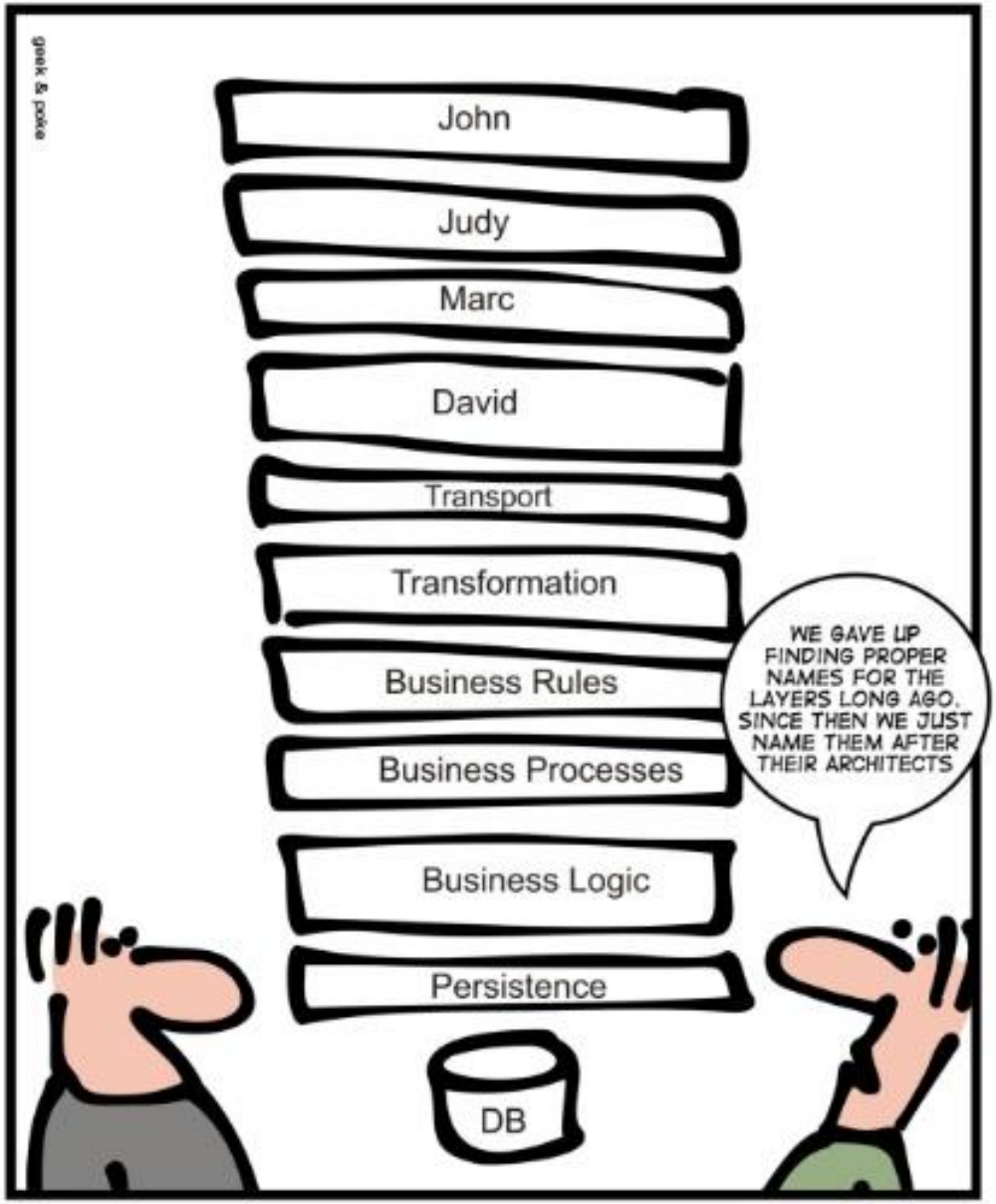
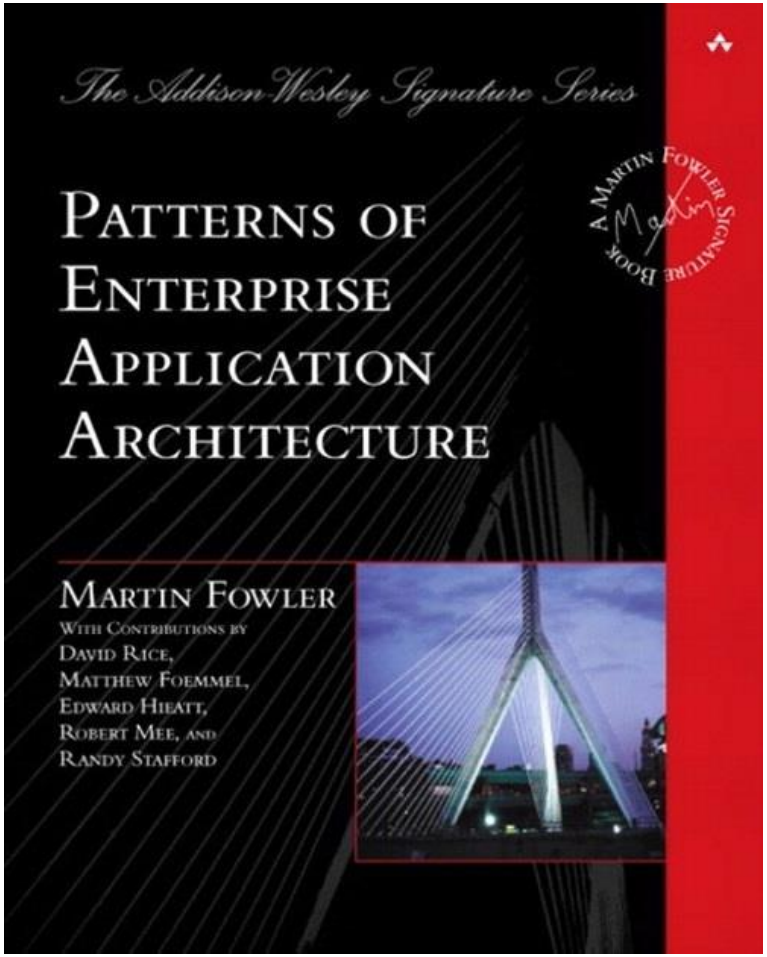
