



STATIC CODE ANALYSIS and MANUAL CODE REVIEW

Jakub Papcun
Jan Svoboda

HELLO!

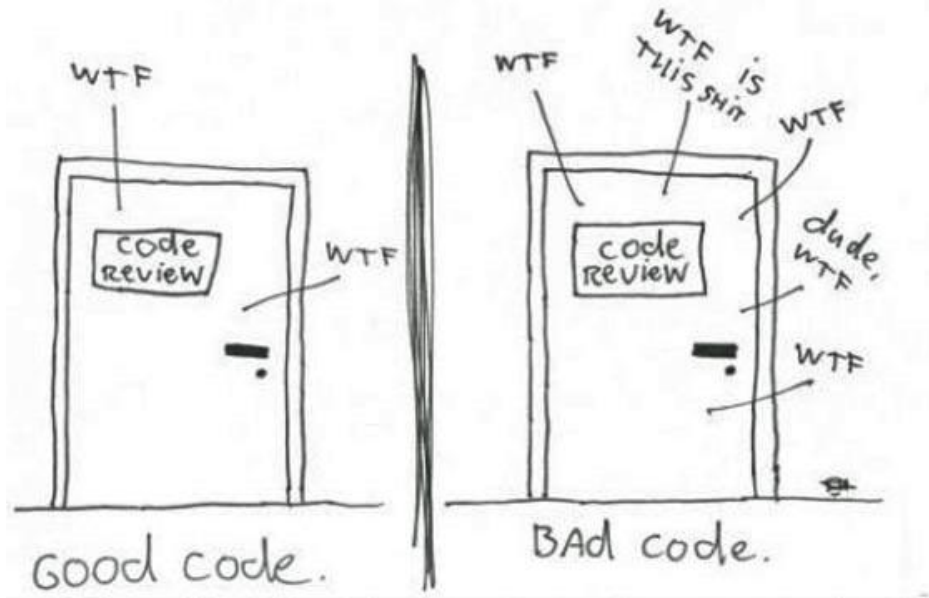
Jakub Papcun

- FI MUNI graduate
- SW Developer since 2011
- DevOps since 2018

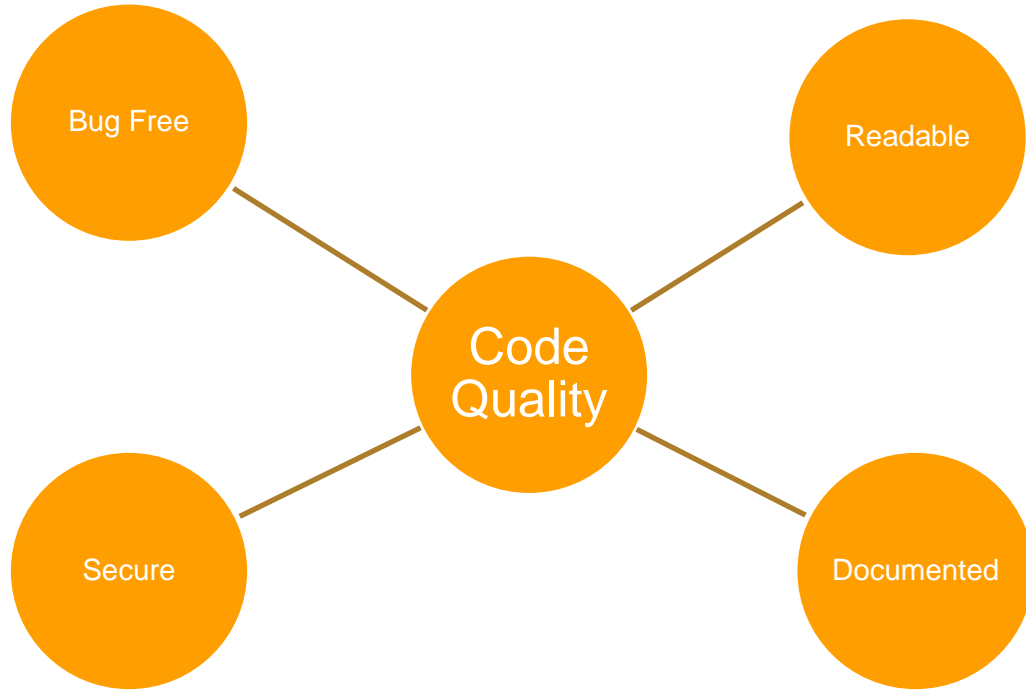
Jan Svoboda

- FI MUNI graduate
- SW Developer since 2011

The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/MINUTE



CODE QUALITY





**THERE IS NO
PERFECT CODE**

Analysis of computer software performed without executing the software.

