Static Code Analysis and Manual Code Review

Jakub Papcun, Jan Svoboda

About Us

- part time Java developers 2011 – 2014
- full time Software Engineers since 2015

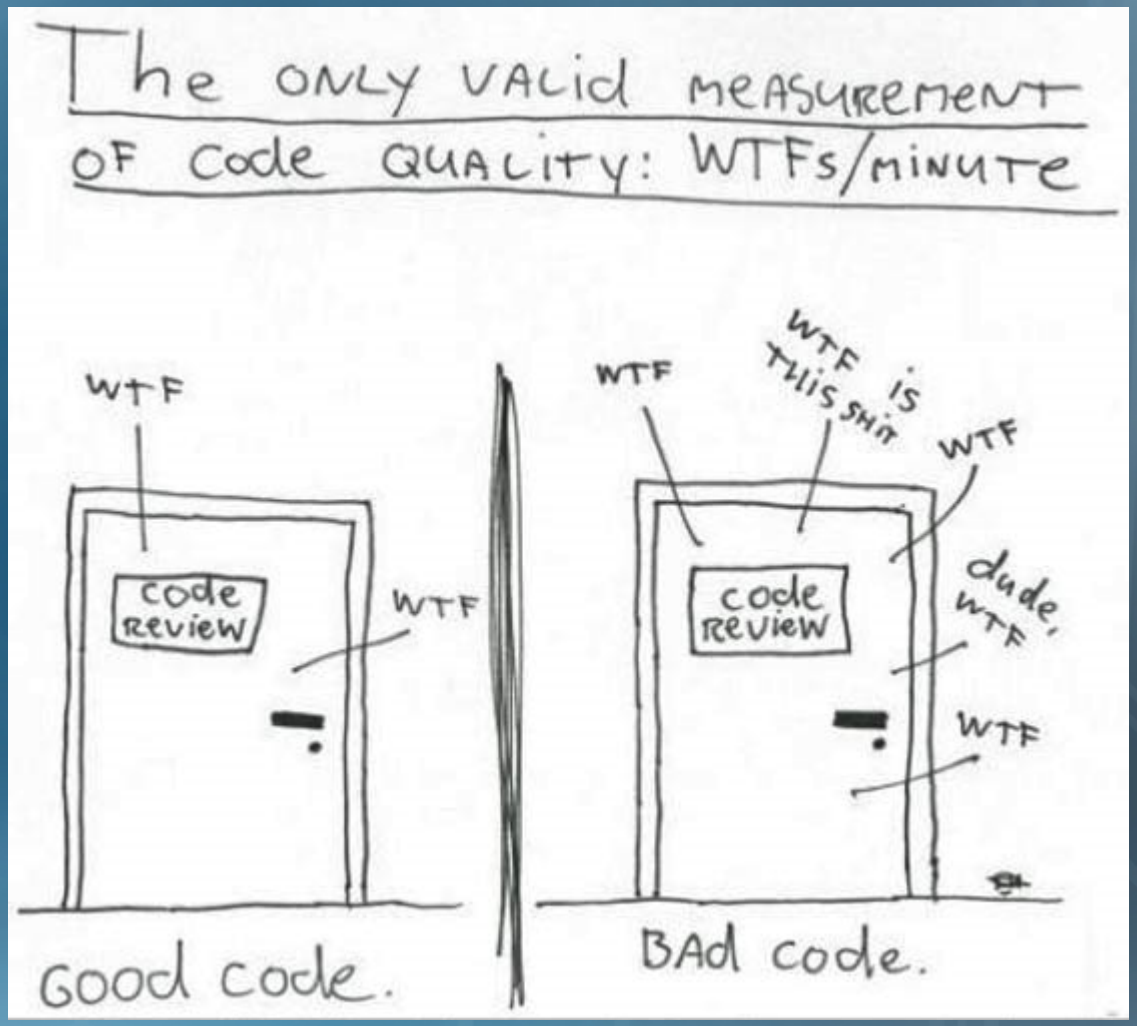
- Experience with full software development cycle, its practices and use of tools
- Some experience with best practices development

- Static Code Analysis and Issue Tracking integration
- Static Code Analysis and Manual Code Review integration

Lecture Outline

- Static Code Analysis, Manual Code Review
  - What it is?
  - Good and Evil sides
  - Why why why
  - Examples

Code Quality



Code Quality

What is Static Code Analysis?

