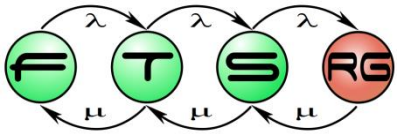
Verifying source code

Akos Hajdu, Istvan Majzik, Zoltan Micskei

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group



Main topics of the course

- Overview (1)
  - V&V techniques, Critical systems
- **Static techniques (2)**
  - Verifying specifications
  - **Verifying source code**
- Dynamic techniques: Testing (7)
  - Developer testing, Test design techniques
  - Testing process and levels, Test generation, Automation
- System-level verification (3)
  - Verifying architecture, Dependability analysis
  - Runtime verification

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 — Compiler
- Good quality — **Coding guidelines**
  - Readable, reusable, maintainable
- Free of bugs — **Static analysis**, testing
- Adheres to the specification — **Code review**, testing

CODING GUIDELINES

Learning outcomes

- List some domain, platform and organization specific coding guidelines and some of their typical categories and elements (K1)

Coding guidelines – introduction

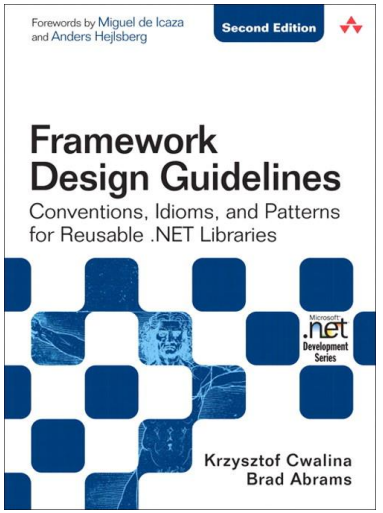
- Set of rules giving recommendations on
  - Style: formatting, naming, structure
  - Programming practices: constructs, architecture
- Main categories
  - Industry/domain specific
    - Automotive, railway, …
  - Platform specific
    - C, C++, C#, Java, …
  - Organization specific
    - Google, CERN,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

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

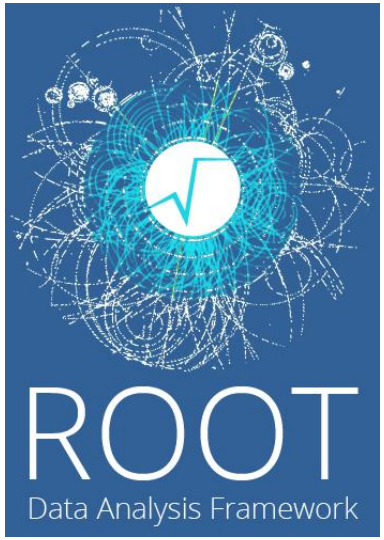
Organization specific: Google

- 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

Learning outcomes

- Apply manual code review on a small unit of code (~50-100 LOC) using common review criteria (K3)

Manual code review

- Performed by humans
  - Typically other team members
  - Usually based on some structured checklist
  - Similar to review techniques for specification (prev. lecture)