ADVANTAGES

- No program execution
- Automated process
- Possibility to run as part of Continuous Integration



C++



eclipse

C#



Visual Studio®

TYPES OF STATIC CODE ANALYSIS

Type Checking

- checks correct assignment of types of objects

Style Checking

- checks style of the code and its formatting

Program Understanding

- helps user make sense of large codebase and may include refactoring capabilities

Security review

- uses dataflow analysis for detection of possible code injection

Bug Finding

- looks for places in the code where program may behave in a different way from the way intended by developer

WHY USE STATIC CODE ANALYSIS

Higher Code Quality

Cheaper defect fixing

Repeatability

Readability

No Input

Education

Coding
Guidelines
Compliance

DRAWBACKS

False
sense
of
security

Only
STATIC
analysis

Possible
overhead

PITFALLS OF STATIC CODE ANALYSIS

	Is a problem	Is NOT a problem
Was found	True Positive	False Positive
Was NOT found	False Negative	

```
public static Pair<Long, String> parseConfigurationString(String configurationStr) {
    Long linkTypeId = null;
    String direction = null;
    int sepPos = 1 configurationStr != null ? configurationStr.indexOf('|') : -1;
    if (sepPos >= 0) {
        String linkTypeIdStr = 2 configurationStr.substring(0, sepPos);
    }
}
```

A "NullPointerException" could be thrown; "configurationStr" is nullable here. ...

2 months ago ▾ L181 🔗

🐛 Bug 📈 Major ✅ Resolved (False Positive) ▾ Not assigned 10min effort Comment

cert, cwe


JP Jakub Papcun sepPos is -1 if configurationStr is null ... the the next IF checks if sepPos is 0 or higher thus by transition checking if configurationStr is null or not.




2 months ago ✎ ✕

HELLO WORLD

```
1 public class HelloWorld {
```

Add a private constructor to hide the implicit public one. [...](#)


2 months ago ▾ L1 




 Code Smell  Major  Open Not assigned 30min effort


 design

```
2
3     public static void main(String[] args) {
4         // Prints "Hello, World" to the terminal window.
5         System.out.println("Hello, World");
```

Replace this usage of System.out or System.err by a logger. [...](#)

2 months ago ▾ L5 

 Code Smell  Major  Open Not assigned 10min effort

 bad-practice, cert

```
6     }
7
8 }
```



LEARN ABOUT YOURSELF

- Lines of Code (LOC)
- Comments Quality
- Code Duplication
- Technical Debt
- Cyclomatic Complexity
- Cognitive Complexity
- Dependency Cycle Detection

RULES

A checker defining possible issues in the code

- Unused local variable
- Memory leaks
- SQL injection
- Call of function on null

- Reliability issues
- May crash at runtime
- May cause extremely unpredictable behavior

- Null pointer dereference
- Memory leaks
- Buffer overflow

```
1 static void printPoint(Point p) {  
2     if (p == null) {  
3         System.err.println("p is null");  
4     }  
5     if (p.x < 0 || p.y < 0) {  
6         System.out.println("Invalid point");  
7         return;  
8     }  
9     System.out.println(p);  
10 }
```

- Security issues
- Crash or corrupt the system
- Open space for attack

- Harcoding credentials
- Data/SQL Injection
- Not securing “cookies”

TYPES – Vulnerability

```
1 public static void main(String[] args) throws Exception {
2     Properties info = new Properties();
3     info.setProperty("user", "root");
4     info.setProperty("password", "^6nR$%");
5     Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3307", info);
6     try {
7         //...
8     } finally {
9         connection.close();
10    }
11 }
```

- Maintainability issues
- Decrease readability, architecture quality etc.

- Unused private method
- Switch statement that do not end with “default” clause
- Classes with too many fields

```
1 static void printErrorMessage(String message) {  
2     System.out.err("An error occurred");  
3 }
```

```
1 Professional john = new Professional("John", 25, "miner");  
2 public boolean checkJohn(Person p) {  
3     return p == john;  
4 }
```

EXCERSISE

```
private Map<String, String> paths = new HashMap<String, String>();

public void addPath(String name, String path) {
    paths.put(name, path);
}

private String getNormalizedPath(String name) throws IOException {
    return paths.get(name).toLowerCase();
}
```



Can return null

A `NullPointerException` is thrown in case of an attempt to dereference a `null` value.