- No program execution
- Performed on Source Code of the software (ideally compiled)
- Automated process

SCA in everyday life

C++

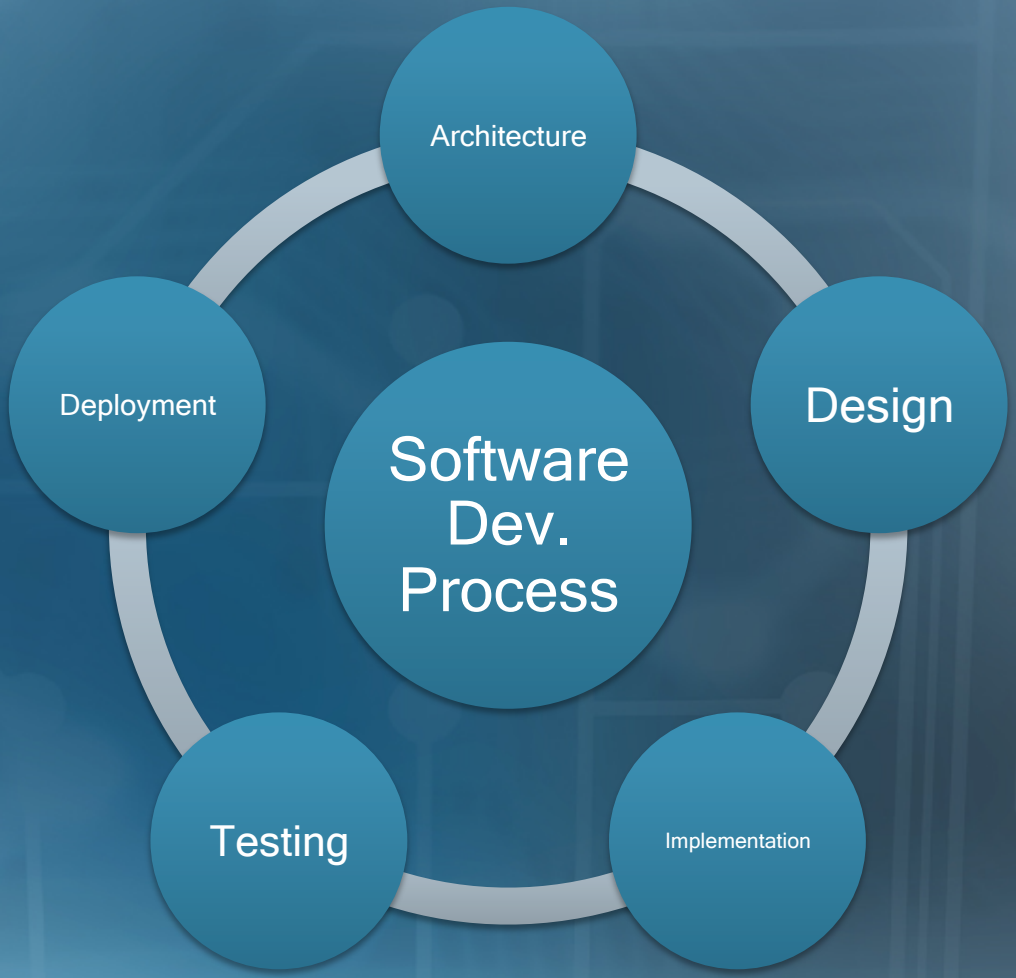
C#

SCA in everyday life

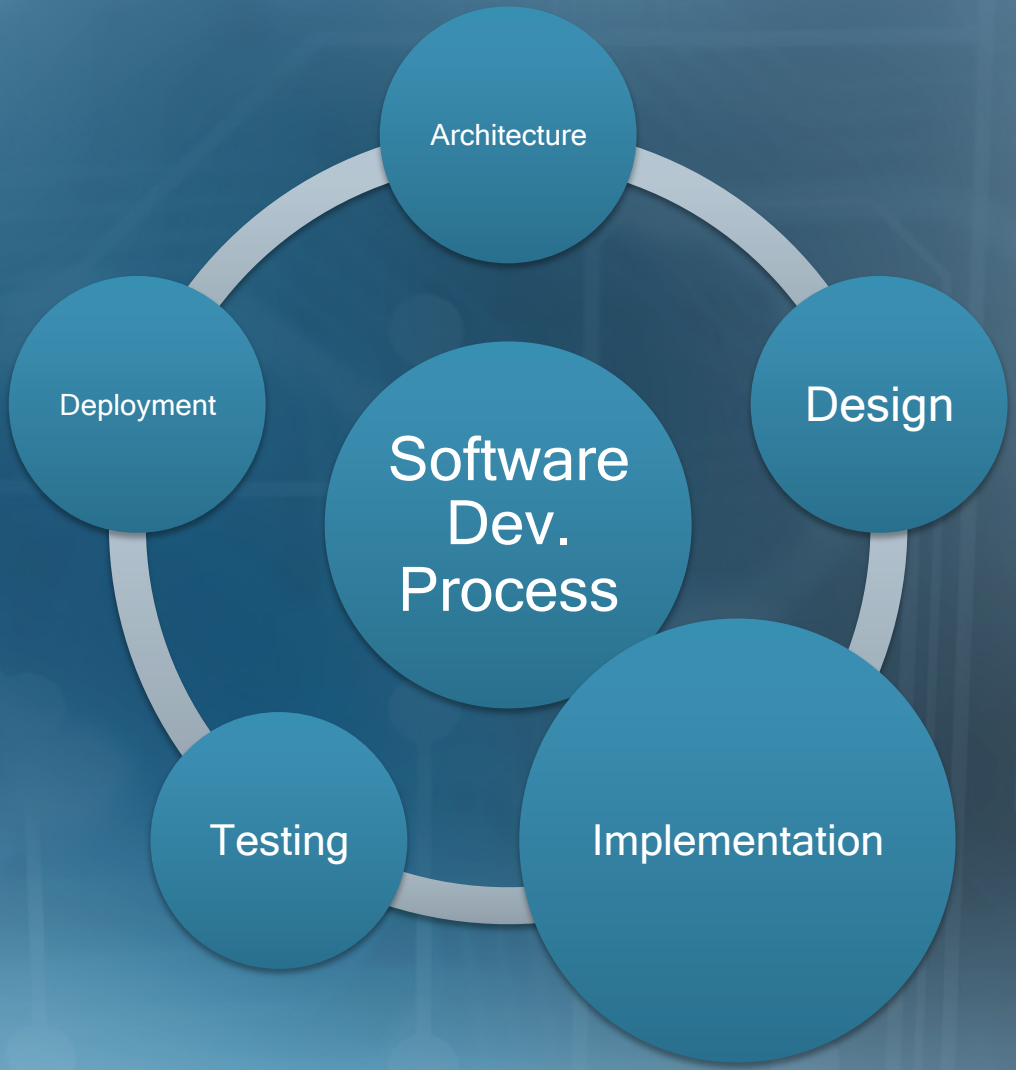
Types of SCA

- Type checking
  - checks for correct assignment of types of objects
- Style checking
  - checks the style of the code and its formatting
- Program Understanding
  - helps user make sense of large codebase and may include refactoring capabilities
- Program verification and property checking
  - attempts to prove that the code correctly implements the specification of the program
- 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

SCA in development cycle



SCA in development cycle



Why Use Static Code Analysis

- Higher Code Quality
- Cheaper defect fixing
- Repeatibility
- Readibility
- No Input
- Education
- Coding Guidelines Compliance

Possible drawbacks

- False sense of security
- Only STATIC analysis
- Possible overhead
- Time consuming if done manually

Dynamic code analysis

- Analysis during execution of program
- DCA process:
  - preparing input data
  - running a test program
  - analyzing the output data
- Able to find run-time errors

Pitfalls of SCA

	Is a Problem	Is not a Problem
Was Reported	True Positive	False Positive
Was not Reported	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    }
```

Metrics

- LOC
- comments quality
- cyclomatic complexity
- dependency cycle detection

Checkers

- Rule defining possible bug/defect
- Examples
  - Unused local variable
  - Memory leaks
  - SQL injection
  - Call of function on null

Severities - Level 1

- Very serious problems
- May crash at runtime
- Examples
  - Null pointer dereference where null comes from condition
  - SQL connection/Input stream is not closed on exit
  - Buffer overflow–array index out of bounds

Severities - Level 1

```
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 }
```

Severities - Level 2

- Serious problems, Security issues
- May crash at runtime
- Examples
  - Modification of unmodifiable collection
  - Data/SQL injection
  - Memory leak possible

Severities - Level 2

```
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 }
```

Severities - Level 3

- May cause moderate problems
- Usually do not crush running program
- Examples
  - Unused private method
  - Possible error in bit operations
  - Incorrect allocation size

Severities - Level 3

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

Severities - Level 4

- Violation of coding standards, possible performance issues
- Very little possibility of program crashing
- Examples
  - Comparing objects with ==
  - Empty catch clause
  - Statement has no effect

Severities - Level 4

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

Example 1

```
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();
}
```

Example 1

```
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.

Example 2

```
private static void foo(){
    int i = 0;
    String s = null;

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

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

Example 2

```
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`

1. Statement is always false and never enters the block

Example 2

```
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

Example 3

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

Example 3

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

& or &&

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

Example 4

