CodeDesign

Drive predictability through Software Design

Application Infrastructure for Clean Architecture

Florin Coroș

florin@onCodeDesign.com

linkedin.com/in/florincoros

CodeDesign

Drive predictability through Software Design

Florin Coroș

Solution Architect Consultant

Technical Trainer

Founder of Code Design

enjoing playing GO

enjoing traveling

oncodedesign.com/webinars/app-infra-24

CodeDesign

Drive predictability through Software Design

Application Infrastructure for Clean Architecture

Florin Coroș

florin@onCodeDesign.com

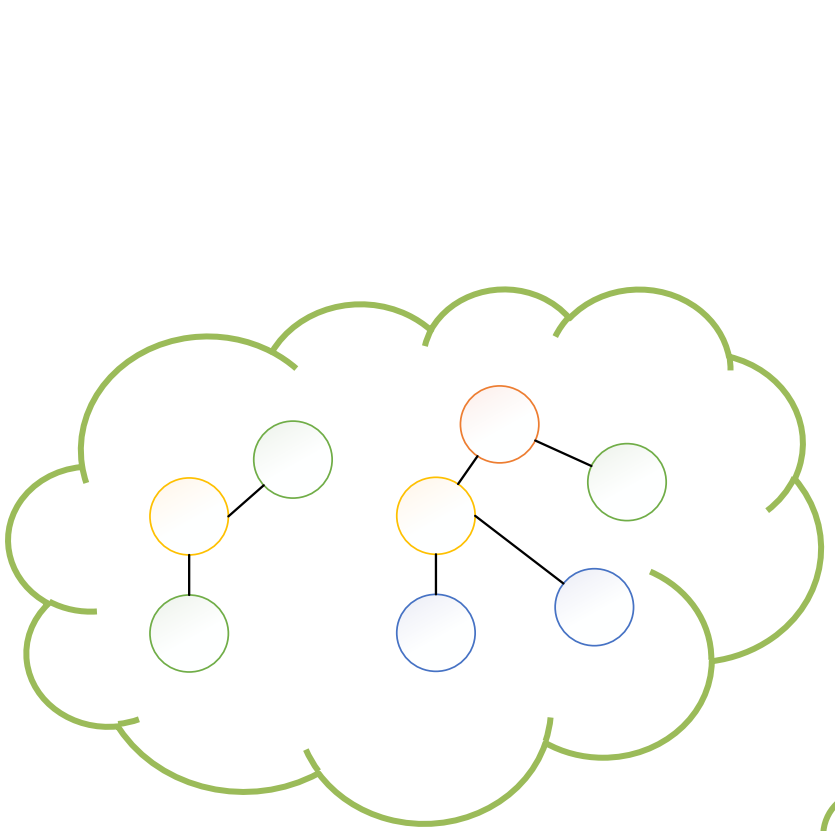
linkedin.com/in/florincoros

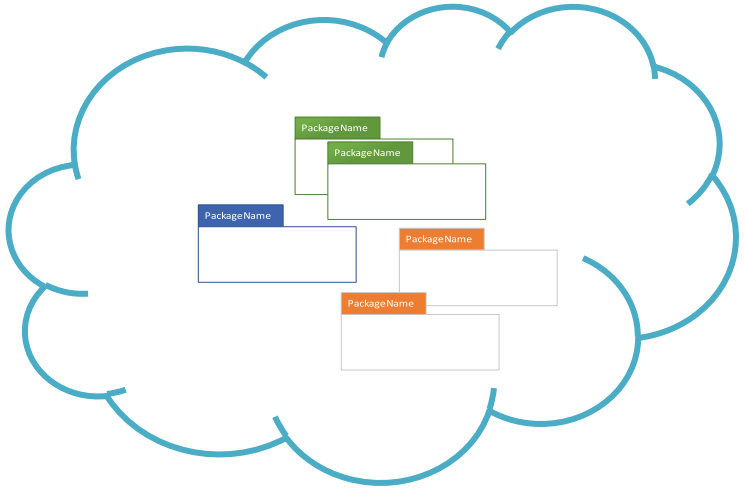
Why is Architecture Important?

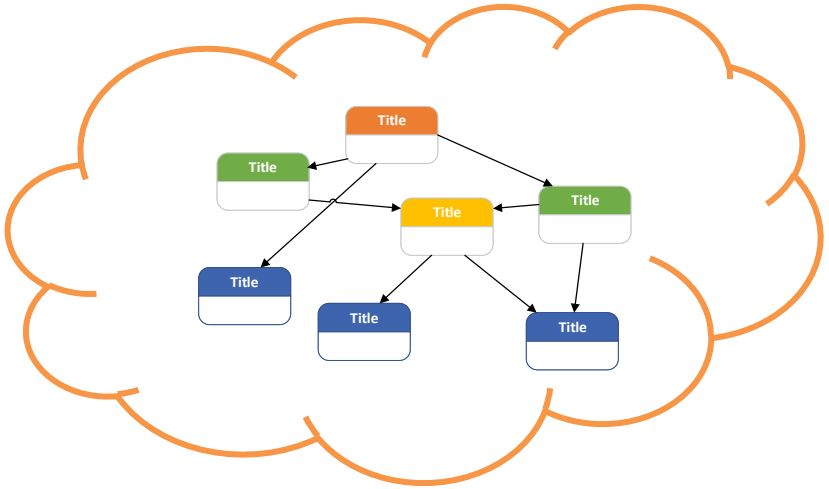
> ***The goal of software architecture is to minimize the human resources required to build and maintain the required system***

-- *Robert C. Martin*, Clean Architecture

Decompose - Separation of Concerns





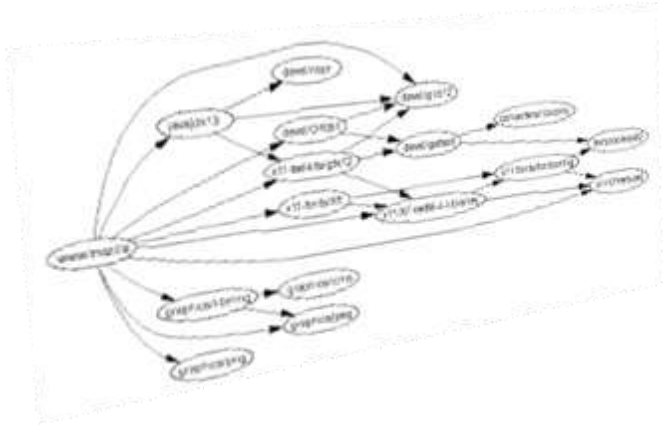


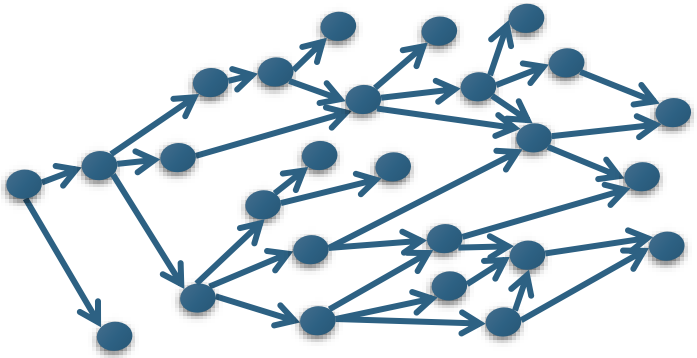
Manage the Complexity and Size

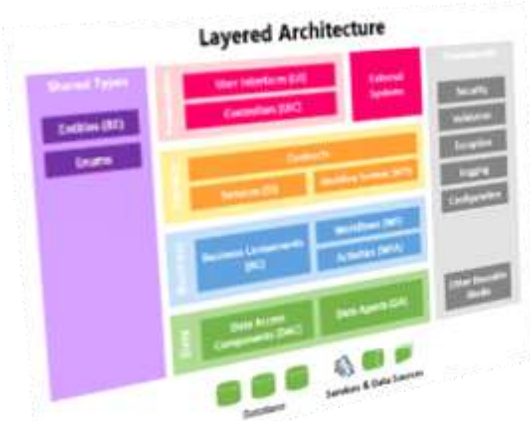
when projects do fail for reasons that are primarily technical, the reason is often

