DAIMLER

Daimler Truck

Clean Code

Q1 2021 Wiessler | TP/XEE

Clean Code

0 **Reusable code**
Why should we ensure clean code

1 **Maintainable code**
Why do we want & what is maintainable code?

2 **Maintainable code rules**
Which are the rules to write maintainable code?

3 **Commitment**
On which rules should we commit?

Why Clean Code

comprehensibility

First and foremost, code is written by people, to be read by people. Secondly, we program for the compiler (or interpreter)

reader != author

- Authors leave teams
- Authors have other projects
- Authors have other tasks, etc ...

ideal

Code should be simple and understandable.

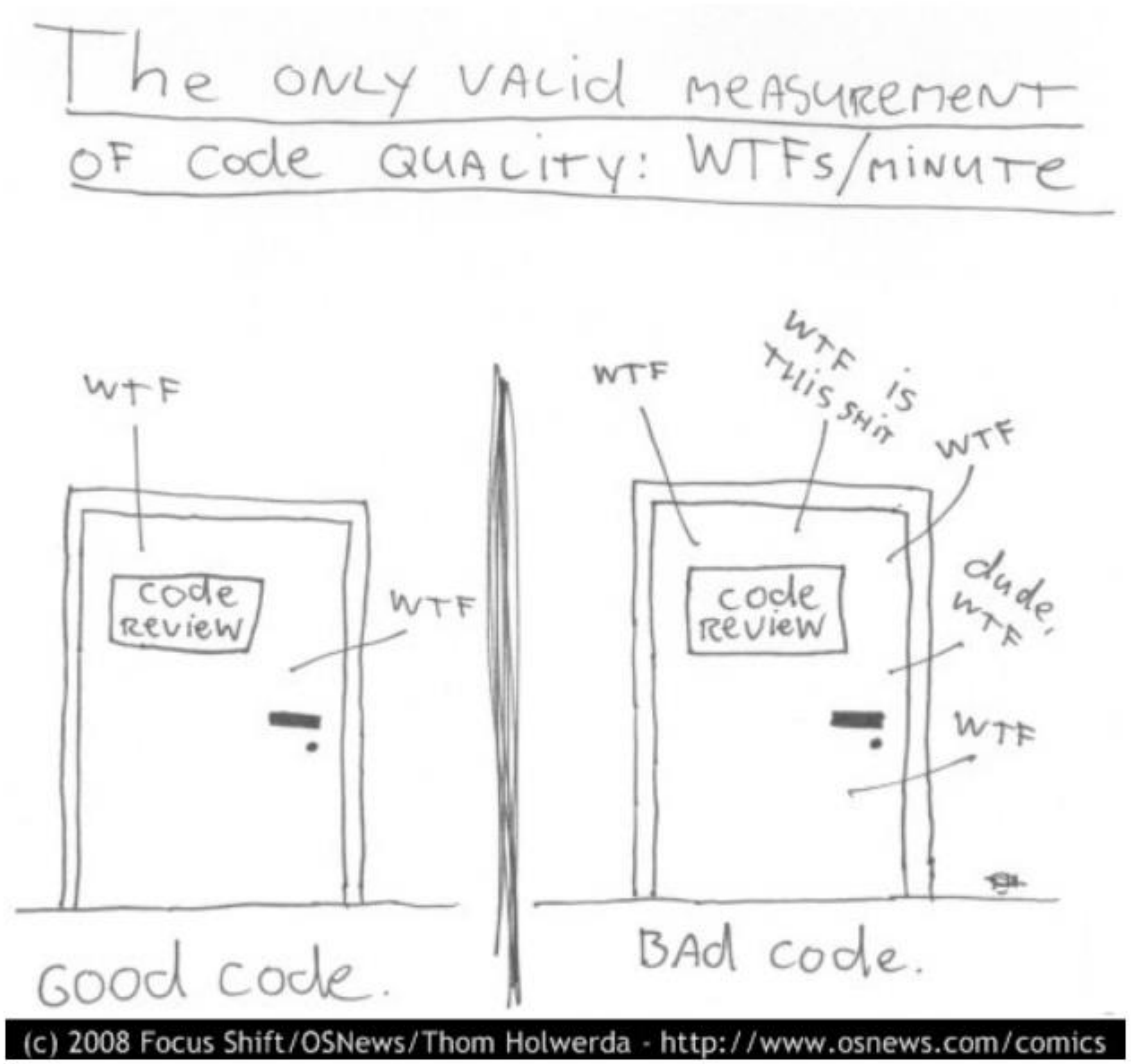
V/S

real work

- normally no code is created like this.
- Things are created in MVPs.
- Requirements and designs change
- Time and cost pressures influence the code quality ("We'll fix it later")
- Different skills and knowledge
- Languages, technologies and idioms, patterns, etc. evolve.

Therefore: Code must be continuously maintained and improved. Clean Code !

Code Quality


The only valid measurement of code quality: WTFs/minute
WTF
code review
WTF
Good code.
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
Bad code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

Example

```
for ind=1:length(temp)
    flag=find(temp2==temp(ind),1);
    if isempty(flag) ~= 1
        temp3(end+1) = temp(ind)
    else
        temp4(end+1) = temp(ind);
    end
end
```

Why is this code hard to read?