```
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;
}
```

Example 4

```
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

1. j variable is never used and thus redundant

Example 4

```
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

Example 5

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


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

Example 5

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


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

may return `null`

REST may fail and return null

Tools

Integration

- World is getting automatized
- Time is money
- Put as much data together as possible

Integration

- Issue Tracking
- Assign Static Code Analysis findings to Issues in Issue Tracking System

Q&A

Manual Code review

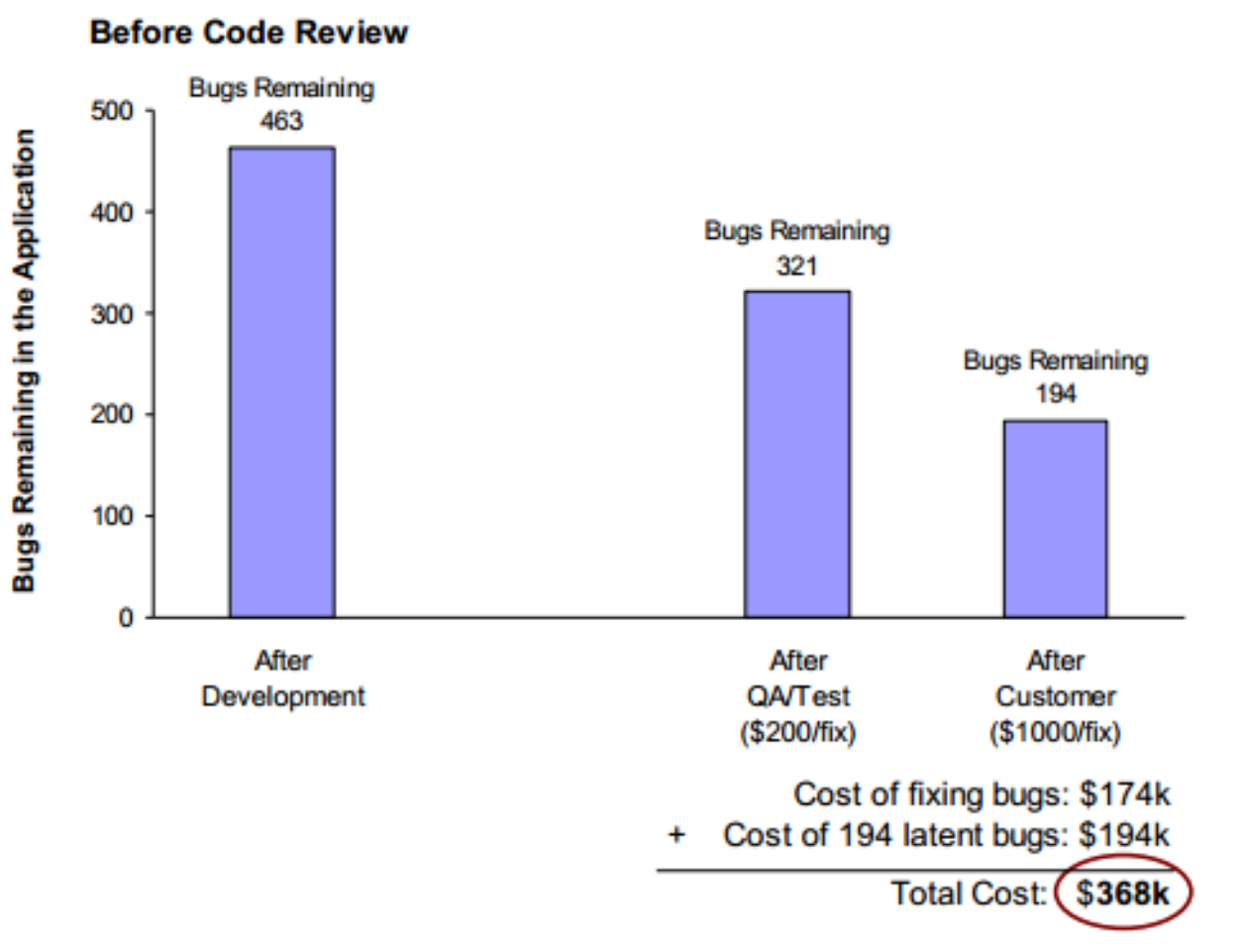
Outline

- What is MCR
- Motivation
- MCR in DEV lifecycle
- Types of MCR
- Pitfalls of MCR
- Relation between MCR and SCA