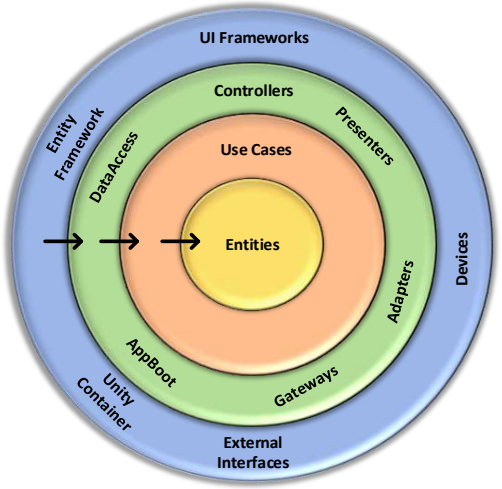
uncontrolled complexity

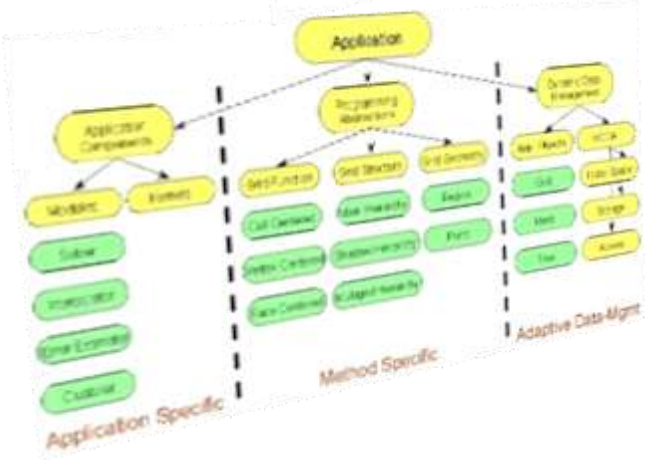
The Architecture is Separation of Concerns and a Set of Rules



