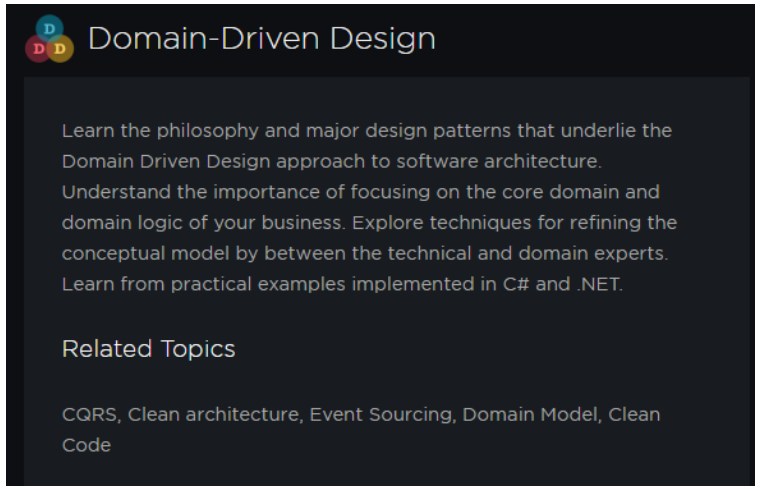
Domain-Driven Design: самое важное

Владимир Хориков

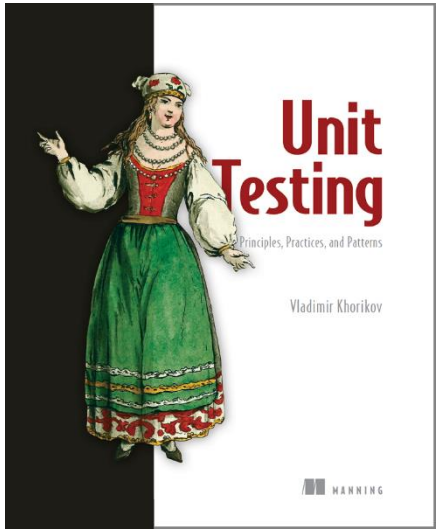
https://enterprisecraftsmanship.com

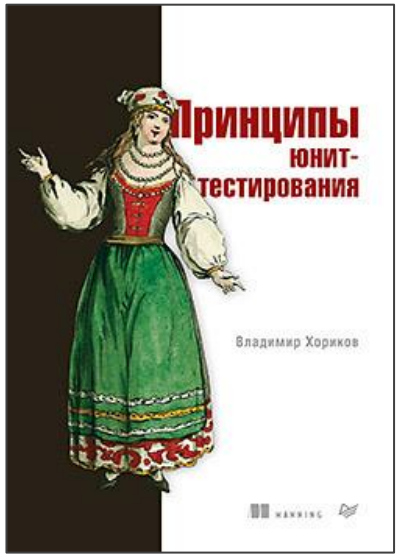
@vkhorikov



Pluralsight Author

https://bit.ly/ps-all





Unit Testing Principles, Practices, and Patterns

http://bit.ly/testing-book

Принципы юнит-тестирования

http://bit.ly/testing-book-ru

План доклада

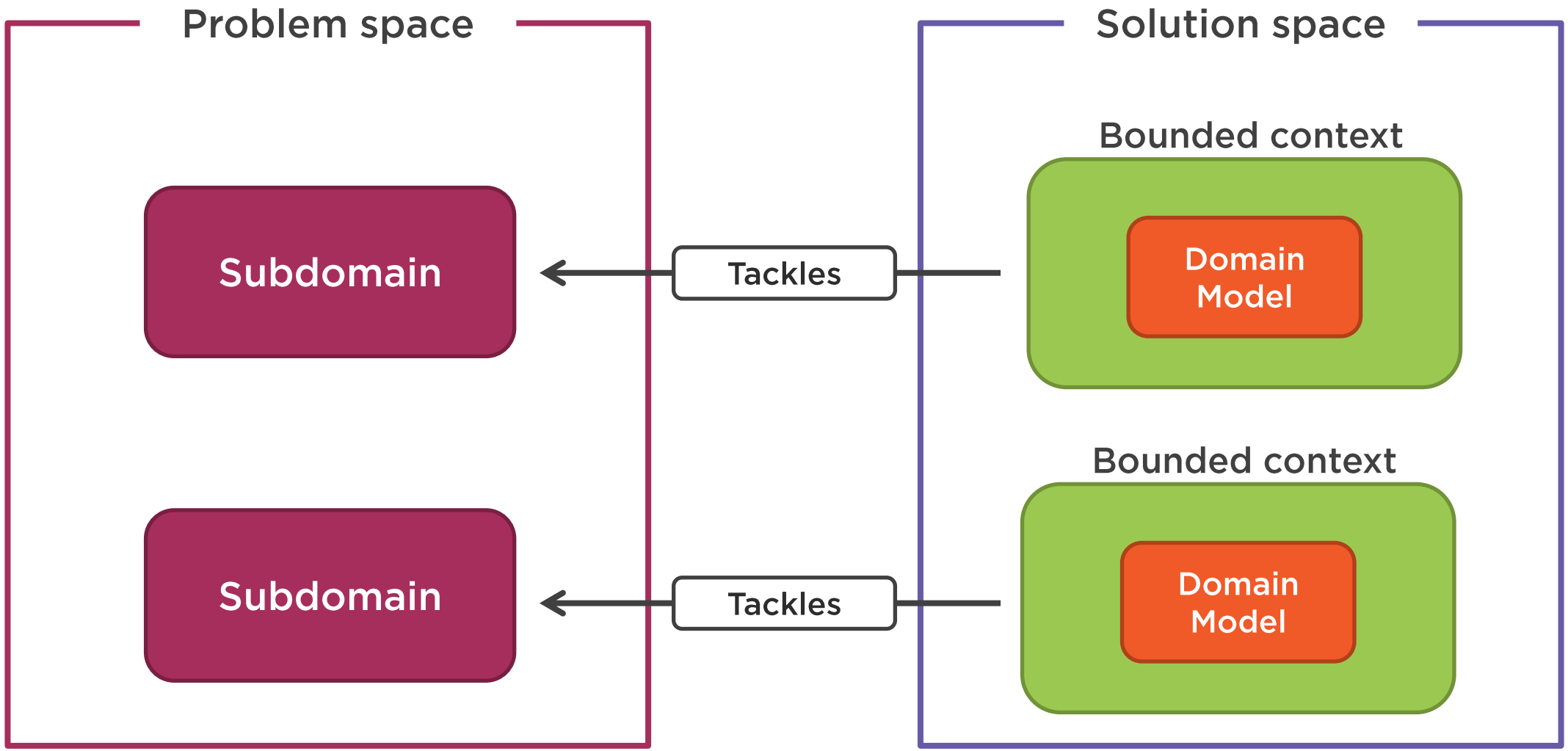
Основные принципы

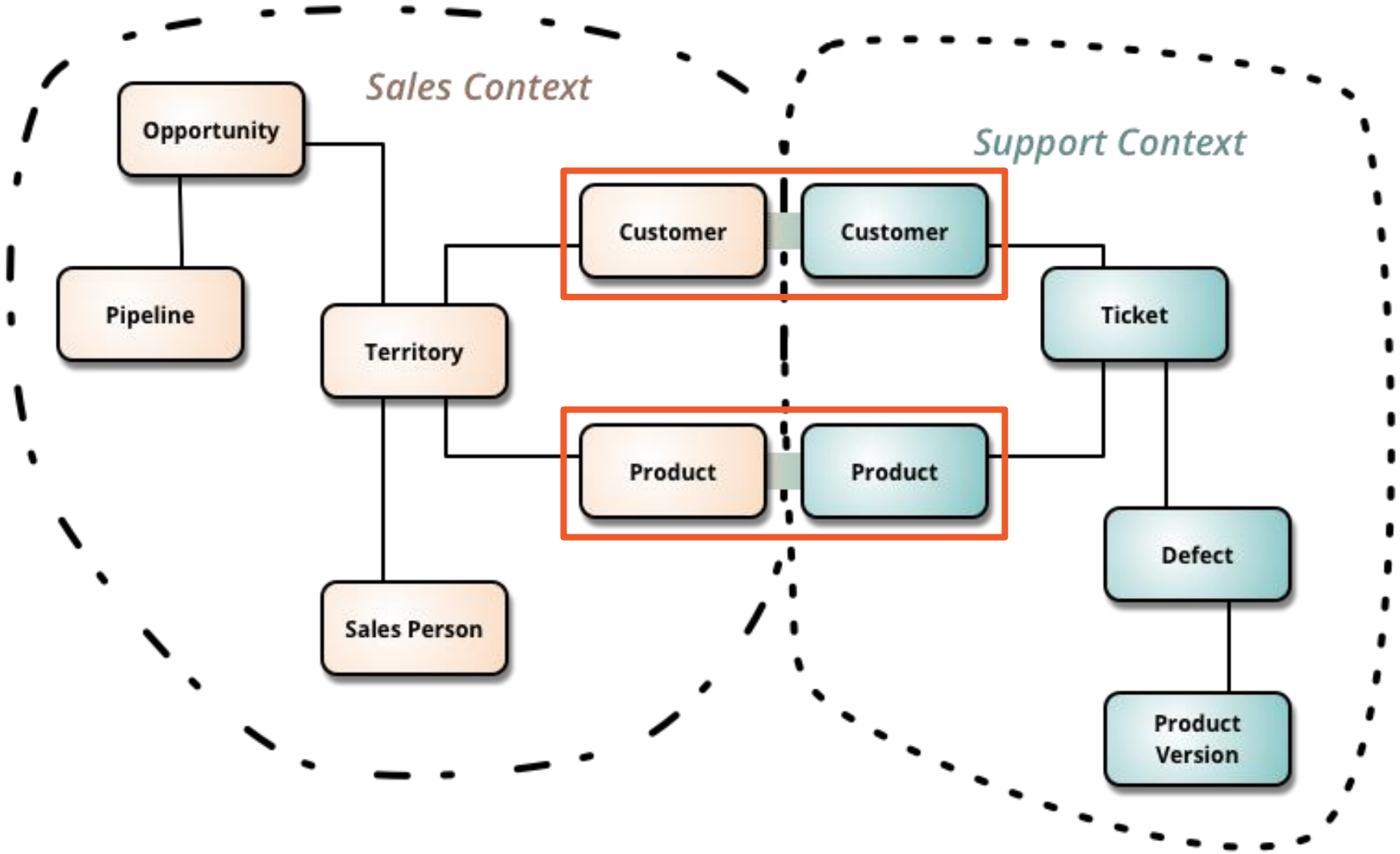
Ограниченные контексты (bounded contexts)