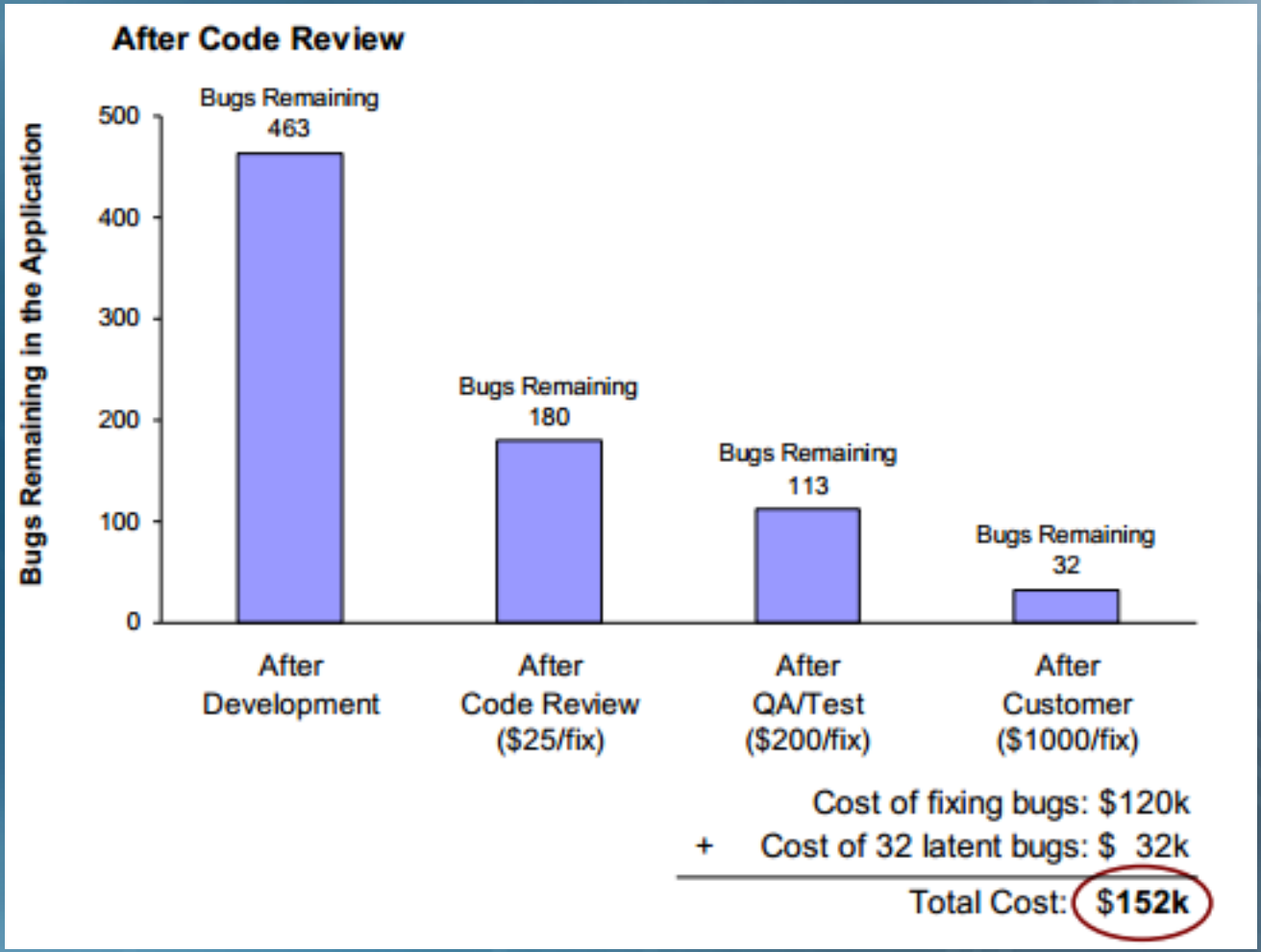
What is MCR

- Systematic examination of the source code
- To be effective
  - The goal of the review needs to be established
  - Some rules need to be obeyed
- Goal determines purpose of review
  - Bug finding
  - Security
  - Architecture compliance

Motivation


Before Code Review
Bugs Remaining in the Application
500
400
300
200
100
0
Bugs Remaining
463
Bugs Remaining
321
Bugs Remaining
194
After
Development
After
QA/Test
($200/fix)
After
Customer
($1000/fix)
Cost of fixing bugs: $174k
+ Cost of 194 latent bugs: $194k
Total Cost: $368k

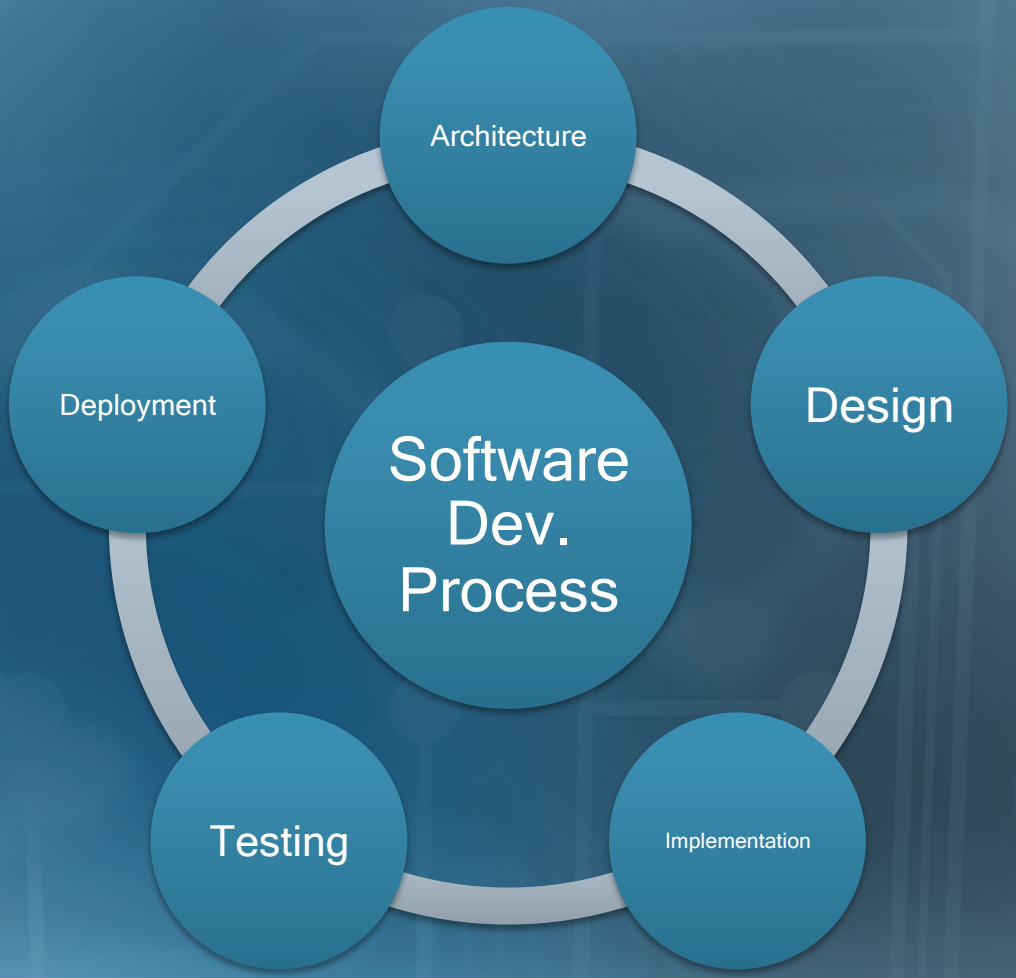
Motivation


After Code Review
Bugs Remaining in the Application
500
400
300
200
100
0
Bugs Remaining 463
Bugs Remaining 180
Bugs Remaining 113
Bugs Remaining 32
After Development
After Code Review ($25/fix)
After QA/Test ($200/fix)
After Customer ($1000/fix)
Cost of fixing bugs: $120k