EXCERSISE

```
private static void foo(){
```

```
    int i = 0;
```

```
    String s = null;
```

```
    if(i > 0){  
        s = "positive";
```

```
    }
```

```
    if(s.contains("pos")){  
        System.out.println(s);
```

```
    }
```

```
}
```

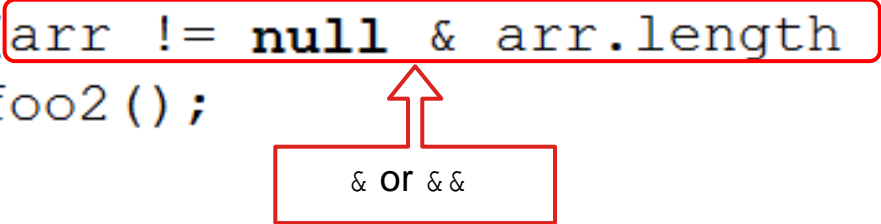
Statement always false

s is always null

1. Statement is always false and never enters the block
2. s variable is always null and NullPointerException may be thrown

EXCERSISE

```
private static void foo(int arr[]) {  
    if(arr != null & arr.length != 0) {  
        foo2 ();  
    }  
    return;  
}
```



Questionable use of bit operation '&' in expression. Did you mean '&&'?

EXCERSISE

```
private static void foo(int j)
    Integer k;
    switch(k)
        case 1: System.out.println("k lower than 2."); break;
        case 2: System.out.println("k equals 2."); break;
        case 3: System.out.println("k bigger than 2."); break;
        default: System.out.println("K = " + k);
    }
    return;
}
```

j is never used

k not initialized

1. j variable is never used and thus redundant
2. k variable is never initialized and thus unusable

EXCERSISE

```
public void foo() {  
    Item item = new Item();  
    if(item.getInfo() != null){  
        String info = item.getInfo().trim();  
    }  
}
```



may return null

```
class Item{  
    public String getInfo() {  
        // Making REST Request  
    }  
}
```

REST may fail and return null



HOW DO I EVEN START?



**IT ALL BEGINS WITH
THE FIRST LINE**



**STOP WITH
REGRESSION**



PERFECTION IS IMPOSSIBLE



DO IT “ON-THE-FLY”



klocwork[®]

a Rogue Wave Company



sonarqube



Pmd

THANKS!

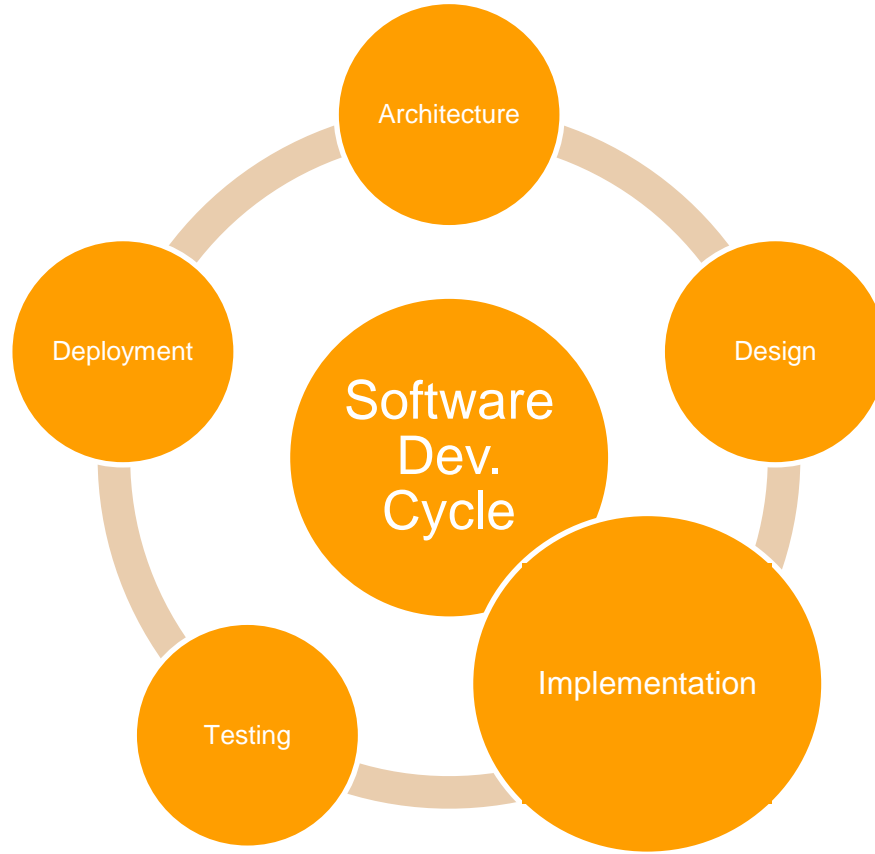
Any questions?