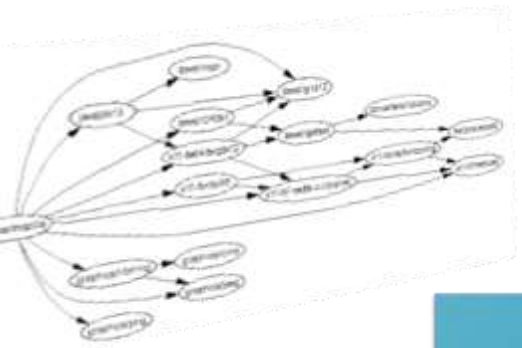


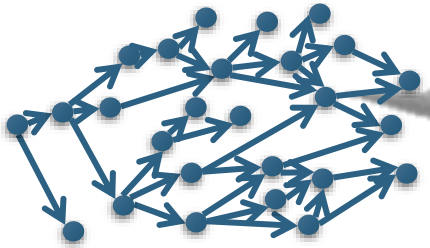


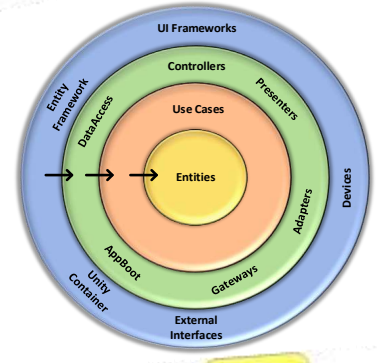


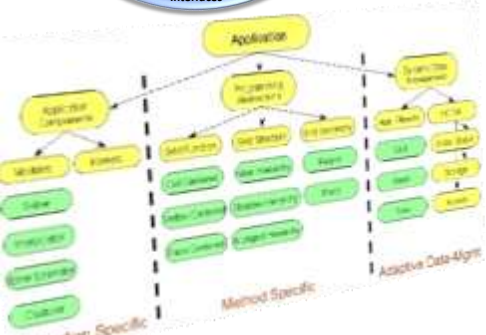


Implementing the Architecture







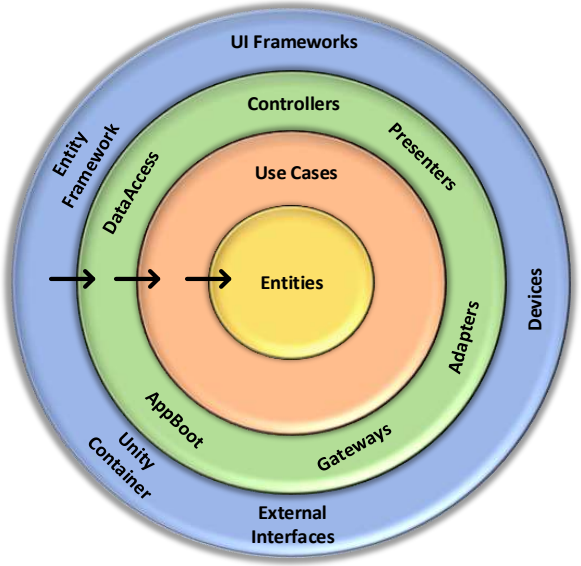


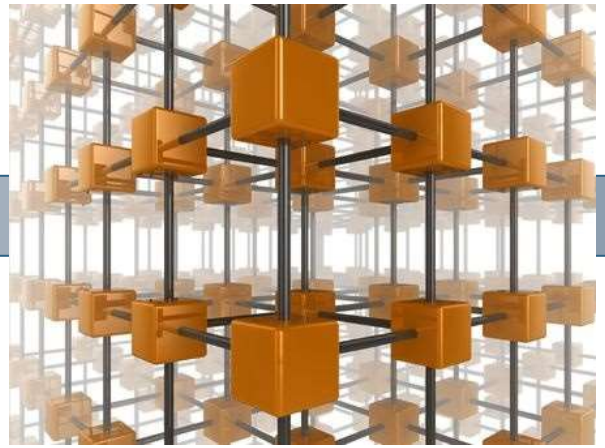
Presentation Layer

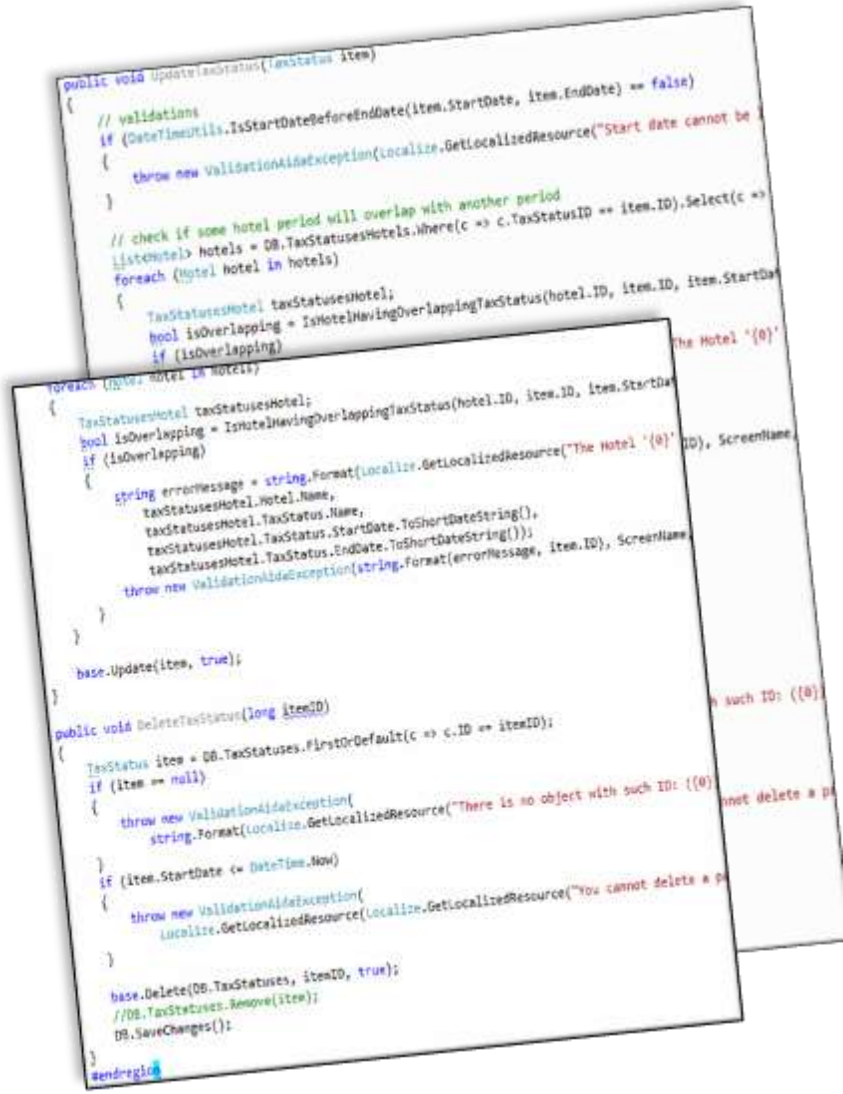
Business Logic Layer

Data Access Layer

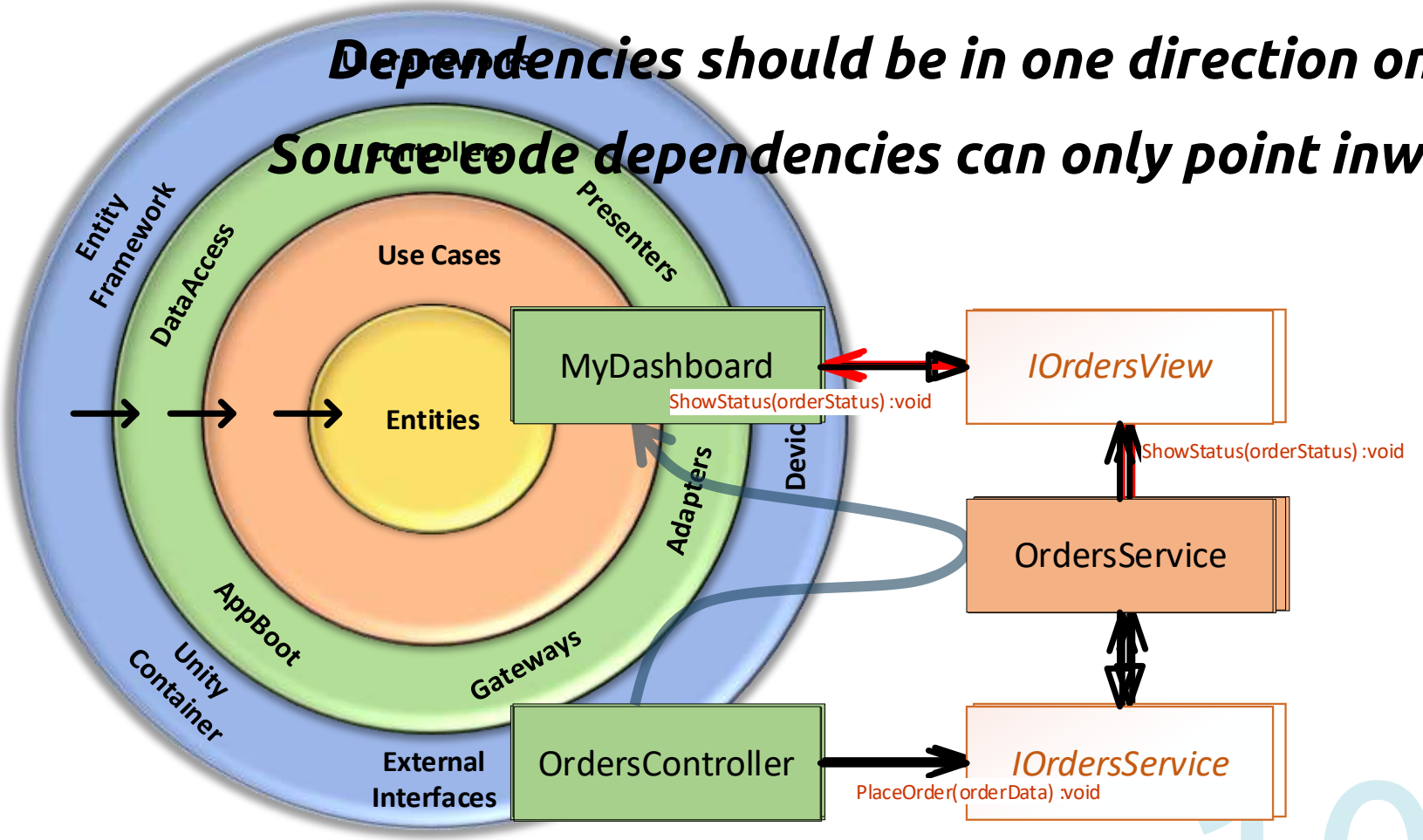
Application Infrastructure Enforces the Architecture Implementation







