



# INTERNET OF THINGS

**Code quality**

**pH.D. Zenoviy Veres**  
**Solution Architect**  
**Assistant Professor @ NULP**



**Tell me, what is a clean code for you?**

```
for (i = 0; i < strlen(line); i++) {  
    short char_code = isupper(line[i]) ? line[i] - 64 :  
line[i] - 96;  
    if (char_code > 0 && char_code < 27) {  
printf("%c", isupper(line[i]) ?  
((char_code + step) % 26 + 64) : ((char_code + step)  
% 26 + 96));  
    } else {  
    printf("%c", line[i]);  
    }  
}
```



```
void CEquation::trio(double array[][rt+1],double alfa[][rt+1])
{
    for (int L = 0; L<rt-1;L++)
    {
        for(int k = 1+L; k<rt+1; k++)
        {
            alfa[L][k] = - (array[L][k]/array[L][L]);
            for (int i=L+1;i<rt;i++)
                array[i][k] = array[i][k] + array[i][L]*alfa[L][k];
        }
    }
    alfa[rt-1][rt+1-1]= - array[rt-1][rt+1-1]/array[rt-1][rt-1];
}
```

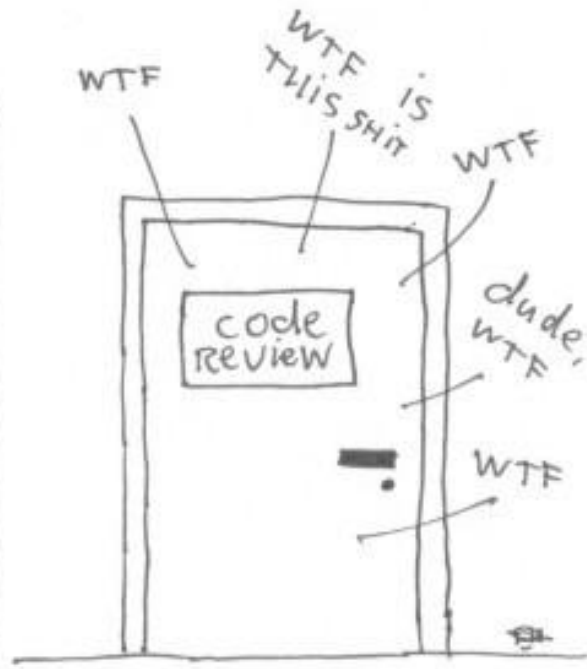


# Good vs Bad

The ONLY VALID MEASUREMENT  
OF CODE QUALITY: WTFs/MINUTE



Good code.



Bad code.



# How to write beautiful code

There are two parts to learning craftsmanship: knowledge and work. You must gain the knowledge of principles, patterns, practices, and heuristics that a craftsman knows, and you must also grind that knowledge into your fingers, eyes, and gut by working hard and practicing.

(Robert C. Martin, Clean code)



# Why its important?

**80% of the lifetime cost of a piece of software goes to maintenance.**

**Hardly any software is maintained for its whole life by the original author.**

**Code quality improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.**

# Why do we see a bad code?

I will not write any more bad code  
I will not write any more bad code  
I will not write any more bad code  
I will not write any more bad code  
I will not write any more bad code  
I will not write any more bad code

**Time:** I don't have a time

We never seem to have time to do it,  
but always seem find time to redo it?!

**Knowledge:** what is a good code?

**Tools:** I don't know about it

**Skills:** I can't do it





# THE BOY SCOUT RULE

---

The Boy Scouts have a rule: "Always leave the campground cleaner than you found it."

---

"Always check a module in cleaner than when you checked it out."

Source: [http://programmer.97things.oreilly.com/wiki/index.php/The\\_Boy\\_Scout\\_Rule](http://programmer.97things.oreilly.com/wiki/index.php/The_Boy_Scout_Rule)

---

# Naming is important



I am the **Baby Name Genie**,  
a baby namer specializing in  
first and middle baby names.

I'll grant your wish for the  
**perfect name** for your  
baby boy or baby girl.

