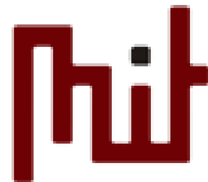
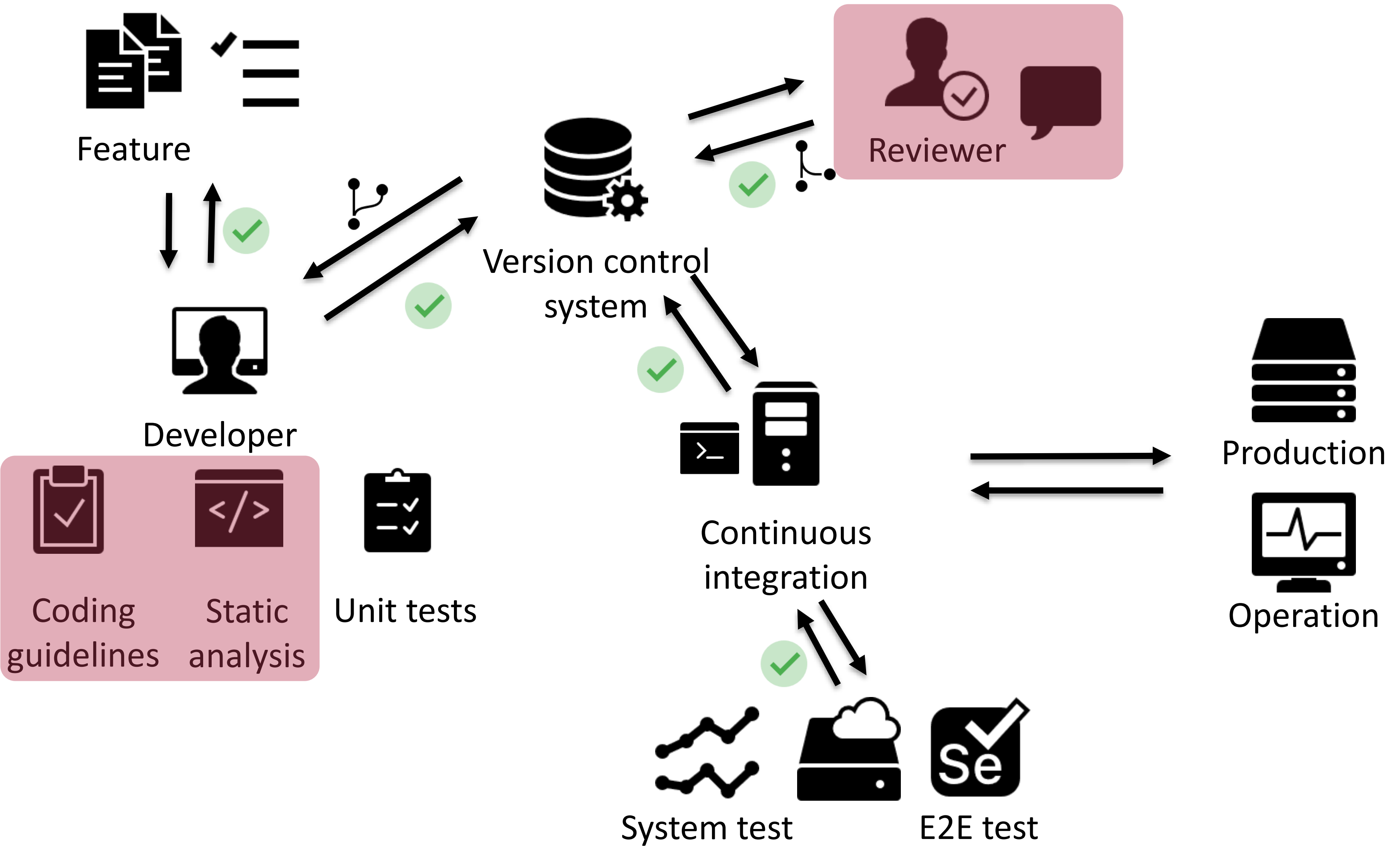
Static Verification Techniques

Ákos Hajdu,
Zoltán Micskei, István Majzik

Department of Measurement and Information Systems



Overview



Icons: icons8.com

Introduction

- Static verification techniques
  - Analyze software without execution

- Advantage: can be performed even if
  - The software is not executable
  - Execution is expensive
  - Input is not yet available