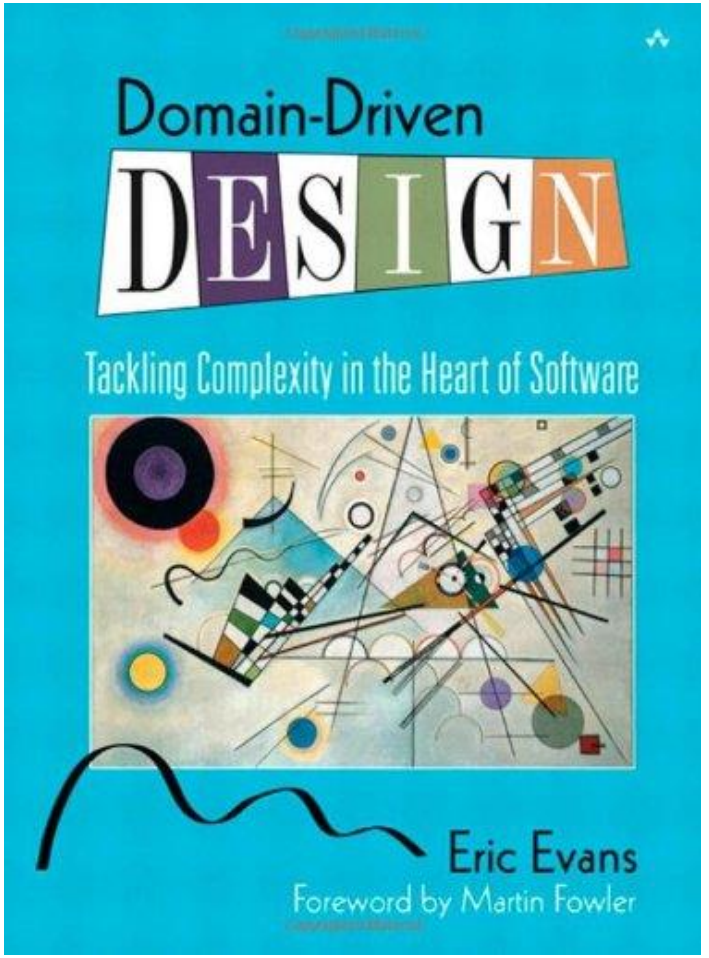
Traditional (Web) Application architecture

