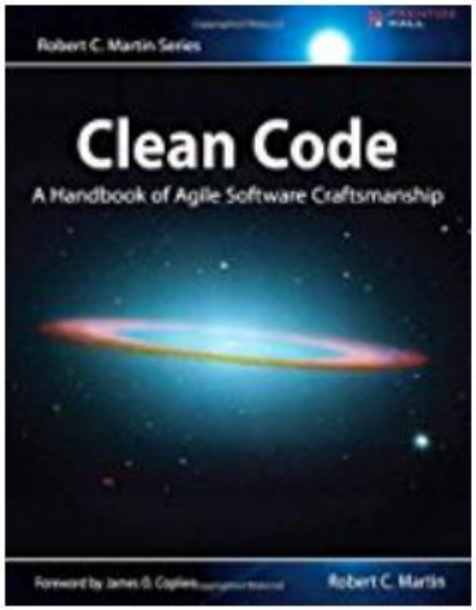
HALVOTEC

CLEAN CODE

LETS GO

LETS GO

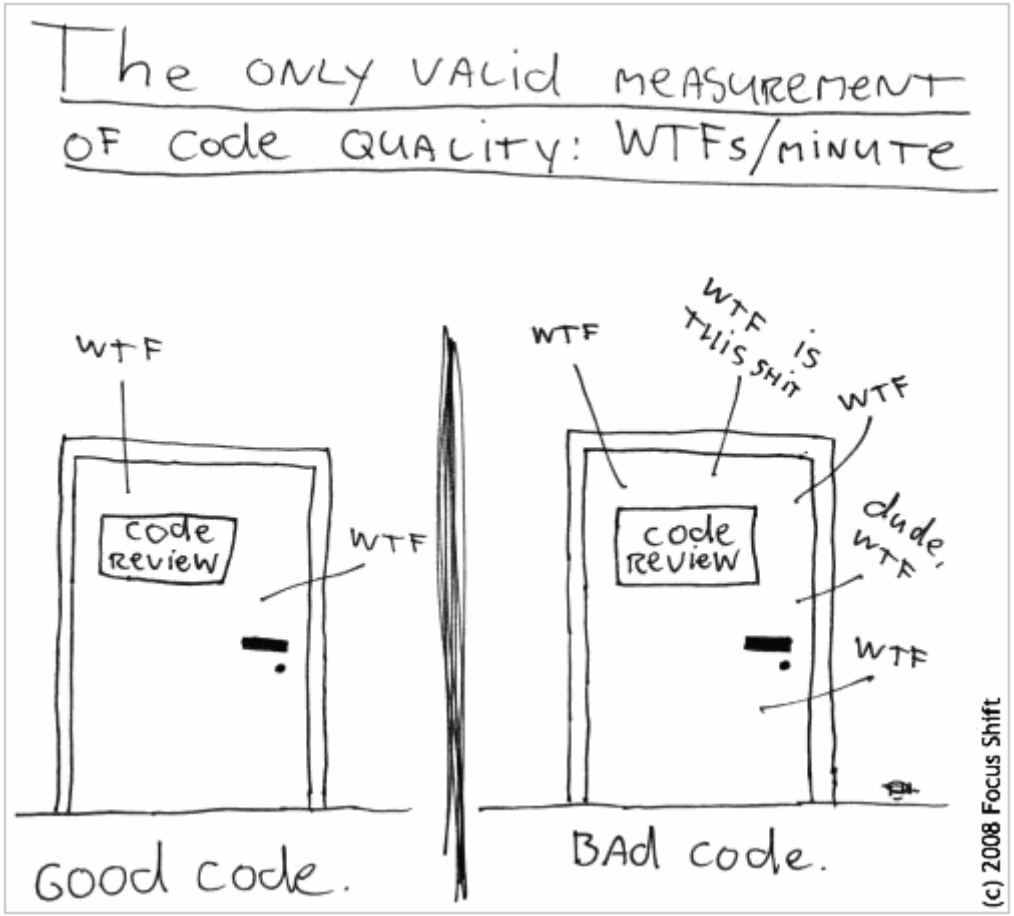

Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Robert C. Martin


Robert C. Martin Series
Clean Architecture
A Craftsman's Guide to Software Structure and Design
Robert C. Martin