Единый язык (ubiquitous language)

Фокус на доменной модели

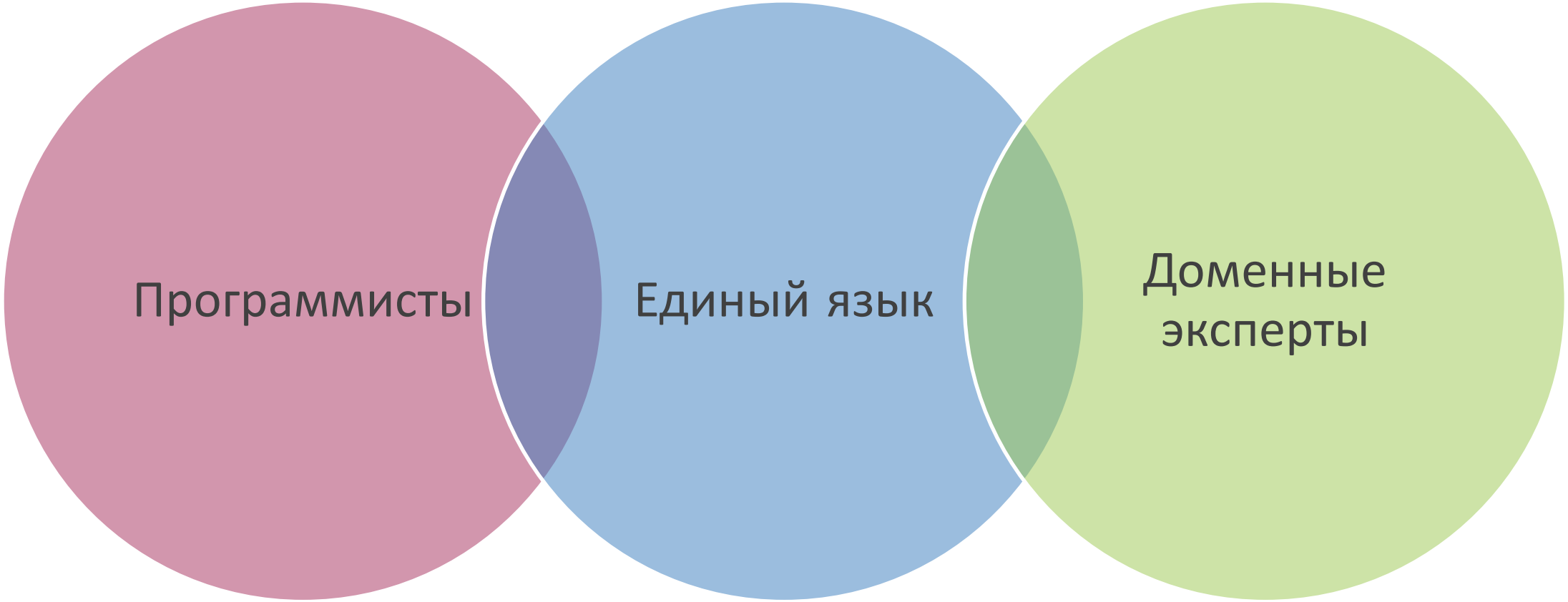
Ограниченные контексты



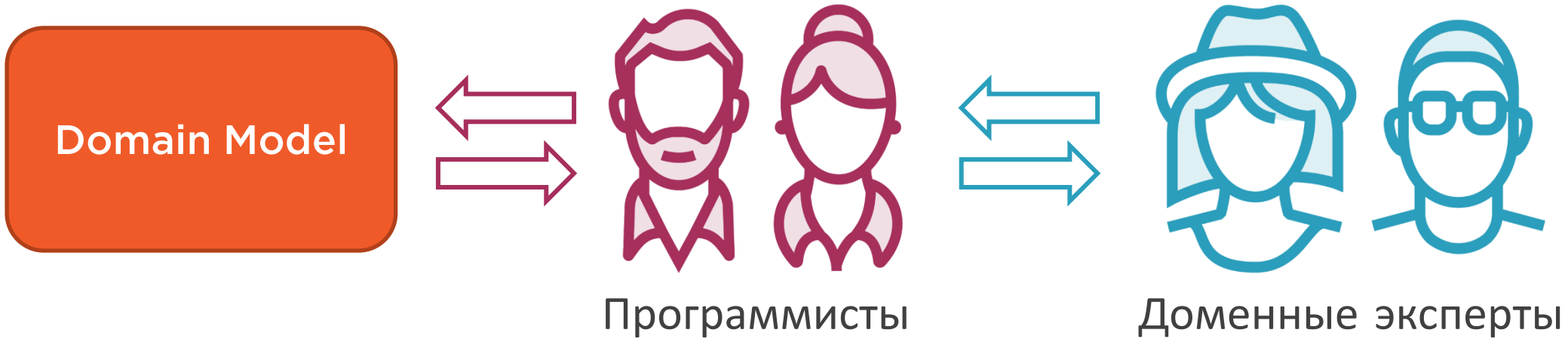


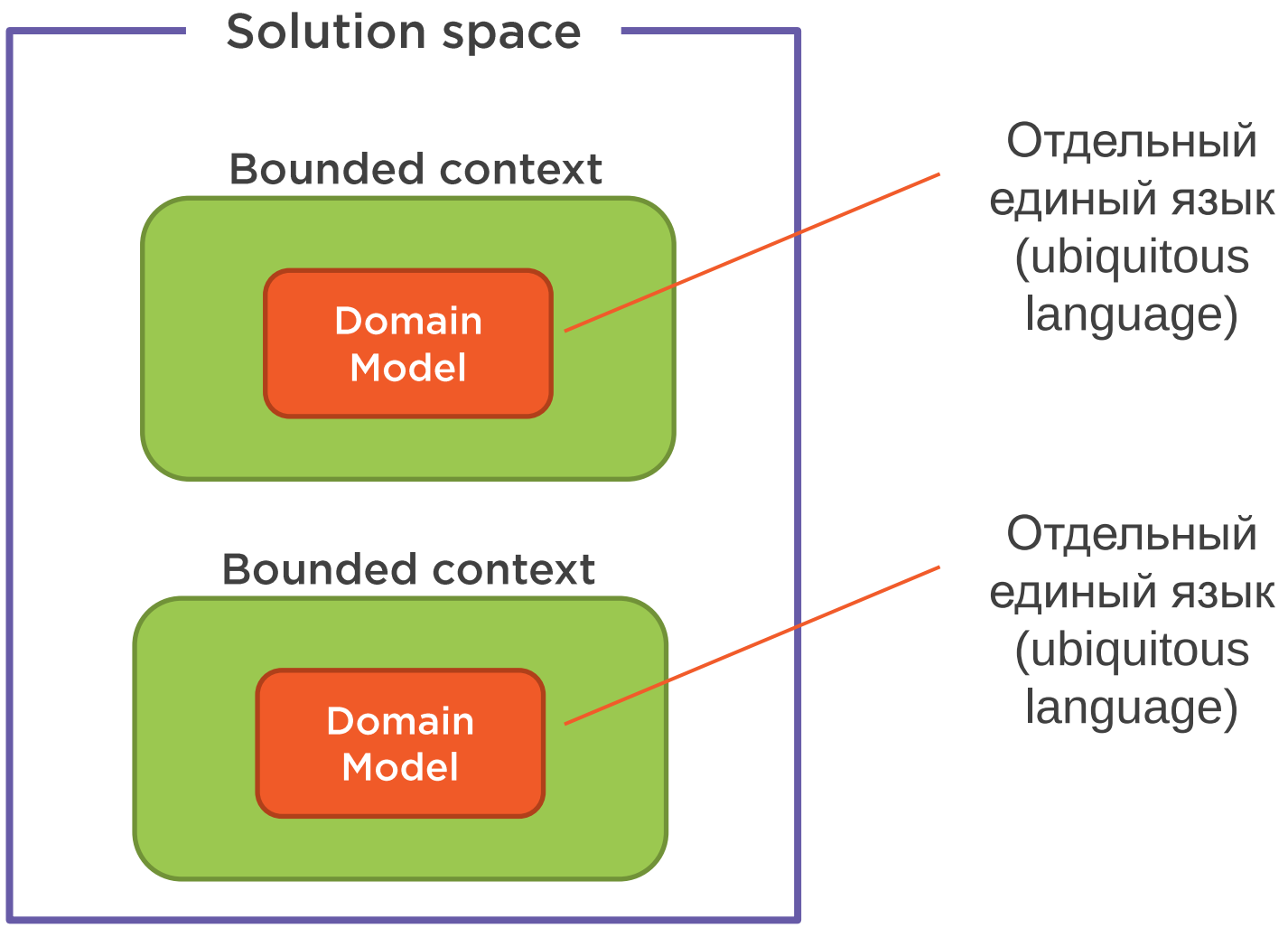
Source: https://martinfowler.com/bliki/BoundedContext.html