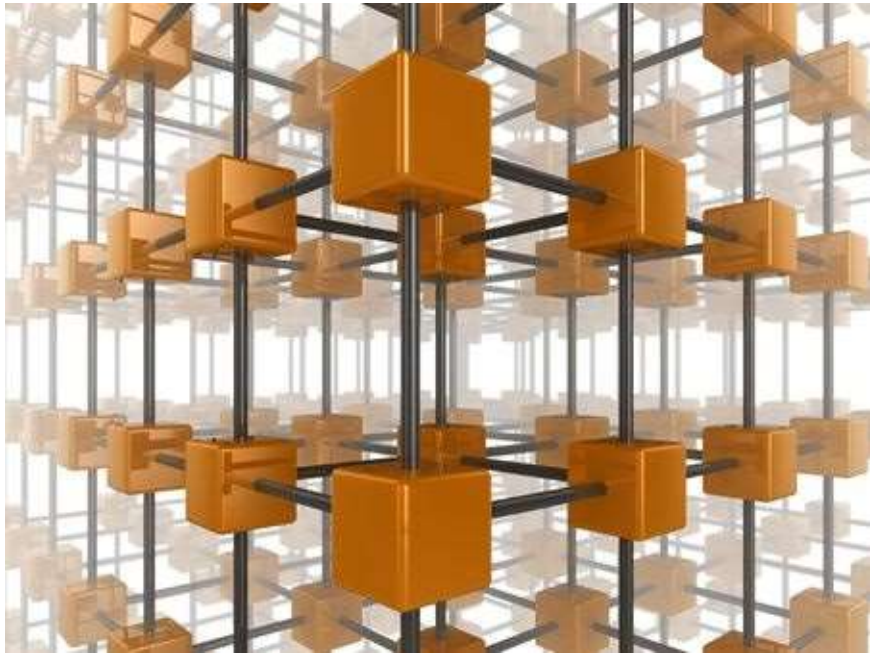
Clean Architecture



http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

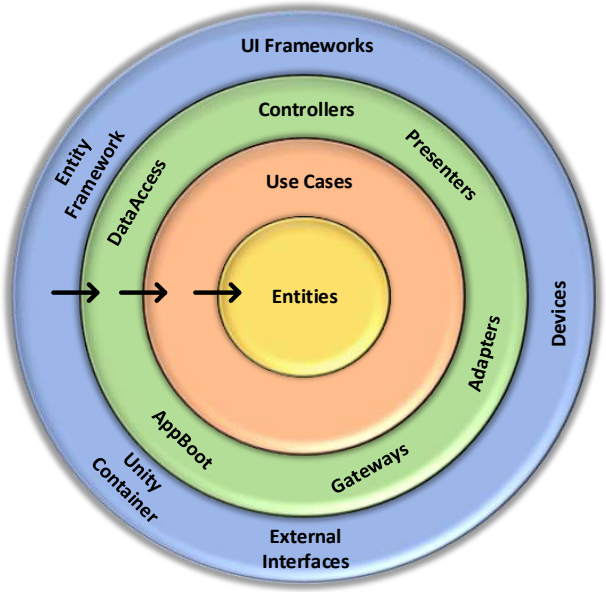
Building an Application Infrastructure

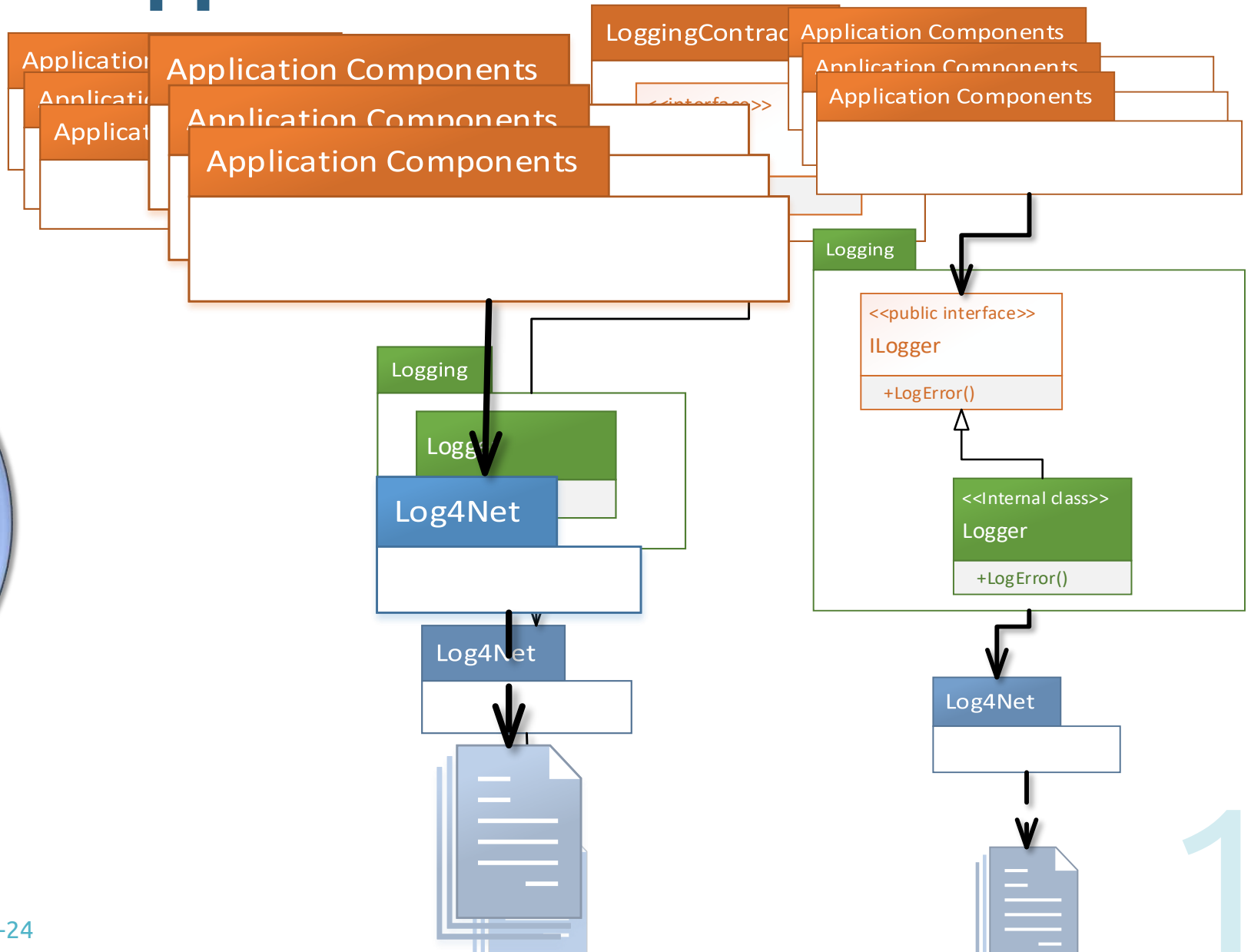
Create a structure that makes it <u>difficult to write bad code</u> and it makes it <u>easy to write good code</u>, code that follows the architecture and the design



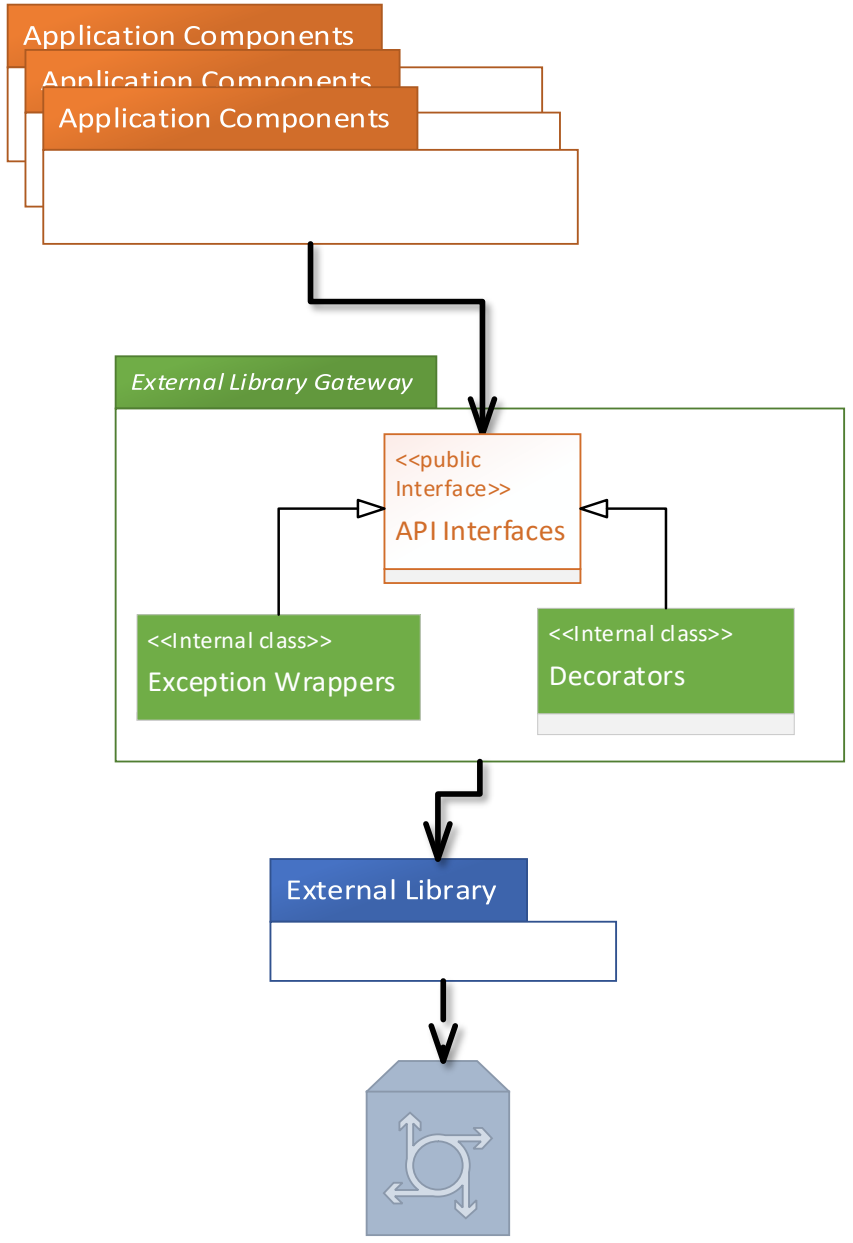
- **Hide external frameworks** to enforce the way they are used
- Use assemblies and references among them to **enforce rules**
- **Enforce *Constructor Dependency Injection*** that encourages *Programming Against Interfaces*

