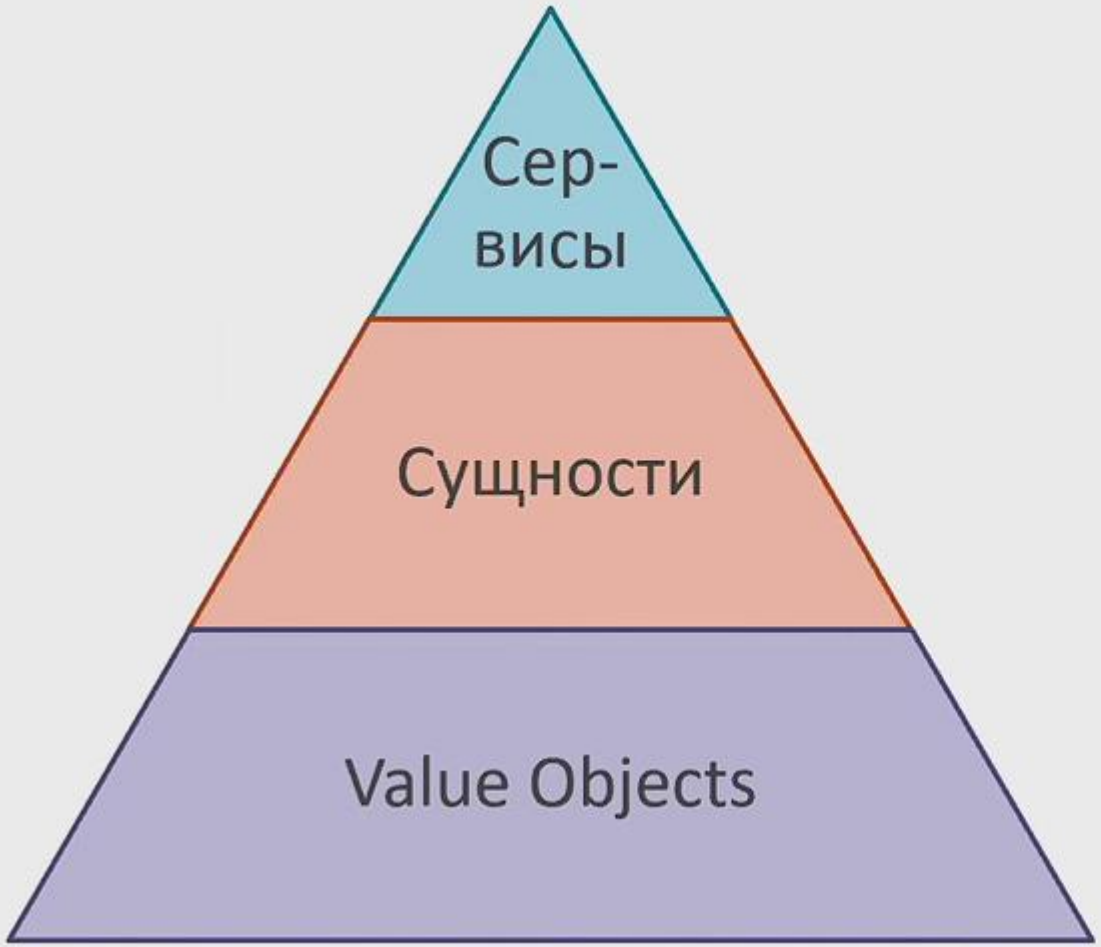
Там класс теперь следит за соблюдением инварианта

КЛИЕНТУ ВАШЕЙ МОДЕЛИ НУЖЕН ОДИН МЕТОД

```
public class CustomerService {
    public void AddOrder(Customer customer, Product product, int quantity)
    {
        customer.Orders.Add(new Order(product, quantity));

        if (customer.Orders.Count > 10)
        {
            customer.CustomerStatus = Status.Advanced;
        }
    }
}
```

```
public class CustomerService {
    public void AddOrder(Customer customer, Product product, int quantity)
    {
        customer.AddOrder(product, quantity);
    }
}
```

ПРОЕКТИРОВАНИЕ НАЧИНАЕТСЯ НЕ С ENTITY



CustomerService

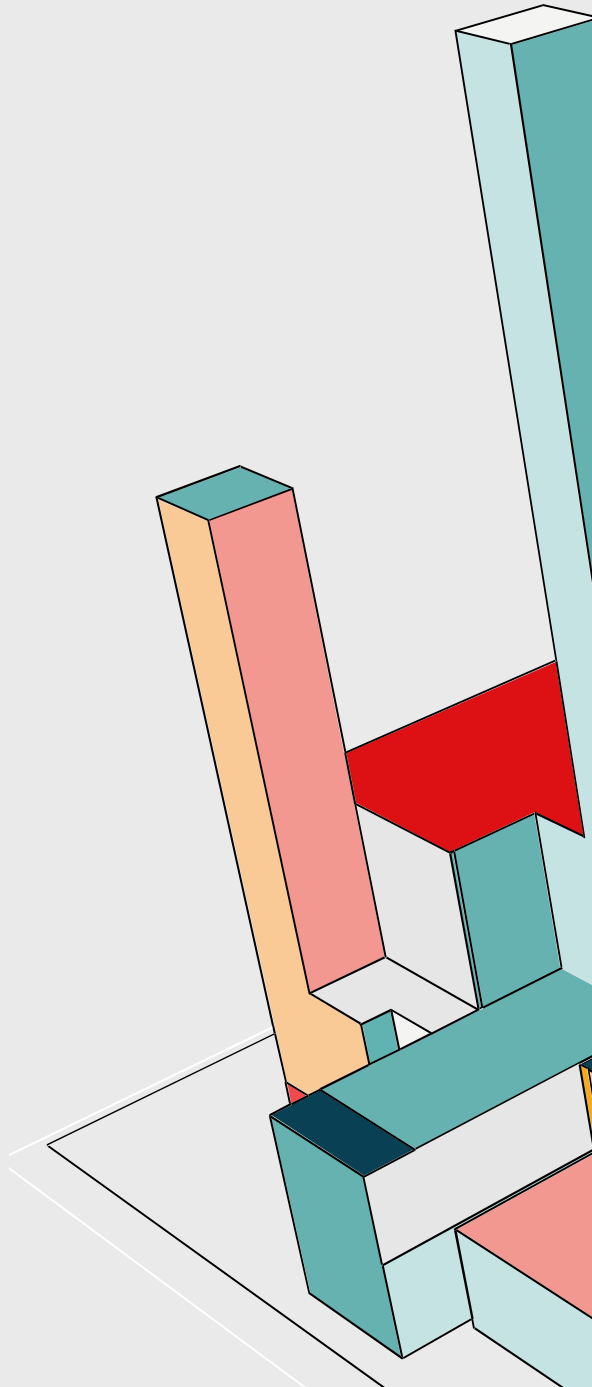
Customer

Email, Status,
Discount

СУЩНОСТЬ (ENTITY)