- Different level of formalization

Code inspection  **“Modern” code review**

Advantages of code reviews

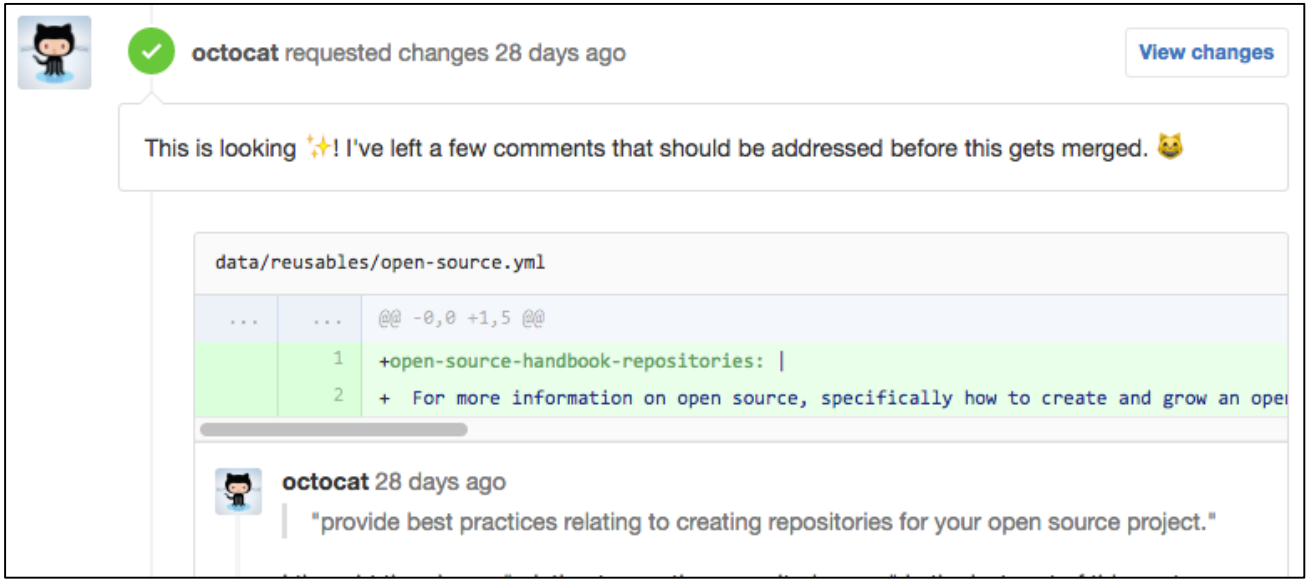
- Code inspection
  - Effective for finding bugs
  - Resource-intensive
- "Modern" code review
  - More informal, good tool support
  - Widespread in industry (MS, Google, FB, …)
  - Further benefits
    - Code understanding
    - Team awareness
    - Change management

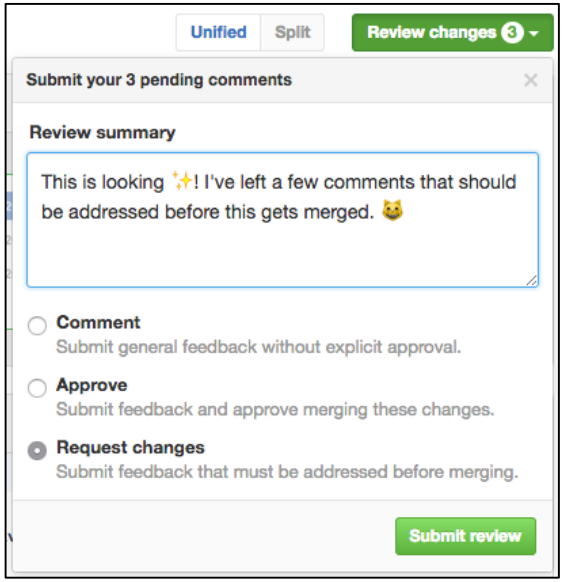
Expectations, outcomes, and challenges of modern code review, ICSE 2013

Tool support

- Supporting code review
  - Discussion, change requests
  - Integrated into the development and CI workflow

- GitHub: pull request reviews (→ Lab)





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

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

Learning outcomes

- List some bugs that can be detected with static analysis (K1)
- Use a static analysis tool to find bugs and mistakes in a non-trivial code base (~100-1000 LOC) (K3)