Systematic examination of the source code

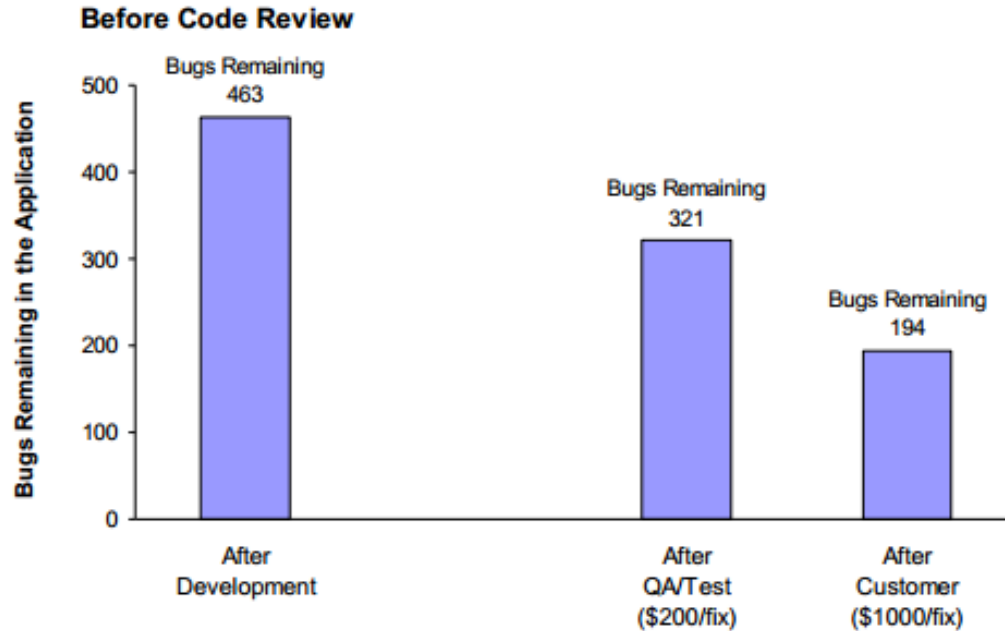
WHY?