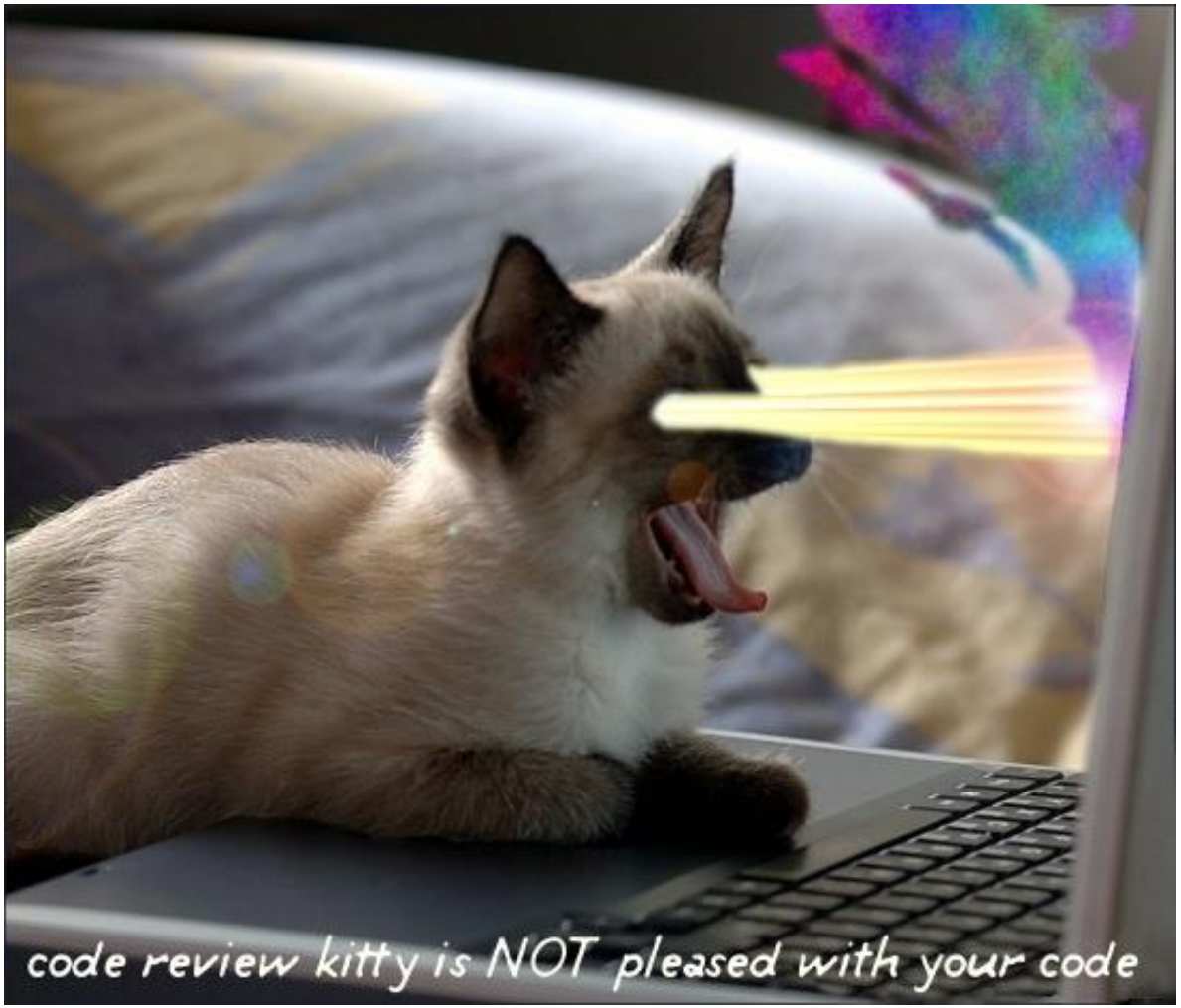

Robert C. Martin Series
The Clean Coder
A Code of Conduct for Professional Programmers
Robert C. Martin

WAS IST CLEAN CODE?

WAS IST CLEAN CODE?