Если какое-то понятие предметной области является уникальным и отличным от всех других объектов в системе, то для его моделирования используется сущность.

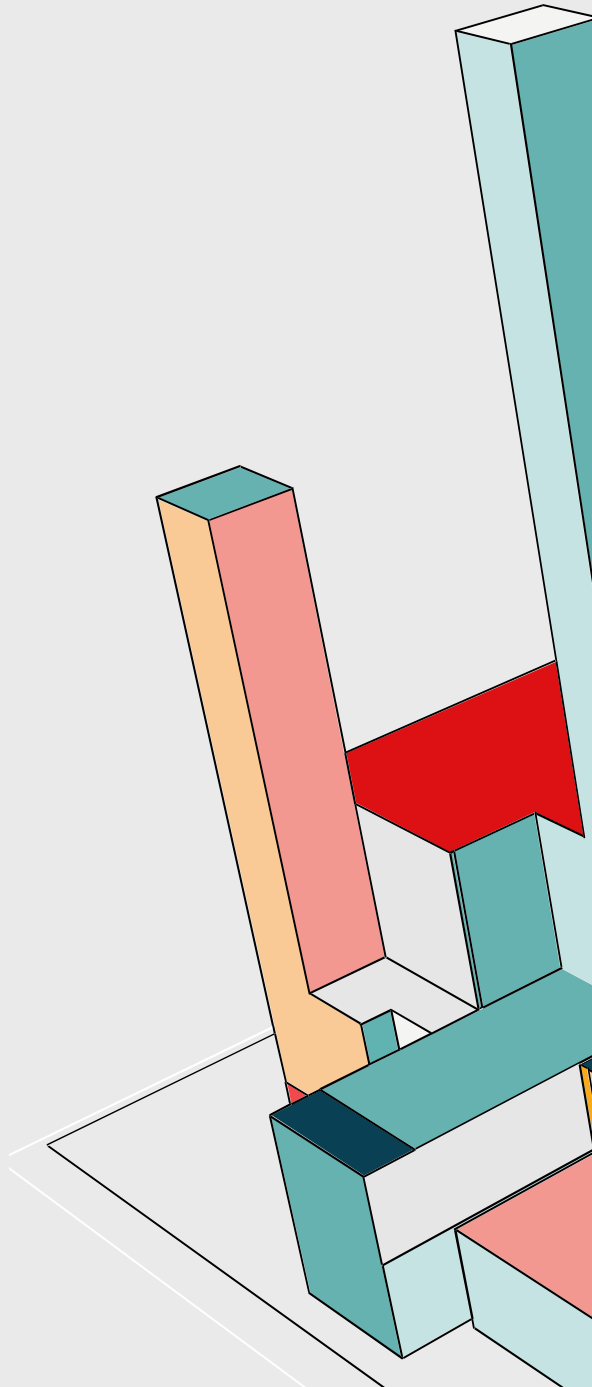
Такие объекты-сущности могут сильно отличаться своей формой за весь цикл существования, тем не менее их всегда можно однозначно идентифицировать и найти по запросу.