Static analysis – introduction

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

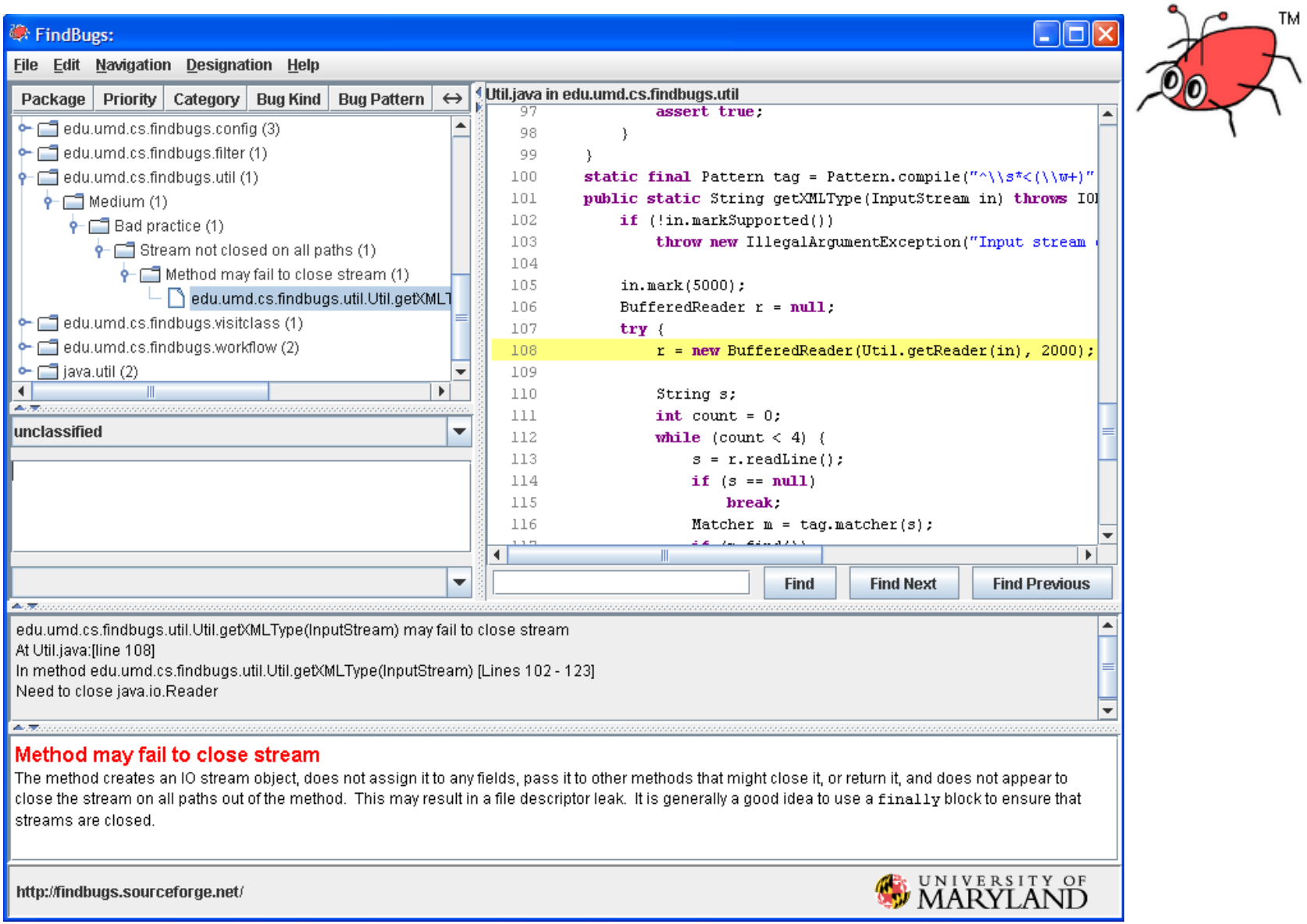
- Pattern-based
  - Basic static properties with error patterns (mostly)
    - E.g., ignored return value, unused variable
  - FindBugs, SonarQube, Coverity
- Interpretation-based
  - Dynamic properties
    - E.g., null pointer dereference, index out of bounds
  - Infer, PolySpace

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)