Early Defect Detection

MCR IN DEVELOPMENT CYCLE



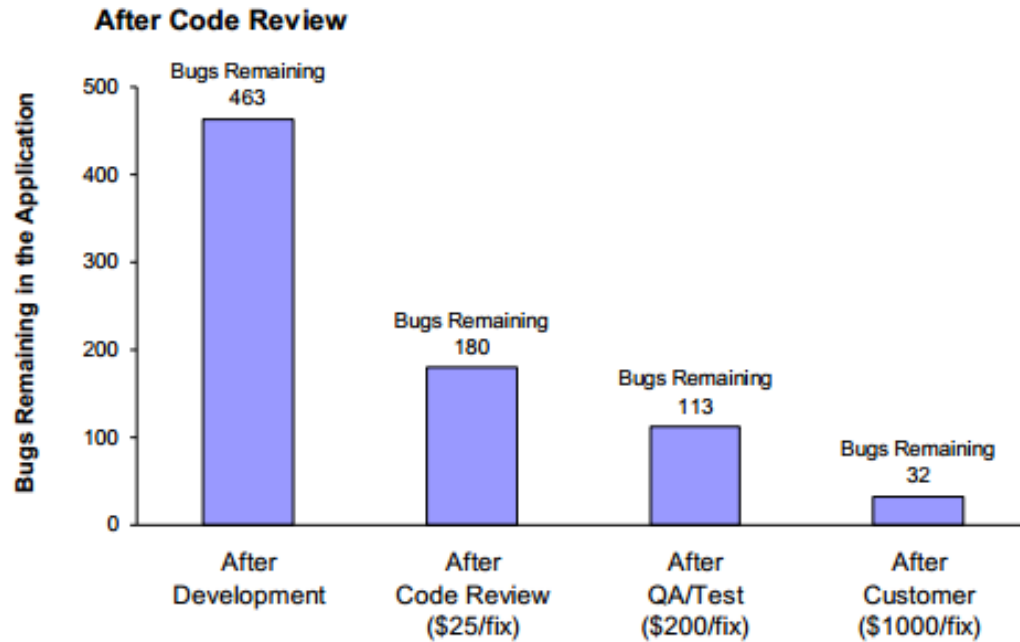
COST OF DEFECT FIX



Cost of fixing bugs: \$174k
+ Cost of 194 latent bugs: \$194k

Total Cost: **\$368k**

COST OF DEFECT FIX



Cost of fixing bugs: \$120k
+ Cost of 32 latent bugs: \$ 32k

Total Cost: **\$152k**

ADVANTAGES

- Different point of view
- Product evolution awareness
- Education

WHAT MAKES GOOD CODE REVIEW?

- Goal
- People
- Technical knowledge

- Formal
- Informal
- Tool-assisted

- Typically, face-to-face meeting
- Roles (moderator, observer, reviewer)
- Participants go through the source code to fulfill goal of review

Pros

- Well documented
- Process oriented

Cons

- Time consuming
- Effort required does not correspond to value gained
- Human Factor

- Typically, two developers (author and reviewer) conducting ad-hoc review
- Over-the-shoulder review
- Extreme programming

Pros

- Simple
- Most effective type of MCR

Cons

- Not documented
- Not process oriented
- Consumes time of two developers

TOOL-ASSISTED CODE REVIEW

- A tool is used for the review
- Designed to mitigate drawbacks of other approaches

Pros

- Documented
- Enforcing process
- Time efficient
- Reviewer has all the time required

Cons

- Cost of the tool
- It is easier for reviewer to cheat

CODE REVIEW TOOL FEATURES

Automated File
Gathering

Automated
Metrics
Collection

Combined
Display

Process
Enforcement



GitHub



GitLab



GIT WORKFLOW



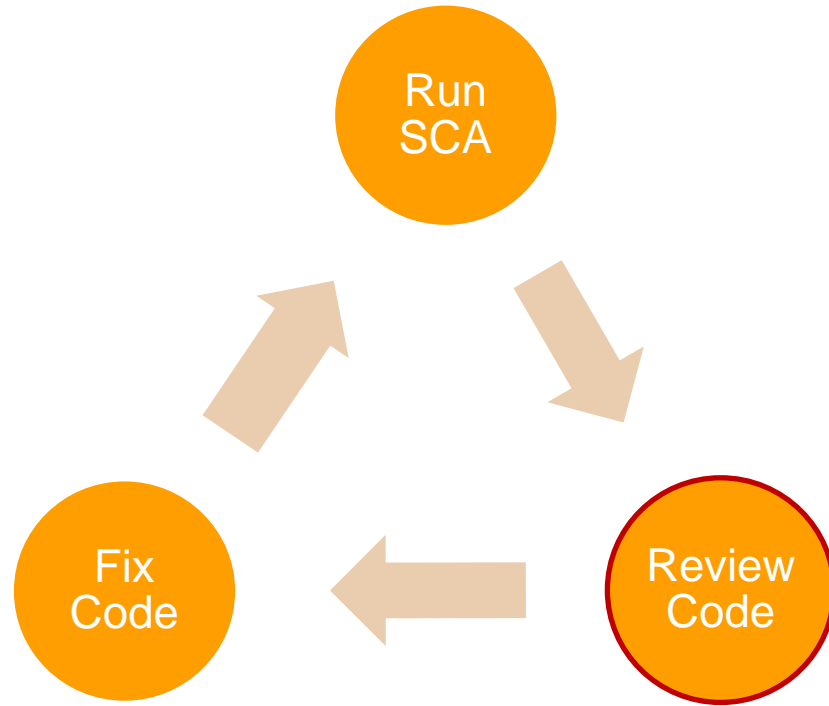


DEMO



Make Code review natural part of development
process

RELATION TO STATIC CODE ANALYSIS?





HUMAN FACTOR

The only factor that ruins manual code review

Effective Code Review

Do it

Don't be
afraid to
have
face-to-
face

Be
honest

Use
proper
and
polite
language

Never be
personal

THANKS!

Any questions?