Motivation – Bad example

```
1 public class Class1
2 {
3   public decimal Calculate(decimal amount, int type, int years) {
4      decimal result = 0;
5    decimal disc = (years > 5) ? (decimal)5/100 : (decimal)years/100;
6     if (type == 1) result = amount;
7     else if (type == 2)
8     {
9       result = (amount - (0.1m * amount)) - disc * (amount - (0.1m * amount));
10    }
11    else if (type == 3) { result = (0.7m * amount) - disc * (0.7m * amount); }
12    else if (type == 4) {
13      result = (amount - (0.5m * amount)) - disc * (amount - (0.5m * amount));
14    }
15    return result;
16  }
17 }
```

http://www.codeproject.com/Articles/1083348/Csharp-BAD-PRACTICES-Learn-how-to-make-a-good-code

Properties of a good source code

Syntactically correct
- Checked by compiler

Good quality
- Readable, reusable maintainable, …
- **Coding guidelines** help

Free of bugs
- **Static analysis**, testing, …

Adheres to specification
- **Code review**, testing, …

CODING GUIDELINES

Coding guidelines – Introduction

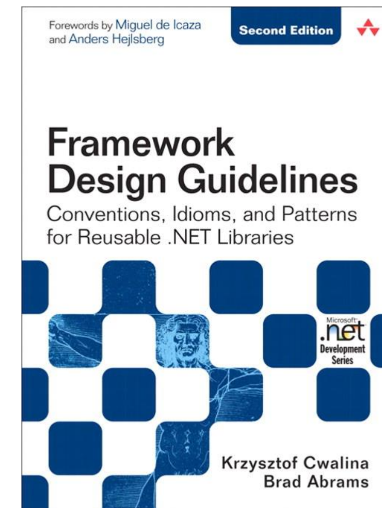
- Set of rules giving recommendations on
  - Style: formatting, naming, structure
  - Programming practices: constructs, architecture
- Main categories
  - Industry/domain specific
    - Automotive, railway, …
  - Platform specific
    - C, C++, C#, Java, …
  - Organization specific
    - Google, CERN, …

Industry specific: MISRA C

- Motor Industry Software Reliability Association
- Focus on safety, security, reliability, portability
- 143 rules + 16 directives
- Tools: SonarQube, Coverity, …
- Examples
  - *RHS of && and || operators shall not contain side effects*
  - *Test against zero should be made explicit for non-Booleans*
  - *Body of* `if`*,* `else`*,* `while`*,* `do`*,* `for` *shall always be enclosed in braces*

Platform specific: .NET

- Framework Design Guidelines (C#)
  - Focus on framework and API development
- Categories
  - Naming, type design, member design, extensibility, exceptions, usage, common design patterns
  - „Do”, „Consider”, „Avoid”, „Do not”
- Tool: StyleCop



https://msdn.microsoft.com/en-us/library/ms229042(v=vs.110).aspx

Platform specific: .NET

- Examples
  - ***DO NOT*** *provide abstractions unless they are tested by developing several concrete implementations and APIs consuming the abstractions.*
  - ***CONSIDER*** *making base classes abstract even if they don’t contain any abstract members. This clearly communicates to the users that the class is designed solely to be inherited from.*
  - ***DO*** *use the same name for constructor parameters and a property if the constructor parameters are used to simply set the property.*

https://msdn.microsoft.com/en-us/library/ms229042(v=vs.110).aspx

Organization specific: Google

- Java Style Guide
- Focus on hard-and-fast rules, avoids advices
- Categories
  - Source file basics
  - Source file structure
  - Formatting
  - Naming
  - Programming practices
  - Javadoc

https://google.github.io/styleguide/javaguide.html

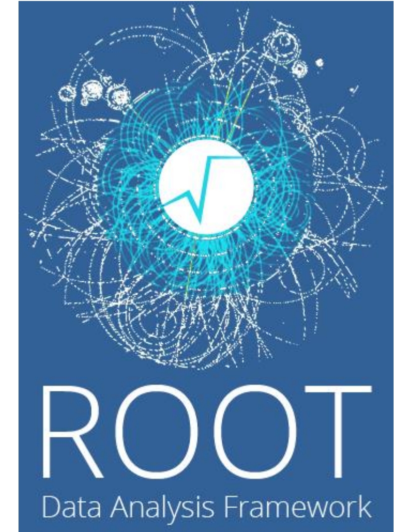
Organization specific: Google

- Examples
  - *Never make your code less readable simply out of fear that some programs might not handle non-ASCII characters properly. If that should happen, those programs are broken and they must be fixed.*
  - *In Google Style special prefixes or suffixes, like those seen in the examples* `name_`, `mName`, `s_name` *and* `kName`, *are not used.*
  - *When a reference to a static class member must be qualified, it is qualified with that class's name, not with a reference or expression of that class's type.*
  - *Local variable names are written in* `lowerCamelCase`.