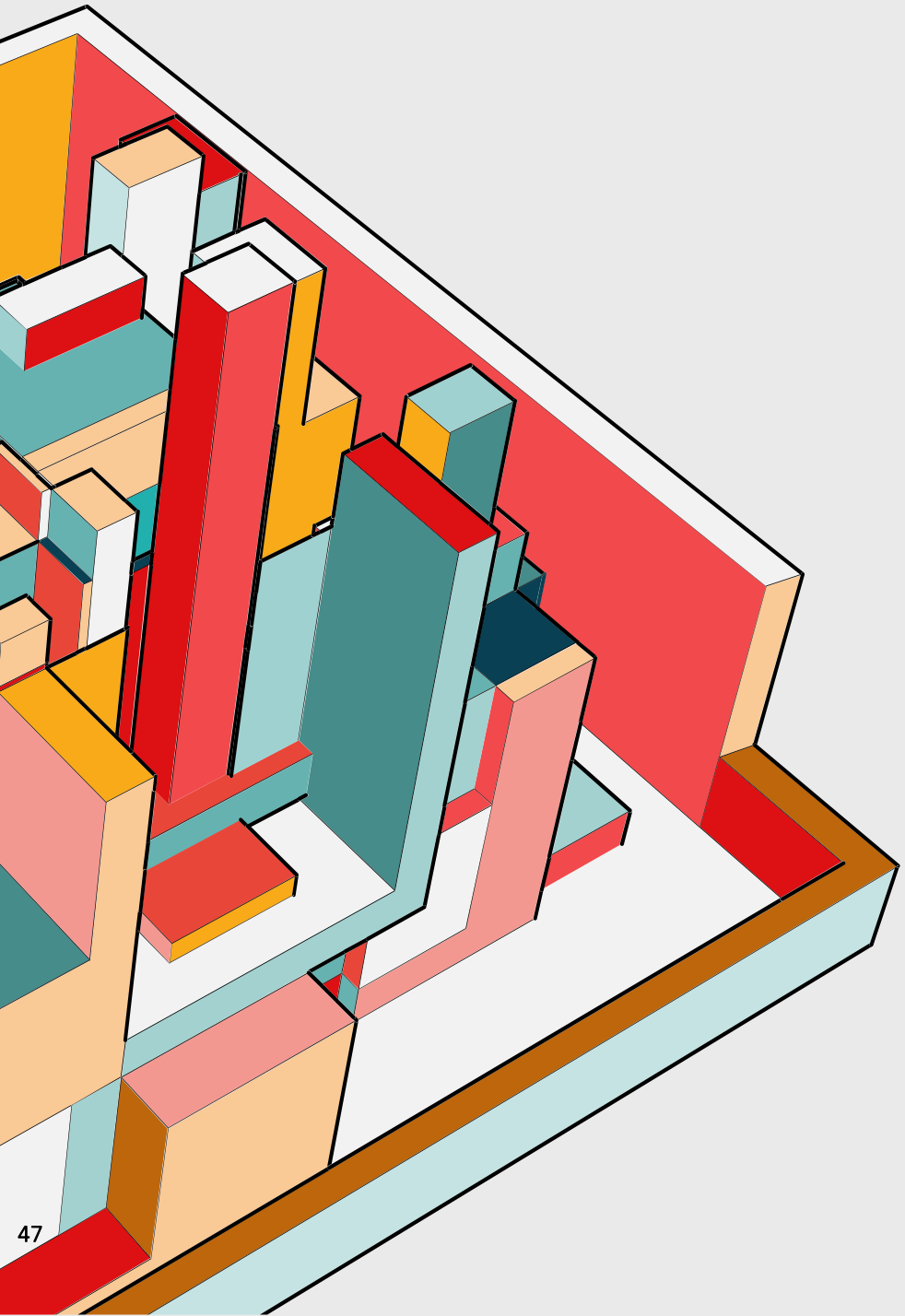


ОБЪЕКТ-ЗНАЧЕНИЕ (VALUE OBJECT)

Обладает большинством из следующих характеристик:

- Оно измеряет, оценивает или описывает объект предметной области;
- Его можно считать неизменяемым;
- Оно моделирует нечто концептуально целостное, объединяя связанные атрибуты в одно целое;
- При изменении способа измерения или описания его можно полностью заменить;
- Его можно сравнивать с другими объектами с помощью отношения равенства значений;
- Оно предоставляет связанным с ним объектам функцию без побочных эффектов.





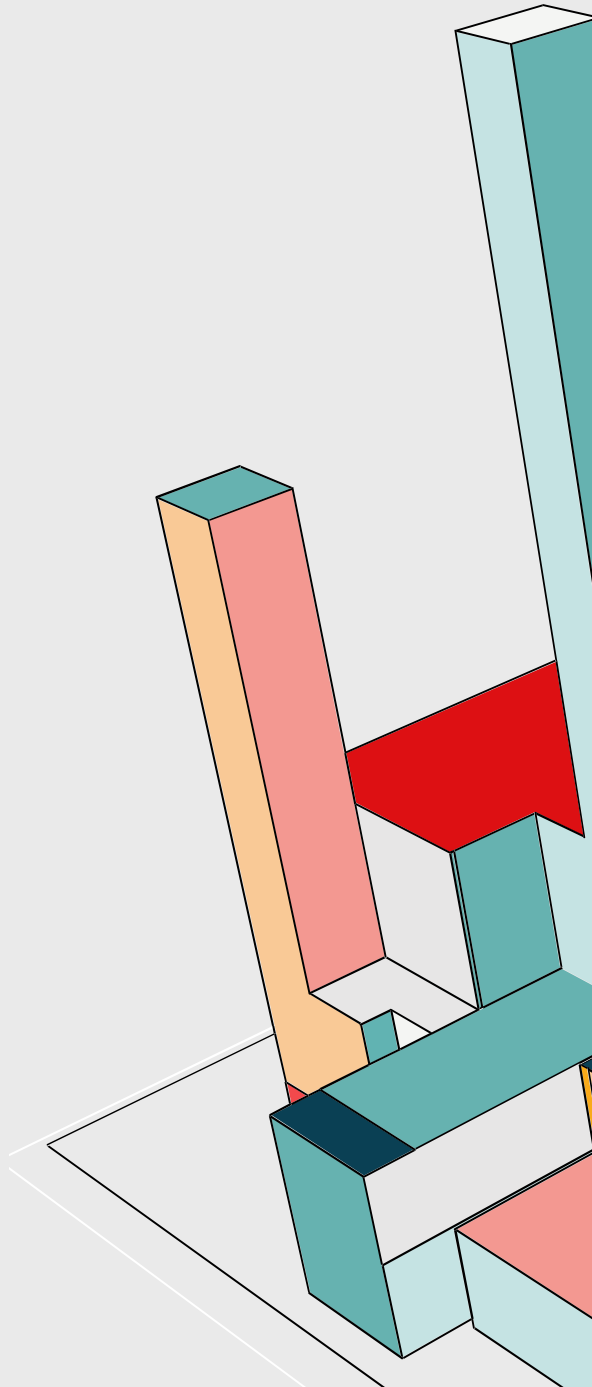
ИЗОЛЯЦИЯ ДОМЕННОЙ МОДЕЛИ

СЛУЖБА ПРЕДМЕТНОЙ ОБЛАСТИ (DOMAIN SERVICE)

Очень часто существуют какие-то действия, которые нельзя отнести к какой-то сущности или к какому-то объекту-значению. Если существует такого рода операция в предметной области, ее объявляют как служба предметной области. Признаки таких действий:

