Clean Code

Agenda

- Meaningful names
- Read your code
- Small steps
- Responsibilities
- Magic numbers
- Exceptions
- Communication between projects

Why?

Why?

- Smart developer
    - Difficult code
    - Has great developing skills
    - r = lowercase url
- Professional developer
    - Readable code
    - Maintanable code
    - Clarity!
    - lowercaseUrlOfPage = lowercase url

Why?

- Bad code?
  - Need to be done fast
  - Tired of project / work
  - Works now, I will clean up later
  - All of us does this

Why?

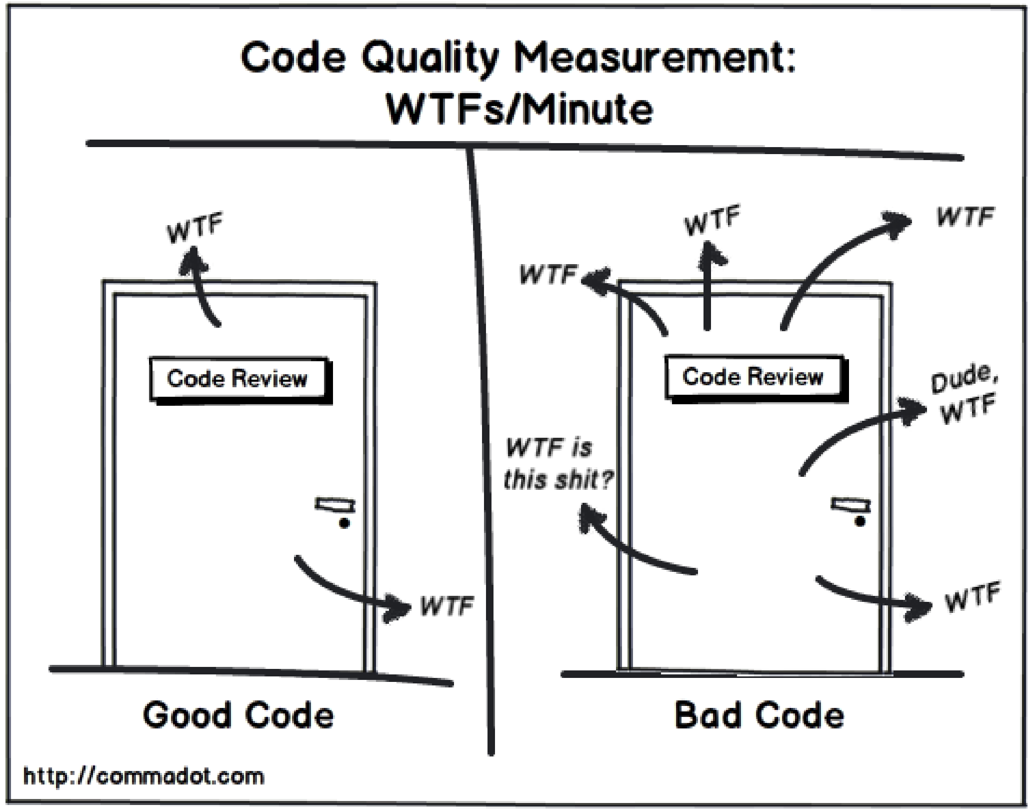
- Bad code?
    - Requirements changed
    - Deadline to close
    - Managers
    - Sales
    - WE - DEVELOPERS

Why?

- DEVELOPERS, DEVELOPERS, DEVELOPERS

Explanation

Explanation



Meaningful names

Meaningful names

```
int x
int y
int temp
return x
```

```
int elapsedTimeInMinutes
int maximumPossibleValue
int daysSinceLastLogin
return result
```

Meaningful names

```
public List<int[]> getThem()
{
   List<int[]> list1 = new ArrayList<int[]>();

   for (int[] x : theList)
      if (x[0] == 4)
        list1.add(x);
   return list1;
}
```

```
public List<Cell> getFlaggedCells()
{
   List<Cell> flaggedCells = new ArrayList<Cell>();

   for (Cell cell : gameBoard)
      if (cell.isFlagged())
         flaggedCells.add(cell);
   return flaggedCells;
}
```

Meaningful names

```
public static void copyChars(char a1[], char a2[])
{
   for (int i = 0; i < a1.length; i++)
   {
      a2[i] = a1[i];
   }
}
```

```
public static void copyChars(char source[], char destination[])
{
   for (int i = 0; i < source.length; i++)
   {
      destination[i] = source[i];
   }
}
```

Meaningful names

```
XYZControllerForEfficientHandlingOfStrings
XYZControllerForEfficientStorageOfStrings
```

```
Array[Item] GetItemsArray()
List<Item> GetItemsList()
```

```
IEnumerable<Item> GetItems()
```

Not too much

```
string strProductTitle = “Some product title”;
double dbProductPrice = 12.34;
```

```
string productTitle = “some product title”;
double productPrice = 12.34;
```

Try to pronouce this

```
class DtaRcrd102
{
	private Date genymdhms;
	private Date modymdhms;
	private final String pszqint = "102";
	/* ... */
};
```

```
class Customer
{
	private Date generationTimestamp;
	private Date modificationTimestamp;;
	private final String recordId = "102";
	/* ... */
};
```

One word per concept

- Get / receive / fetch
- Controller / manager / driver

Read your code

Read your code

- Max 150 chars per line
- Max 20 lines per method
- Blocks ({ })

Read your code

- Method parameters
  - Search(int minValue, int maxValue, int pageNumber, int pageSize)
  - Search(Conditions conditions)
  - Conditions

  {int minValue, int maxValue, int pageNumber, int pageSize }

Read your code

```
if(account.Id > 0 && account.Name != null)
{
        // do something
}
```

```
if(AccountIsValid(account))
{
        // do something
}

//where account is bussinnes class
if(account.IsValid)
{
        // do something
}
```