- Application Layering (Стандартные слои приложения)
- Domain Model vs Anemic Domain Model (Анемичная vs богатая доменная модель)
- Layers vs Slices



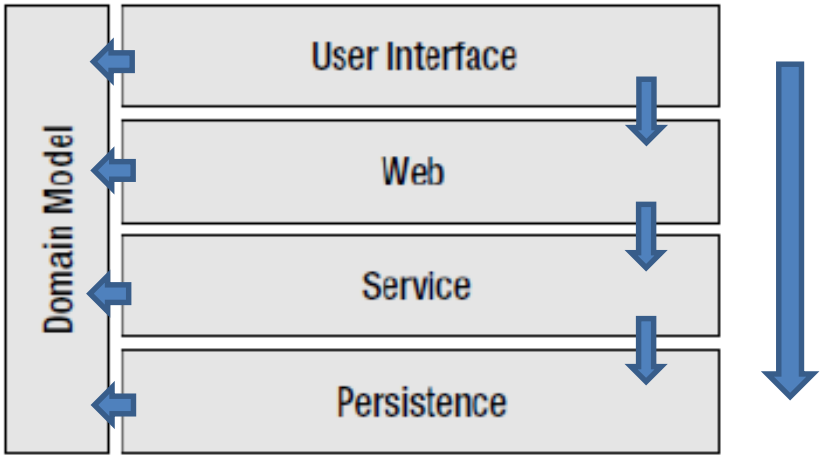
A GOOD ARCHITECT LEAVES A FOOTPRINT


The Addison Wesley Signature Series
A Martin Fowler Signature Book
Patterns of Enterprise Application Architecture
Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Application Layering

- Presentation Layer
  - User interface
  - Web
- Service Layer
- Domain Layer
  - Domain object model
- Infrastructure Layer
  - Repository
  - Persistence



Application Layering

- Зачем нужны слои
  - Разделение ответственности (Single responsibility principle)
  - Понятная архитектура
  - Гибкая архитектура и упрощение модификации (замены реализации)
  - Упрощение тестирования
- Принципы выделения слоев
- Связь между слоями
- Ответственность слоев

Examples

- *com.apress.prospringmvc.bookstore.domain* - **the domain layer**
- *com.apress.prospringmvc.bookstore.service* - **the service layer**
- *com.apress.prospringmvc.bookstore.repository* - **the data access layer**

https://github.com/olivergierke/whoops-architecture/tree/master/packages-before/src/main/java/de/olivergierke/whoops

https://github.com/mdeinum/pro-spring-mvc-code/tree/master/bookstore-shared/src/main/java/com/apress/prospringmvc/bookstore

Domain Layer/Model Layer

- Уровень предметной области (Domain Layer) или Уровень модели (Model Layer)
  - Отвечает за представление понятий прикладной предметной области
    - *Книга, Счет, Заказ*
  - Содержит логику поведения предметной области (business rules)
    - Добавить/удалить книгу в корзину, Не оформлять заказ с 0 книг
  - Этот уровень является главной, алгоритмической частью программы
    - Подсчитать общую сумму заказа с учетом новогодней скидки и стоимостью доставки в выходной
- Объекты предметной области, избавленные от необходимости выводить самих себя на экран, хранить в базах данных, распределять задачи и т.п.