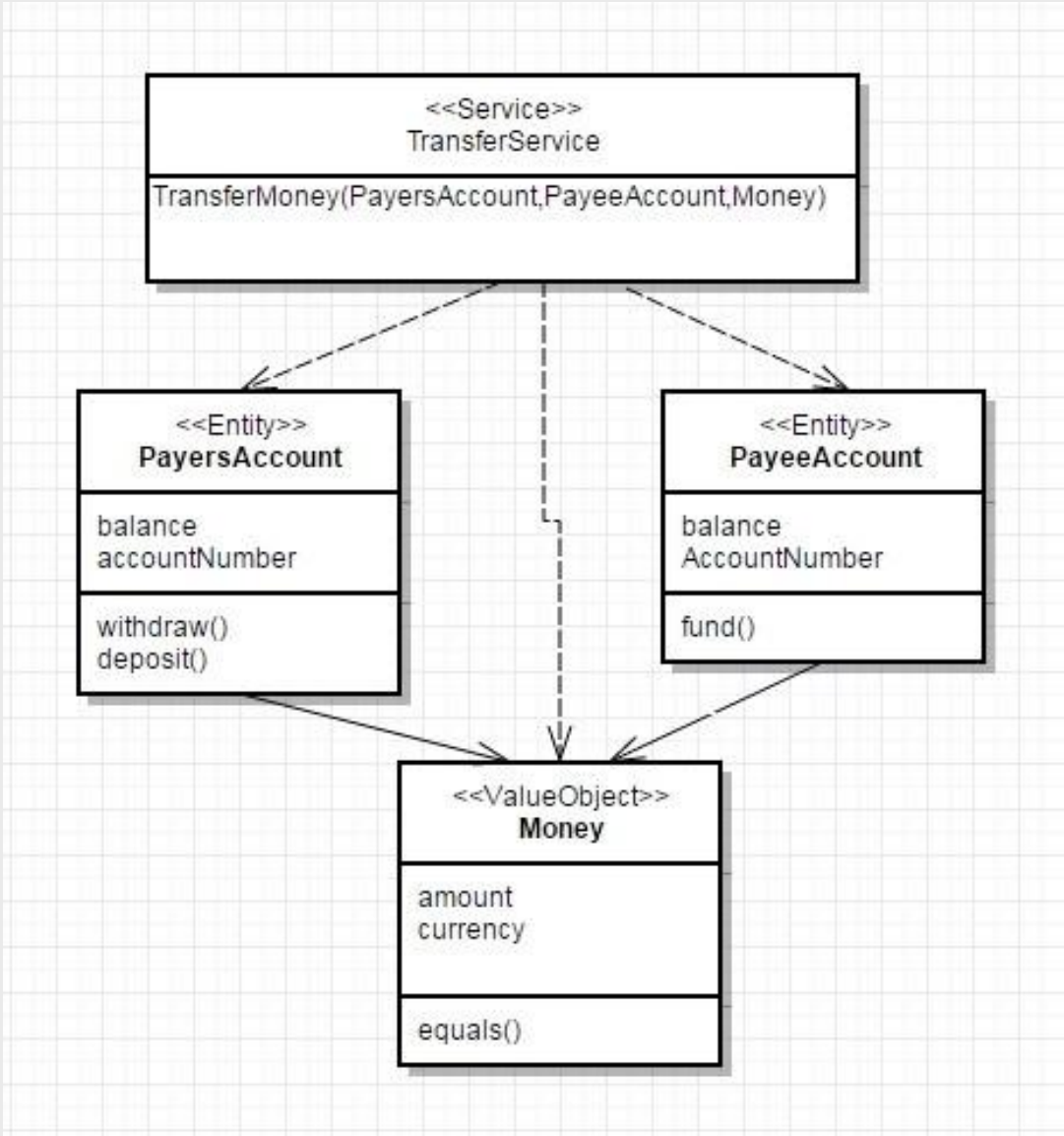
1. Операция, выполняемая службой, относится к концепции предметной области, которая не принадлежит ни одной из существующих сущностей;
2. Операция выполняется над различными объектами модели предметной области;
3. Операция не имеет состояния.

Не нужно злоупотреблять использованием служб. Это приводит к созданию анемичной модели предметной области

