



STATIC CODE ANALYSIS and MANUAL CODE REVIEW

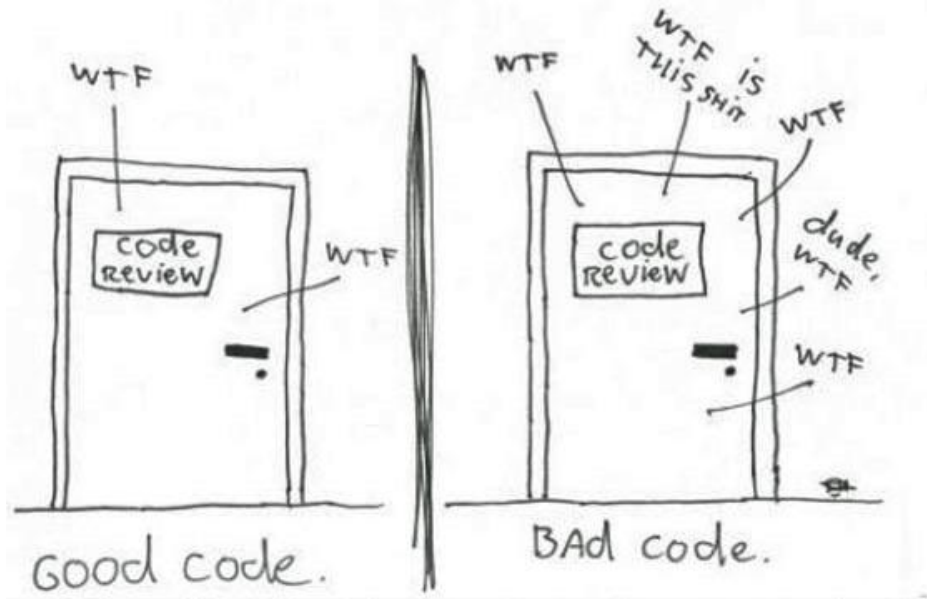
Jakub Papcun
Jan Svoboda

HELLO!

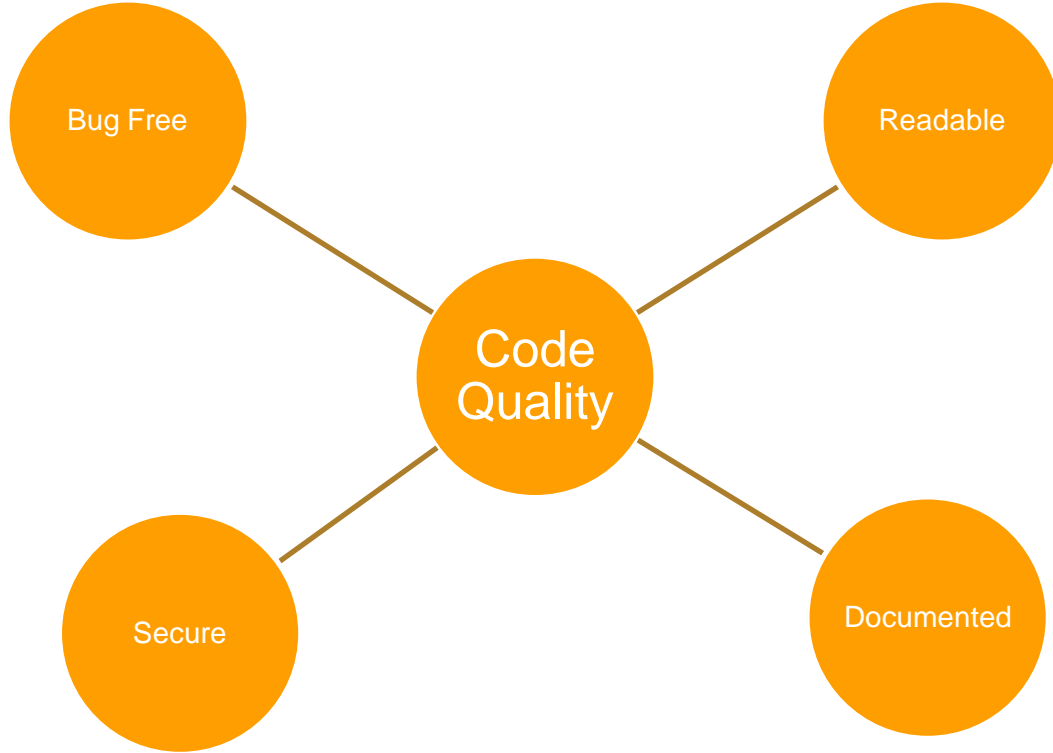
Jakub Papcun, Jan Svoboda

- FI MUNI graduates
- Java Developers since 2011
- Scrum Masters

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
- Possible to run as part of Continuous Integration