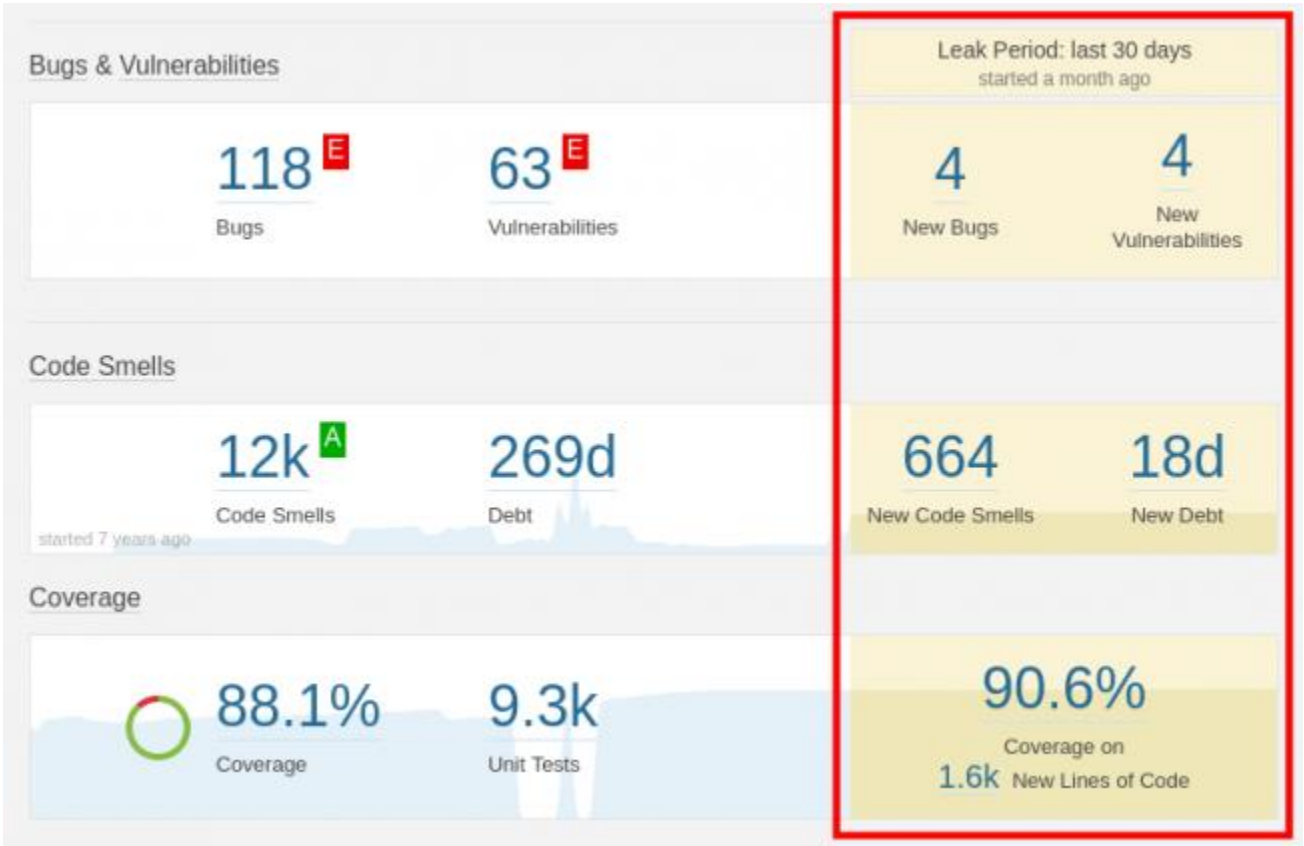
SonarQube

- Code quality management platform

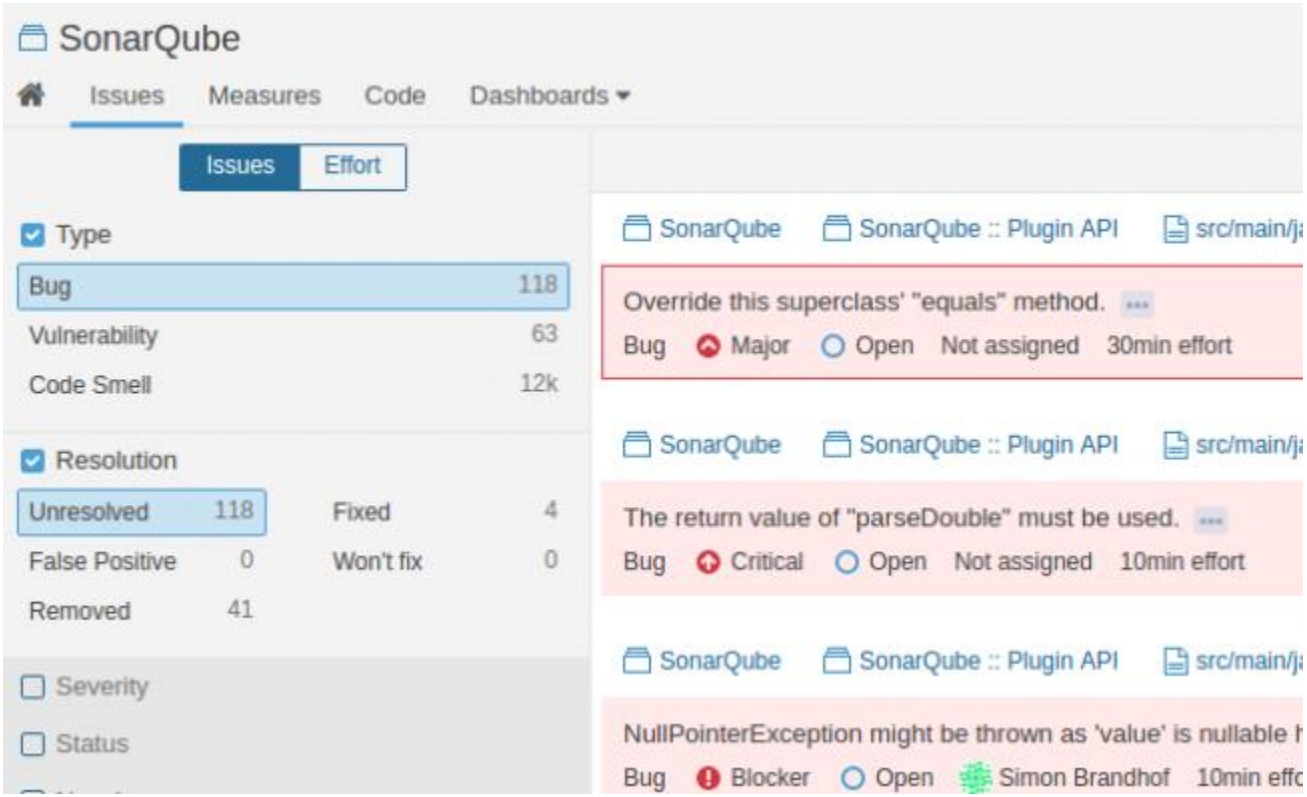
- 20+ programming languages (Java, C, C++, C#, …)
- Features
  - Examines coding standards, duplicated code, test coverage, code complexity, potential bugs and vulnerabilities, technical debt
  - Produces reports, evolution graphs
  - Integrates with external tools: IDEs, CI tools, …

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 positive (false alarm)
  - An error found may not cause a real failure
  - Ignore rule / one occurrence
    - Always explain why it is not an error
- False negative
  - No errors found does not mean correct software

Confusing terms! Define full confusion matrix!

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

More information

- J. Carmack. In-Depth: Static Code Analysis
  - “**it is irresponsible to not use it**”
  - “there was an epic multi-programmer, multi-day bug hunt that wound up being traced to something that /analyze had flagged, but I hadn't fixed yet.”

- A few Billion Lines of code Later using static Analysis to find Bugs in the Real World

coverity®

  - Turning a prototype into commercial tool