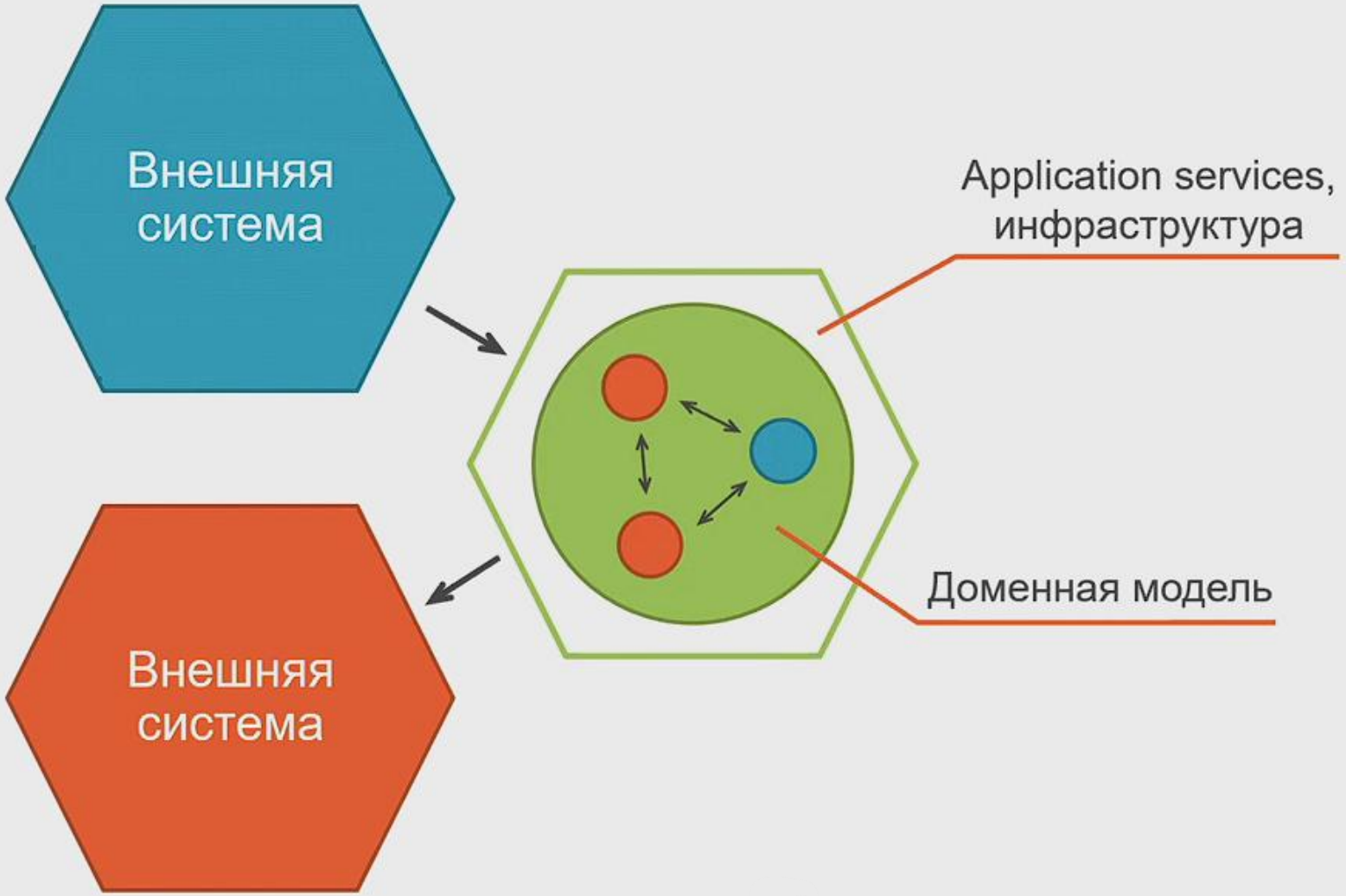


ПРИМЕР

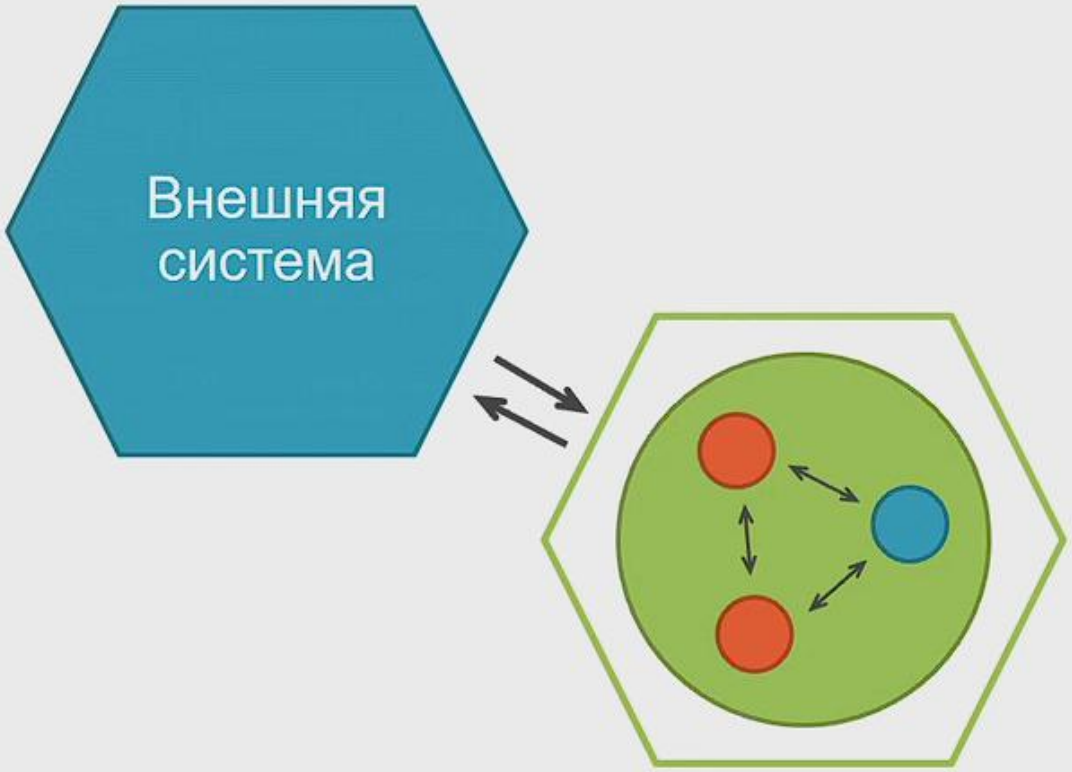
Для примера можно взять службу перевода денег с одного счета плательщика в счет получателя. Совершенно неясно, в каком объекте хранить метод перевода, поэтому используется служба



ИЗОЛИРУЙТЕ ДОМЕННУЮ МОДЕЛЬ



ИЗОЛИРОВАННАЯ ДОМЕННАЯ МОДЕЛЬ



- Разделение ответственностей между доменной моделью и сервисами приложения
- One-way flow of dependencies
- Взаимодействия между приложениями

ПРИМЕР НЕ ИЗОЛИРОВАННОЙ ДОМЕННОЙ МОДЕЛИ

```
public class Customer
{
    public void Save()
    {
        /* Save to database */
    }

    public void Restore(long id)
    {
        /* Load from database */
    }
}
```

Active Record

Доменная модель не изолирована (impure)