Creating the App Infra for Clean Architecture

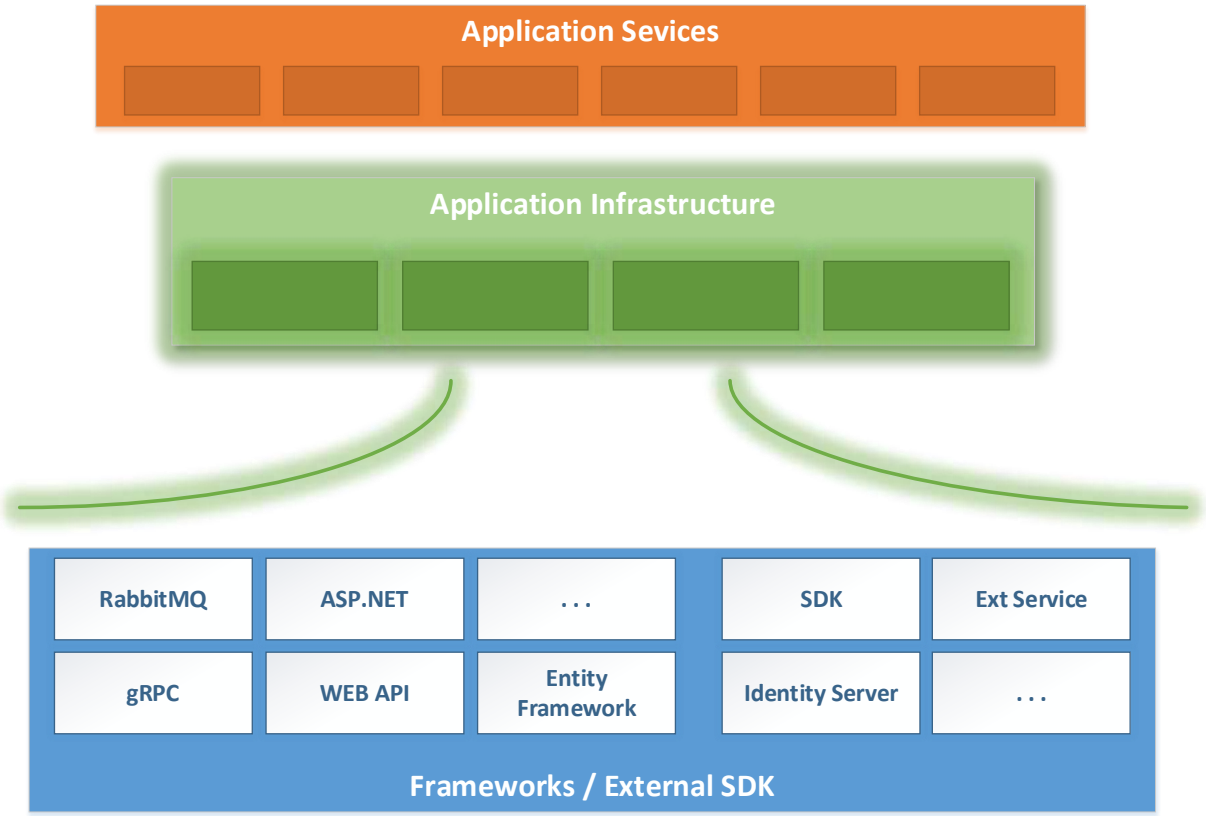




Hide External Libraries from the App Code



App Infrastructure is a Set of Tech Components



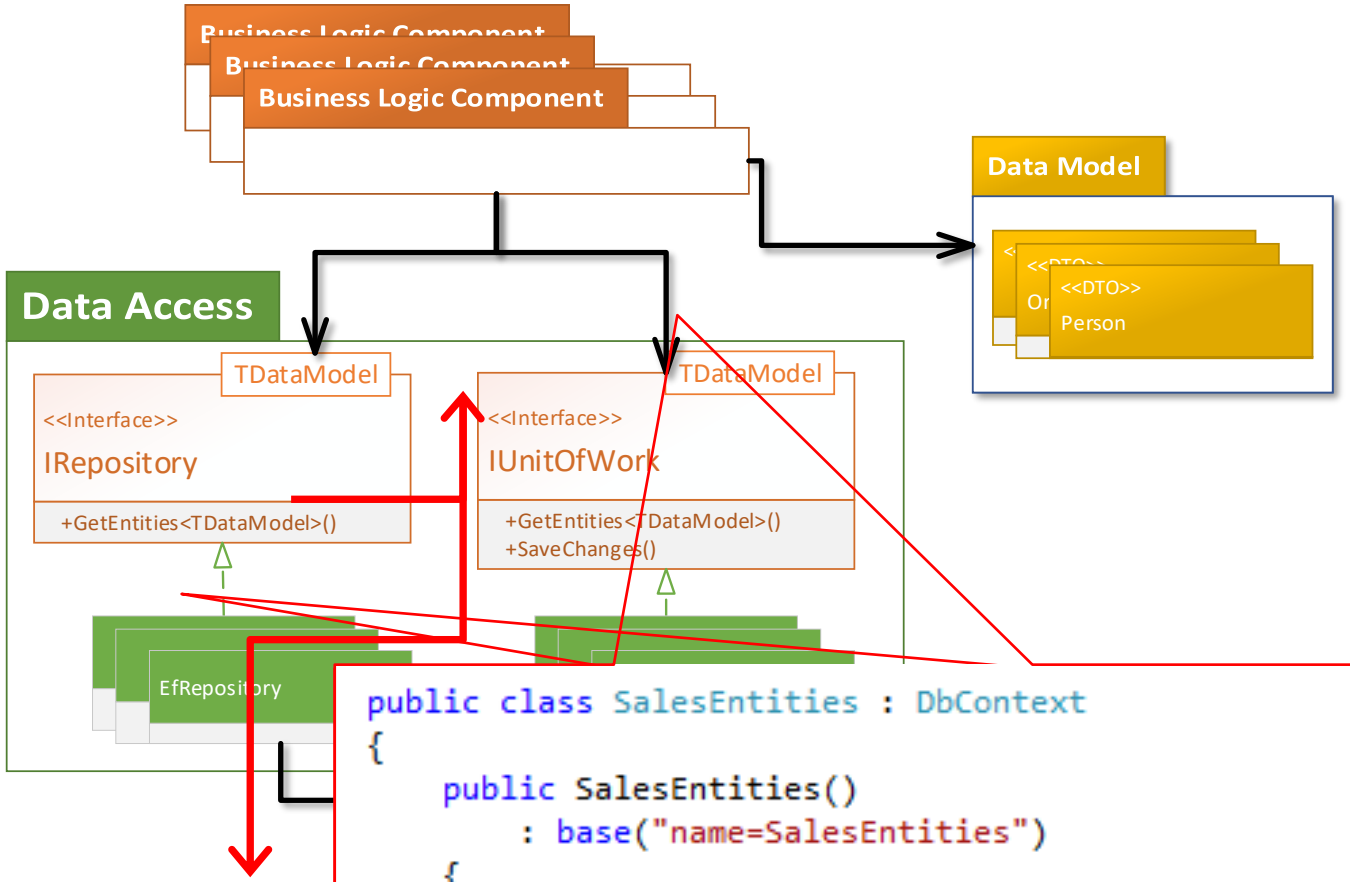
do not depend on Frameworks

CONSISTENCY

STRUCTURE

HIDE COMPLEXITY

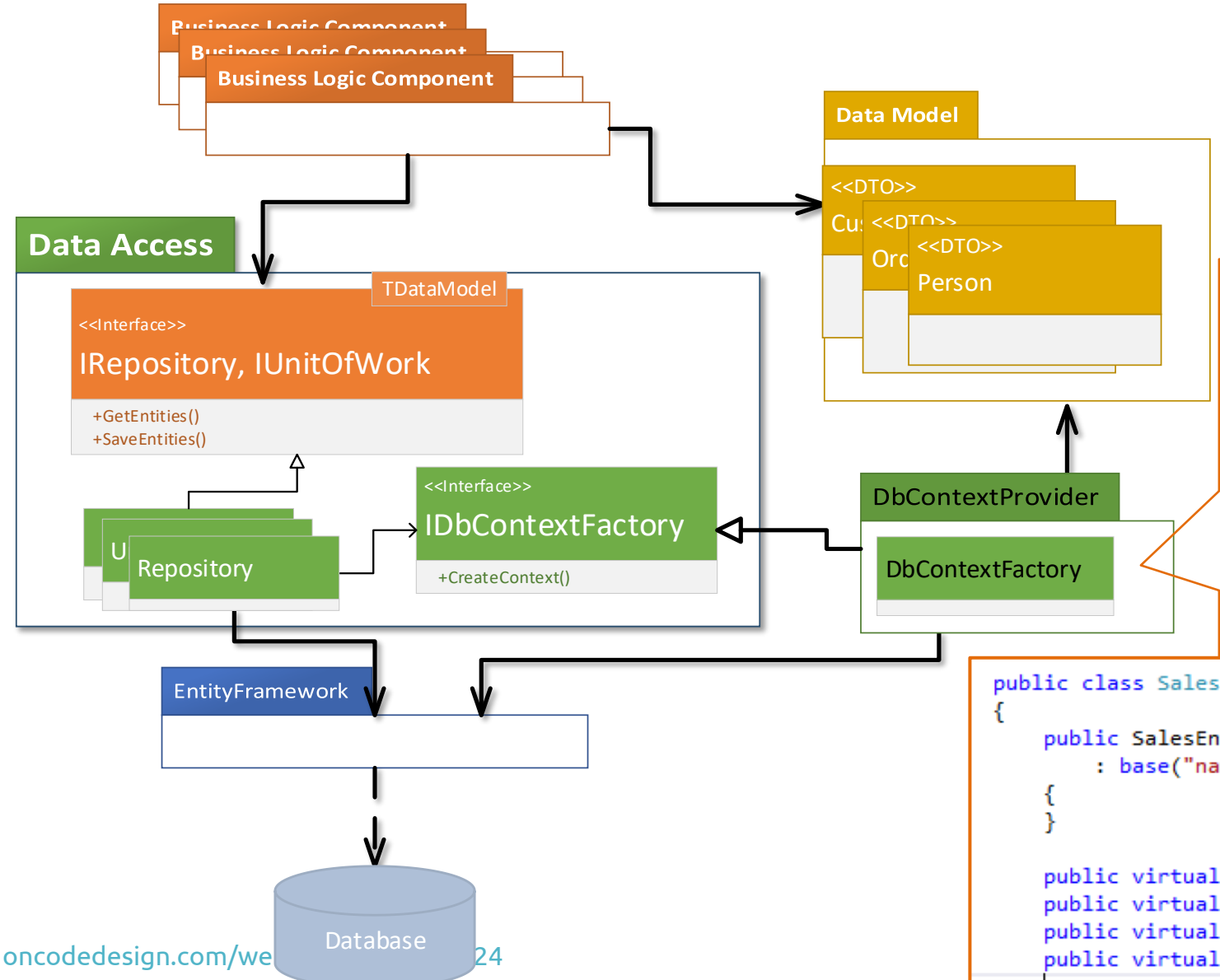
Enforce Separation of Data Access Concerns




```
public class SalesEntities : DbContext
{
    public SalesEntities()