Presentation Layer

- Интерфейс пользователя (User Interface) или Уровень представления (Presentation Layer)
  - Отвечает за вывод информации пользователю и интерпретирование его команд
  - Внешним действующим субъектом может быть не человек, а другая компьютерная система
  - Стандартная архитектура MVC (Model View Controller pattern)

Service Layer/Application Layer

- Выполняется координирование задач и распределение работы между совокупностями объектов предметной области
- В нем **не** содержатся business rules
- Определяет набор use cases которые система предоставляет пользователю (уровню представления/другой системе/разным клиентам)
- Определяет границы бизнес-транзакций и безопасности

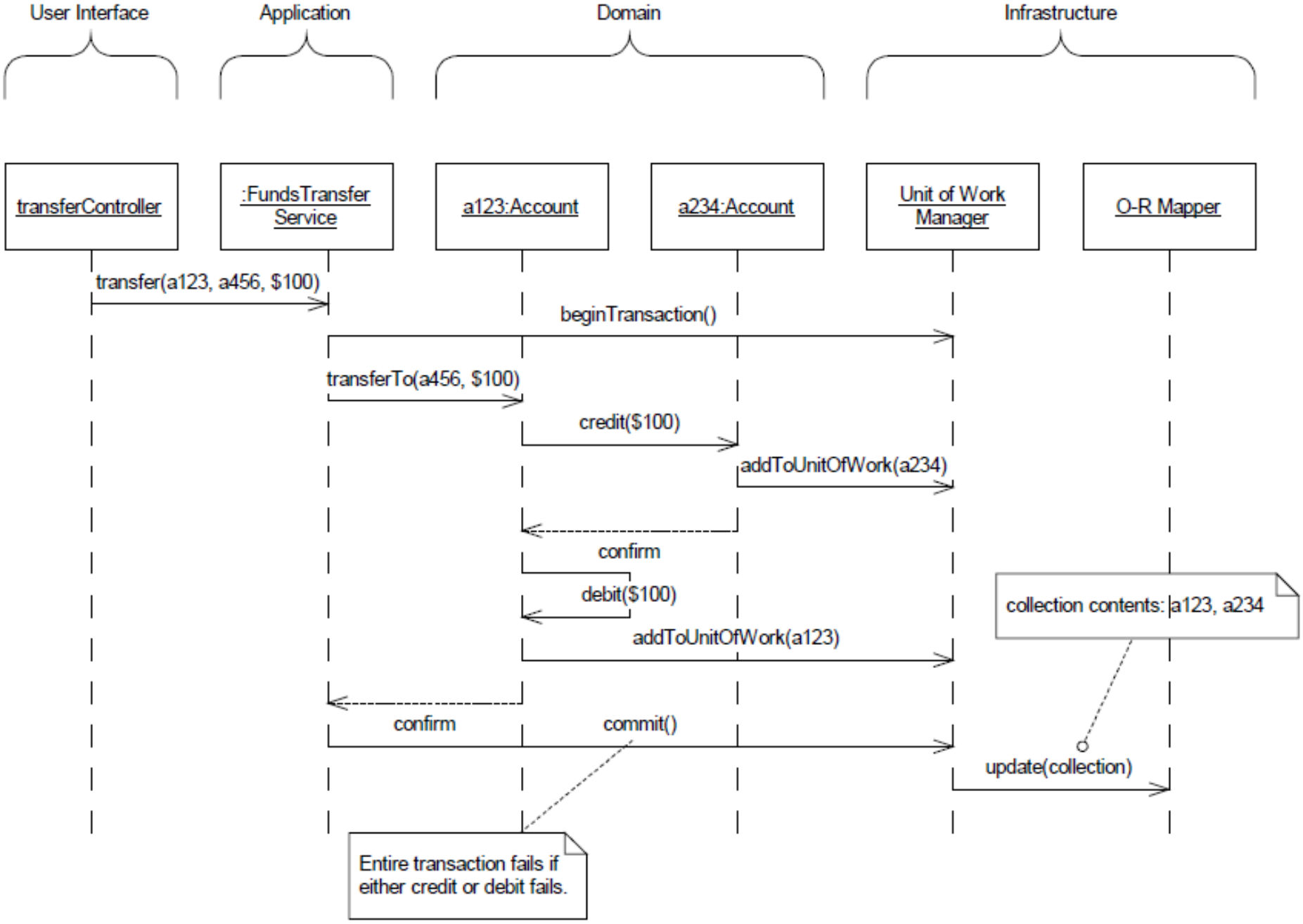
Service Layer

- The service layer provides a
  - stateless,
  - coarse-grained interface for clients to use for system interaction,
  - single point of entry
  - apply security at this layer
- Each method in the service layer typically represents one use case
- Each method is also one transactional unit of work