ПРИМЕР НЕ ИЗОЛИРОВАННОЙ ДОМЕННОЙ МОДЕЛИ

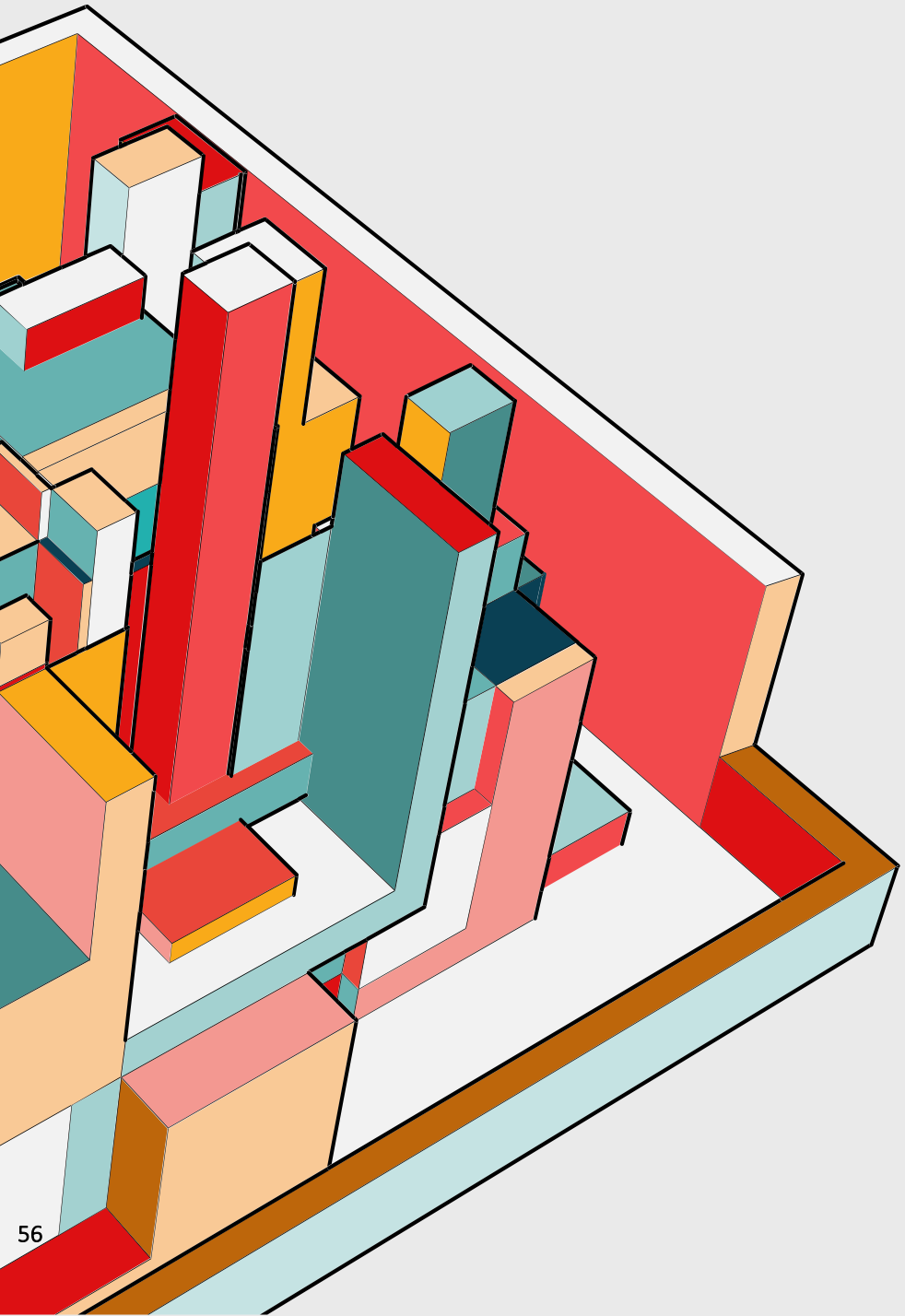
```
public void AddOrder(Product product, int quantity)
{
    var order = new Order(product, quantity, DateTime.Now);
    _orders.Add(order);
}
```

ПРИМЕР НЕ ИЗОЛИРОВАННОЙ ДОМЕННОЙ МОДЕЛИ

```
public void AddOrder(Product product, int quantity)
{
    var order = new Order(product, quantity, DateTime.Now);
    _orders.Add(order);
}

public void AddOrder(Product product, int quantity, Func<DateTime> getTimeFunc)
{
    var order = new Order(product, quantity, getTimeFunc());
    _orders.Add(order);
}
```

ПРИМЕР НЕ ИЗОЛИРОВАННОЙ ДОМЕННОЙ МОДЕЛИ

```
public void AddOrder(Product product, int quantity)
{
    var order = new Order(product, quantity, DateTime.Now);
    _orders.Add(order);
}

public void AddOrder(Product product, int quantity, Func<DateTime> getTimeFunc)
{
    var order = new Order(product, quantity, getTimeFunc());
    _orders.Add(order);
}

public void AddOrder(Product product, int quantity, DateTime now)
{
    var order = new Order(product, quantity, now);
    _orders.Add(order);
}
```



DDD ТРИЛЕММА

```
public class CustomerController
{
    public string ChangeEmail(int customerId, string newEmail)
    {
        Customer customer = _repository.GetById(customerId);
        customer.ChangeEmail(newEmail);
        _repository.Save(customer);

        return "OK";
    }
}
```

```
public class CustomerController
{
    public string ChangeEmail(int customerId, string newEmail)
    {
        Customer existing = _repository.GetByEmail(newEmail);
        if (existing != null && existing.Id != customerId)
            return "Email is already taken";

        Customer customer = _repository.GetById(customerId);
        customer.ChangeEmail(newEmail);
        _repository.Save(customer);

        return "OK";
    }
}
```

```
public class CustomerController
{
    public string ChangeEmail(int customerId, string newEmail)
    {
        Customer existing = _repository.GetByEmail(newEmail);
        if (existing != null && existing.Id != customerId)
            return "Email is already taken";

        Customer customer =  repository.GetById(customerId);
        customer.ChangeEmail(newEmail);
        _repository.Save(customer);

        return "OK";
    }
}
```