https://google.github.io/styleguide/javaguide.html

Organization specific: CERN

- ROOT: data analysis tool/framework for high energy physics (C++)
- Categories
  - Naming
  - Exceptions
  - Namespaces
  - Comments
  - Source layout
- Tool: Artistic Style (astyle)



https://root.cern/coding-conventions

Organization specific: CERN

- Examples
  - *Avoid the use of raw C types like* `int`, `long`, `float`, `double` *when using data that might be written to disk.*
  - *For naming conventions we follow the Taligent rules. Types begin with a capital letter (*`Boolean`*), base classes begin with „T" (*`TContainerView`*), members begin with „f" (*`fViewList`*), …*
  - *Each header file has the following layout: Module identification line, Author line, Copyright notice, Multiple inclusion protection macro, Headers file includes, Forward declarations, Actual class definition.*

https://root.cern/coding-conventions

Coding guidelines – Summary

- How to enforce
  - Base functionality in many IDEs
  - External tools
  - Tool integrated in the workflow

- Important
  - Always use a common guideline
  - As a minimum, common IDE formatter settings
    - Can usually be committed to version control as a settings file

Coding guidelines – Summary

- Which one is the best? Which one to select?
- In many cases it is already determined
  - By the industry, platform or organization
  - Consistency with the current code base
- Sometimes it can be determined
  - There may be no single best one
    - They can be even inconsistent with each other
    - Combination is possible
  - Do not reinvent the wheel
    - Makes it harder for new developers

CODE REVIEW

Code review – Introduction

- Manual process performed by humans
  - Reading, examining, reviewing the code
  - Usually based on a structured checklist
- Different levels (informal → formal)

Level	Characteristics
Informal review	• Informal • Performed by other team members or team lead
Walkthrough	• Mostly informal • Guided by the author of the code
Technical review	• Well defined, documented process • Including experts
Inspection	• Formally defined, documented process • Including external experts, moderators

http://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html

Code review process

Phase	Activities
Planning	• Specifying documents, participants and criteria • Distributing tasks
Kick-off	• Introducing the process to participants • Getting the code to the reviewer
Preparation	• Reviewing the code • Documenting problems
Review meeting	• Discussing and documenting problems • Suggestions for fixes
Rework	• Performing the fixes • Documenting modifications
Follow up	• Checking fixes • Checking exit criteria

Code review – Advantages

- Formal inspection
  - Effective in finding errors
  - Time consuming, tiresome work
- Modern techniques
  - Less formal, more tool support
  - Used in the industry (Microsoft, Google, Facebook, …)
  - Other advantages besides finding errors
    - Knowledge transfer
    - Team spirit
    - Alternative solutions

http://dl.acm.org/citation.cfm?id=2486882

Code review – Checklist

- Checklist: structured enumeration of criteria
- Similar categories as in coding guidelines
  - Readability, maintainability
  - Security, vulnerability
  - Performance
  - Programming patterns and practices
- Advices
  - Many code review checklists can be found online
  - Strive for automation
    - E.g., formatting can be checked by a tool

Code review – Tools

- Supporting code review
  - Attach notes and conversations to code
  - Integrated into development workflow
- GitHub: pull request reviews (→ LAB)
  - Comments, accepting, requesting changes

octocat requested changes 28 days ago

View changes

This is looking ✨! I've left a few comments that should be addressed before this gets merged. 😺

```
data/reusables/open-source.yml
...   ...   @@ -0,0 +1,5 @@
      1     +open-source-handbook-repositories: |
      2     +  For more information on open source, specifically how to create and grow an ope
```

octocat 28 days ago

"provide best practices relating to creating repositories for your open source project."