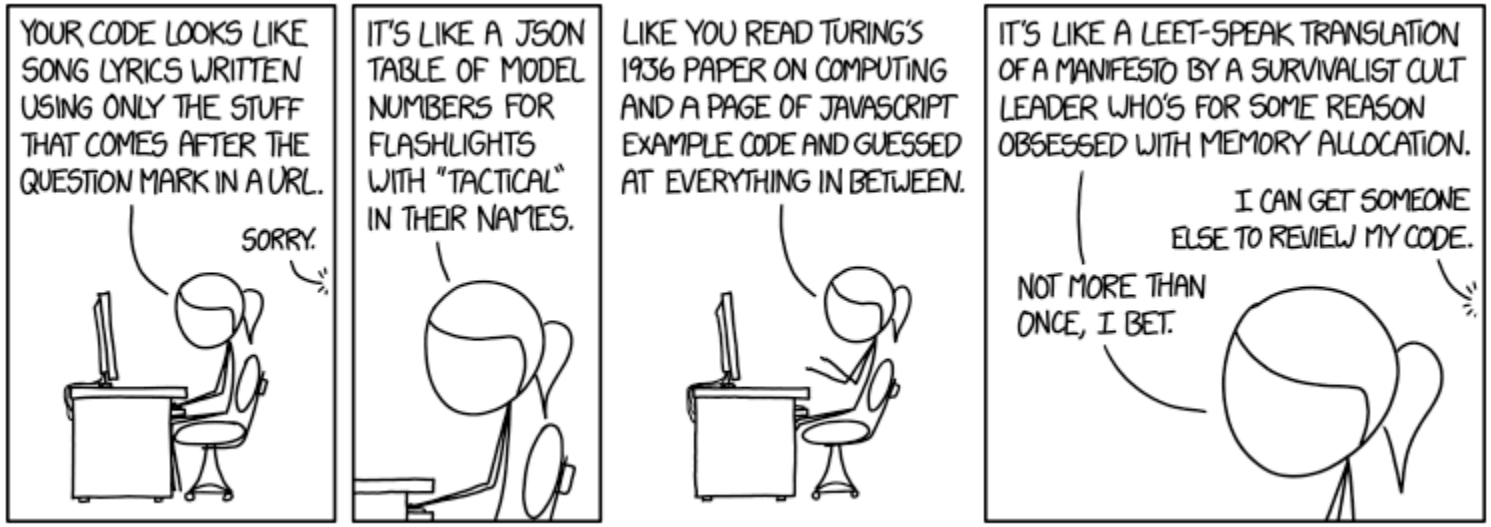


WAS IST CLEAN CODE?



WAS IST CLEAN CODE?

- bla



https://xkcd.com/1833/

Clean Code…

… does one thing well

Bjarne Stroustrup

Clean Code…

… looks like it was written by someone who cares

Michael Feathers

ANYONE...

... can write code that a computer understands, but few programmers know how to write code that a human can understand

Martin Fowler


IT'S
HAPPENING

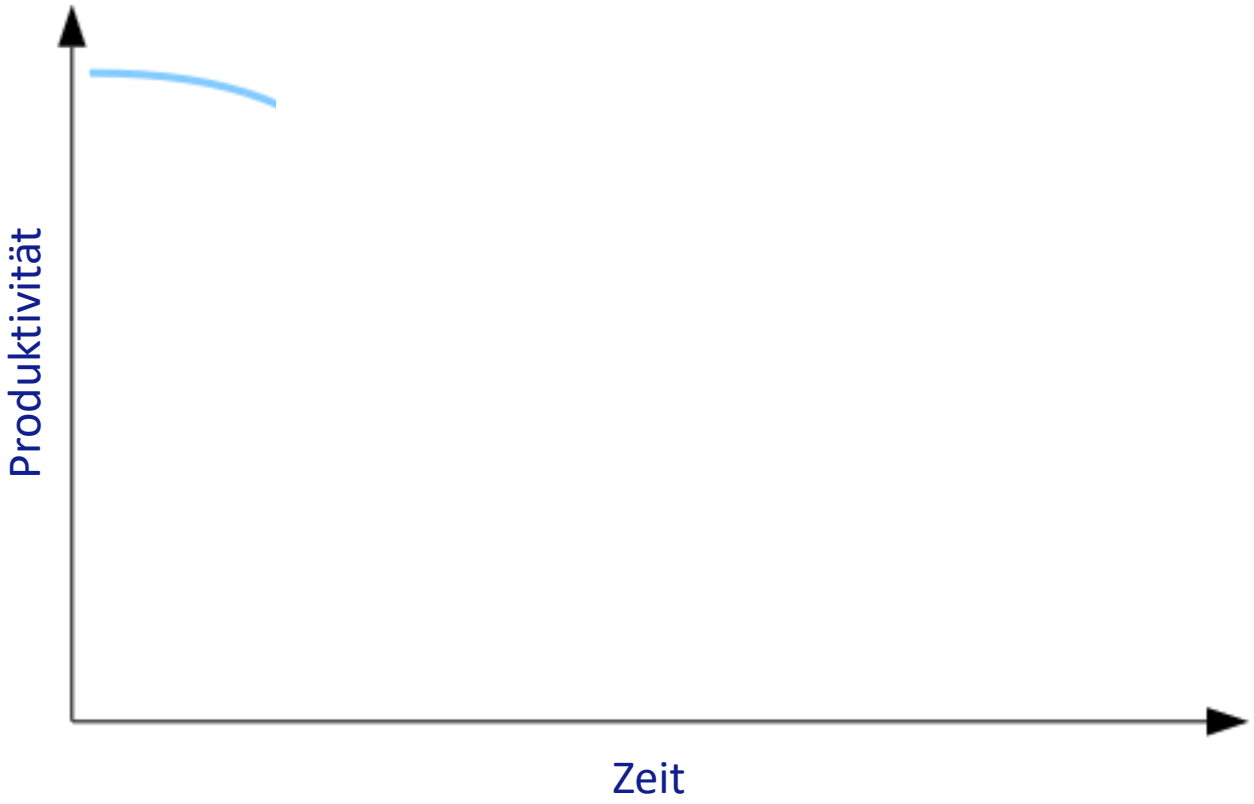
WARUM CLEAN CODE?