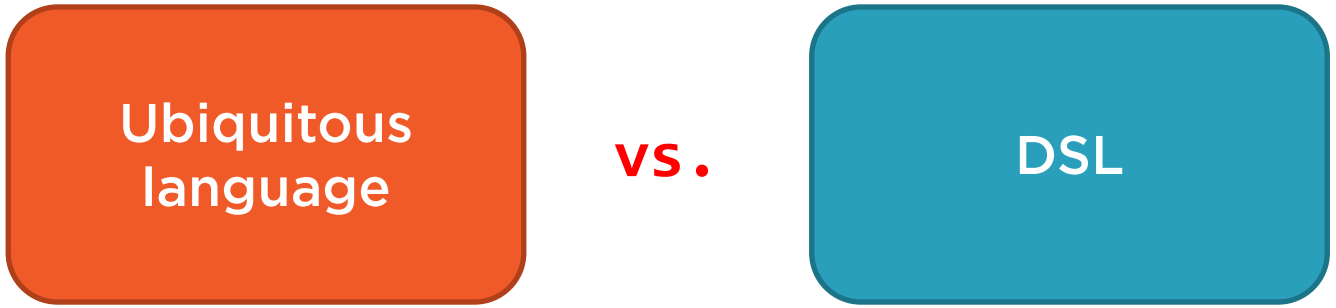
Единый язык



Единый язык требует постоянной поддержки и рефакторинга

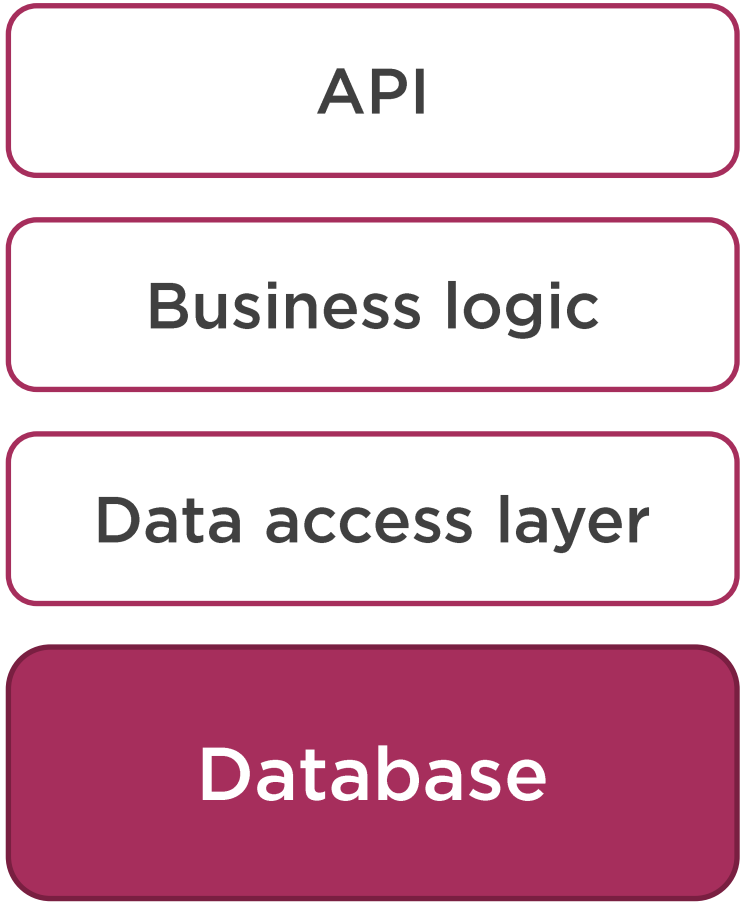

Domain Model
Программисты
Доменные эксперты


Solution space
Bounded context
Domain Model
Bounded context
Domain Model
Отдельный единый язык (ubiquitous language)
Отдельный единый язык (ubiquitous language)