Read your code

```
BadWay()
{
   CodeToCallDB
   CodeToConvertItemsToModel
   CodeToConvertItemsToModel
   CodeToConvertItemsToModel
   CodeToCalculate
   CodeToCalculate
   CodeToCalculate
   CodeToCalculate
   CodeToSaveCalculations
   CodeToSaveCalculations
   CodeToSaveCalculations
   return
}
```

```
BetterWay()
{
   var items = GetItems();
   var results = ProcessItems(items);
   SaveResults(results);
   return results;
}

GetItems() { }
ProcessItems(itemsToProcess){ }
SaveResults(resultsToSave){ }
```

Don’t repeat your self

```
GetItems(){
   Account account = accountsService.GetAccount();
   if(account.IsValid)
   {
      return itemsService.GetItems(account);
   }
}

GetItem(int itemId){
   Account account = accountsService.GetAccount();
   if(account.IsValid)
   {
      return itemsService.GetItem(itemId);
   }
}
```

```
GetItems(){
   if(ValidateAccount())
   {
      return itemsService.GetItems(account);
   }
}

GetItem(int itemId){
   if(ValidateAccount())
   {
      return itemsService.GetItem(itemId);
   }
}

ValidateAccount(){
   Account account = accountsService.GetAccount();
   return account.IsValid;
}
```

Small steps

Small steps

- Scout rule
- Always leave it better than you found it
- Check your code after some time

Conditions – leave when is not ok

```
if(checkCondition1)
{
   some logic here
   if(checkCondition2)
   {
      long logic here
      long logic here
      long logic here
   }
   else
   {
      return / throw
   }
}
else
{
   return / throw
}
```

```
if(!checkCondition1)
{
   return / throw
}

some logic here

if(!checkCondition2)
{
   return / throw
}

long logic here
long logic here
long logic here
```

Responsibilities

SOLID

- S – Single responsibility principle
- O – Open / closed principle
- L – Liskov substitution principle
- I – Interface segregation principle
- D – Dependency inversion principle

Single responsibility principle

```
class Person
{
   string Name;
   string Surname;
   int Age;
   ...

   int CalculateDistance(Place);
   void SaveInDB()
   ...

   and so on ...
   ... for next couple k lines
}
```

```
class Person
{
   string Name;
   string Surname;
   int Age;
}
class World
{
   CalculateDistance(Person, Place)
}
class PersonRepository
{
   Save(Person);
}
```

Magic Numbers

MNDD – Magic Numbers Driven Development

Magic number

```
if(report.Type == 2)
{

}

// guess what is 2 
```

```
if(report.Type == ReportType.Cool)
{
}

enum ReportType
{
   Normal = 1,
   Cool = 2,
   Basic = 3
}
```

Magic string

```
if(report.CategoryName == „special”)
{
}
```

```
if(report.CategoryName == StringResources.SpecialCategoryName)
{
}
```

Exceptions

Exception hierarchy

```
try
{
   var result = DoSomething();
}
catch(Exception ex)
{
   if(ex.Message.Equals(„range”))
   {
   }
   else if(ex.Message.StartsWith(„_”))
   {
   }
}
```

```
OutOfRangeException
InvalidOperationException
InvalidUserException
InsufficientPrivilegesException

try
{
   var result = DoSomething();
}
catch(OutOfRangeException){}
catch(InvalidUserException){}
catch(InsufficientPrivilegesException){}
catch(Exception)
{
   // something went really bad
   // as I was not expecting that 
}
```

„Empty” object

```
Var account = GetAccount(id);

if(account == null || account.Id <= 0)
{
   throw new InvalidAccountException();
}

rest of logic
```

```
GetAccountReturns Empty instead of null

if(account == Account.Empty)
{
   throw new InvalidAccountException();
}

rest of logic

class Account
{
   public int Id { get; set; }
   public Account Empty
   {
      return new Account { Id = 0};
   }
}
```

Communication between projects

External / internal

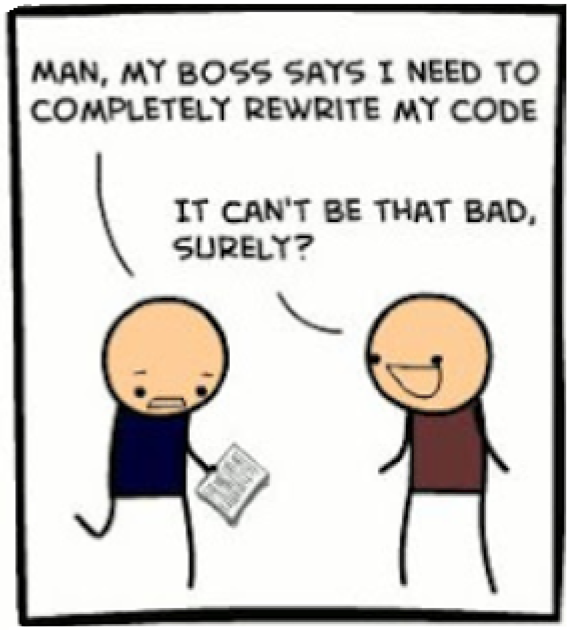
- IPersonRepository
  - Person Get(id)
- Model
  - Person
- DBPersonRepository : IPersonRepository
  - DBPerson
  - DBPerson To Person Mapper

- Easy to change repository ex. from EF to Azure Tables

- WebAPI
  - Always use DTO (ex Model.Person)

- PersonController
  - PersonDto Get(id)

- Contract is resistant to internal changes of model


MAN, MY BOSS SAYS I NEED TO COMPLETELY REWRITE MY CODE
IT CAN'T BE THAT BAD, SURELY?