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

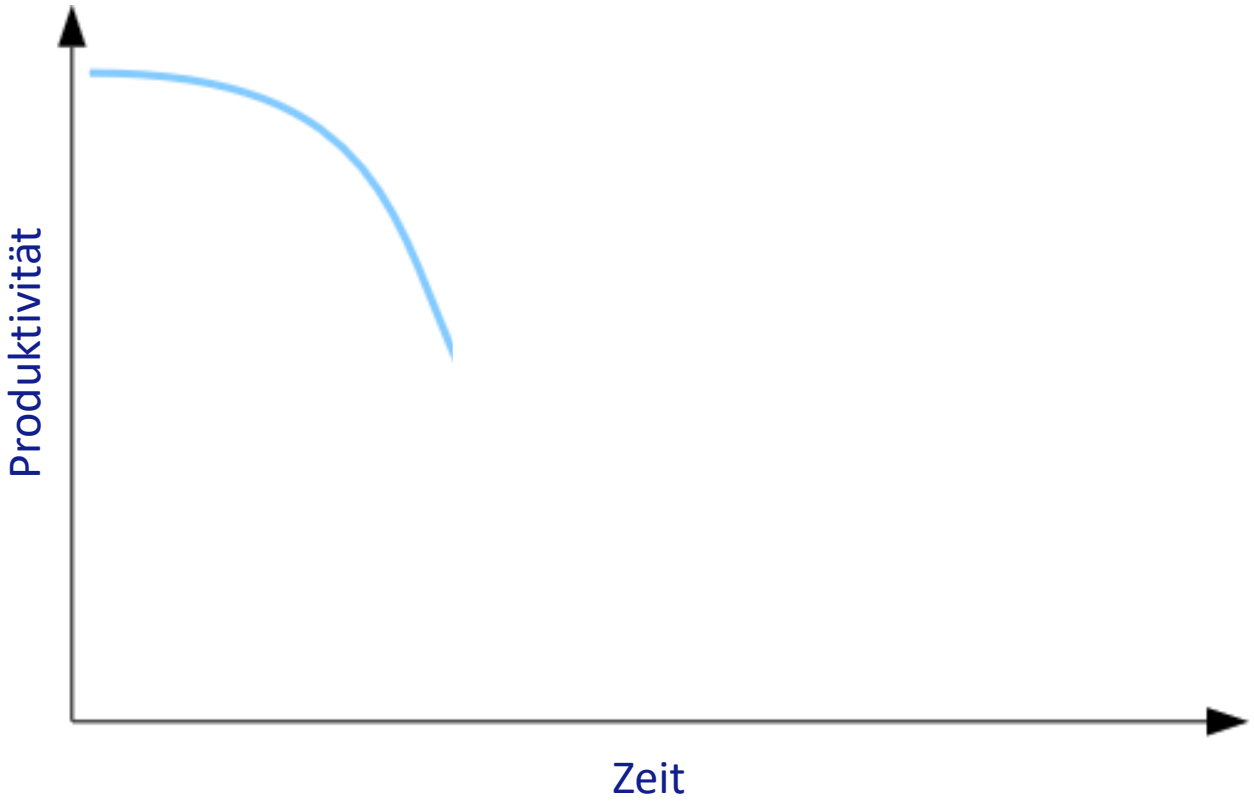
„The true cost of Software is its maintenance“