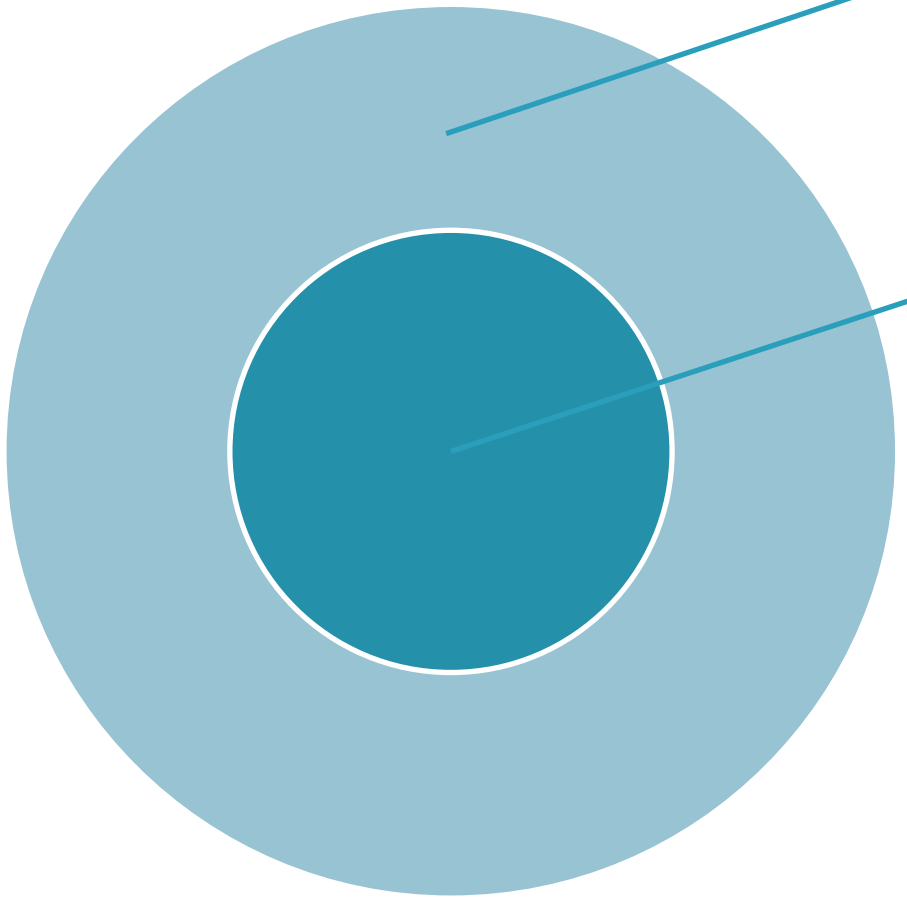

Ubiquitous language
vs.
DSL

Фокус на доменной модели

Доменная модель – краеугольный камень всего приложения



~~Inside-out development~~

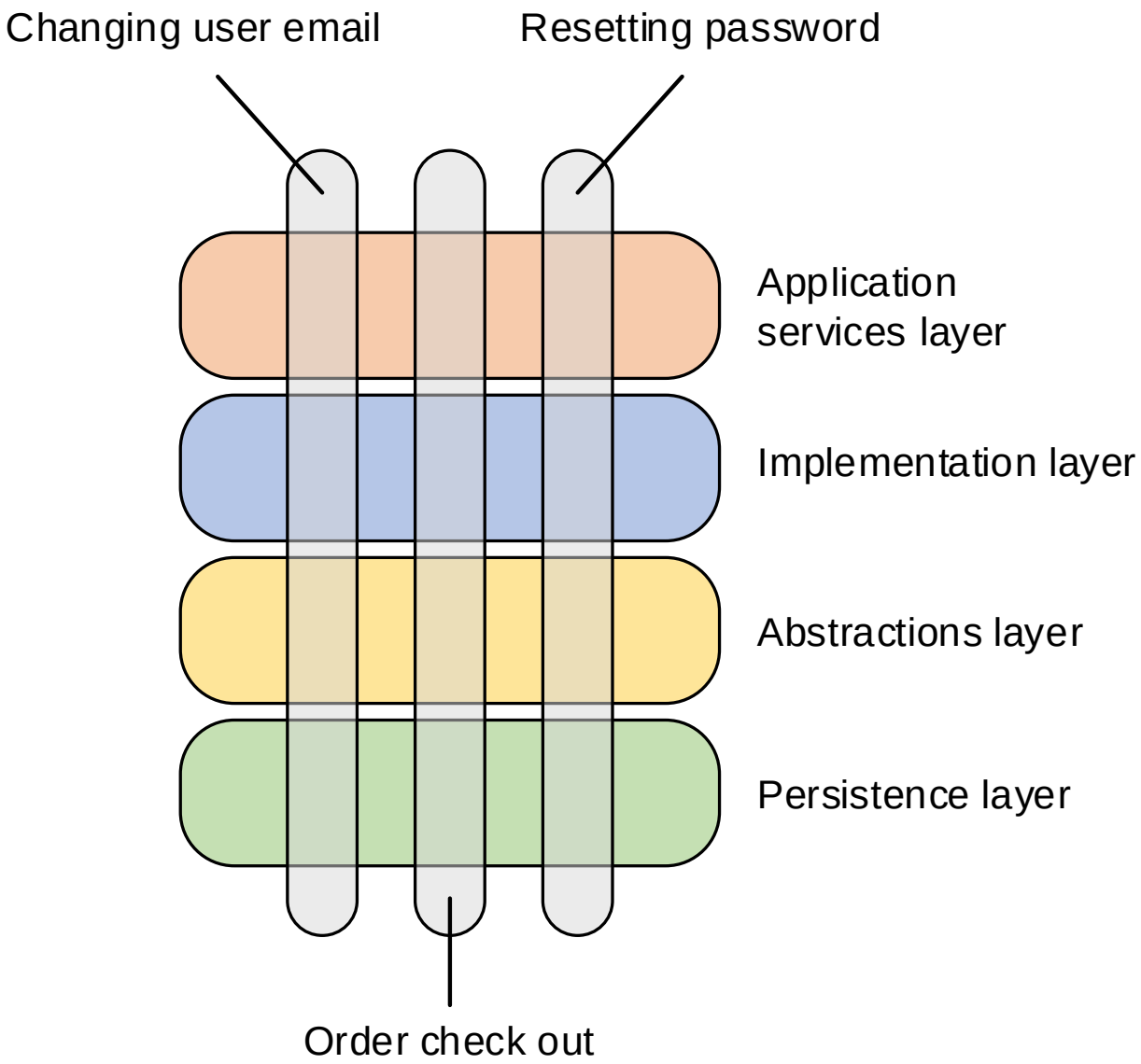


Application
Services

Infrastructure

Business logic
(domain model)

Middle-out development



Доменная логика должна иметь отдельное, четко обозначенное место в проекте

Microsoft Dynamics AX

✓ Хороши для generic subdomains

✗ Плохи для core subdomains

План доклада

Самое важное в DDD

- Основные принципы 
- Разница между анемичной и богатой моделью
- Изоляция доменной модели
- DDD трилемма

Богатая модель

Полная инкапсуляция
Domain model
Новая фича занимает 2 дня

Анемичная модель

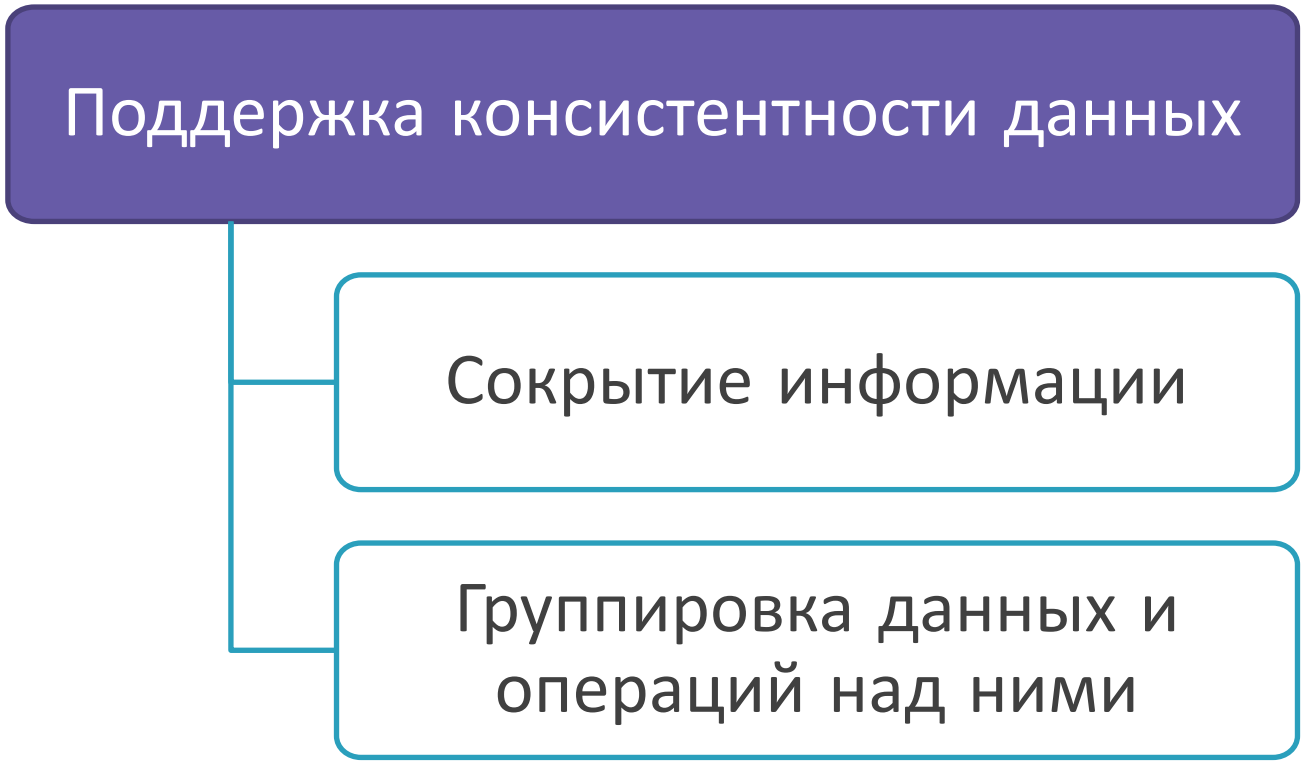
Логика в хранимках
Transaction script
Новая фича занимает 2 месяца

Анемичная модель


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

Нет ограничений по изменению данных

Богатая модель = Инкапсулированная модель



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

Пример анемичной модели

```
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; }
}
```

Создает дополнительную когнитивную нагрузку

Большой риск ошибок

Анемичная модель и функциональное программирование

```
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?

Анемичная модель и функциональное программирование

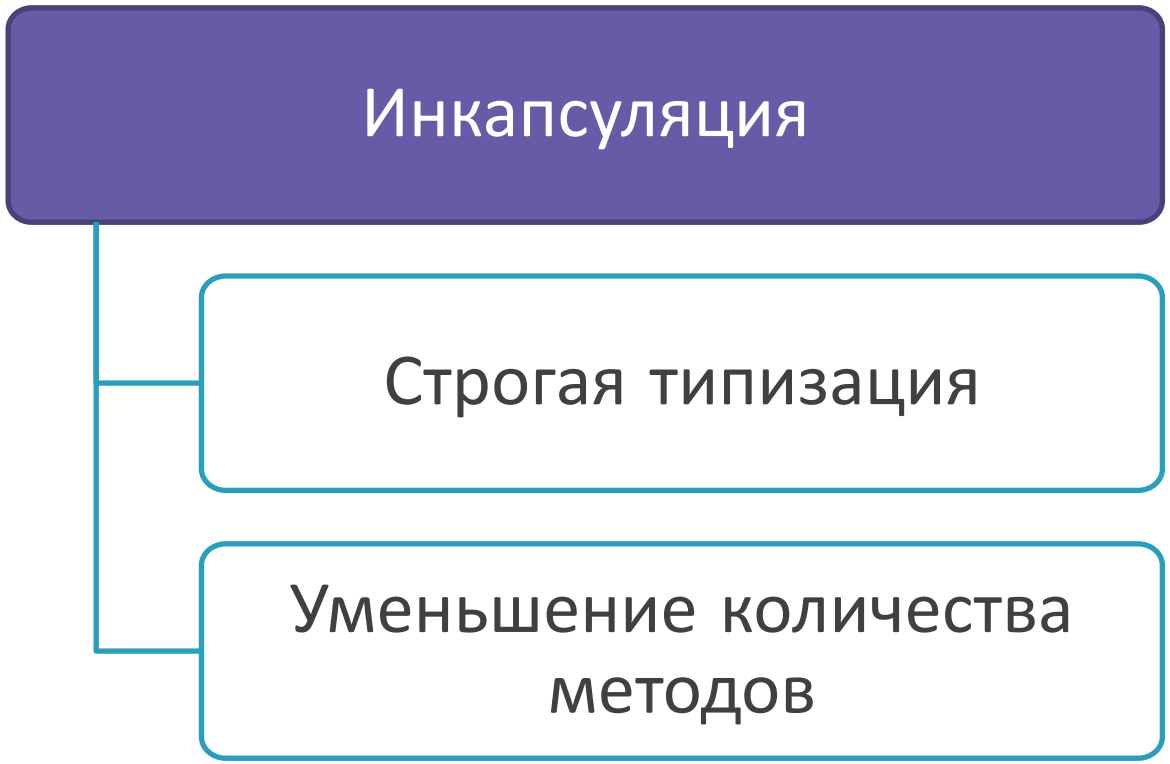
“Объектно-ориентированное программирование делает код понятным через инкапсуляцию подвижных частей. Функциональное программирование делает код понятным через минимизацию подвижных частей.”

Michael Feathers

Анемичная модель и функциональное программирование

```
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



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; }
```

Value Objects!