+ Cost of 32 latent bugs: $ 32k
Total Cost: $152k

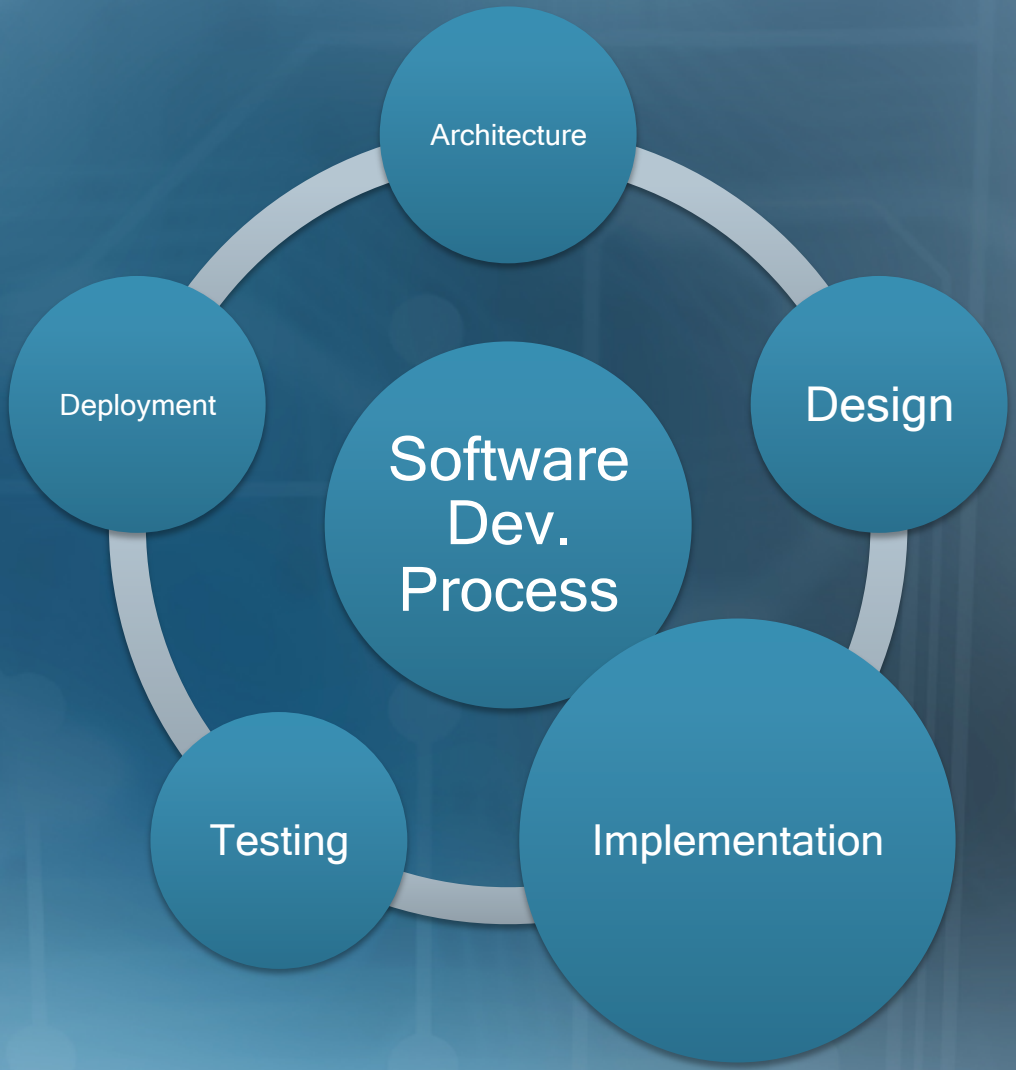
Motivation

- Improves code quality
  - Reviewer has different point of view
- Decreases cost of defect fixing
- Education

MCR in development cycle



MCR in development cycle



Types of MCR

- Formal
- Informal
- Tool-assisted

Formal Review

- 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

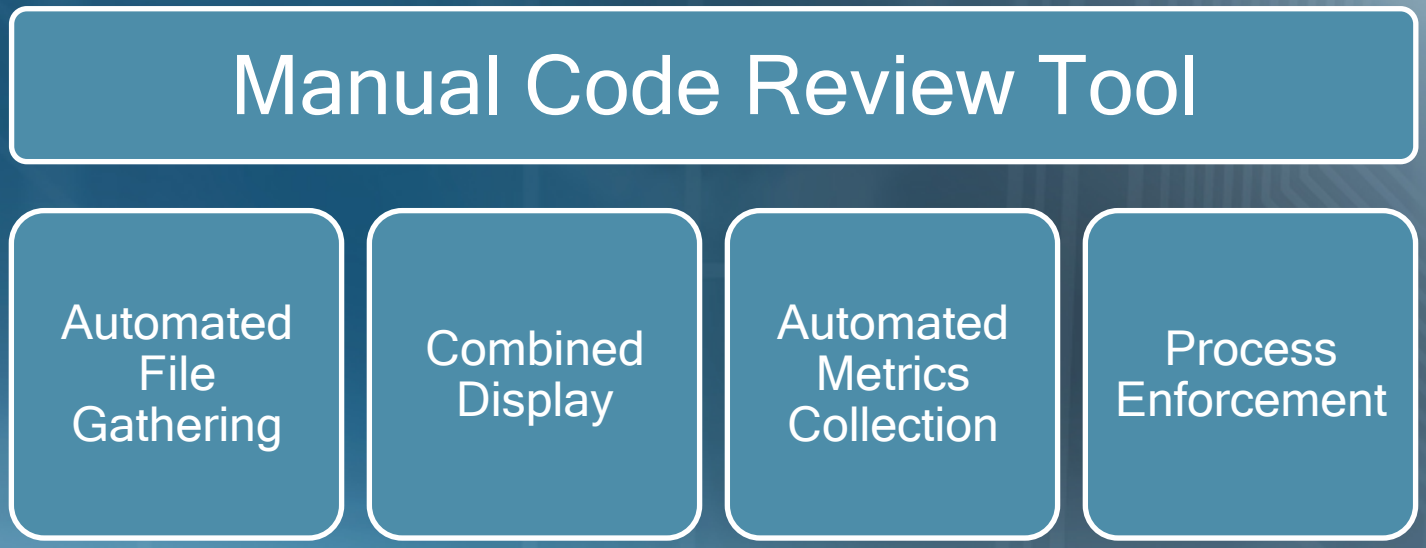
Informal review

- 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 • Higher chance to miss an issue

Tool-assisted review

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



Tool-assisted review

Pros

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

Cons

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

Tools for MCR

Atlassian Crucible

CR-3 (1)

- AllUpperCaseFormatter.java
- AllUpperCaseFormatterTest.java
- CSVFieldFormatter.java
- CSVFormatterConfigParser.java
- CSVFormatterConfigParserTest.java
- CSVFormatterFactory.java
- DoNothingFormatter.java
- FirstWordFormatter.java
- FirstWordFormatterTest.java
- FormatterConfiguration.java
- LastWordFormatter.java
- LastWordFormatterTest.java
- TrimWordFormatter.java
- TrimWordFormatterTest.java
- mapping
  - CSVBeanMapping.java
  - CSVFieldMapping.java
  - CSVMappingParser.java
- taglets
  - CSVBeanMappingTaglet.java
  - CSVFieldMappingTaglet.java
  - CSVFormatterMappingTaglet.java
- CSVParser.java
- CSVParserFactory.java (1)
- CSVParserFactoryTest.java
- CSVParserTest.java
- CSVParserTimedTest.java
- CSVReader.java
- CSVReaderTest.java
- test/net/sf/anupam/csv/beans
  - Designation.java
  - Employee.java
- .cvsignore
- README
- build.number
- build.properties
- build.xml

Prefs · Filter · 2m · Mark all comments as read · Keyboard shortcuts

/CSVParser/src/.../csv/CSVParserFactory.java Changed · View · FishEye · File Outdated

1.2 — 1.3 · Leave Unread