## Ask The Genie

Baby's Last Name:

Baby's Gender:

Surprise Me ▼

**MAKE A WISH**

# The name should represent the developer's idea

```
int d; // time from beginning
```

```
int elapsedTimeInDays;  
int daysSinceCreation;  
int daysSinceModification;
```



# The name should represent the developer's idea

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

What kinds of things are in theList?

What is the significance of the zeroth subscript of an item in theList?

What is the significance of the value?

How would I use the list being returned?



# The name should represent the developer's idea

```
public List<int[]> getFlaggedCells() {  
    List<int[]> flaggedCells = new ArrayList<int[]>();  
    for (int[] cell : gameBoard)  
        if (cell[STATUS_VALUE] == FLAGGED)  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```



# Use meaningful difference

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

**VS**

**source** and **destination** are used for the argument names



# Use meaningful names

```
private Date genymdhms;  
private Date modymdhms;  
private final String pszqint = "102";
```

**VS**

```
private Date generationTimestamp;  
private Date modificationTimestamp;;  
private final String recordId = "102";
```




# Use searchable names



```
#define CFT_CNT 13  
#define SSIM_CLS_CNT 4096  
#define PX_BUFF_CNT 10000
```



# Class names



Classes and objects should have noun or noun phrase names like Customer, WikiPage, Account, and AddressParser.

Avoid words like Manager, Processor, Data, 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** according to the javabean standard.

# Method Names Should Say What They Do



```
Date newDate = date.add(5);
```

Would you expect this to **add five days to the date**? Or is it **weeks**, or **hours**?

Is the date **instance changed** or does the function just return a new Date **without changing** the old one?

*You can't tell from the call what the function does.*



# Functions (aka methods)

```
public void pay() {  
    for (Employee e : employees) {  
        if (e.isPayday()) {  
            Money pay = e.calculatePay();  
            e.deliverPay(pay);  
        }  
    }  
}
```

```
public void pay() {  
    for (Employee e : employees)  
        payIfNecessary(e);  
}  
  
private void payIfNecessary(Employee e) {  
    if (e.isPayday())  
        calculateAndDeliverPay(e);  
}  
  