Clean Code

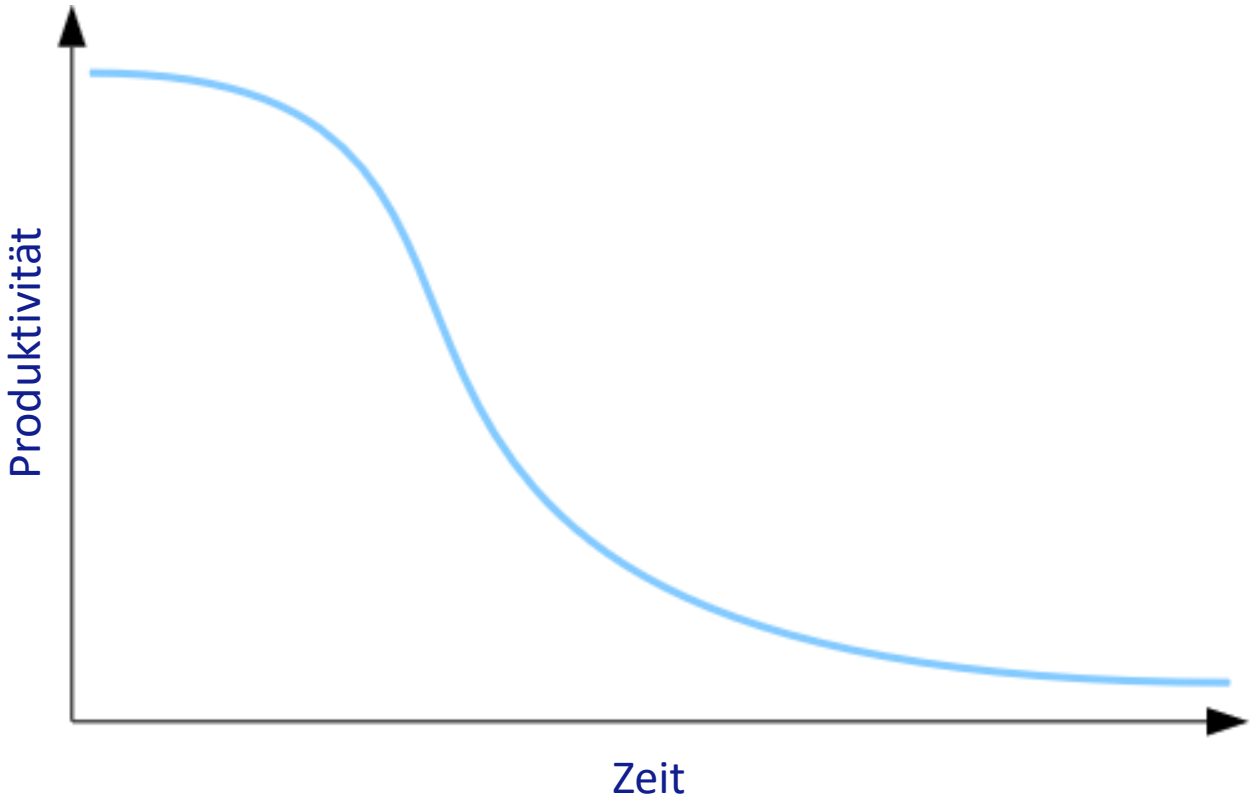
WARUM CLEAN CODE?



WARUM CLEAN CODE?



WARUM CLEAN CODE?



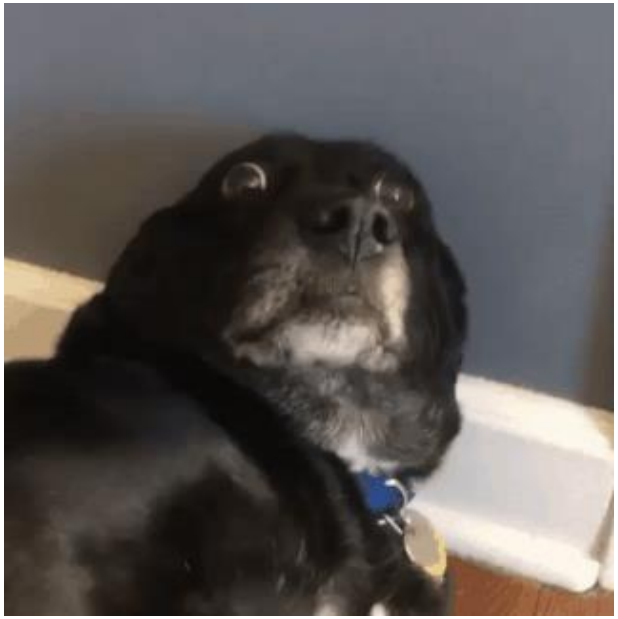
WARUM CLEAN CODE?