1 Meaningless names for variables

2 Unnecessary complex logic

Example

```
for ind=1:length(engineSpeedSignal)
    flag=find(engineSpeedSignal(ind)>=engineSpeedThreshold,1);
    if isempty(flag) ~= 1
        engineSpeedOverThreshold(end+1) = engineSpeedSignal(ind)
    else
        engineSpeedBelowThreshold(end+1) = engineSpeedSignal(ind);
    end
end
```

1 Meaningless names for variables

2 Unnecessary complex logic

Example

```
for engineSpeed=engineSpeedSignal
    if engineSpeed >= engineSpeedThreshold
        engineSpeedOverThreshold(end+1) = engineSpeed;
    else
        engineSpeedBelowThreshold(end+1) = engineSpeed;
    end
end
```

1 Meaningless names for variables

2 Unnecessary complex logic

Example

```
engineSpeedOverThreshold  = engineSpeedSignal(engineSpeedSignal>=engineSpeedThreshold);
engineSpeedBelowThreshold = engineSpeedSignal(engineSpeedSignal< engineSpeedThreshold);
```

1 Meaningless names for variables

2 Unnecessary complex logic

Meaningful variable names

Searchable/ exposed names

```
int d; // elapsed time in days ✗
int elapsedTimeInDays; ✓
int daysSinceCreation; ✓
int daysSinceModification; ✓
int fileAgeInDays; ✓
```

Pronounceable names/ decoding

```
int genymdhms; ✗
int modymdhms; ✗
String dsc; ✗
int generationTimestamp; ✓
int modificationTimestamp; ✓
String description ✓
```

Functions and methods

One meaningful verb per operation

```
calc() ✗
calcWeightedMean() ✓
setName() ✓
getState() ✓
Complex.fromRealNumber(29.3) ✓
```

Rules for functions

1. Should be smaller than that.
2. < 150 characters per line
3. < 20 lines
4. Functions should do one thing. They should do it well. They should do it only. (no side effects)
5. Avoid globals or tartegt reference passing
6. 10 Inputs: function may be to complex?

Meaningful variable names

Searchable/ exposed names *

```
int d; // elapsed time in days ✗
int elapsedTimeInDays; ✓
int daysSinceCreation; ✓
int daysSinceModification; ✓
int fileAgeInDays; ✓
```

Pronounceable names/ decoding *

```
int genymdhms; ✗
int modymdhms; ✗
String dsc; ✗
int generationTimestamp; ✓
int modificationTimestamp; ✓
String description ✓
```

* Also for the authors own understanding of the code logic

Functions and methods

One meaningful verb per operation *

```
calc() ✗
calcWeightedMean() ✓
setName() ✓
getState() ✓
Complex.fromRealNumber(29.3) ✓
```

Matlab scecific function header & expansion

```
function ouput = myVerySpecificFunction(input1, input2,…) ✗

function ouput = myVerySpecificFunction(varargin) ✓

   for i = 1:2:nargin
         if strcmp(varargin{i},‘input1‘)
            input1 = varargin{i+1};
         elseif …
         else …
         end
   end
   …
end
```

key-value pairs simulating dictionary

~~**containers.Map**~~

Comments

Comments do not make up for bad code

```
// don‘t comment bad code, rewrite it !

// Check to see if the employee is eligible for full benefits
if ((employee.flags && HOURLY_FLAG) && (employee.age > 65)) ✗
if (employee.isEligibleForFullBenefits()) ✓
```

Clarification ✓

```
assertTrue(a.compareTo(b) == -1) // a < b

assertTrue(a.compareTo(b) == 1) // b > a
```

Legal Comments ✓

```
// Copyright (C) 2021 by Wiessler. All
rights reserved. Released under the
terms of the GNU General Public license
```

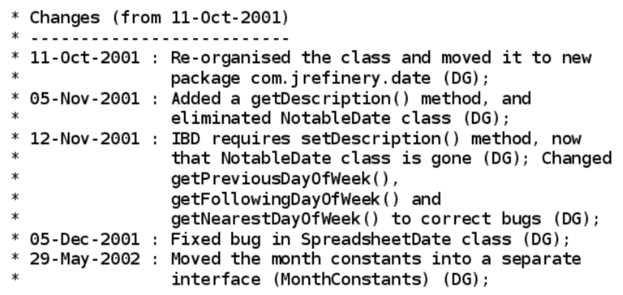
Noise/ redundant comments ✗

```
/**
 * Default constructor
 */
protected UserInteraction();

/** The name */
private String name

/** The version. */
private String version
```

Journal ✗



```
* Changes (from 11-Oct-2001)
* --------------------------
* 11-Oct-2001 : Re-organised the class and moved it to new
*               package com.jrefinery.date (DG);
* 05-Nov-2001 : Added a getDescription() method, and
*               eliminated NotableDate class (DG);
* 12-Nov-2001 : IBD requires setDescription() method, now
*               that NotableDate class is gone (DG); Changed
*               getPreviousDayOfWeek(),
*               getFollowingDayOfWeek() and
*               getNearestDayOfWeek() to correct bugs (DG);
* 05-Dec-2001 : Fixed bug in SpreadsheetDate class (DG);
* 29-May-2002 : Moved the month constants into a separate
*               interface (MonthConstants) (DG);
*
```

Position markers & closing brace ✗

```
// Actions ///////////////////

while (lineCounter < 10){
  // do stuff
}// while
```