```
anupamsg 1.1  114 113  *
                115 132  * @param fieldMapping
                         *          the field for which formatters should be created
anupamsg 1.3      133    * @param fieldMapping the field for which formatters should be created
                  134    * @throws net.sf.anupam.csv.exceptions.CSVOException
```

Jan Svoboda
This is wrong
Reply · Edit · Delete · Add to favourites · 16:55

I dissagree

Quote source

Comment · Keep as draft · Discard · Autosaved at 4:56 PM

```
                  135    *          thrown if the specified formatters cannot be created
anupamsg 1.3  116 136    */
anupamsg 1.1  117        private void createFormattersFor(final CSVFieldMapping fieldMapping) {
              118            final CSVFormatterFactory formatterFactory = CSVFormatterFactory
              119                    .getSingleton();
anupamsg 1.3      137    private void createFormattersFor(final CSVFieldMapping fieldMapping)
                  138            throws CSVOException {
anupamsg 1.3  120 139
anupamsg 1.1  121        final CSVFieldFormatter formatter = formatterFactory
anupamsg 1.3      140    final CSVFieldFormatter formatter = FORMATTER_FACTORY
anupamsg 1.3  122 141            .createFormatterFor(fieldMapping.getReformatterName());
              123 142    fieldMapping.setFormatter(formatter);
              124 143