```
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; }
}
```

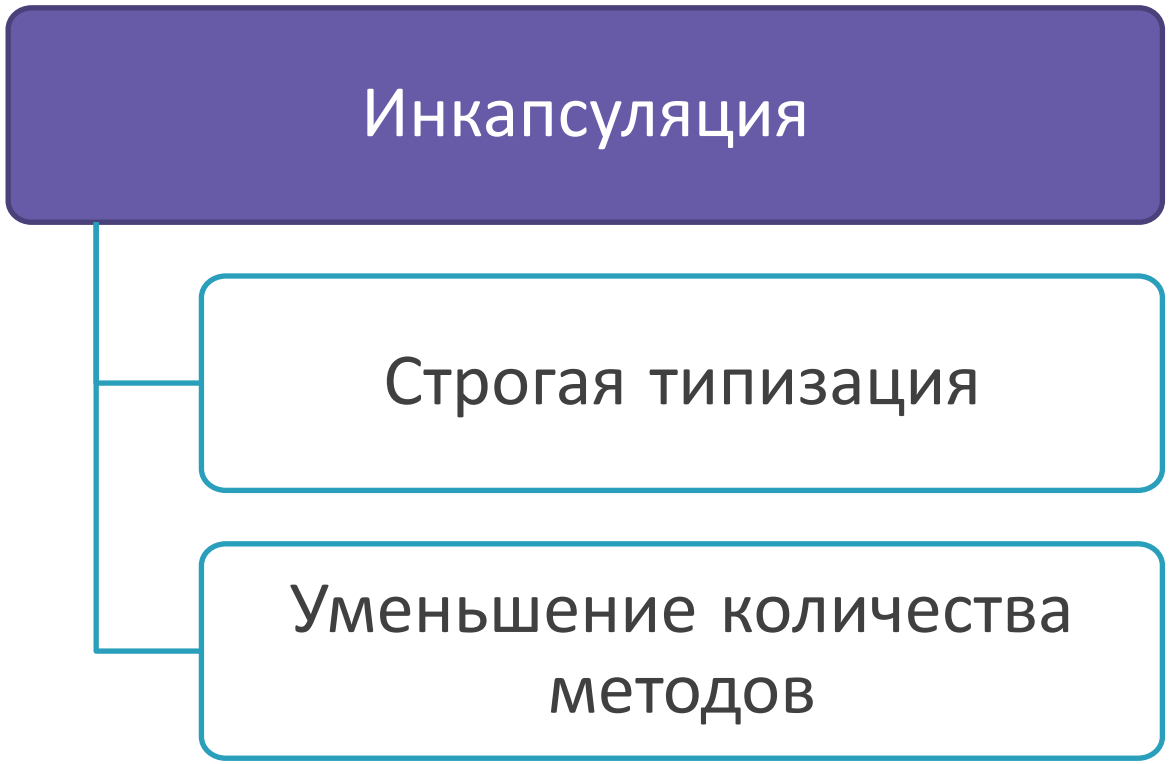
```
public class Customer {
    public string Name { get; set; }
    public Email Email { get; set; }
    public Status Status { get; set; }
    public List<Order> Orders { get; set; }
    public Discount CurrentDiscount { get; set; }
}
```

Value Objects

```
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



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;
        }
    }
}
```

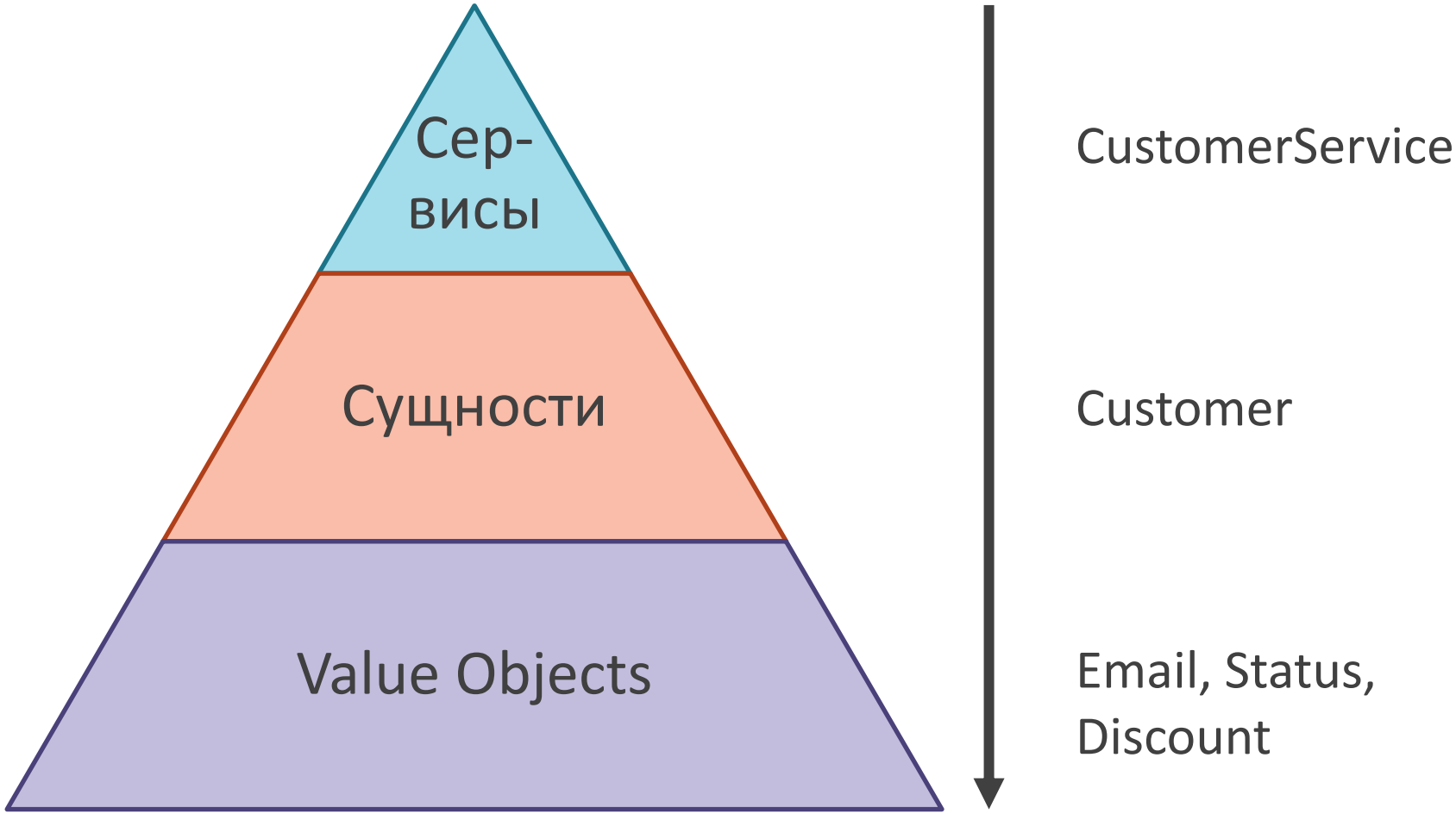
```
public class Customer {
    public string Name { get; }
    public Email Email { get; }
    public Status Status { get; private set; }
    public Discount CurrentDiscount => Status.GetDiscount();

    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

```
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);
    }
}
```