Repository/Data Access Layer

- Data Access Layer несет ответственность за взаимодействие с механизм сохранения/получения данных
  - каким образом получать и сохранять данные доменной модели в БД/файле

```
public interface AccountRepository {
    Account findByUsername(String username);
    Account findById(long id);
    Account save(Account account);
}
```


User Interface
Application
Domain
Infrastructure
transferController
:FundsTransfer Service
a123:Account
a234:Account
Unit of Work Manager
O-R Mapper
transfer(a123, a456, $100)
beginTransaction()
transferTo(a456, $100)
credit($100)
addToUnitOfWork(a234)
confirm
debit($100)
collection contents: a123, a234
addToUnitOfWork(a123)
confirm
commit()
update(collection)
Entire transaction fails if either credit or debit fails.

Coarse-Grained Service Layer Interface

Coarse-Grained

```
public interface AccountService {
    Account save(Account account);
    Account getAccount(String username);
    Account login(String username, String password) throws
    AuthenticationException;
}
```

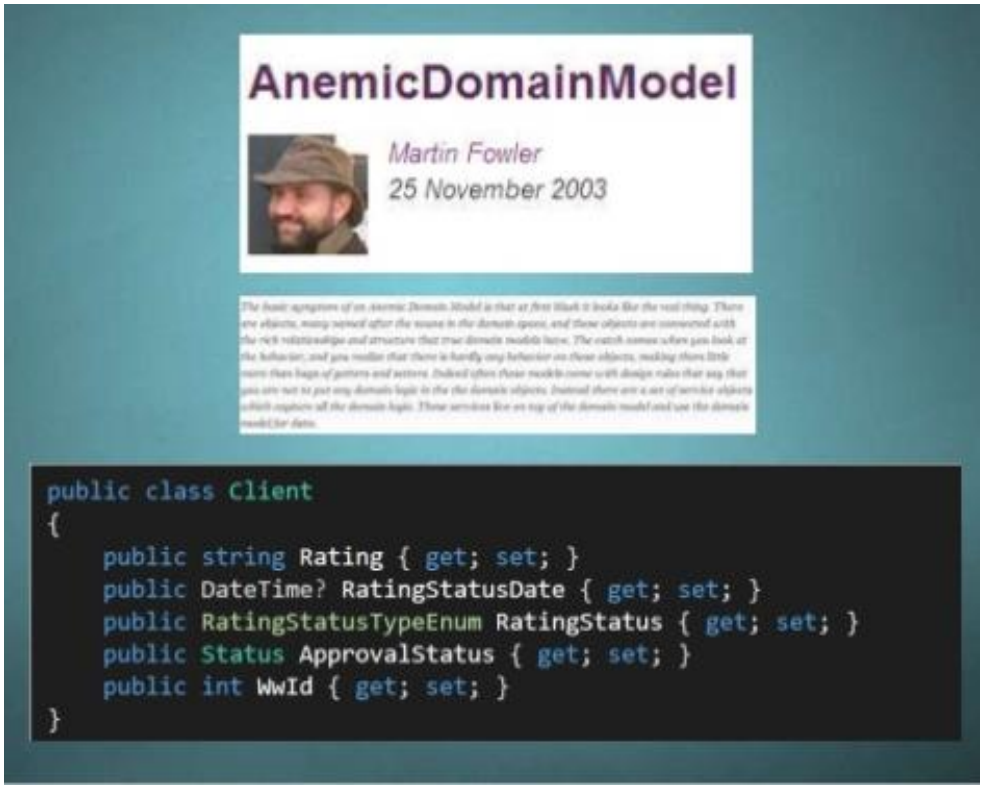
Fine-Grained

```
public interface AccountService {
    Account save(Account account);
    Account getAccount(String username);
    void checkPassword(Account account, String password);
    void updateLastLogin(Account account);
}
```