Unified | Split | Review changes 3

Submit your 3 pending comments

Review summary

This is looking ✨! I've left a few comments that should be addressed before this gets merged. 😺

Comment
Submit general feedback without explicit approval.

Approve
Submit feedback and approve merging these changes.

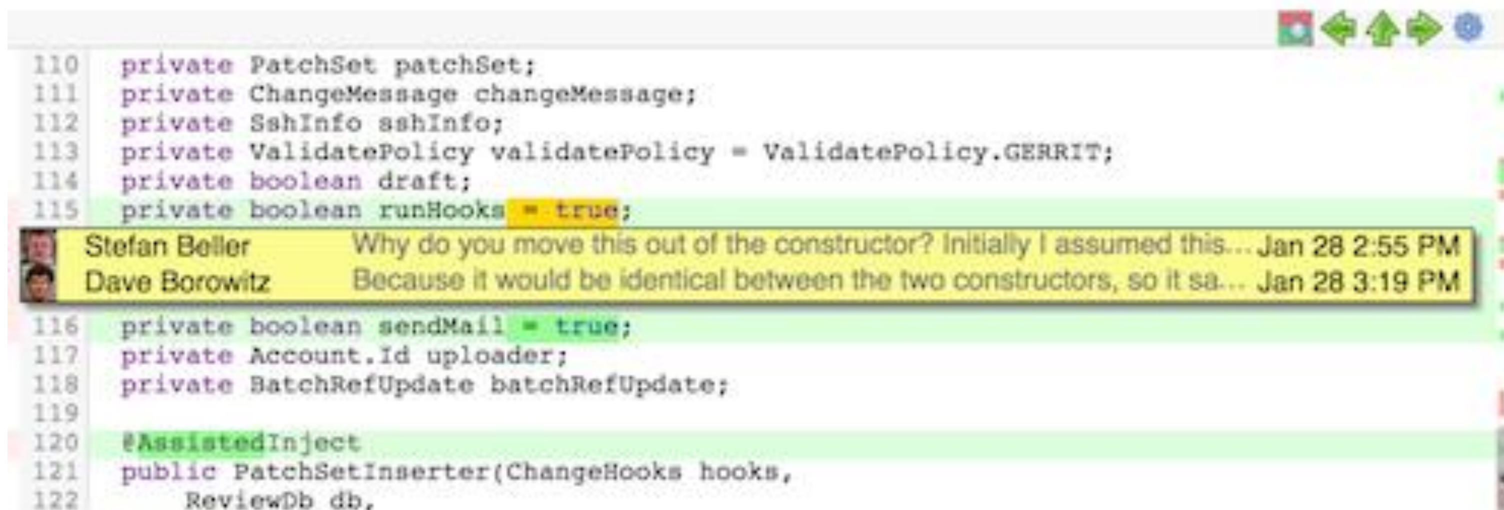
Request changes
Submit feedback that must be addressed before merging.

Submit review

https://help.github.com/articles/about-pull-request-reviews/

Code review – Tools

- Gerrit
  - Web-based code review
  - Git support
  - Managing workflow



https://www.gerritcodereview.com/

STATIC ANALYSIS

Static analysis – Example

```
1   public class Sample {
2       public static void main(String[] args) {
3           String str = null;
4           try {
5               Scanner scanner = new Scanner("file.txt");
6               str = scanner.nextLine();
7               scanner.close();
8           } catch (Exception e) {
9               System.out.println("Error opening file!");
10          }
11          str.replace(" ", "");
12          System.out.println(str);
13      }
14  }
```

Scanner not closed in case of exception

str may be null

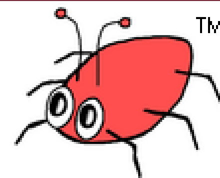
str immutable

Static analysis – Introduction

- Definition: analysis of software without execution
  - Usually automated tools
  - Human analysis (code review)

- Pattern-based
  - Basic static properties with error patterns (mostly)
    - E.g., ignored return value, unused variable
  - FindBugs, SonarQube, Coverity
- Interpretation-based
  - Dynamic properties
    - E.g., null pointer dereference, index out of bounds
  - Infer, PolySpace

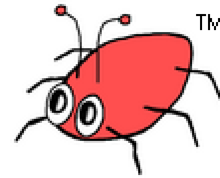
FindBugs (Java)