Рефакторинг анемичной модели

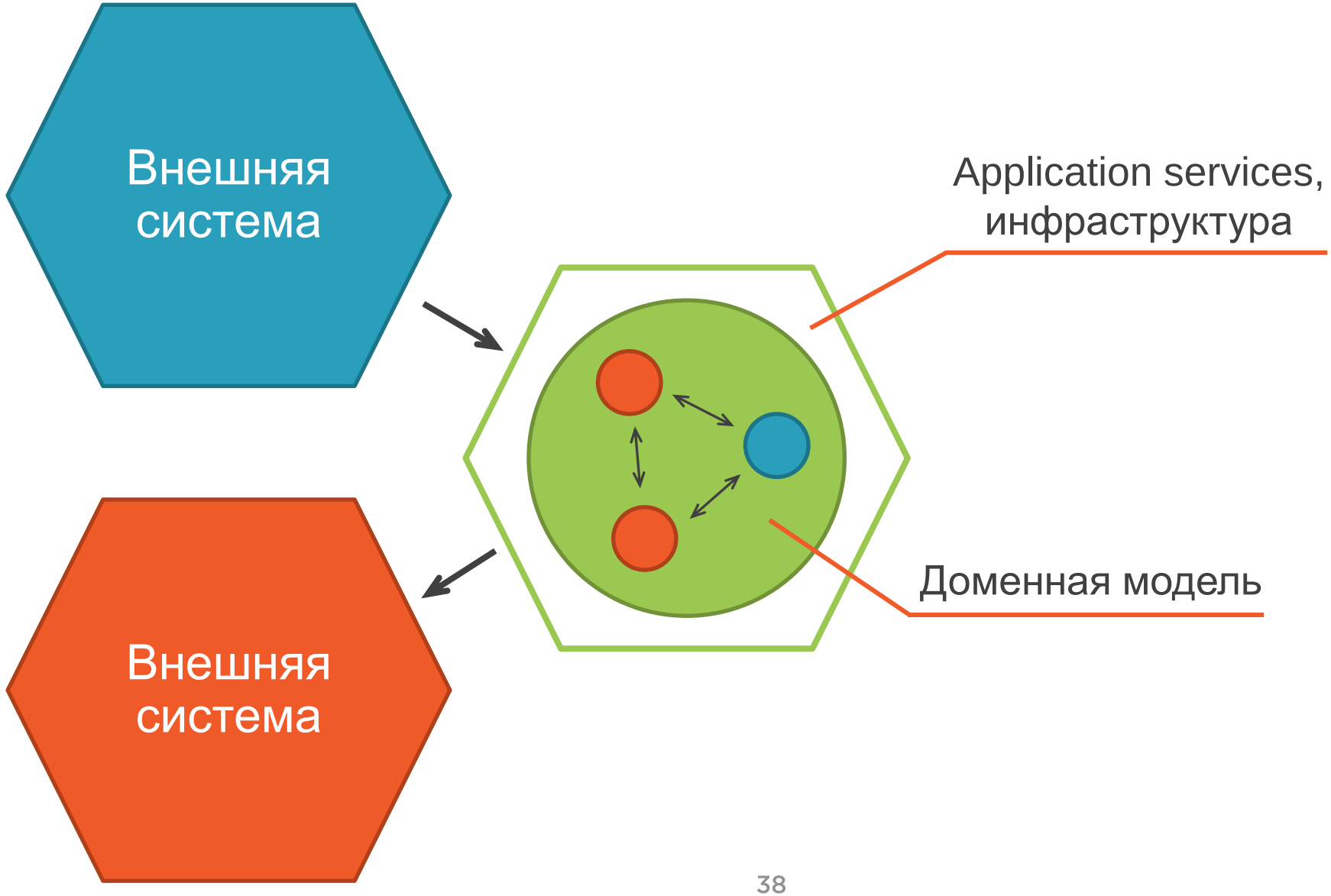


План доклада

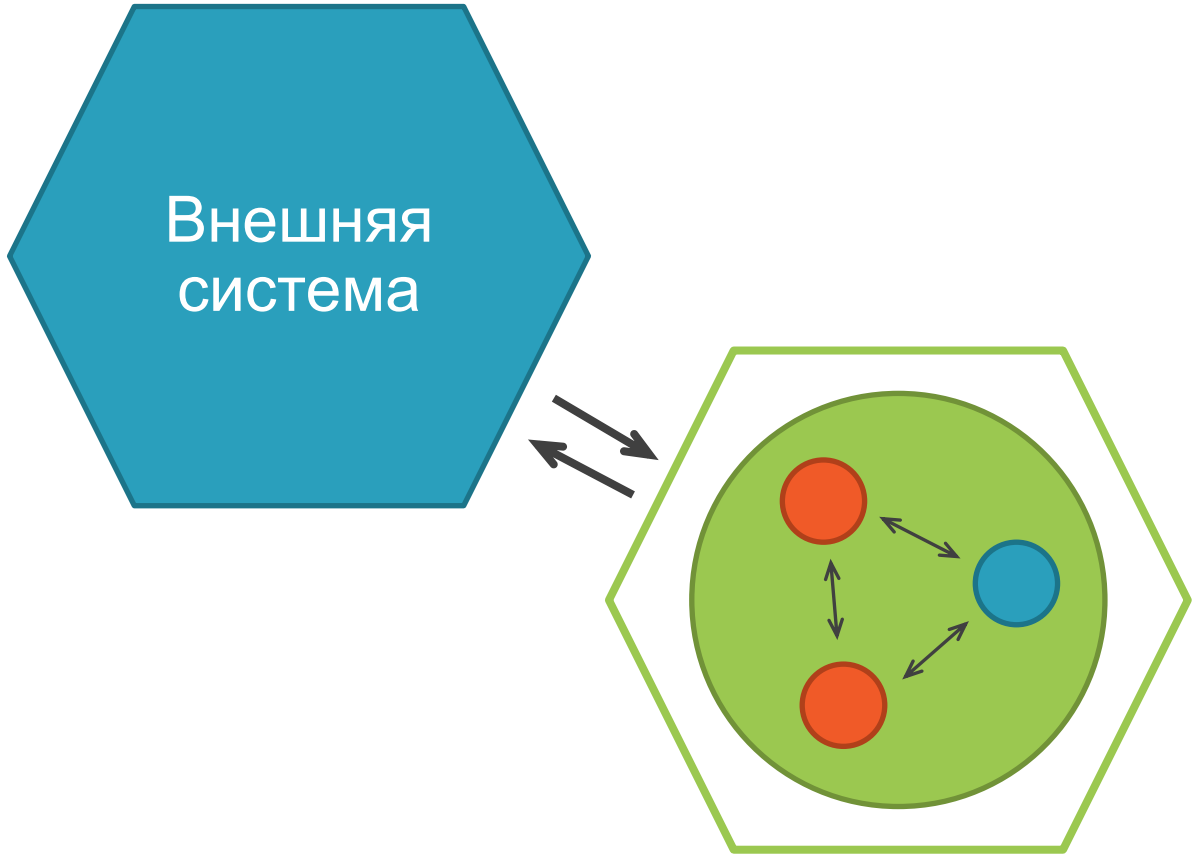
Самое важное в DDD

- Основные принципы 
- Разница между анемичной и богатой моделью 
- Изоляция доменной модели
- DDD трилемма

Изоляция доменной модели



Изоляция доменной модели



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

Изоляция доменной модели

(Domain model purity)

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

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

Active Record

Доменная модель не изолирована (impure)

Изоляция доменной модели

(Domain model purity)

```
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);
}
```

План доклада

Самое важное в DDD

- Основные принципы 
- Разница между анемичной и богатой моделью 
- Изоляция доменной модели 
- 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";
    }
}
```

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();
    }
}
```

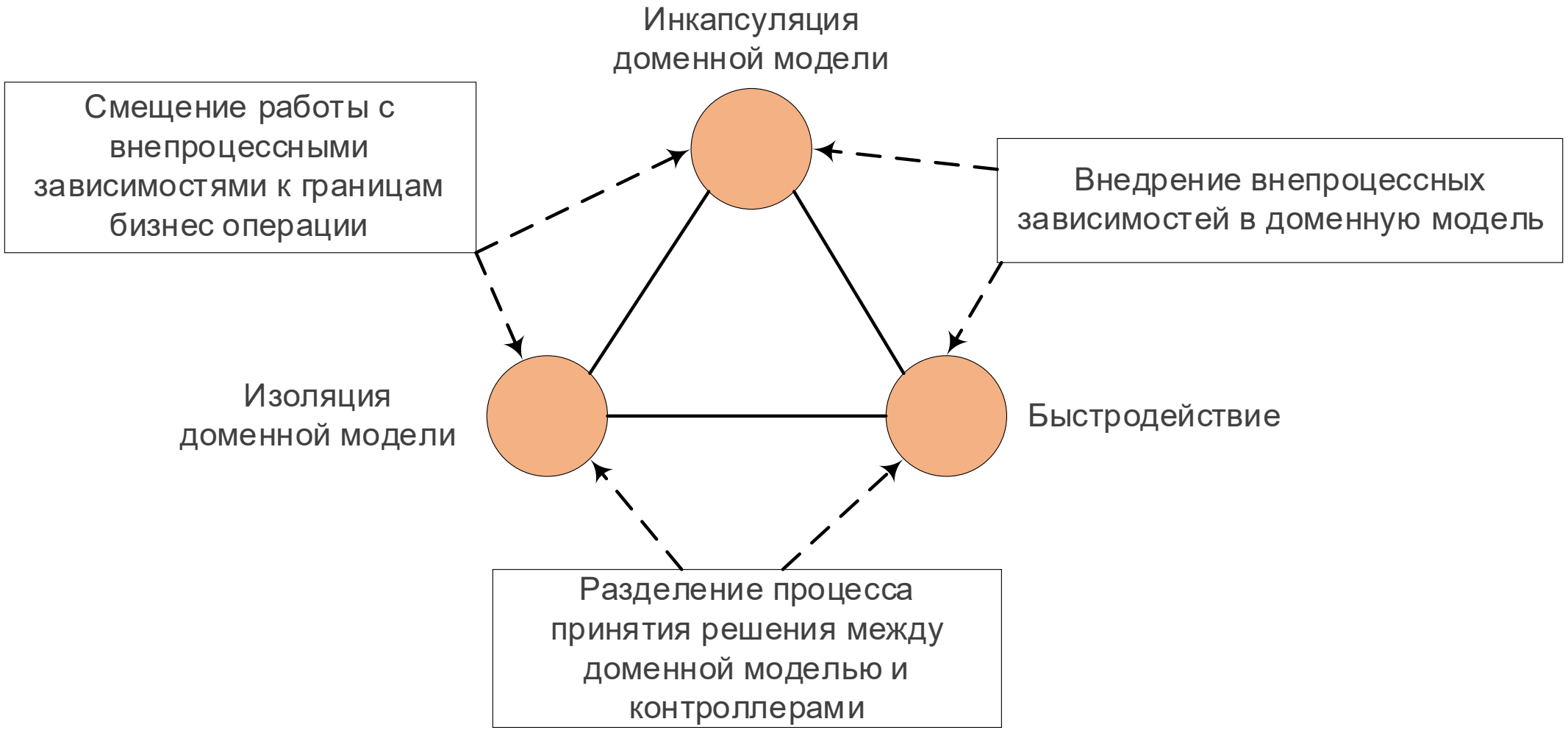
Доменная модель не изолирована

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

        Email = newEmail;