Rich Domain vs Anemic Domain Model

http://martinfowler.com/bliki/AnemicDomainModel.html

- **Anemic domain model** is the use of a software domain model where the domain objects contain little or no business logic
- Needs a service layer when sharing domain logic (business logic) across differing consumers of an object model



https://github.com/link-intersystems/blog/tree/master/anemic-vs-rich-domain-model

Anemic model drawbacks

- **Anemic models are procedural programming**
- Objects combine data and logic while anemic models separate them
- Anemic model is contradictory with fundamental object-oriented principles like: encapsulation, information hiding
- Anemic model has no logic that ensures that object is in a legal state at any time

http://www.link-intersystems.com/blog/2011/10/01/anemic-vs-rich-domain-models/

```c
struct order_item {
int amount;
double price;
char *name;
};

struct order {
int total;
struct order_item items[10];
};

int main(){
struct order order1;
struct order_item item;
item.name = "Domain-Driven";
item.price = 30.0;
item.amount = 5;
order.items[0] = item;
calculateTotal(order1);
}

void calculateTotal(order o){
int i, count;
count = 0;
for(i=0; i < 10; i++) {
order_item item = o.items[i];
o.total = o.total + item.price * item.amount;
}
}
```

Anemic Domain Model example

```
public class Client
{
    public string Rating { get; set; }
    public DateTime? RatingStatusDate { get; set; }
    public RatingStatusTypeEnum RatingStatus { get; set; }
    public Status ApprovalStatus { get; set; }
    public int WwId { get; set; }
}

public partial class ClientService
{
    public ClientService(IClientRepository clientRepository)...

    public void SaveClient(Client client)
    {
        _clientRepository.Save(client);
    }
}
```

Applying business rules

```
public partial class ClientService
{
    public ClientService(IClientRepository clientRepository)...

    public void SaveClient(Client client)
    {
        if (client.RatingStatus != RatingStatusType.Approved
            && !string.IsNullOrEmpty(client.Rating))
        {
            throw new Exception("A client rating must be approved.");
        }

        if (!string.IsNullOrEmpty(client.Rating) && client.WwId == 0)
        {
            throw new Exception("A rated client must have a Wwid.");
        }

        _clientRepository.Save(client);
    }
}
```

```
public class Client
{
    public string Rating { get; set; }
    public DateTime? RatingStatusDate { get; set; }
    public RatingStatusType RatingStatus { get; set; }
    public Status ApprovalStatus { get; set; }
    public int WwId { get; set; }
}
```

What if we have business rules on the client Status as well?