anupamsg 1.3  128 147    * Resolves bean references for the specified field, and sets the bean
              129 148    * mapping hierarchy accordingly.
```

Pitfalls of MCR

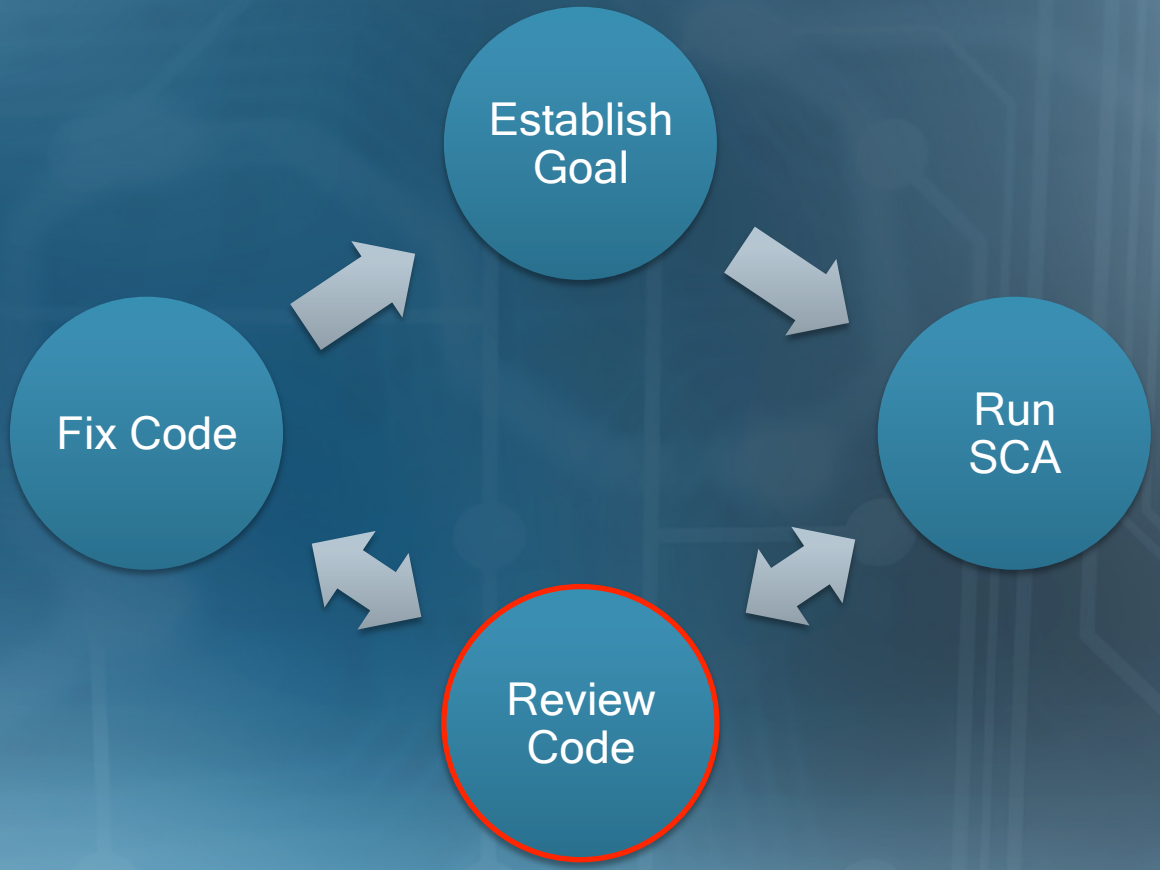
- Human factor
  - Communication skill is one of the most important ones for MCR
  - Honest feedback is foundation stone of each successful review
  - Reviewer perspective
    - Might leave out/soften some of the findings in order not to offend author
    - Might use improper language and offend author
  - Author perspective
    - Might feel confrontated in case of many findings
    - Softening/leaving out findings ruins education benefit

Pitfalls of MCR

- Review of complex code
  - A reviewer needs to study code in more depth to understand it
  - Often help of the author is needed
  - Time consuming
  - The reviewer might tend to check only common and obvious mistakes

Relation between SCA and MCR

- MCR is natural part of SCA



Conclusion

- MCR are very effective if done properly
  - Choose proper review method
  - Establish goal of the review
  - Be honest
  - Use proper and polite language
  - Never be personal

Thank You

Q&A