- Large and extensible set of rules
- Command line, GUI, Eclipse plug-in
- Examples
  - Bad practice: *random object created and used only once*
  - Correctness: *bitwise add of signed byte value*
  - Vulnerability: *expose inner static state by storing mutable object into a static field*
  - Multithreading: *synchronization on Boolean could lead to deadlock*
  - Performance: *invoke toString() on a string*
  - Security: *hardcoded constant database password*
  - Dodgy: *useless assignment in return statement*

http://findbugs.sourceforge.net/

FindBugs (Java)



FindBugs:

File Edit Navigation Designation Help

Package | Priority | Category | Bug Kind | Bug Pattern

- edu.umd.cs.findbugs.config (3)
- edu.umd.cs.findbugs.filter (1)
- edu.umd.cs.findbugs.util (1)
  - Medium (1)
    - Bad practice (1)
      - Stream not closed on all paths (1)
        - Method may fail to close stream (1)
          - edu.umd.cs.findbugs.util.Util.getXMLT
- edu.umd.cs.findbugs.visitclass (1)
- edu.umd.cs.findbugs.workflow (2)
- java.util (2)

unclassified

Util.java in edu.umd.cs.findbugs.util

```
 97                 assert true;
 98             }
 99         }
100     static final Pattern tag = Pattern.compile("^\\s*<(\\w+)"
101     public static String getXMLType(InputStream in) throws IOI
102         if (!in.markSupported())
103             throw new IllegalArgumentException("Input stream
104
105         in.mark(5000);
106         BufferedReader r = null;
107         try {
108             r = new BufferedReader(Util.getReader(in), 2000);
109
110             String s;
111             int count = 0;
112             while (count < 4) {
113                 s = r.readLine();
114                 if (s == null)
115                     break;
116                 Matcher m = tag.matcher(s);
```

Find | Find Next | Find Previous

edu.umd.cs.findbugs.util.Util.getXMLType(InputStream) may fail to close stream
At Util.java:[line 108]
In method edu.umd.cs.findbugs.util.Util.getXMLType(InputStream) [Lines 102 - 123]
Need to close java.io.Reader