1



C++



NetBeans

C#



Visual Studio®

TYPES OF STATIC CODE ANALYSIS

Type Checking

- correct assignment of types of objects

Style Checking

- style of the code and its formatting

Program Understanding

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

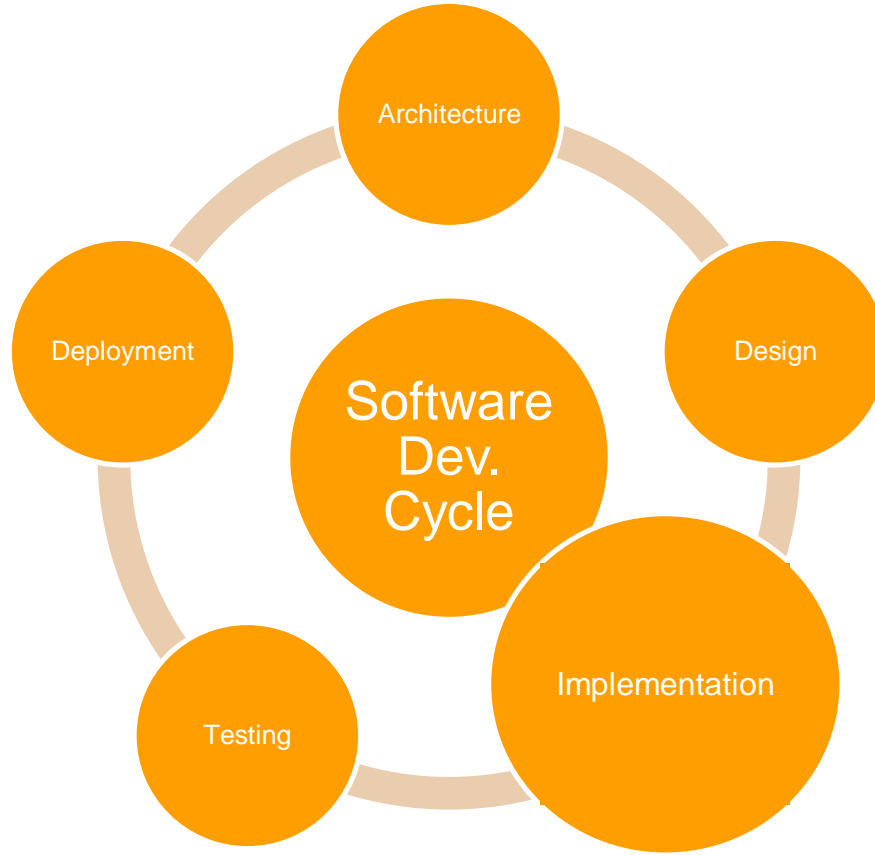
Security review

- 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

MCR IN DEVELOPMENT CYCLE



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

Time
consuming
if done
manually

PITFALLS OF STATIC CODE ANALYSIS

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

```
24 private static final Map<Integer, Integer> PARAM_STATUS_NAME_MAPPING =
25     ImmutableMap.of(PARAM_OPEN_ID, OPEN_STATUS_ID,
26                     PARAM_CLOSED_ID, CLOSED_STATUS_ID);
27
28 private Predicate getPredicateForType(int type, Parameters params, RegularTimePeriod cursor){
29     Predicate result = null;
30     switch (type){
31         case(PARAM_OPEN_ID):
32             int status = PARAM_STATUS_NAME_MAPPING.get(type);
33             result = // do something;
34             break
35         case(PARAM_CLOSED_ID):
36             int status = PARAM_STATUS_NAME_MAPPING.get(type);
37             result = // do something;
38             break
39     }
40     return result;
41 }
```



**YOU CAN LEARN A LOT
ABOUT YOUR CODE**

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

Rule defining possible bug/defect

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

- Very serious problems
- May crash at runtime

- Null pointer dereference
- SQL connection not closed
- 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

- Modification of unmodifiable collection
- Data/SQL Injection
- Memory leaks

```
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 }
```

- Moderate issues
- Do not usually crush running program

- Unused private method
- Possible error in bit operations
- Incorrect allocation size