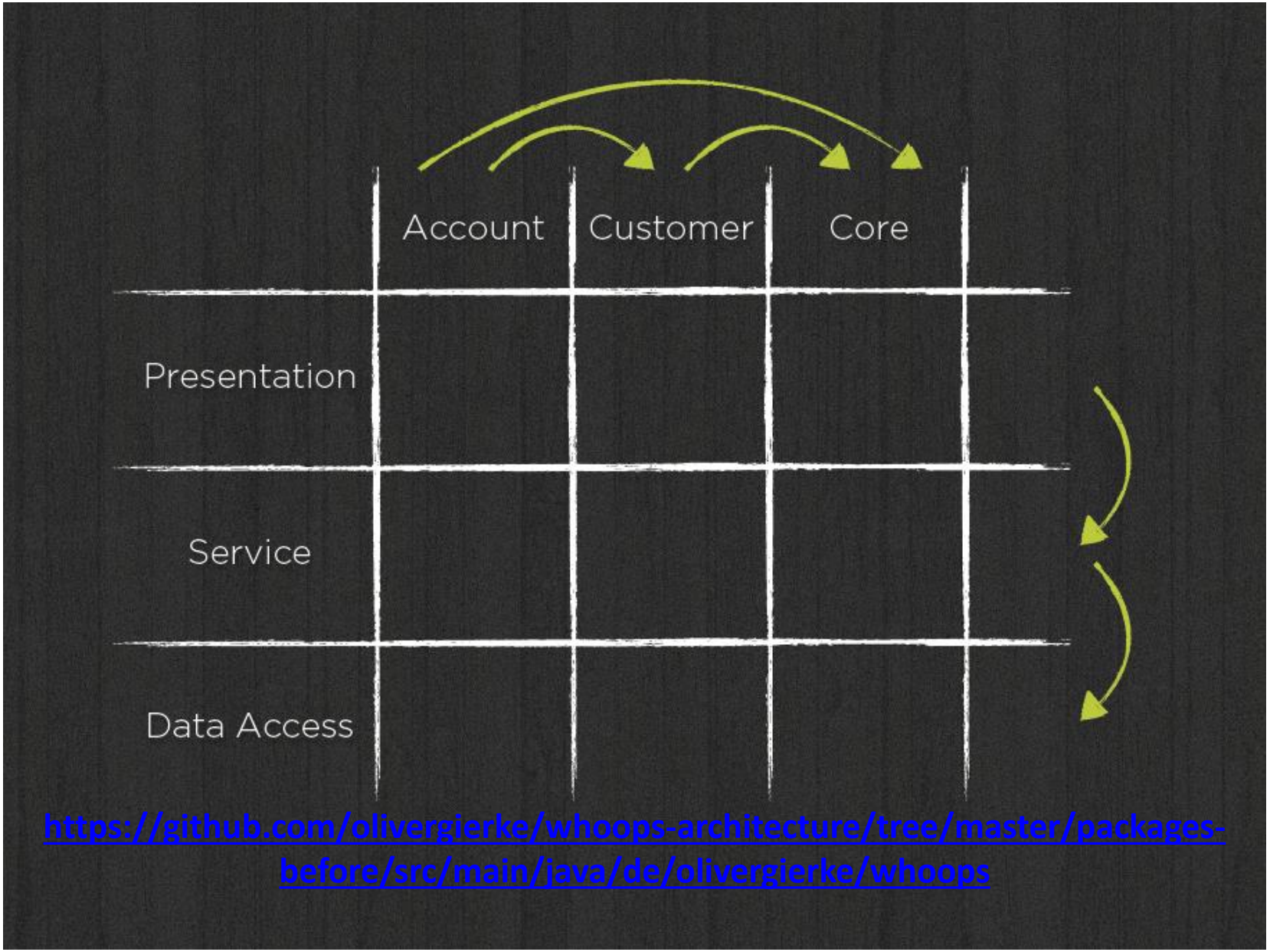
```
public partial class ClientService
{
    public ClientService(IClientRepository clientRepository)...

    public void SaveClient(Client client)
    {
        // other business rules
        ...

        var existingClient = _clientRepository.Get(client.Id);

        if (client.ApprovalStatus == Status.Accepted)
        {
            if (existingClient.ApprovalStatus == Status.Initial)
            {
                throw new Exception("Client must be confirmed before it can be accepted");
            }
        }
        else if ...

        else ...

        _clientRepository.Save(client);
    }
}
```

Things get messy quickly...