private void calculateAndDeliverPay(Employee e) {  
    Money pay = e.calculatePay();  
    e.deliverPay(pay);  
}
```

# Functions (aka methods)

## ***F1: Too Many Arguments***

Functions should have a small number of arguments. No argument is best, followed by one, two, and three.

## ***F2: Output Arguments***

*Output arguments are counterintuitive. Readers expect arguments to be inputs, not outputs. If your function must change the state of something, have it change the state of the object it is called on.*

# Functions (aka methods)

## ***F3: Flag Arguments***

Boolean arguments loudly declare that the function does more than one thing. They are confusing and should be eliminated

## ***F4: Dead Function***

Methods that are never called should be discarded. Keeping dead code around is wasteful. Don't be afraid to delete the function. Remember, your source code control system still remembers it.



**I'll leave the project when complete all my technical debt**



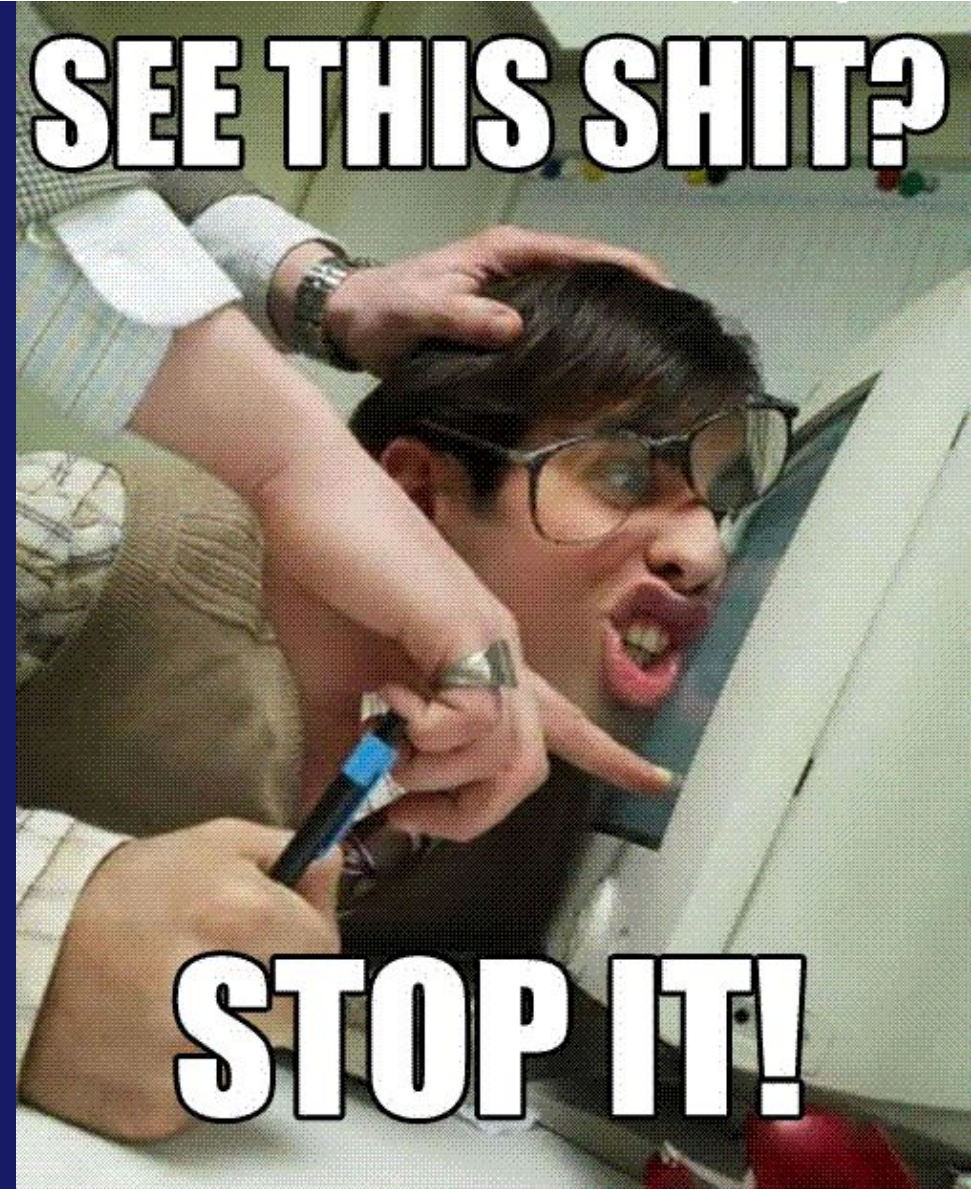
# Code review

Each team member review the code and writing comments/recomendations

It's a primary responsibility of TL/Senior at the project

Code have to be understandable by your colleagues (from other teams)

Good news – we have automatic code quality check tools available



# Comments

- **C1: *Inappropriate Information***

- Change histories(?)
- Authors(?)
- Date of last update(?)

- **C2: *Obsolete Comment***

- It is best not to write a comment that will become obsolete
- If you find an obsolete comments – update or erase it ASAP!

# Comments

- ***C3. Redundant Comment***

- Don't comment what a code does – I can read the code for that—keep it DRY

## Example 1

```
i++; // increment i
```

## What about example 2?