  - “False positives do matter. In our experience, more than 30% easily cause problems. People ignore the tool. True bugs get lost in the false.”

DYNAMIC PROPERTIES WITH STATIC ANALYSIS

Learning outcomes

- Explain the main concept of abstract interpretation (K2)

Dynamic properties

- Motivation: detect **run time** failures without executing the software
  - Examples: null pointer, index out of bounds, uninitialized data, arithmetic error, overflow, dead code,…
- Can be performed by control-flow and data-flow analysis
  - Calculate interval for each variable
  - Propagate intervals based on control-flow

Example

```
0: k=ioread32();
1: i=2;
2: j=k+5;
3: while (i<10) {
4:       i=i+1;
5:       j=j+3;
6: }
7: // end of loop
8: k=k/(i-j);
```

Division by zero?
For what *k*?

Example

Possible (i,j,k) values

- $X0=\{(0,0,k) \mid k \in [-2^{31}, 2^{31}-1]\}$
- $X1=\{(2,j,k) \mid (i,j,k) \in X0\}$
- $X2=\{(i,k+5,k) \mid (i,j,k) \in X1\}$
- $X3= X2 \cup X6$
- $X4=\{(i+1,j,k) \mid (i,j,k) \in X3, i<10\}$
- $X5=\{(i,j+3,k) \mid (i,j,k) \in X4\}$
- $X6= X5$
- $X7=\{(i,j,k) \mid (i,j,k) \in X3, i=10\}$
- $X8=\{(i,j,k/(i-j)) \mid (i,j,k) \in X7\}$

```
0: k=ioread32();
1: i=2;
2: j=k+5;
3: while (i<10) {
4:      i=i+1;
5:      j=j+3;
6: }
7: // end of loop
8: k=k/(i-j);
```