        : base("name=SalesEntities")
    {
    }

    public virtual DbSet<Customer> Customers { get; set; }
    public virtual DbSet<SalesOrderHeader> SalesOrderHeaders { get; set; }
    public virtual DbSet<SalesOrderDetail> SalesOrderDetails { get; set; }
    public virtual DbSet<CountryRegionCurrency> CountryRegionCurrencies { get; set;
```

\iQuarc\DataAccess

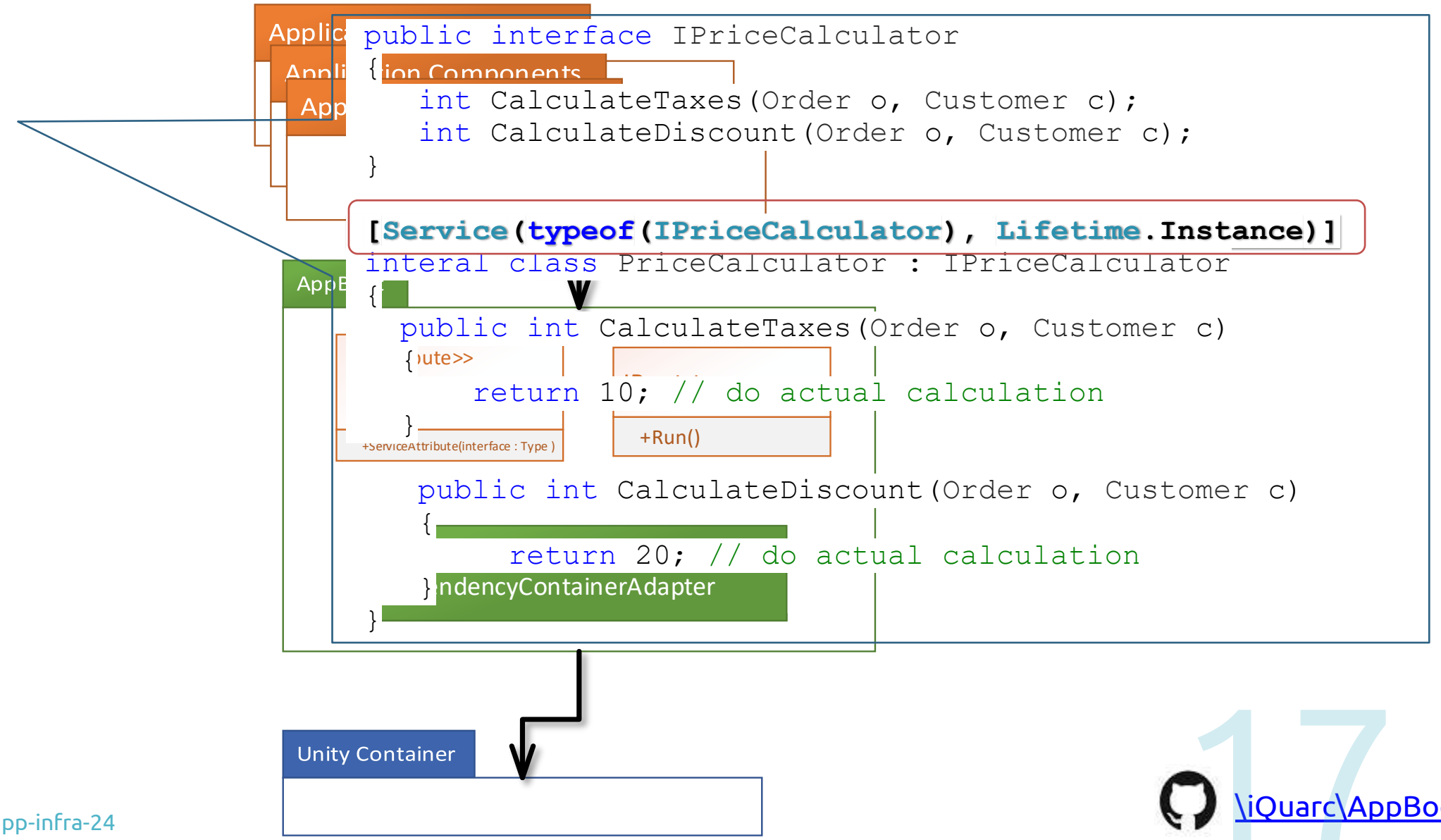
DIP to Enforce Separation of Data Access Concern