Empowering the domain: Encapsulating data and operations in domain objects

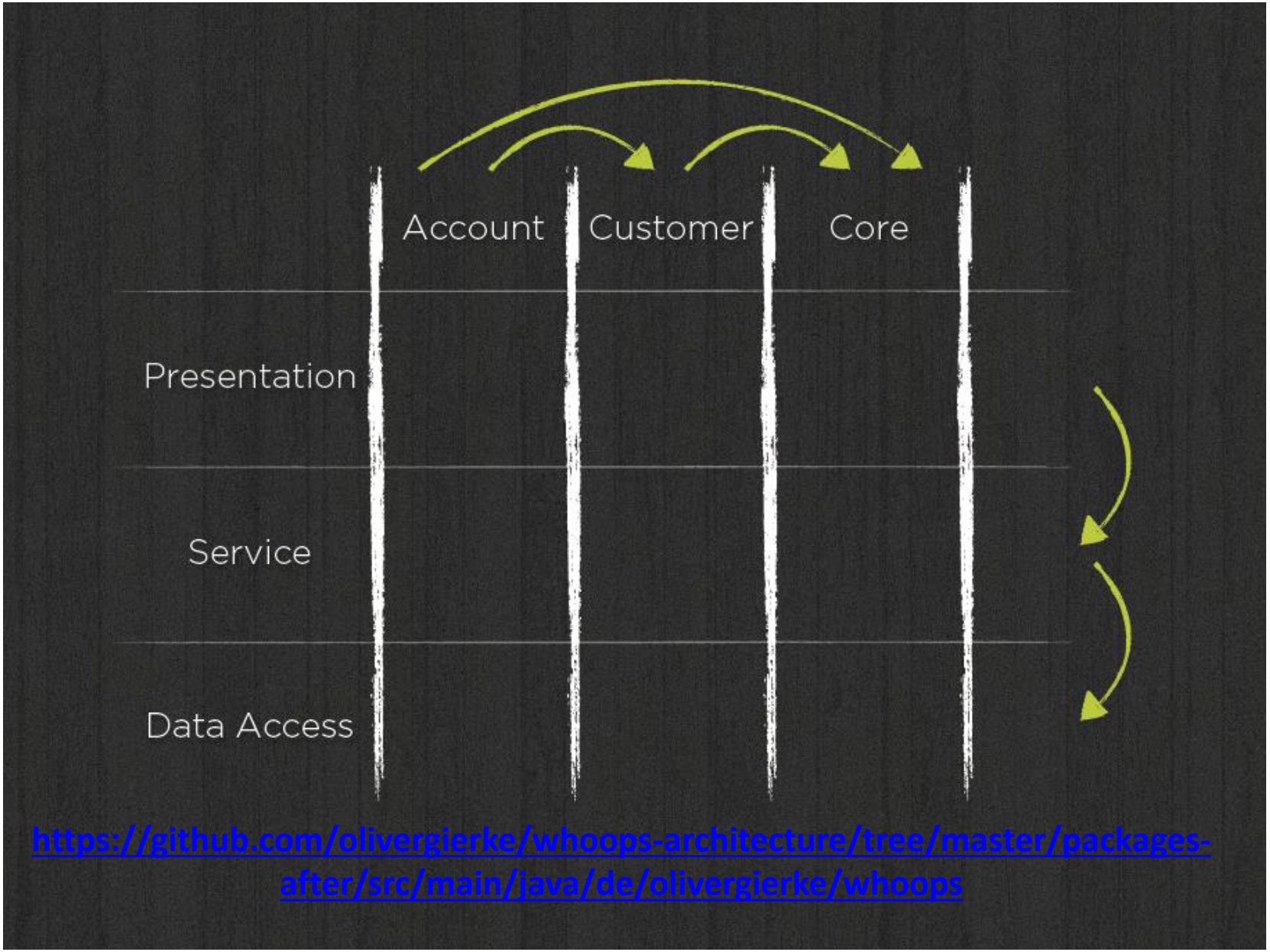
```
public class Client
{
    public string Rating { get; private set; }
    public DateTime? RatingStatusDate { get; private set; }
    public RatingStatusType RatingStatus { get; private set; }
    public Status ApprovalStatus { get; set; }
    public int WwId { get; set; }

    public void AssignRating(string rating, RatingStatusType status)
    {
        if (WwId == 0)
        {
            throw new Exception("A rated client must have a Wwid.");
        }

        // other business rules for rating
        ...

        Rating = rating;
        RatingStatusDate = DateTime.UtcNow;
        RatingStatus = status;
    }
}
```

Layers vs Slices



https://github.com/olivergierke/whoops-architecture/tree/master/packages-before/src/main/java/de/olivergierke/whoops

Slices -> Subsystems



https://github.com/olivergierke/whoops-architecture/tree/master/packages-after/src/main/java/de/olivergierke/whoops

- http://olivergierke.de/2013/01/whoops-where-did-my-architecture-go/
- http://www.codingthearchitecture.com/2015/03/08/package_by_component_and_architecturally_aligned_testing.html