Example

- $X0=\{(0,0,k) \mid k \in [-2^{31}, 2^{31}-1]\}$

  $X0=\{(0,0,k) \mid k \in [-2^{31}, 2^{31}-1]\}$

- $X1=\{(2,j,k) \mid (i,j,k) \in X0\}$

  $X1=\{(2,0,k) \mid k \in [-2^{31}, 2^{31}-1]\}$

- $X2=\{(i,k+5,k) \mid (i,j,k) \in X1\}$

  $X2=\{(2,k+5,k) \mid k \in [-2^{31}, 2^{31}-1]\}$

- $X3 = X2 \cup X6$

  $X3=\{(i,j,k) \mid k \in [-2^{31}, 2^{31}-1], i \in [2,10], j=k+3i-1\}$

- $X4=\{(i+1,j,k) \mid (i,j,k) \in X3, i<10\}$

  $X4=\{(i,j,k) \mid k \in [-2^{31}, 2^{31}-1], i \in [3,10], j=k+3i-4\}$

```
0: k=ioread32();
1: i=2;
2: j=k+5;
3: while (i<10) {
4:       i=i+1;
5:       j=j+3;
6: }
7: // end of loop
8: k=k/(i-j);
```

Loop invariant: $j=k+5+3(i-2)$

Example

- $X5=\{(i,j+3,k) \mid (i,j,k) \in X4\}$

  $X5=\{(i,j,k) \mid k \in [-2^{31}, 2^{31}-1], i \in [3,10], j=k+3i-1\}$

- $X6= X5$

  $X6=X5$

- $X7=\{(i,j,k) \mid (i,j,k) \in X3, i=10\}$