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

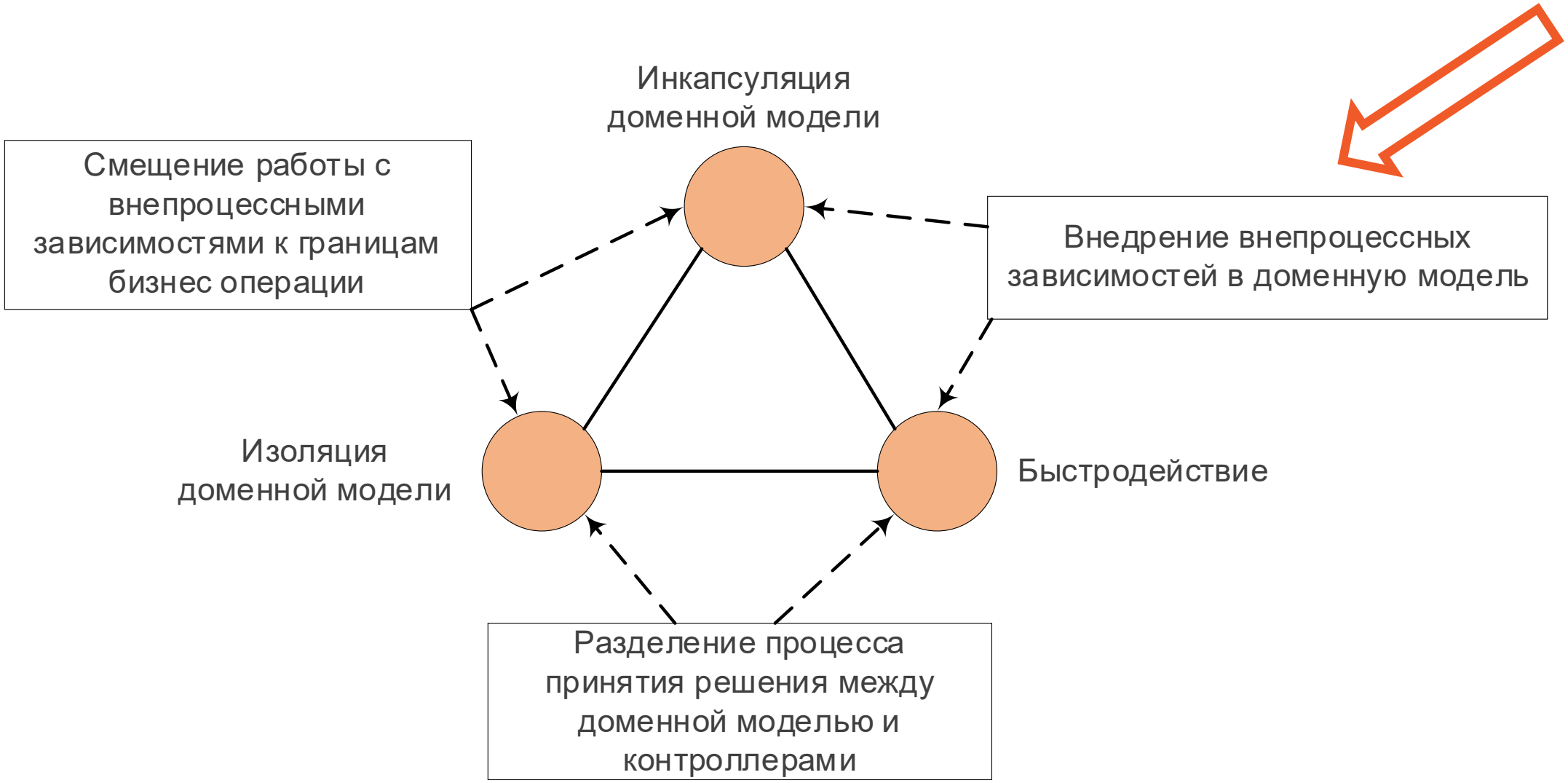
Доменная модель не изолирована

`CustomerRepository` **vs.** `ICustomerRepository`

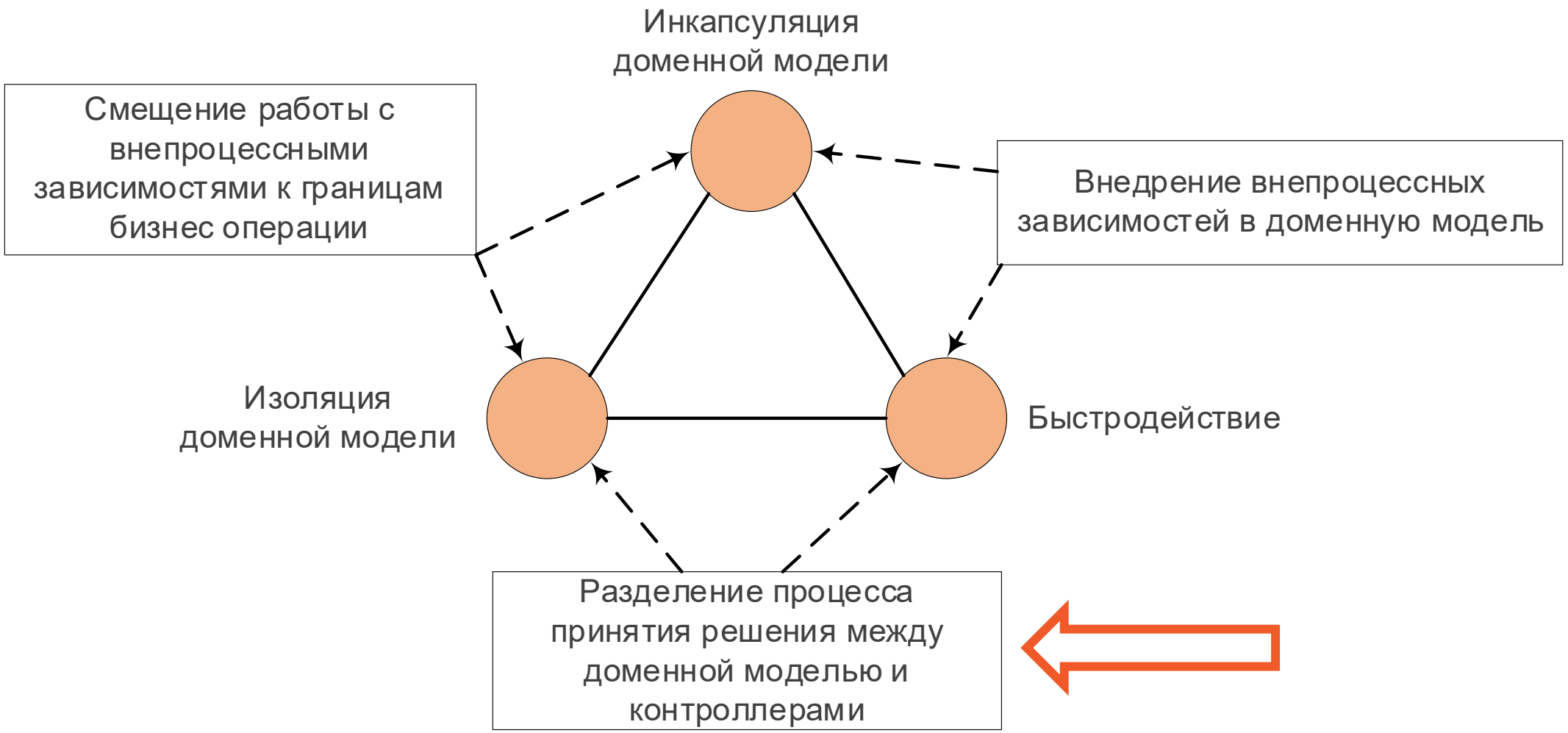
DDD трилемма



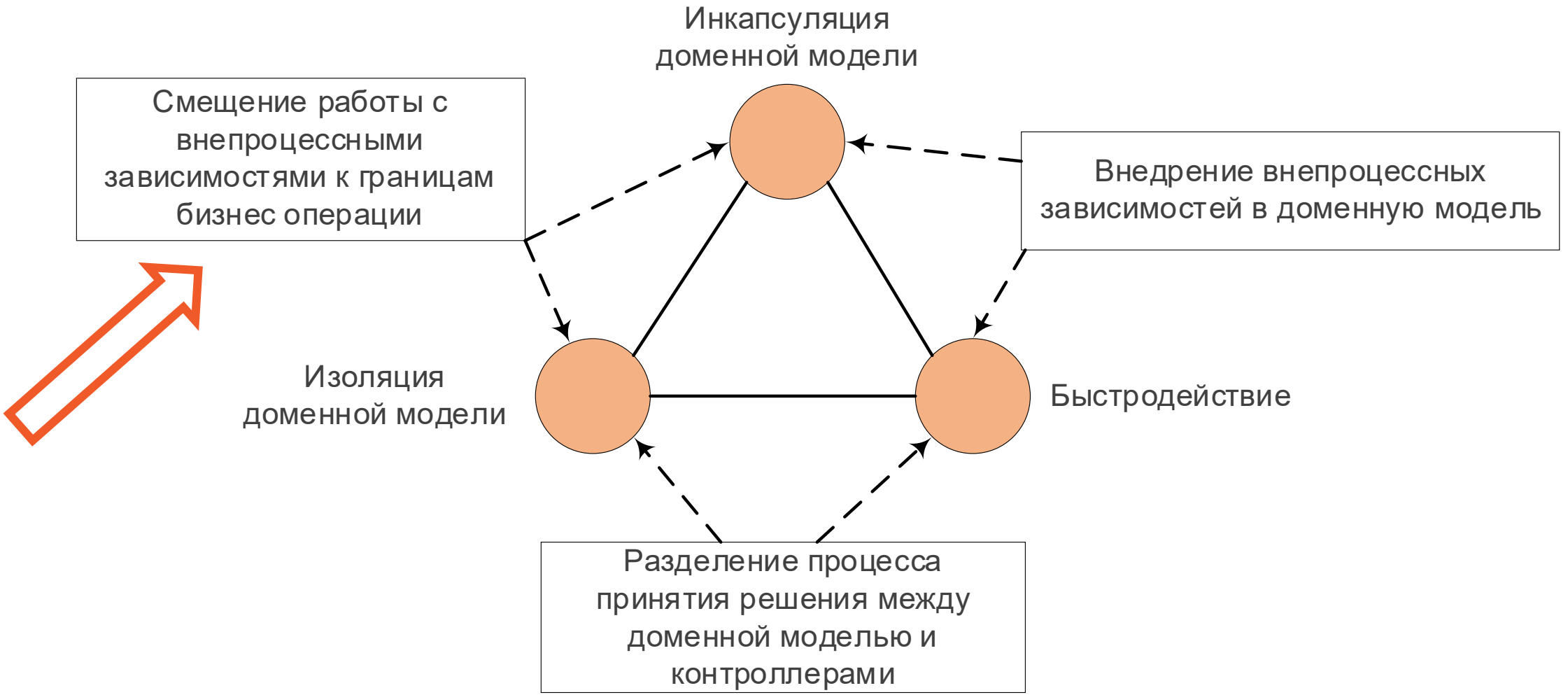
DDD трилемма



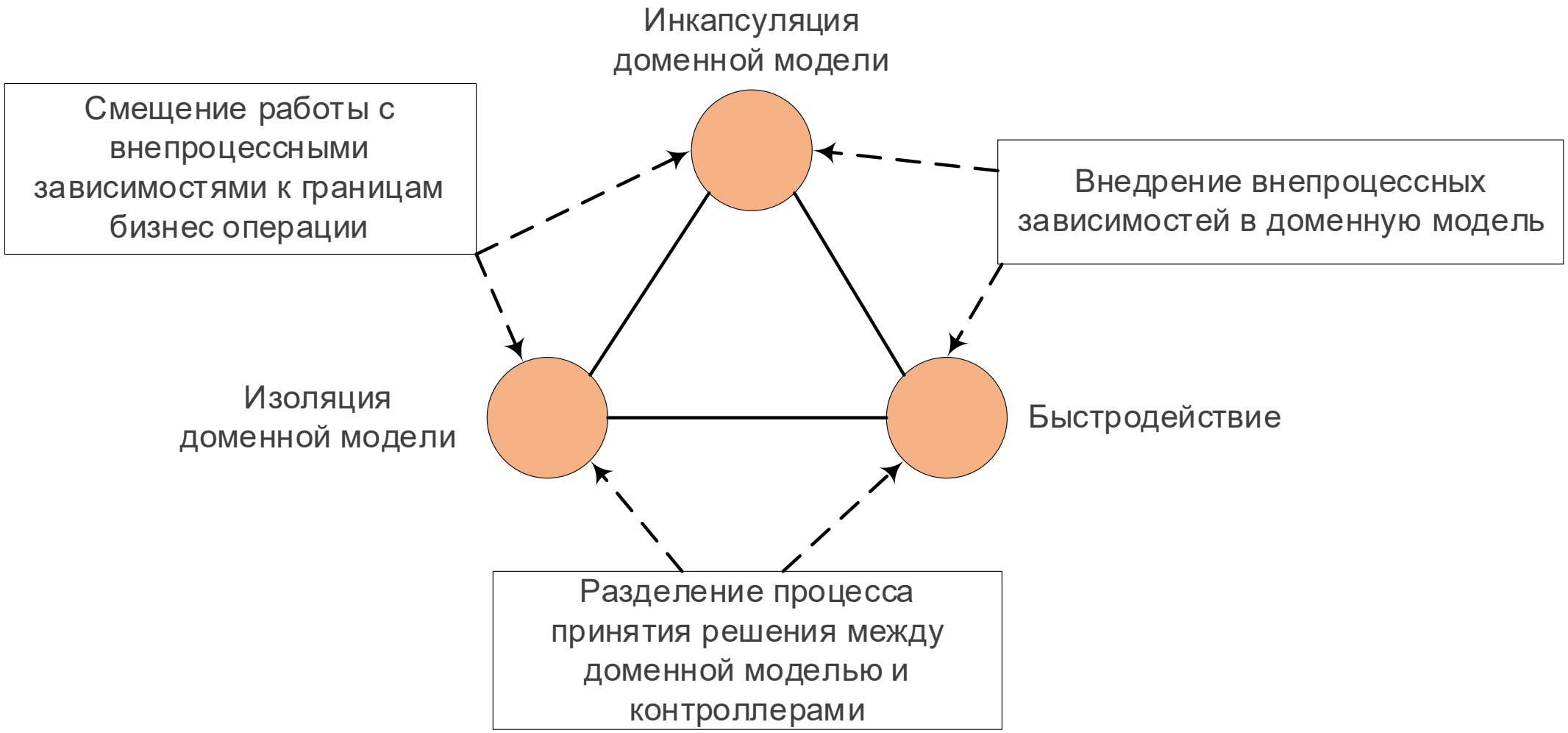
DDD трилемма



DDD трилемма

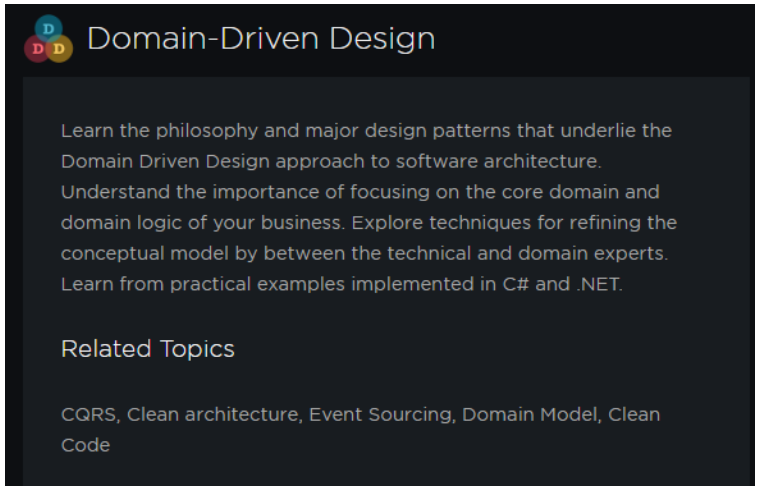


DDD трилемма



Изоляция > Инкапсуляция > Быстродействие

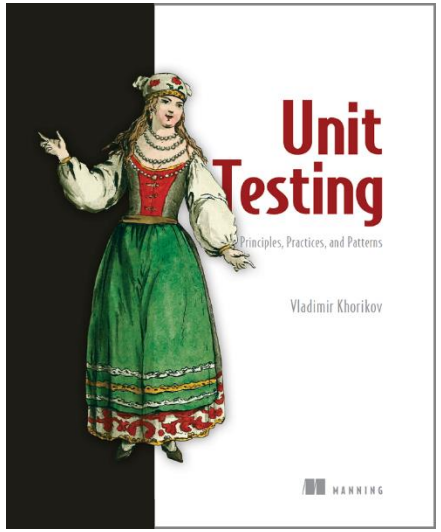

Самое важное в DDD
Основные принципы
Разница между анемичной и богатой моделью
Изоляция доменной модели
DDD трилемма



http://bit.ly/ddd-path

vlad@enterprisecraftsmanship.com

DDD trilemma series:
http://bit.ly/ddd-trilemma



Unit Testing Principles, Practices, and Patterns

http://bit.ly/testing-book

Принципы юнит-тестирования

http://bit.ly/testing-book-ru