Total cost of owning a mess

Clean Code

TOTAL COST OF OWNING A MESS

„Der Kunde wünscht sich noch diese kleine Feature-Erweiterung…“

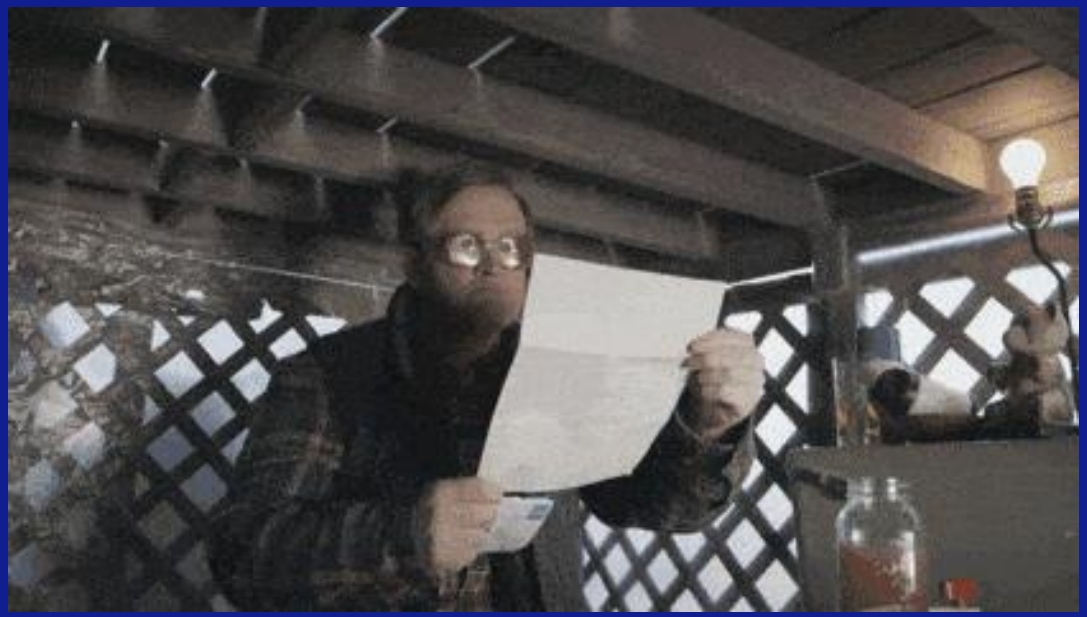


„Ja, cool… Ähm…

Wir müssen die Anwendung neubauen…

Die Architektur gibt das nicht her…

We read x times more code than we write



Clean Code

WAS IST CLEAN CODE?

- Je „schlechter“ der Code, desto mehr Zeit wird für das „Verstehen“ benötigt
- Man muss erst verstehen was der bestehende Code tut bevor man ihn erweitert / verändert (sollte man zumindest ☺)

Was kann man tun?*

Meaningful Names

„… should describe what things are, how they are measured, what they do …“

BENENNUNGEN

- Namen von Variablen, Klassen, Namespaces, Methoden, …
- Die Intention soll klar erkennbar sein
- Methodennamen sollten mit einem Verb beginnen
- Klassennamen sollten Nomen sein
- Desinformation vermeiden
- „Dont be cute“
- Ein Wort pro Konzept
- Kontext liefern

BENENNUNGEN

- Die Intention soll klar erkennbar sein

```
// Frist in Tagen
private int _d;


private int _fristInTagen;
```

BENENNUNGEN

- Die Intention soll klar erkennbar sein

```
public decimal Berechne(decimal betrag, decimal prozent)
{
        return (betrag * prozent) + betrag;
}
```

```
public decimal ErhöheBetragUmProzent(decimal betrag, decimal prozent)
{
        return (betrag * prozent) + betrag;
}
```

BENENNUNGEN

- Methodennamen sollten mit einem Verb beginnen

```
public decimal ErhöheBetragUmProzent(decimal betrag, decimal
prozent)

public User GetUserById(int id)
```

BENENNUNGEN

- Klassennamen sollten Nomen sein
- Customer, User, Antrag, Document

BENENNUNGEN

- Desinformation vermeiden

```
class MeaningfulNames
{
    public void PrintDocument(Document c)
    {
        SendToPrinter(c);
        SendEmailToAdmin(c);
    }
}
```

BENENNUNGEN

- „Dont be cute“

```
public void MachWas() { }

public void Bäämmm() { }
```

BENENNUNGEN

- Ein Wort pro Konzept, Synonyme für das Gleiche sind verwirrend

```
public void GetUser() { }

public void LoadUser() { }

public void ReadUser() { }
```

BENENNUNGEN

- Kontext liefern

```
class User
{
        public void AddToRole(Role role){}
}
```

```
public void AddRoleToUser(Role role, User user) { }
```