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

- Coding standards, Performance issues
- Comparing objects with ==
- Empty catch clause
- Statement has no effect

```
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.

EXERCISE

```
private static void foo(){  
    int i = 0;  
    String s = null;  
  
    if(i > 0){  
        s = "positive";  
    }  
  
    if(s.contains("pos")){  
        System.out.println(s);  
    }  
}
```

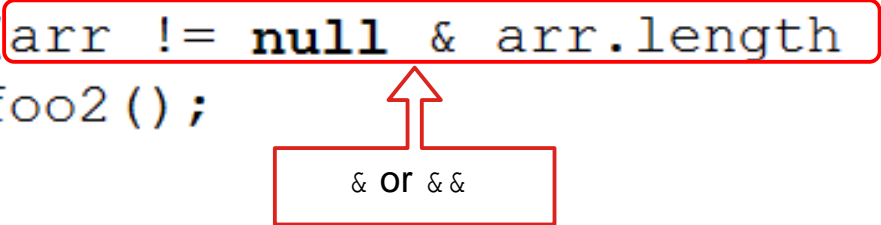
Statement always false

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;  
}
```



The diagram shows a red box around the expression `arr != null & arr.length != 0` in the `if` statement. A red arrow points from a box containing the text `& or &&` to the `&` operator in the expression.

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



klocwork[®]

a Rogue Wave Company



sonarqube



Pmd

THANKS!

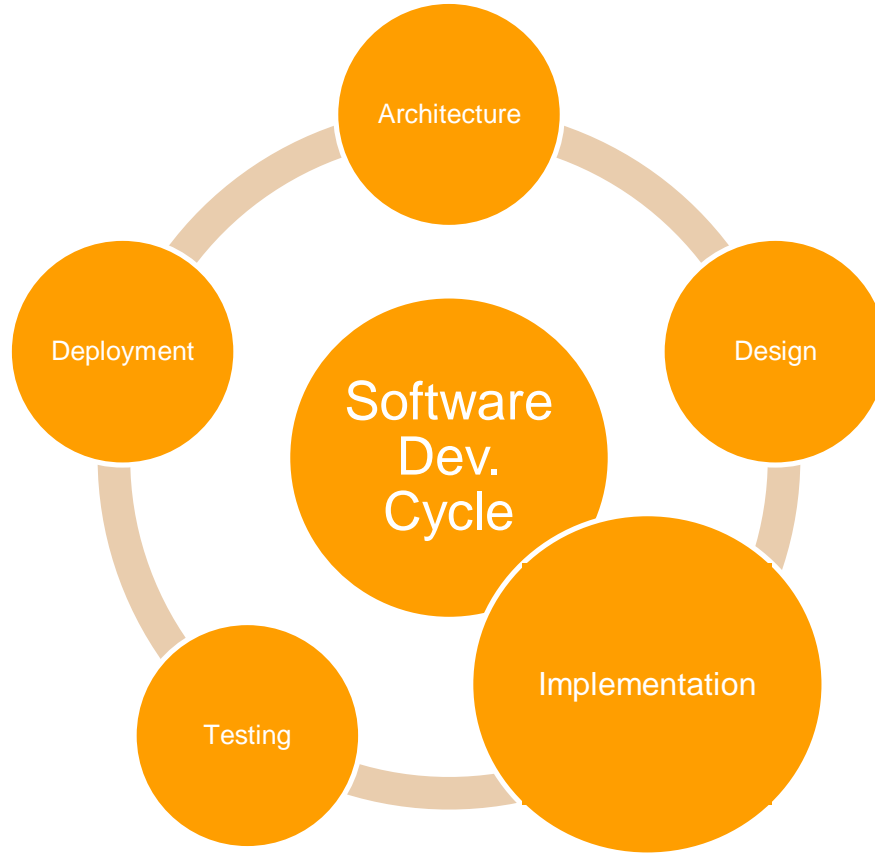
Any questions?

Systematic examination of the source code

WHY?

