Clean Code

Why Clean Code?

Bad Code

It can destroy your company!

- Bugs aren‘t repaired from one release to the next.
- Load times grew.
- Crashes increased

-> User acceptance decreases.

Total Cost of owning a mess

At first the developer team is moving very fast and after two years they find themselves at a snail‘s pace.

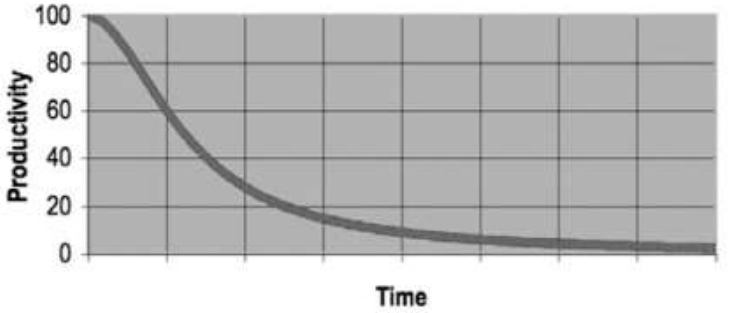

Figure 1-1
Productivity vs. time

Every change:

- Breaks two or three other parts of the code.
- Are absolutely not trivial.

Every addition or modification:

- The tangles, twists and knots must be understood

-> so we can more tangles, twists and knots. ☺

Redesign of the software

The code base is so awful, that the team rebels. Actually the management bends and authorize a redesign.

One team (best and brightest) does the the implementation on the greenfile, the remaining one does the modification of the legacy system.

-> The teams now are at a race.

The new software must do everything what the old one does. Furthermore it must keet up with the contiuous changes.

Attitude

It‘s our fault as developers that good code rots into bad code.

Not theirs:

- Managers
- Customers
- Changed requirements
- To tight schedules

We are complicit in the planning of the project and share a great responsibility for any failures, especially if those failures have to do with bad code!

What is clean code?

Bjanre Stroustrup (inventor of c++):

„I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well.“

We are authors

The ratio of time spent reading vs. writing is well over 10:1.

-> The code you are trying to write today will be hard or easy to write depending on how hard or easy the surrounding code is easy to read.

The boy scout rule

It‘s not enough to write code well. The code has to be kept clean over time.

Rule:

Leave the campground cleaner than you found it.

Don‘t try to save the world at a single blow.

Change one variable name for the better, break up function that‘s little to large, eliminate one small bit of duplication and so on…

Meaningful names

Use intention-revealing names

Name of a variable, function or class should reveal:

- Why it exists
- What it does
- How it is used

If a name required a comment, then the name does not reveal its intent.

Exercise time

int d; // elapsed time in days

Why it‘s not a good name?

Find examples for better ones.

Exercise time

```
public List<int[]> getThem() {
    List<int[]> list1 = new ArrayList<int[]>();
    for (int[] x : theList)
      if (x[0] == 4)
        list1.add(x);
    return list1;
  }
```

What are the weak spots of this code? And why it is so hard to understand it?

Avoid disinformation

- Avoid leaving false clues that obscure the meaning of the code.

For example:

- Hp, aix and sco are poor variable names because they are name of unix platforms or variants.
- Do not refer to a grouping of accounts as an *accountList* unless it's actually a List. If the container is not a list, it will lead to false conclusions. Name it *accounts* instead.

Avoid disinformation

For example:

- Beware of using names which vary in small ways. How long does it take to spot the subtle difference between *XYZControllerForEfficientHandlingOfStrings* and *XYZControllerForEfficientStorageOfStrings*

Make meaningful distinctions

Don‘t add number series or noise words to names, so that the compiler is satisfied. If names must be different, then the should also mean something different.

Exercise time

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

Examples

What is the difference between?

class Product

Class ProductInfo

Class ProductData

Don’t use noise words for meaningless distinction.

Use pronounceable names

If you can’t pronounce it, you can’t discuss it without sounding like an idiot.

For example:

Genymdhms (gerneration date, year, month, day, hour, minute, and second)

-> New developers had to have the variables explained to them.

Use pronounceable names

If you can’t pronounce it, you can’t discuss it without sounding like an idiot.

For example:

Genymdhms (gerneration date, year, month, day, hour, minute, and second)

-> New developers had to have the variables explained to them.

Exercise time

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

This should represent a customer record. Rename it.

Use searchable names

The length of a name should correspond to the size of its scope.

Use single letter names ONLY as local variables inside short methods!

WORK_DAYS_PER_WEEK // good name for searching

For (int i = 0; i < WORK_DAYS_PER_WEEK; i++) // sufficient since small scope

Example

Compare

```
for (int j=0; j<34; j++) {
s += (t[j]*4)/5;
}
```

To

```
int realDaysPerIdealDay = 4;
const int WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for (int j=0; j < NUMBER_OF_TASKS; j++) {
int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;
int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);
sum += realTaskWeeks;
}
```

Hungarian notation

Don‘t add the type of a variable to the name. This was necessary 30 years ago.

PhoneNumber phoneString;

// Name not changed when type changed!

Class names

Classes and object should have noun or noun prhase names like Customer, WikiPage, Account and AddressParser.

Avoid words like Manager, Processor, Data or Info in the name of a class.

A class name should not be a verb.

Method names

Methods should have verb or verb phrase names like postPayment, deletePage or save.

Accessors, mutators and predicates should be named for their value and prefixed with get, set and is

Examples

```
Public class Customer

Private readonly string _name;
Private readonly bool _isActivated;

Public string GetName() => return _name;
Public void SetName(string name) { _name = name; }
Public bool IsActivated => return _ isActivated;
```

Don‘t pun

Avoid using the same word for two purposes.

```
Public class CustomerReadService()
        // Reads customer object from database

Public class BankAccountReadController
        // Also reads bank account object from database
```

-> Very bad! You should always follow the „one word per concept“ rule.