Methoden

METHODEN

- Sollten kurz sein
- Viele kurze Methoden sind besser als eine große
- Eine Funktion sollte genau einen Zweck erfüllen
- Möglichst switch-Statements vermeiden
- Argumente bei Überladungen immer in der gleichen Reihenfolge
- Möglichst keine bool Argumente (DoStuff(true,false,true,true,false))
- Maximal drei Argumente => besser einen komplexen Typ (DeleteOptions, …)

METHODEN

- Commands von Queries trennen

```
class User
{
        public User Update(User user) { }
}
```

DRY

„dont repeat yourself“

DON´T REPEAT YOURSELF

- Methoden und Klassen sind zum Kapseln von Logik da

Comments

„are bad* code“

COMMENTS

- „Jeder Kommentar lügt“
- Evtl. nicht mehr gepflegt
- Erweiterung des umgebenden Codes hat die Bedeutung verändert
- ABER:
  - Keine generelle Verteufelung von Kommentaren, aber 2 mal Überlegen ob man einen Kommentar macht oder den Code so refactored das klar ist was passiert.

Line Length

„zwischen 140 – 240 Zeichen“

Style Guide

„an Konventionen des vorhandenen Codes halten wenn möglich“

Boyscout Rule

„Verlasse den Platz sauberer als du ihn vorgefunden hast“

PFADFINDER REGEL (BOYSCOUT RULE)

- Kontinuierliche, kleine Verbesserungen an bestehendem Code der verändert / erweitert wird
- Benennungen verdeutlichen
- Methoden auftrennen
- Code besser strukturieren

SOLID

Single Responsibility Principle

„A class should have only one reason to change“

SINGLE RESPONSIBILITY PRINCIPLE

```
public interface IBasket
{
    IEnumerable<Item> GetItems();
    void AddItem(Item item);
    void RemoveItem(Item item);
    decimal CalculateTotal();
    void HandlePayment(Account account, decimal amount);
}
```

SINGLE RESPONSIBILITY PRINCIPLE

```
public interface IBasket
{
    IEnumerable<Item> GetItems();
    void AddItem(Item item);
    void RemoveItem(Item item);
}


public interface IPaymentProcessor
{
    decimal CalculateTotal(IBasket basket);
    void HandlePayment(Account account, decimal amount);
}
```

Open Closed Principle

„A class should be easy to extend without changes“

OPEN CLOSED PRINCIPLE

```
public void HandlePayment(Account account, decimal amount, PaymentService
paymentService)
{
     switch (paymentService)
     {
          case PaymentService.CreditCard:
               PayWithCreditCard(account, amount);
               break;
          case PaymentService.PayPal:
               PayWithPayPal(account, amount);
               break;
          // ...
     }
}
```

```
public enum PaymentService
{
     PayPal,
     CreditCard,
     NoPaymentBecauseIllOpenItAndSendItBackAnyway
}
```

OPEN CLOSED PRINCIPLE

```
public class PaymentProcessor
{
    public void HandlePayment(Account account, decimal amount,
    IPaymentService paymentService)
    {
        paymentService.Charge(account, amount);
    }
}

public class PayPalService : IPaymentService
{
    public void Charge(Account account, decimal amount)
    {
    // Magic...
    }
}
```

Lizkov Substitution Principle

„A subclass musn´t change expected behavior“

LIZKOV SUBSTITUTION PRINCIPLE

```
public class Greeter
{
    public virtual string GetMessage(string name)
    {
        return $"Hello {name}";
    }
}
```

LIZKOV SUBSTITUTION PRINCIPLE

```
public class Greeter
{
    public virtual string GetMessage(string name)
    {
        return $"Hello {name}";
    }
}



public class SurpriseGreeter : Greeter
{
    public override string GetMessage(string name)
    {
        return string.Equals(name, "flo")
        ? throw new ArgumentException(nameof(name))
        : "Hello NOT FLO";
    }
}
```

Interface Segregation Principle

„A interface shouldn´t force classes to implement useless methods“

INTERFACE SEGREGATION PRINCIPLE

```
public interface ICache
{
    void Connect(string credentials);
    void Add(string key, object item);
    object Get(string key);
}
```

INTERFACE SEGREGATION PRINCIPLE

```
public class FileSystemCache : ICache
{
    public void Connect(string credentials)
    {
        // NOOP
    }
}
```

INTERFACE SEGREGATION PRINCIPLE

```
public interface ICache
{
    void Add(string key, object item);
    object Get(string key);
}
```

```
public interface IRequireConnection
{
   void Connect(string credentials);
}
```

Dependency Inversion Principle

„High-Level and Low-Level code should depend on abstractions“

DEPENDENCY INVERSION PRINCIPLE

```
public class Logger
{
    public void Log(string message)
    {
        File.AppendAllText("my.log", message);
    }
}
```