Defect finding

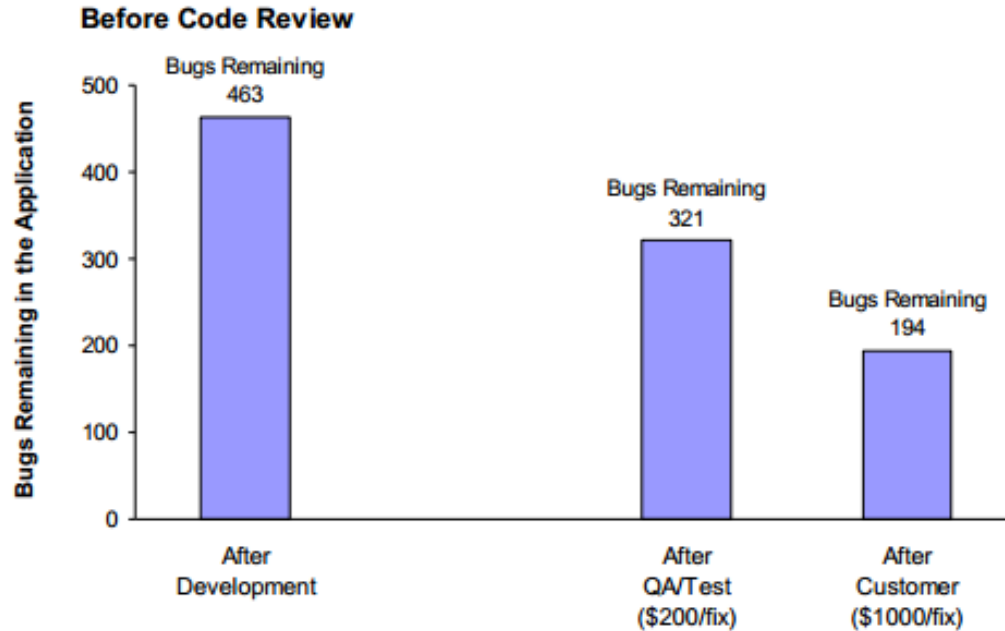
MCR IN DEVELOPMENT CYCLE



ADVANTAGES

- Different point of view
- Decreases cost of defect fixing
- Education

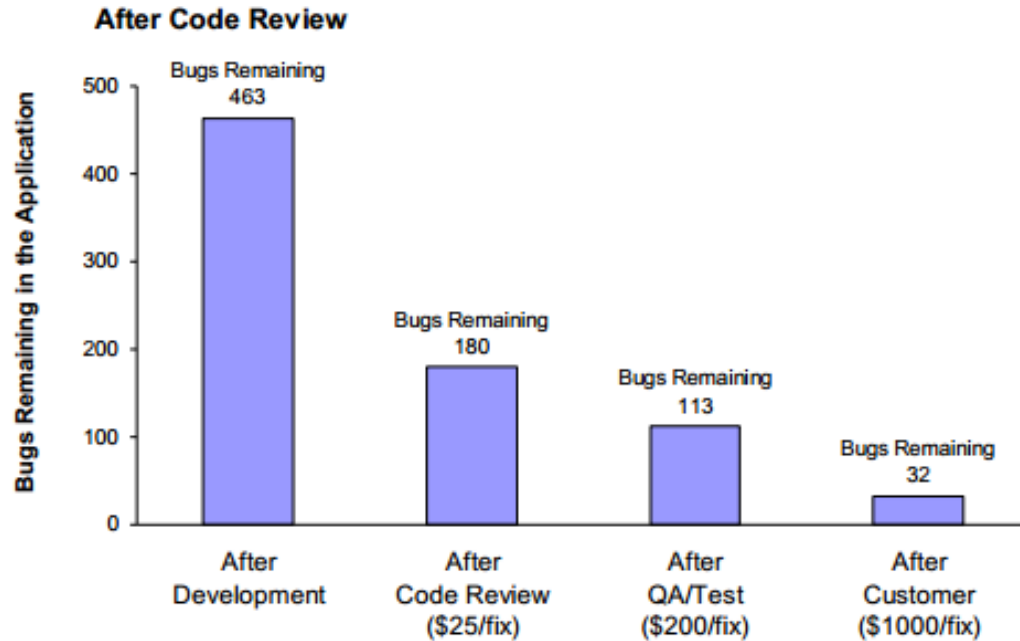
COST OF DEFECT FIX



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**

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
- Saves time and resources

Cons

- Not documented
- Not process oriented

- 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



klocwork[®]

a Rogue Wave Company

Atlassian



Stash



Atlassian



Crucible


ATLASSIAN STASH


Overview **Diff** Commits



Changed files  / webapp / default / src / main / resources / **stash-plugin.xml** **MODIFIED** 