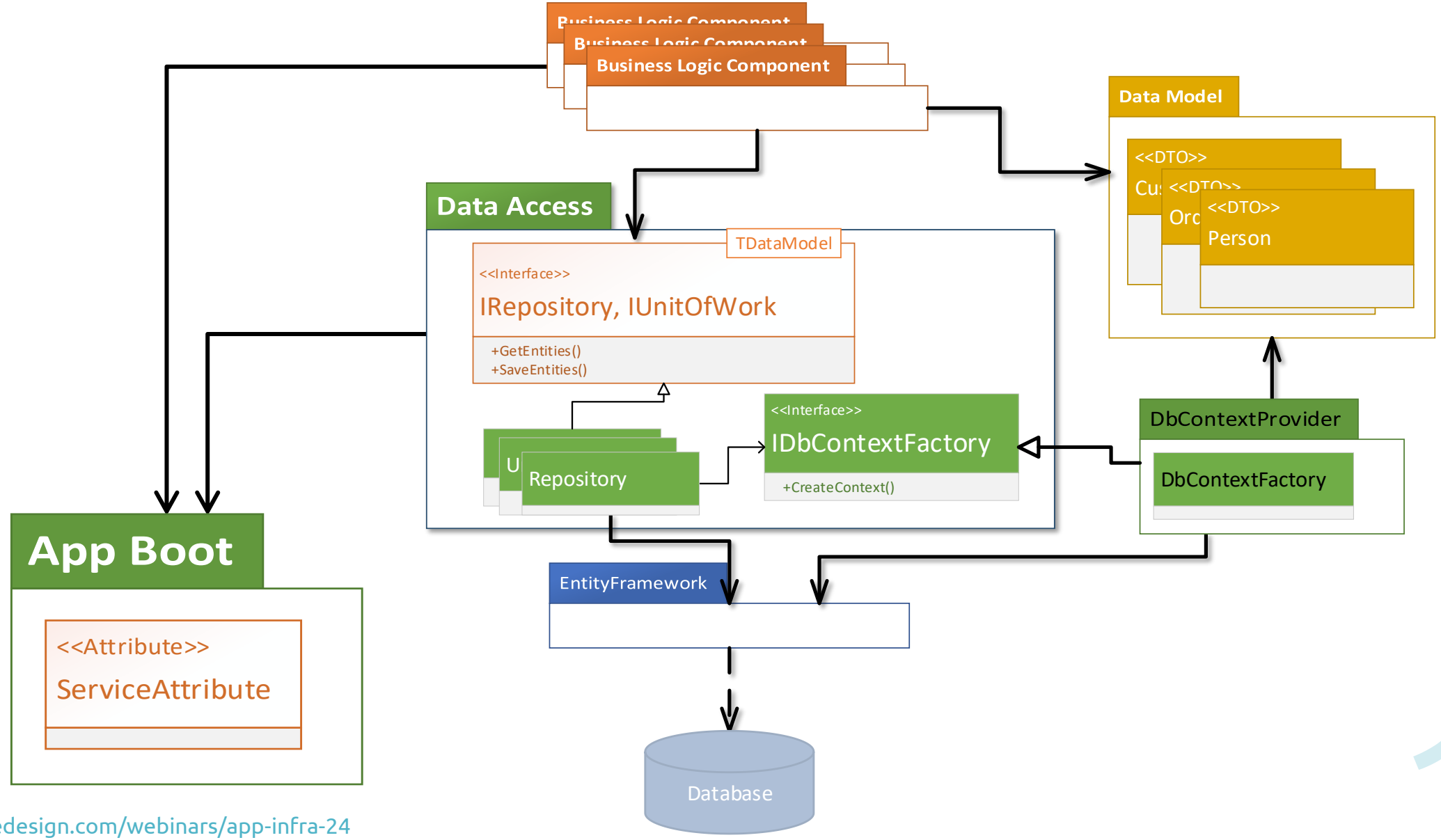

```
[Service(typeof(IDbContextFactory))]
public class DbContextFactory : IDbContextFactory
{
    public IDbContextWrapper CreateContext()
    {
        var context =  new SalesEntities();
        return new DbContextWrapper(context);
    }
}
```

```
public class SalesEntities : DbContext
{
    public SalesEntities()
        : base("name=SalesEntities")
    {
    }

    public virtual DbSet<Customer> Customers { get; set; }
    public virtual DbSet<SalesOrderHeader> SalesOrderHeaders { get; set; }
    public virtual DbSet<SalesOrderDetail> SalesOrderDetails { get; set; }
    public virtual DbSet<CountryRegionCurrency> CountryRegionCurrencies { get; set; }
```

AppBoot Hides the DI Framework under Abstractions



AppBoot: DI Abstractions & Type Discovery



Patterns on how most of the code is written

```
[Service(typeof (IOrderingService))]
private class OrderingService : IOrderingService
{
   private readonly IRepository repository;
   private readonly IPriceCalculator calculator;
   private readonly IApprovalService orderApproval;

 public OrderingService(IRepository repository, IPriceCalculator calculator, IApprovalService orderApproval)
   {
       this.repository = repository;
       this.calculator = calculator;
       this.orderApproval = orderApproval;
   }

    public SalesOrderInfo[] GetOrdersInfo(string customerName)
    {
        var orders = repository.GetEntities<SalesOrderHeader>()
        ...
        return orders.ToArray();
    }

    public SalesOrderResult PlaceOrder(string customerName, OrderRequest request)
    {
        ...
    }
}
```