```
/**  
 * @param sellRequest  
 * @return  
 * @throws ManagedComponentException  
 */  
public SellResponse beginSellItem(SellRequest sellRequest)  
    throws ManagedComponentException
```

# Comments

- **C4: Poorly Written Comment**

- Comments should say *Why* or *purpose*, not *how*

- **C5: Commented-Out Code**

- *Who knows how old it is? Who knows whether or not it's meaningful? Yet no one will delete it because everyone assumes someone else needs it or has plans for it.*

# General

- **G1: *Multiple Languages in One Source File***
  - a Java source file might contain snippets of XML, HTML, YAML, JavaDoc, English, JavaScript or
  - in addition to HTML a JSP file might contain Java, a tag library syntax, English comments, Javadocs, XML, JavaScript, and so forth.
  - The ideal is for a source file to contain one, and only one, language. Realistically, we will probably have to use more than one.

# General

- **G5: *Duplication***

- Every time you see duplication in the code, it represents a missed opportunity for abstraction.
- Result of copy/paste programming – to separate method
- switch/case or if/else chain that appears again and again in various modules, always testing for the same set of conditions – to use polymorphism

# General

- **G5: *Duplication (cont)***

- modules that have similar algorithms, but that don't share similar lines of code – to use Template Method or Strategy design patterns.

**Find and eliminate duplication wherever you can!**

- ***G6: Code at Wrong Level of Abstraction***

- Good software design requires that we separate concepts at different levels and place them in different containers

```
public interface Stack {  
    Object pop() throws EmptyException;  
    void push(Object o) throws FullException;  
  
    double percentFull();  
  
class EmptyException extends Exception {}  
class FullException extends Exception {}
```





- ***G7: Base Classes Depending on Their Derivatives***

- The most common reason for partitioning concepts into base and derivative classes is so that the higher level base class concepts can be independent of the lower level derivative class concepts.

# General

- **G9: *Dead Code***

- You find it in the body of an if statement that checks for a condition that can't happen. You find it in the catch block of a try that never throws. You find it in little utility methods that are never called or switch/case conditions that never occur.

# General

- ***G11: Inconsistency***

- If you do something a certain way, do all similar things in the same way.
- If you name a method **processVerificationRequest**, then use a similar name, such as **processDeletionRequest**, for the methods that process other kinds of requests.

# General

- **G23: Prefer Polymorphism to If/Else or Switch/Case**

- **"ONE SWITCH" rule:**

*There may be no more than one switch statement for a given type of selection. The cases in that switch statement must create polymorphic objects that take the place of other such switch statements in the rest of the system.*

# General

- **G28: *Encapsulate Conditionals***

```
if (shouldBeDeleted(timer))
```

is preferable to

```
if (timer.hasExpired() &&  
    !timer.isRecurrent())
```

# General

- **G29: Avoid Negative Conditionals**

```
if (buffer.shouldCompact())
```

is preferable to

```
if (!buffer.shouldNotCompact())
```

# General

- **G35: *Keep Configurable Data at High Levels***

```
class Arguments {  
    public static final String DEFAULT_PATH = ".";  
    public static final String DEFAULT_ROOT = "FitNesseRoot";  
    public static final int DEFAULT_PORT = 80;  
    public static final int DEFAULT_VERSION_DAYS = 14;  
}  
  