- webapp/default/src/main
 - resources
 - stash-plugin.xml
 - webapp/static
 - images/icons
 - icon-approved.png
 - icon-approved.svg
 - widget/avatar
 - avatar-ie8.less
 - avatar.less

1566	1566	<stash-resource key="avatar" name="Avatar widget">
- 1567		<resource type="download" name="avatar.css" location="/static/widget
+ 1567		<resource type="download" name="avatar-ie8.css" location="/static/wi
1568	1568	<param name="source" value="webContextStatic"/>
+ 1569		<param name="conditionalComment" value="lte IE 8"/>
1569	1570	</resource>
1570	1571	
1571	1572	<resource type="download" name="avatar/" location="/static/images/av
1572	1573	<param name="source" value="webContextStatic"/>
1573	1574	</resource>
+ 1575		<resource type="download" name="icons/icon-approved.png" location="/
+ 1576		<param name="source" value="webContextStatic"/>
+ 1577		</resource>
+ 1578		
+ 1579		<resource type="download" name="icons/icon-approved.svg" location="/
+ 1580		<param name="source" value="webContextStatic"/>
+ 1581		<param name="content-type" value="image/svg+xml"/>
+ 1582		

 **Giancarlo Lionetti**
Add inline comments to start a discussion with your teammates.
[Reply](#) · [Edit](#) · 3 days ago

 **Seb Ruiz**
Threaded comments allow you to have conversations around code changes.
[Reply](#) · [Delete](#) · 3 days ago



[Comment](#) [Cancel](#) Tip: Cmd + Enter to post your comment

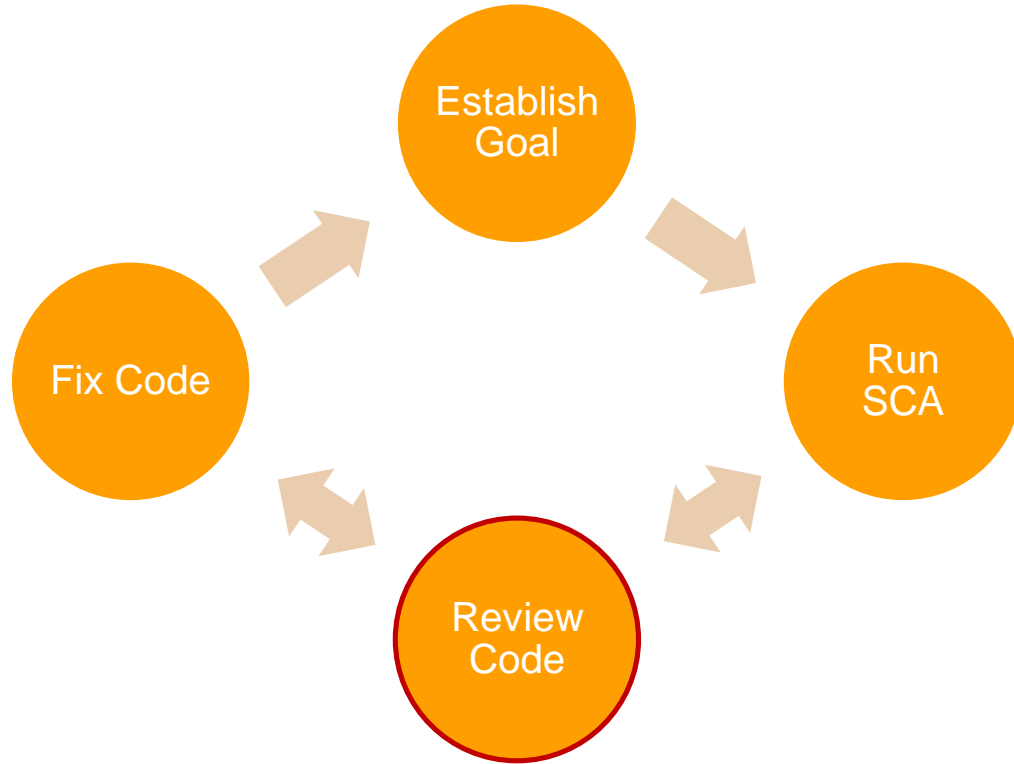


It is better to make Code review natural part of
development process

GIT WORKFLOW



RELATION TO STATIC CODE ANALYSIS?





HUMAN FACTOR

The only factor that ruins manual code review

Effective Code Review

Choose
proper
review
method

Establish
the goal

Be
honest

Use
proper
and
polite
language

Never be
personal

THANKS!

Any questions?