Patterns for Read-Only data

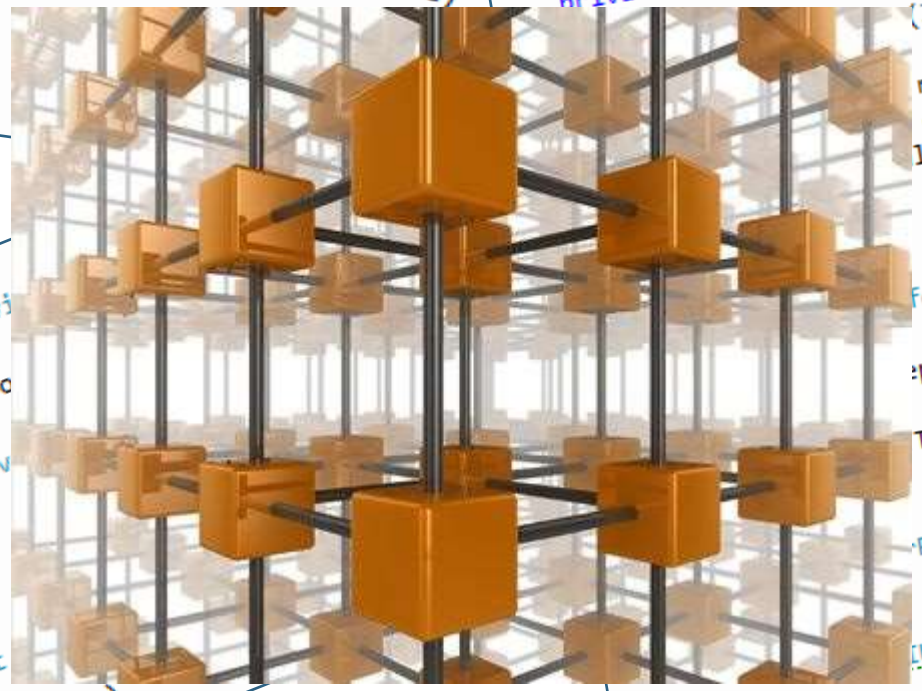
```
public class OrdersController : Controller
{
  private readonly IRepository repository;
  public OrdersController(IRepository repository)
  {
     this.repository = repository;
  }

  public IActionResult Index(string customer)
  {
     var orders = repository.GetEntities<SalesOrderHeader>()
    .Where(soh => soh.Customer.Person.LastName == customer)
    .Select(soh => new OrdersListViewModel
    {
       CustomerName = customer,
       Number = soh.SalesOrderNumber,
       SalesPersonName = soh.SalesPerson,
       DueDate = soh.DueDate,
    });

    return View(orders);
  }
...
}
```

Patterns for Read-Write data

```
public class OrdersController : Controller
{
...
  [HttpPost]
  [ValidateAntiForgeryToken]
   public IActionResult PlaceOrder(OrderRequestViewModel model)
   {
     ...
     using (IUnitOfWork uof = repository.CreateUnitOfWork())
     {
         SalesOrderHeader order = uof.GetEntities<SalesOrderHeader>()
         .FirstOrDefault(o => o.CustomerID == c.ID && o.OrderDate.Month == DateTime.Now.Month);
         if (order == null)
         {
           order = new SalesOrderHeader {Customer = c};
               uof.Add(order);
      }
      AddRequestToOrder(model, order);

         uof.SaveChanges();
       }
     ...
    }
}
```

Create Development Patterns in Code

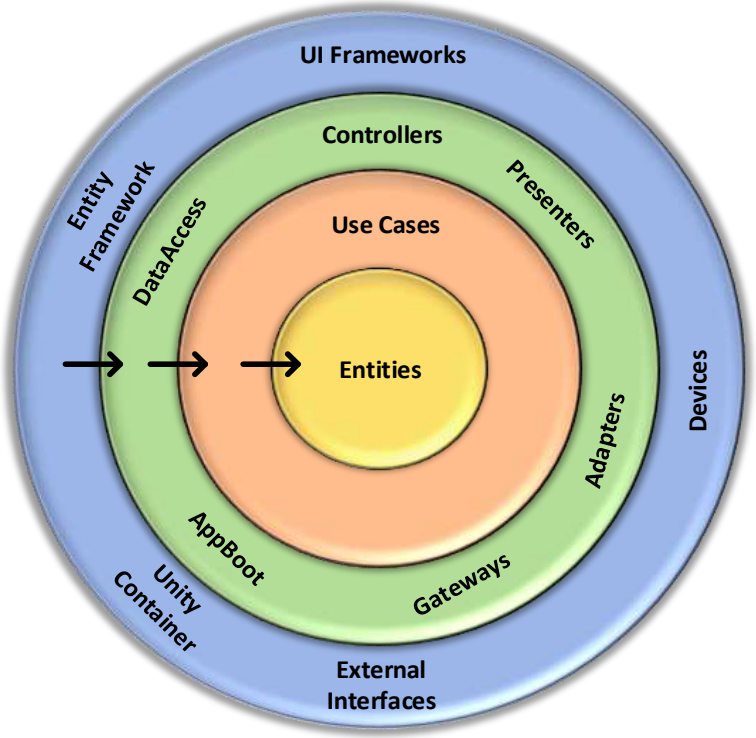
```
[Service(typeof (IPriceCalculator), Lifetime.Instance)]
public class PriceCalculator : IPriceCalculator
{
    public decimal CalculateTaxes(OrderRequest o, Customer c)
    {
        // do actual calculation
        return 10;
    }
```

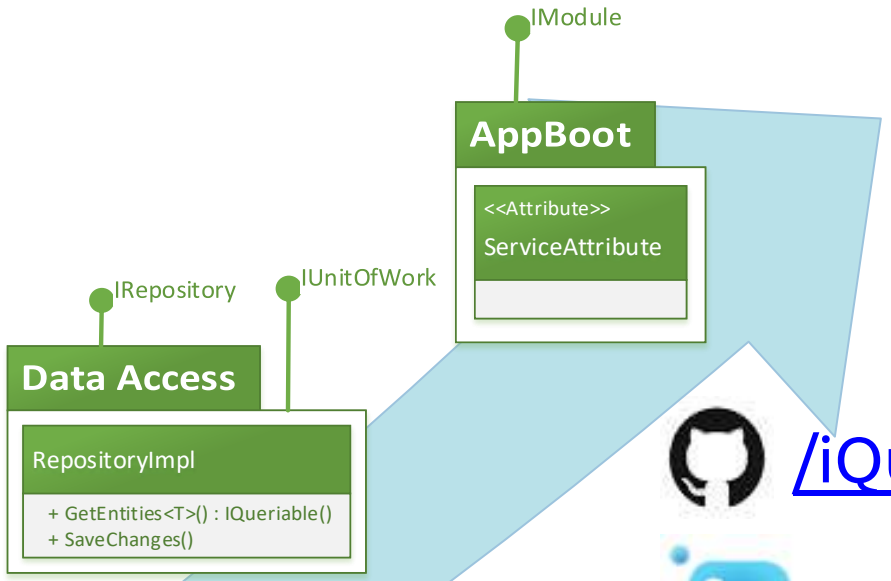


```
[Service(typeof (IOrderingService))]
class OrderingService : IOrderingService
{
    private readonly IRepository repository;
    private readonly IPriceCalculator calculator;
    private readonly IApprovalService orderApproval;

    (IRepository repository, IPriceCalculator calculator, IApprova

    repository;
    calculator;
    l = orderApproval;

    fo[] GetOrdersInfo(string customerName)

    epository.GetEntities<SalesOrderHeader>();

    ToArray();

    Result PlaceOrder(string customerName, OrderRequest request)

    IUnitOfWork uof = repository.CreateUnitOfWork())

    {
        ..
    }

var taxes = calculator.CalculateTaxes(order, customer);
var discount = calculator.CalculateDiscount(order, customer);
...
new SalesOrderResult {State = OrderResultState.Invalid};
```

```
ervice(typeof(IApprovalService))]
lass CompositeApprovalService : IApprovalServi
{   private readonly IApprovalService[] appro

    public CompositeApprovalService(IApprov
    {   this.approvals = approvals;
    }

    public bool Approve(ApproveRequest
    {   foreach (var approval in approvals)
        {
```

Implementing Clean Architecture with Application Infrastructure





/iQuarc

/iQuarc

Hide external frameworks to enforce the way they are used

Use assemblies and references among them to **enforce rules**

Enforce *Constructor Dependency Injection* that encourages *Programming Against Interfaces*

CodeDesign

Drive predictability through Software Design

Application Infrastructure for Clean Architecture

florin@onCodeDesign.com

linkedin.com/in/florincoros

oncodedesing.com/training

calendly.com/code-design/florin-short-call

Florin Coroș

Solution Architect Consultant

Technical Trainer