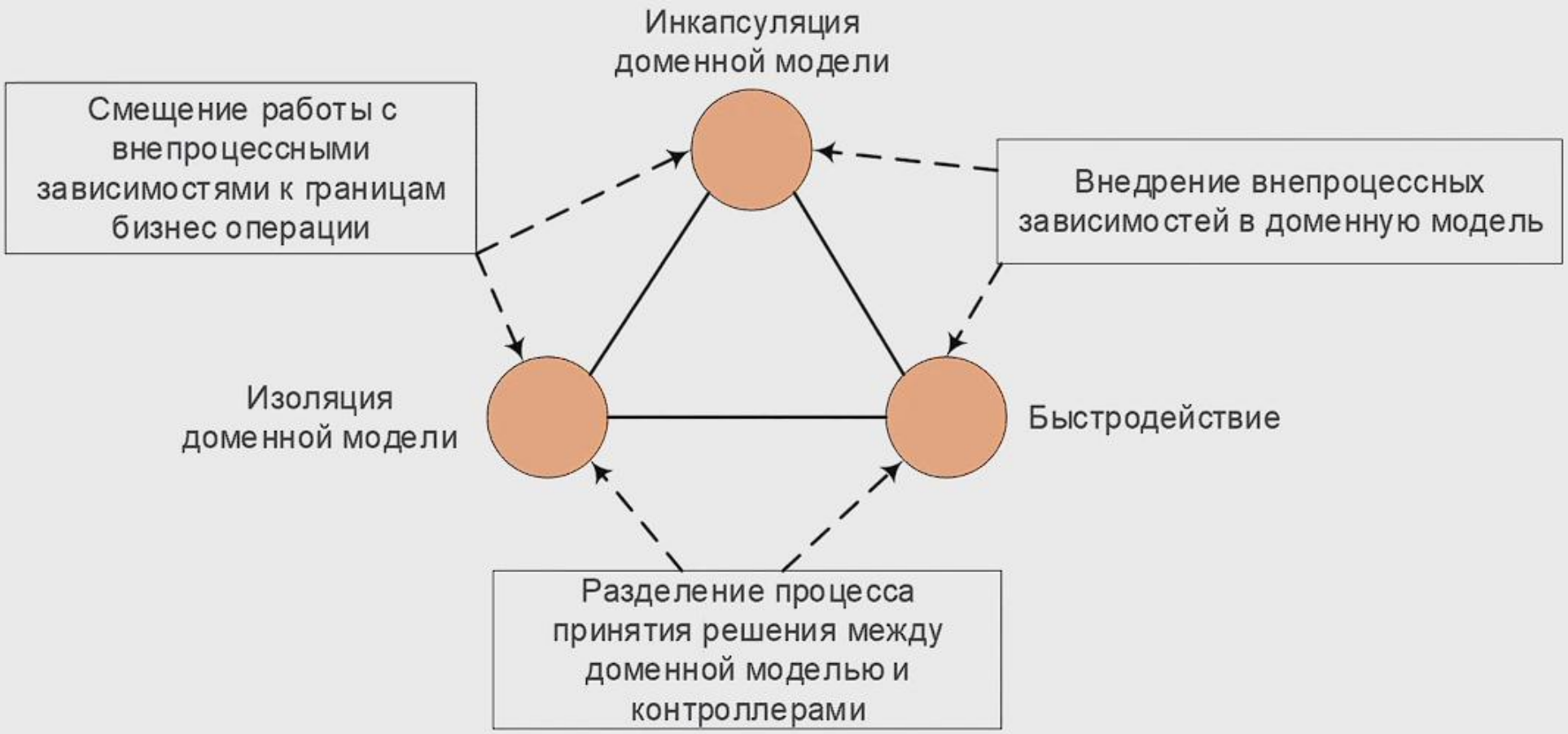
Доменная модель не инкапсулирована

```
public class CustomerController {
    public string ChangeEmail(int customerId, string newEmail) {
        Customer customer = _repository.GetById(customerId);

        Result result = customer.ChangeEmail(newEmail, _repository);
        if (result.IsFailure)
            return result.Error;

        _repository.Save(customer);
        return "OK";
    }
}

public class Customer {
    public Result ChangeEmail(Email newEmail, CustomerRepository repository) {
        Customer existing = repository.GetByEmail(newEmail);
        if (existing != null && existing != this)
            return Result.Failure("Email is already taken");

        Email = newEmail;

        return Result.Success();
    }
```

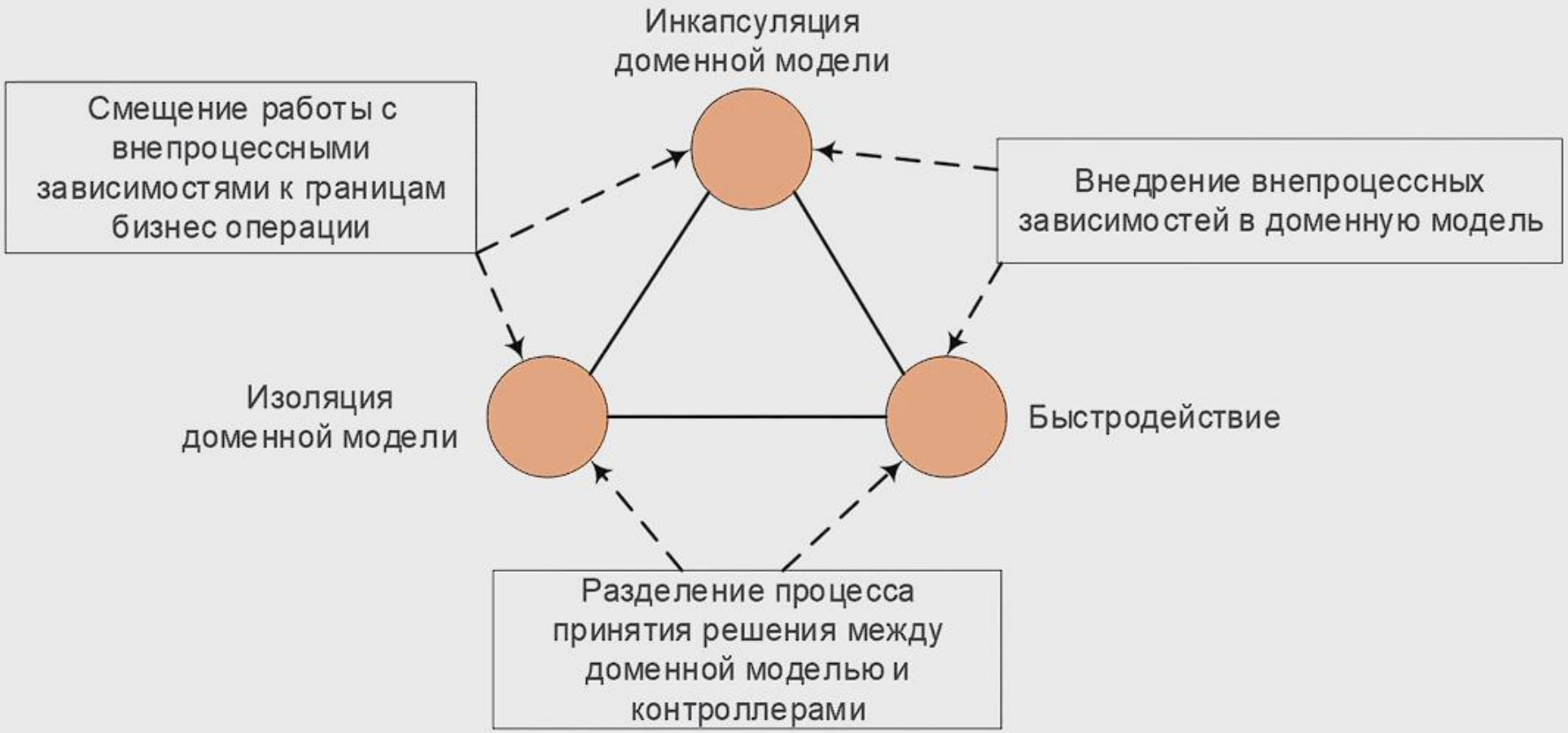
```
public class Customer
{
    public Result ChangeEmail(Email newEmail, CustomerRepository repository)
    {
        Customer existing = repository.GetByEmail(newEmail);
        if (existing != null && existing != this)
            return Result.Failure("Email is already taken");

        Email = newEmail;

        return Result.Success();
    }
}
```