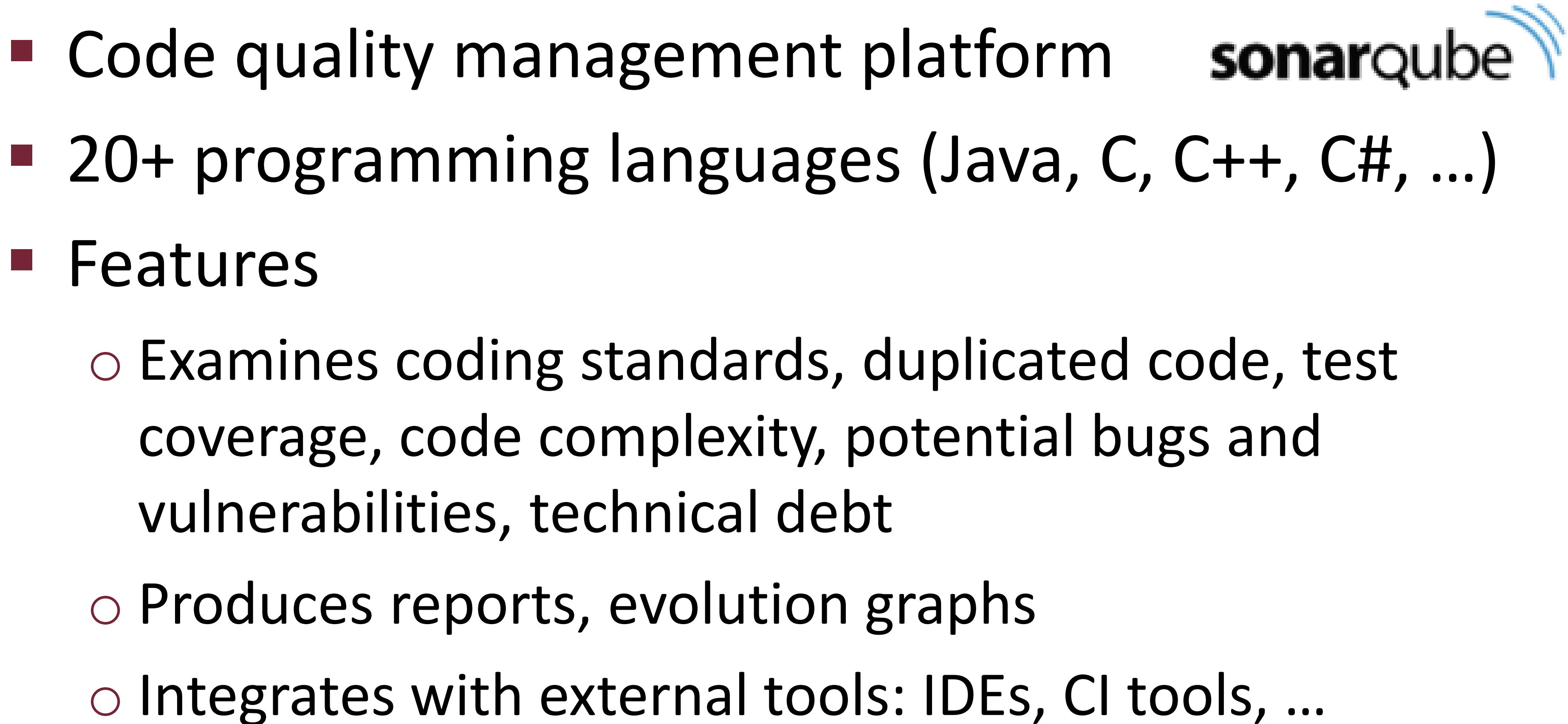
Method may fail to close stream

The method creates an IO stream object, does not assign it to any fields, pass it to other methods that might close it, or return it, and does not appear to close the stream on all paths out of the method. This may result in a file descriptor leak. It is generally a good idea to use a `finally` block to ensure that streams are closed.

http://findbugs.sourceforge.net/

UNIVERSITY OF MARYLAND

SonarQube



http://www.sonarqube.org/

SonarQube

Bugs & Vulnerabilities

118 E
Bugs

63 E
Vulnerabilities

Leak Period: last 30 days
started a month ago

4
New Bugs

4
New Vulnerabilities

Code Smells

12k A
Code Smells

269d
Debt

started 7 years ago

664
New Code Smells

18d
New Debt

Coverage

88.1%
Coverage

9.3k
Unit Tests

90.6%
Coverage on
1.6k New Lines of Code

SonarQube

SonarQube

Issues | Measures | Code | Dashboards

Issues | Effort

Type

Bug	118
Vulnerability	63
Code Smell	12k

Resolution

Unresolved	118	Fixed	4
False Positive	0	Won't fix	0
Removed	41		

Severity

Status

SonarQube · SonarQube :: Plugin API · src/main/ja

Override this superclass' "equals" method.

Bug · Major · Open · Not assigned · 30min effort

SonarQube · SonarQube :: Plugin API · src/main/ja

The return value of "parseDouble" must be used.

Bug · Critical · Open · Not assigned · 10min effort

SonarQube · SonarQube :: Plugin API · src/main/ja

NullPointerException might be thrown as 'value' is nullable h

Bug · Blocker · Open · Simon Brandhof · 10min effo

Coverity

- Static analyzer of the Synopsys suite
- C, C++, C#, Java, JavaScript
- Used by CERN, NASA, …
- Examples: resource leaks, null pointers, uninitialized data, concurrency issues, …
- Coverity Scan: free service for open source projects
  - Integrated with GitHub and Travis CI

http://www.synopsys.com/software/coverity/Pages/default.aspx https://scan.coverity.com/

Using static analysis tools efficiently

- Integrate to build process
  - Perform check before/after each commit
  - Generate reports, send e-mails
- Use from the start of a project
  - Too many problems would discourage developers
- Configure the tools
  - Filter based on severity or category
  - Add custom rules

Using static analysis tools efficiently

- Review the results carefully
  - False positives and false negatives are possible
- False negative
  - No errors found does not mean correct software
- False positive
  - An error found may not cause a real failure
  - Ignore rule / one occurrence
    - Always explain why it is not an error

Advantages of static analysis

- Analyzing software without execution
  - Analysis before software is executable or input is present
  - Execution may be expensive
- Find subtle errors
  - Interesting even for expert programmers
- Automatic process
  - Integrated into development process

Static verification techniques – Summary

- Coding guidelines
  - Industry, platform, organization specific
- Code review
  - Manual inspection based on checklist
- Static analysis tools
  - Code analysis without execution