  $X7=\{(10,j,k) \mid k \in [-2^{31}, 2^{31}-1], j=k+29\}$

- $X8=\{(i,j,k/(i-j)) \mid (i,j,k) \in X7\}$

  $X8=\{(10,j,k/(i-j)) \mid k \in [-2^{31}, 2^{31}-1], j=k+29\}$

- Error at X8, if i-j=0

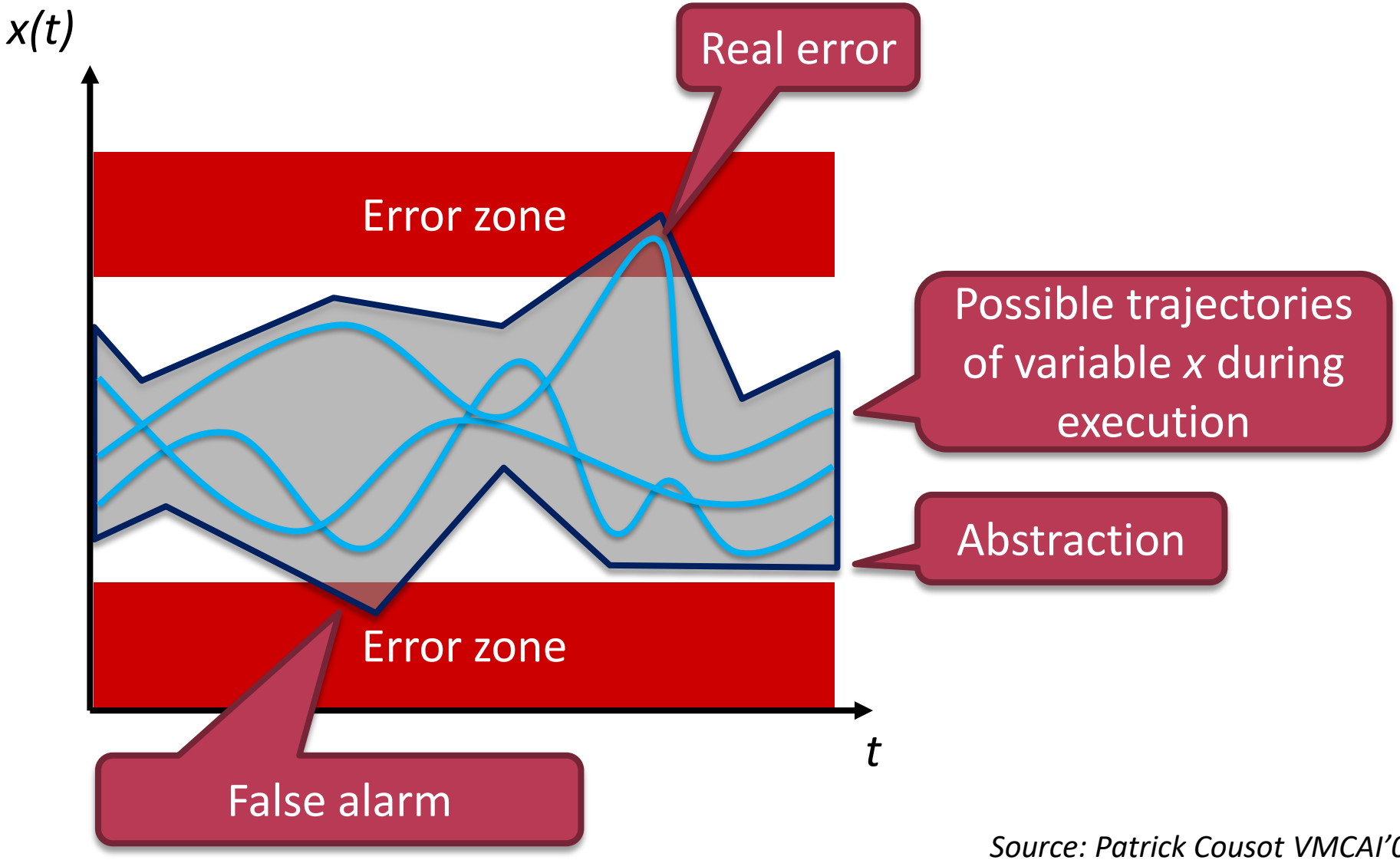
  Since i=10, this can happen if k=-19 → $X8_{error}=\{(10,10,-19)\}$

```
0: k=ioread32();
1: i=2;
2: j=k+5;
3: while (i<10) {
4:      i=i+1;
5:      j=j+3;
6: }
7: // end of loop
8: k=k/(i-j);
```

Analyzing dynamic properties

- Based on analyzing control-flow and data-flow
  - Operations with intervals and constraints
  - Loops: determine loop invariants
- Calculating loop invariants
  - Undecidable in general
  - Approximations are required
- Abstraction: over-approximate intervals
  - All errors are detected
  - False positives (false alarms) are possible
    - Can be treated as a hint for further analysis