DEPENDENCY INVERSION PRINCIPLE

```
public class Logger
{
    private readonly IWriter _writer;

    public Logger(IWriter writer)
    {
        _writer = writer;
    }

    public void Log(string message)
    {
        _writer.append(message);
    }
}
```

TDD

Test Driven Development

TEST DRIVEN DEVELOPMENT (TDD)

- Schreibe einen Test
- Der Test schlägt fehl
- Schreibe nur soviel Produktiv-Code bis der Test durchläuft
- Beginne von vorne

http://butunclebob.com/files/downloads/Bowling%20Game%20Kata.ppt

DDD

Domain Driven Design

DOMAIN DRIVEN DESIGN

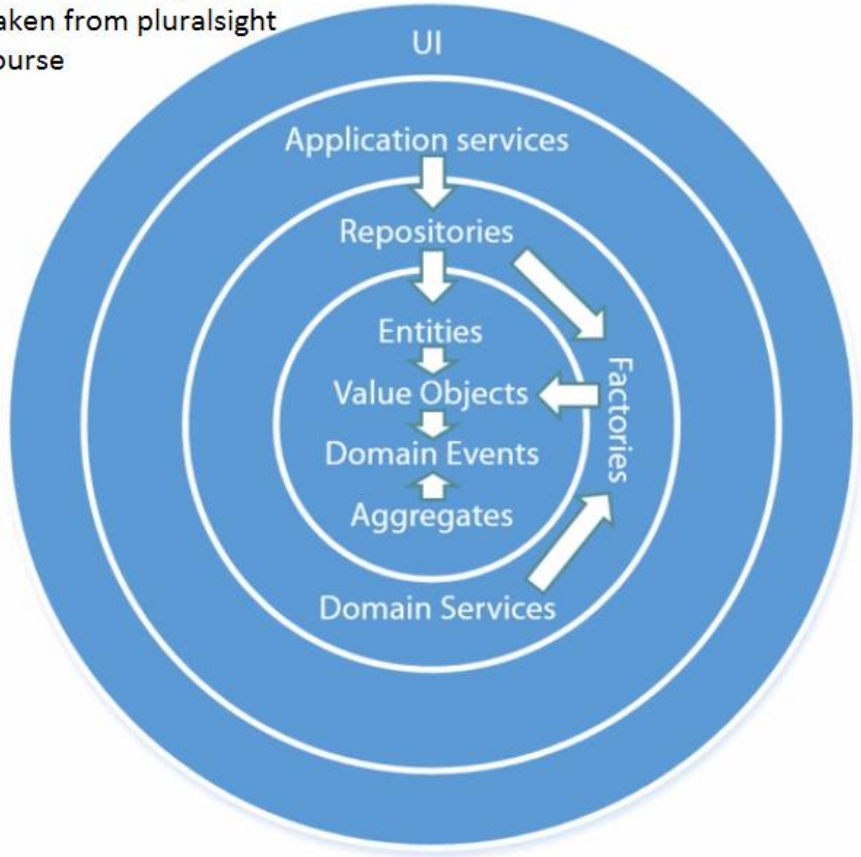
Domain-driven Design ist nicht nur eine Technik oder Methode. Es ist viel mehr eine Denkweise und Priorisierung zur Steigerung der Produktivität von Softwareprojekten im Umfeld komplexer fachlicher Zusammenhänge.[2] Domain-driven Design basiert auf folgenden zwei Annahmen:

Der Schwerpunkt des Softwaredesigns liegt auf der Fachlichkeit und der Fachlogik.
Der Entwurf komplexer fachlicher Zusammenhänge sollte auf einem Modell der Anwendungsdomäne, dem Domänenmodell basieren.

https://de.wikipedia.org/wiki/Domain-driven_Design

DOMAIN DRIVEN DESIGN

DDD. Classic approach.
Taken from pluralsight course



BDD

Behavior Driven Development

BEHAVIOR DRIVEN DEVELOPMENT

- Umsetzung anhand von Szenarien
- Gegeben ist…,
- Und …
- Wenn …
- Dann …

Szenario 1: Rückgegebene Ware kommt wieder ins Lager

- **Gegeben** ist, dass ein Kunde eine schwarze Hose gekauft hat
- **Und** wir 3 schwarze Hosen im Lager haben,
- **Wenn** er die Hose zurückgibt und dafür einen Gutschein erhält,
- **Dann** sollten wir 4 schwarze Hosen im Lager haben.

https://de.wikipedia.org/wiki/Behavior_Driven_Development

Für Fragen stehe ich Euch gerne zur Verfügung!