Доменная модель не изолирована

DDD ТРИЛЕММА



DDD ТРИЛЕММА



Изоляция > Инкапсуляция > Быстродействие

📼 Что такое DDD:
https://www.youtube.com/watch?v=pMuiVlnGqjk

📼 Ограниченные контексты:

https://www.youtube.com/watch?v=am-HXycfalo

📼 Пример моделирования DDD:

https://www.youtube.com/watch?v=T29WzvaPNc8&t=1718s

Статьи основным принципам DDD

📕 часть 1: https://habr.com/ru/post/316438/

📕 часть 2: https://habr.com/ru/post/316890/

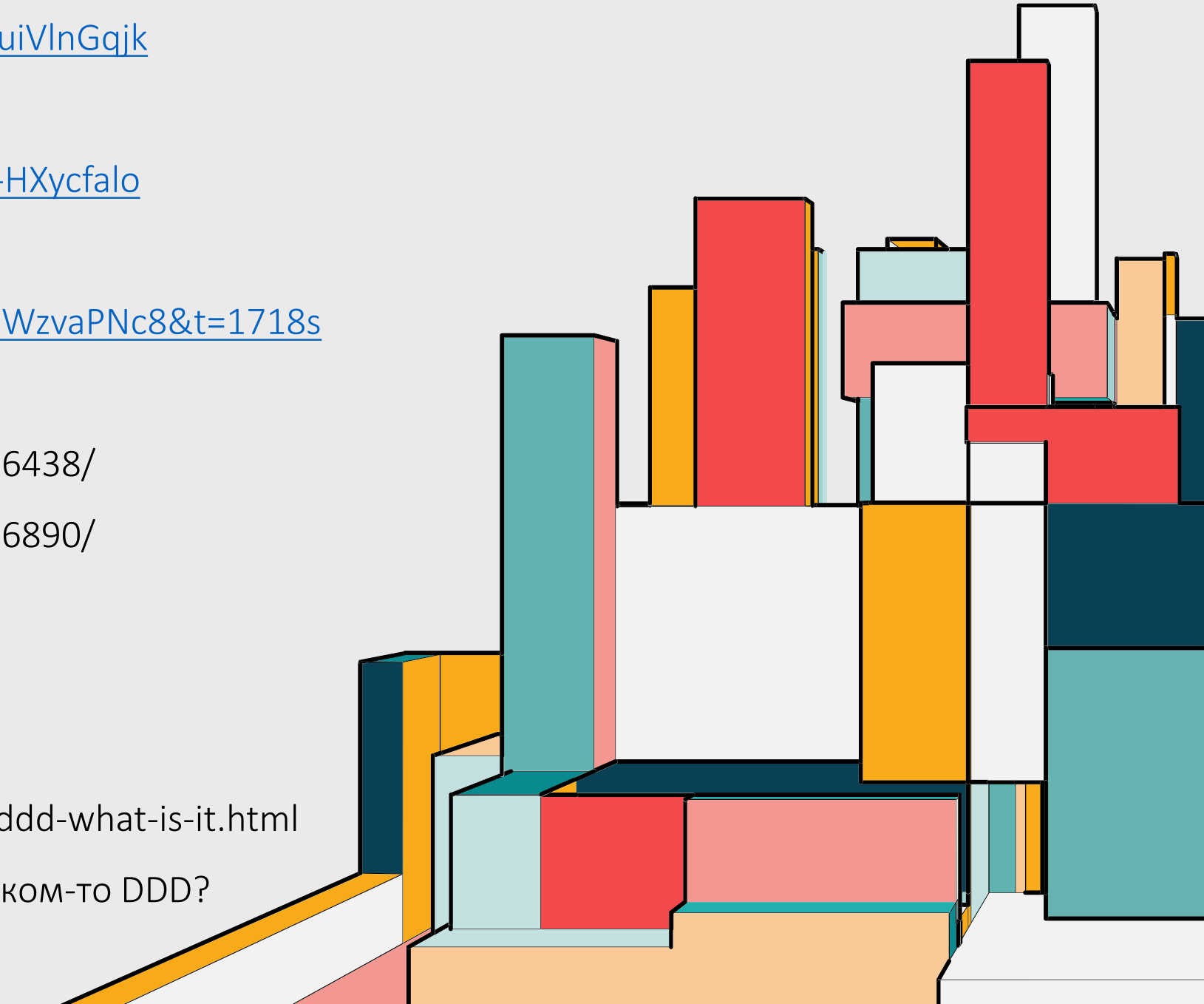
Небольшие заметки про DDD

📌 Что такое DDD?

https://blog-programmista.ru/post/132-ddd-what-is-it.html

📌 Почему вы должны заботиться о каком-то DDD?

https://habr.com/ru/post/497656/



СПАСИБО!

Виденин Сергей

@videninserg