Abstraction illustrated



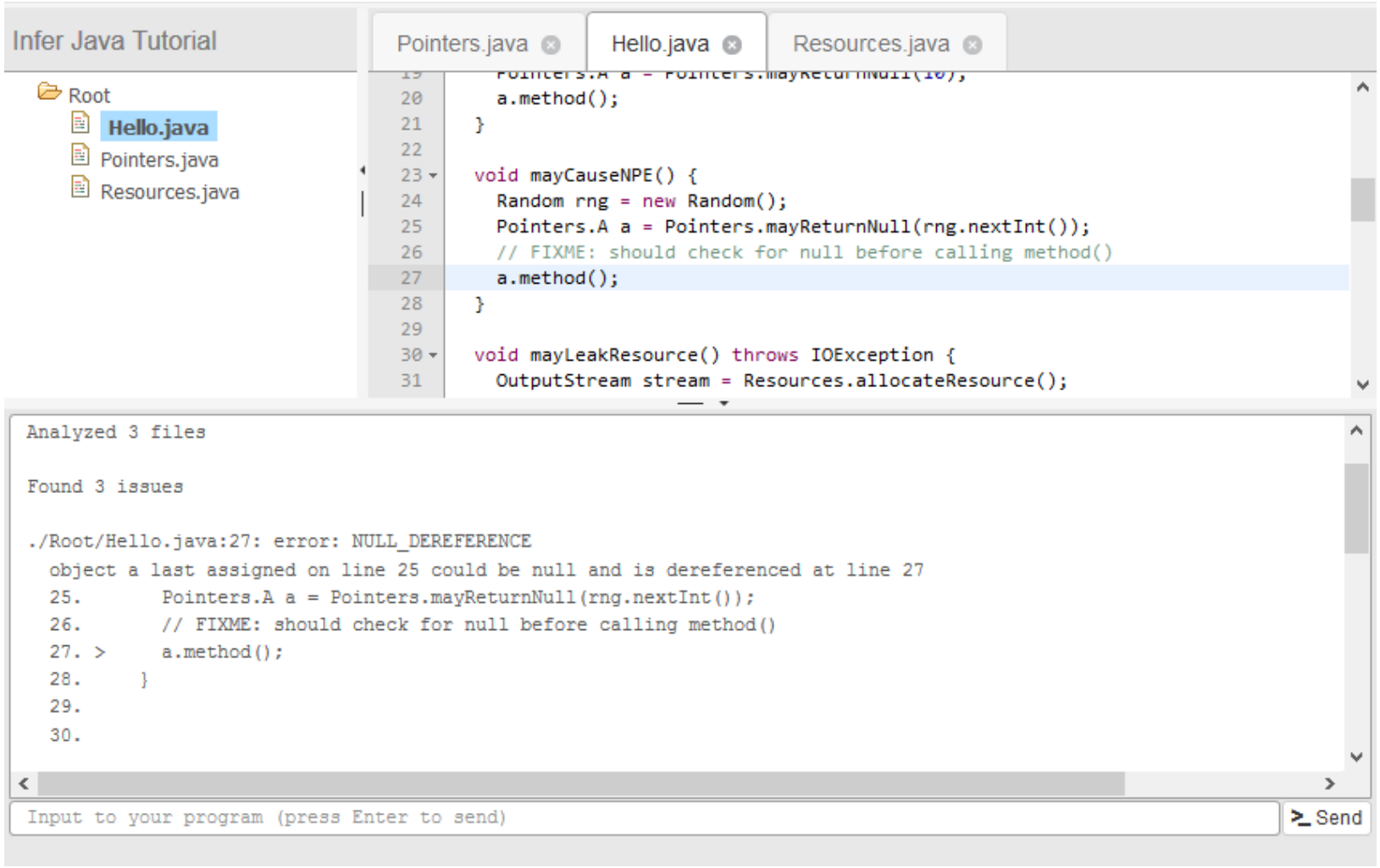
Source: Patrick Cousot VMCAI'05

Infer

- Static analysis tool by Facebook
  - Focus on mobile development
  - Users: Facebook, Instagram, Oculus, Spotify, WhatsApp, …
- Android and Java
  - Null pointers, resource leaks, context leak
- iOS and Objective-C
  - Null pointers, memory leaks, resource leaks

http://fbinfer.com/

Infer


Infer Java Tutorial
Root
Hello.java
Pointers.java
Resources.java
Pointers.java
Hello.java
Resources.java
20 a.method();
21 }
22
23 void mayCauseNPE() {
24 Random rng = new Random();
25 Pointers.A a = Pointers.mayReturnNull(rng.nextInt());
26 // FIXME: should check for null before calling method()
27 a.method();
28 }
29
30 void mayLeakResource() throws IOException {
31 OutputStream stream = Resources.allocateResource();
Analyzed 3 files
Found 3 issues
./Root/Hello.java:27: error: NULL_DEREFERENCE
object a last assigned on line 25 could be null and is dereferenced at line 27
25. Pointers.A a = Pointers.mayReturnNull(rng.nextInt());
26. // FIXME: should check for null before calling method()
27. > a.method();
28. }
29.
30.
Input to your program (press Enter to send)
Send

PolySpace

- Static analysis tool by MathWorks for C/C++
- Bug Finder
  - Run time errors, concurrency issues, vulnerabilities
  - Coding guidelines
- Code prover
  - Can prove absence of overflow, division by zero, index out of bounds
  - Color codes: safe, definite error, unproven, unreachable

http://www.mathworks.com/products/polyspace/

MathWorks®

Verifying source code – summary

- Coding guidelines
  - Industry, platform, organization specific
- Static analysis tools
  - Analyze software without execution
- Dynamic properties with static analysis
  - Abstract interpretation