public static void main(String[] args) {  
    Arguments arguments = parseCommandLine(args);  
}
```

# One more

- **Clear, not Clever**

- Don't be clever, instead be clear

**“I never make stupid mistakes. Only very, very clever ones”**

John Peel



# What is “code smell”?

Wiki say:

In computer programming, **code smell** is any symptom in the source code of a program that possibly indicates a deeper problem. Code smells are usually not bugs—they are not technically incorrect and don't currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future.



# The most “popular” code smells

Duplication

Unnecessary complexity

Useless/misleading comments

Long classes

Long methods

Poor naming

Code that’s not used

Improper use of inheritance

Convolutted code

Tight coupling

Over abstraction

Design Pattern overuse

Trying to be clever

...

# Bad vs. Clean code

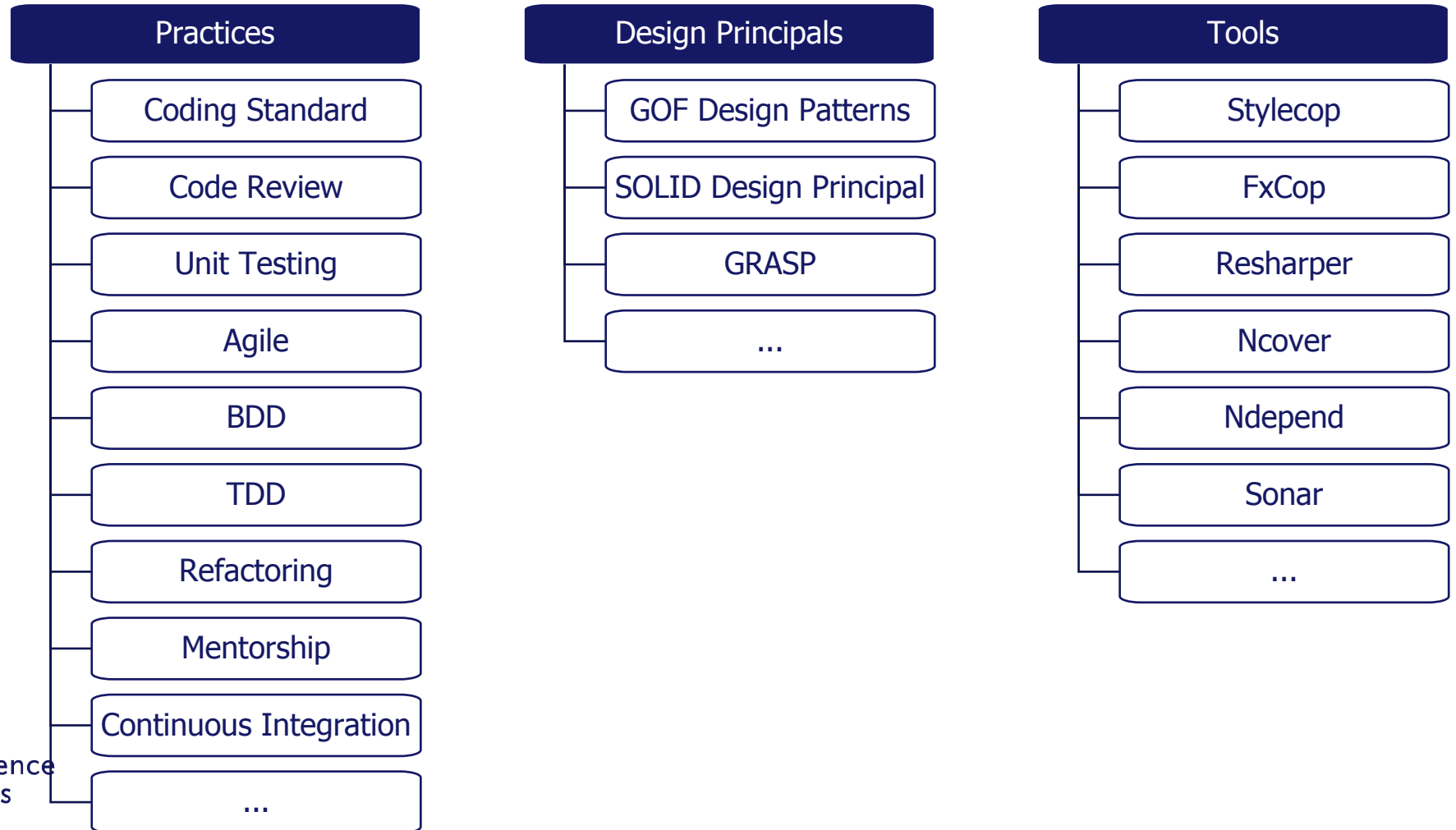
- What is the clean code?
- What is the bad code?
  
- Characteristics of quality code
- Metrics to measure quality
- Ways to identify and build quality



# What do you know now?

- What is Code Smell
  - It's a feeling or sense that something is not right in the code
  - You can't understand it
  - Hard to explain
  - Does some magic
  
- Can we measure it?

# How to improve code quality?





INTERNET  
OF THINGS

**Thank you for attention!**

**[www.iot.lviv.ua](http://www.iot.lviv.ua)**

**[www.facebook.com/iotlvivua](https://www.facebook.com/iotlvivua)**

**[viruslviv@gmail.com](mailto:viruslviv@gmail.com)**