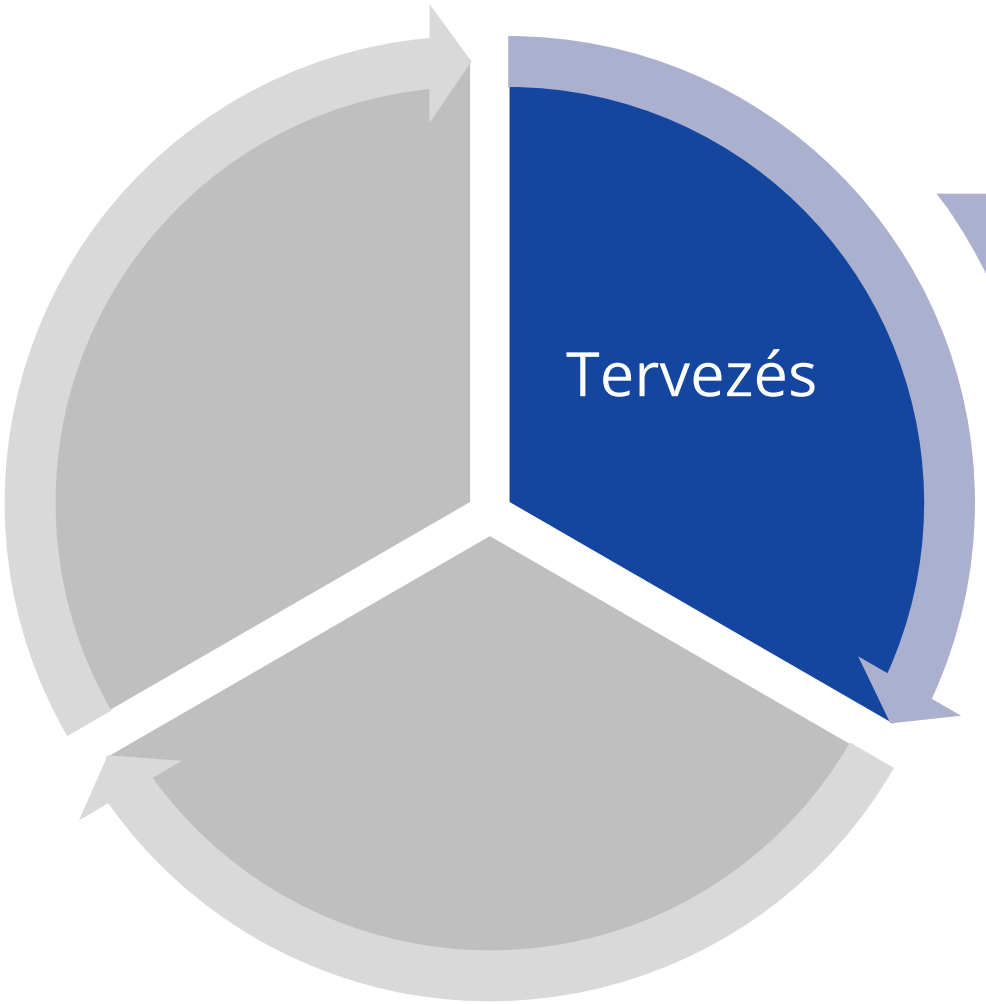
Static Analysis

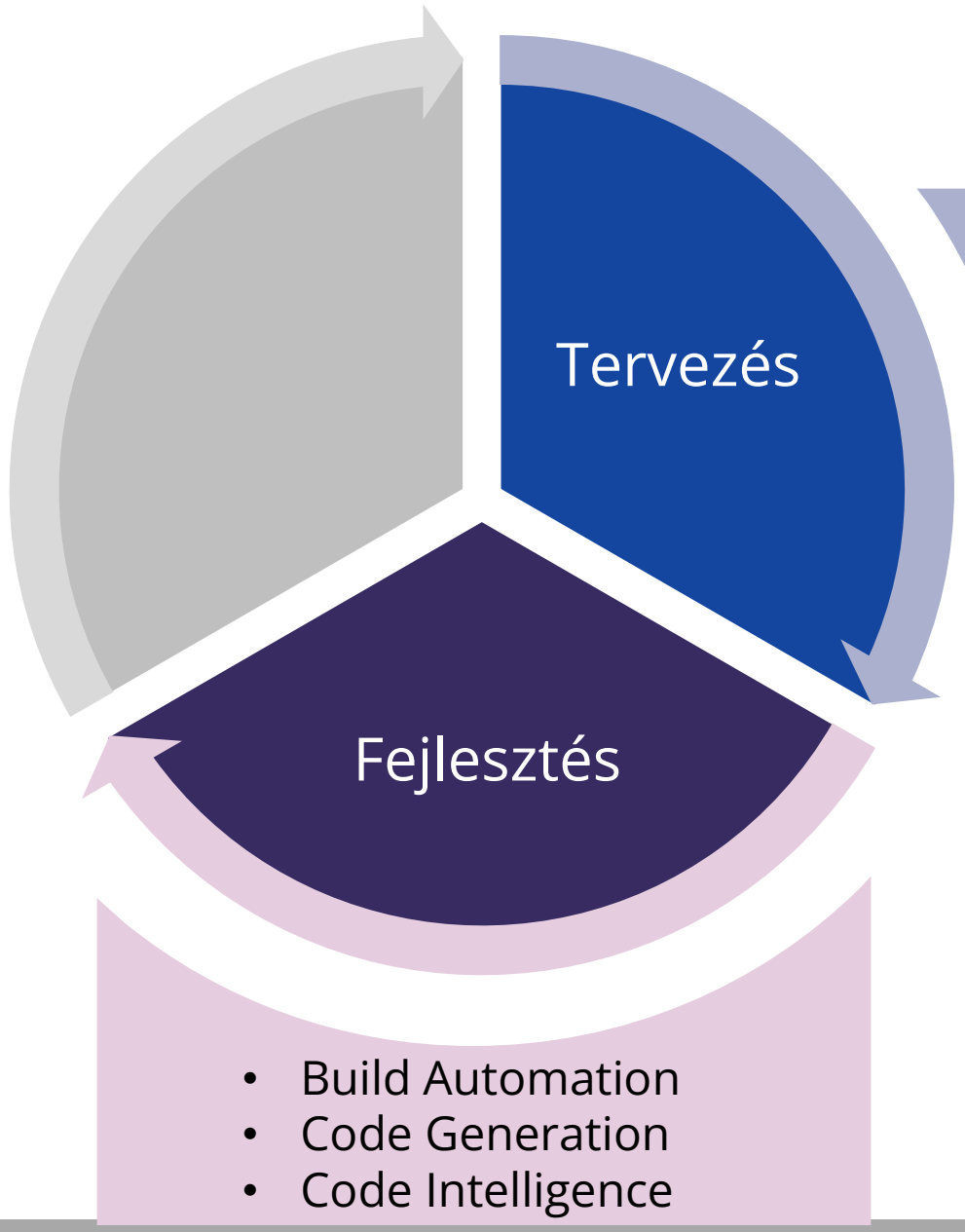
Critical Systems Research Group

Overview



- Graph-Based Modeling
- Textual Modeling
- Code Generation
- Model Intelligence
- Model Checking

Overview

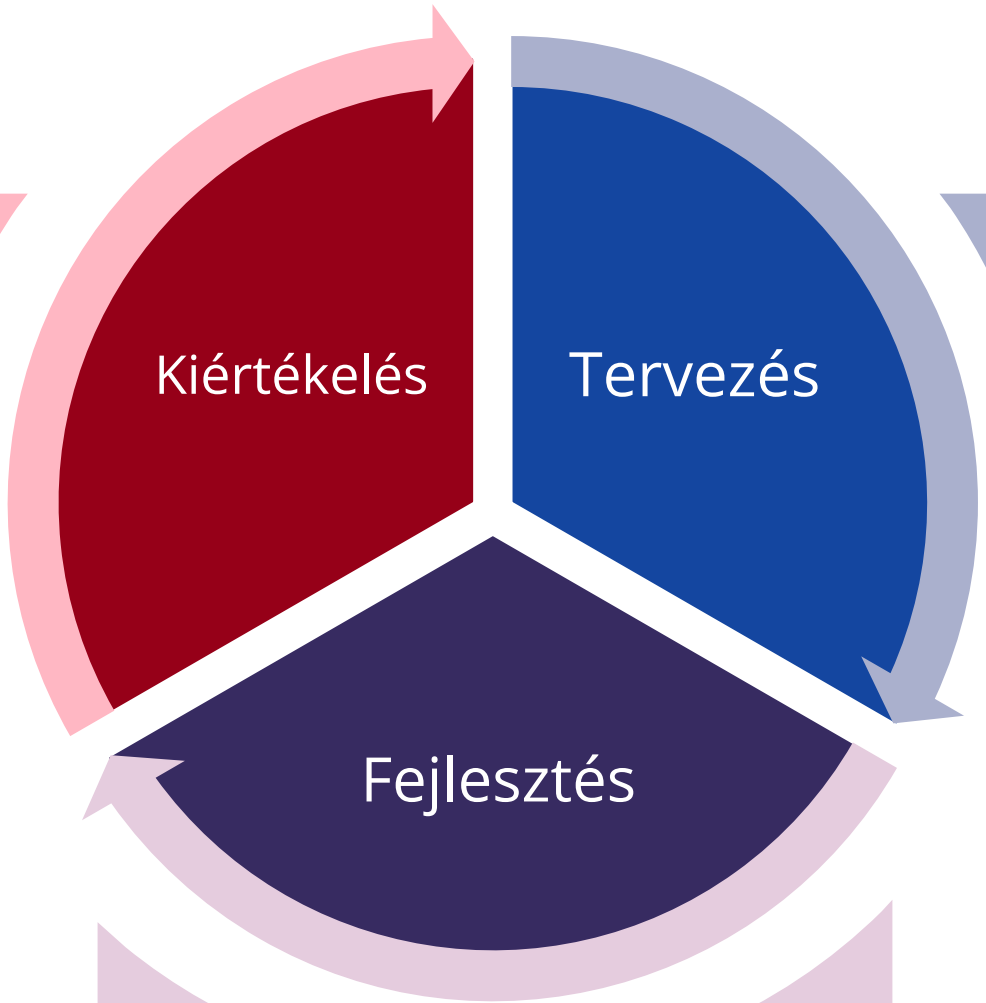

Tervezés
Fejlesztés
• Graph-Based Modeling
• Textual Modeling
• Code Generation
• Model Intelligence
• Model Checking
• Build Automation
• Code Generation
• Code Intelligence

Overview

- Performance evaluation
- Data Analysis
- Code Quality
- Static Analysis
- Testing & Coverage

Next two lesions

- Code Quality
- Static Analysis
- Testing
- Coverage



- Graph-Based Modeling
- Textual Modeling
- Code Generation
- Model Intelligence
- Model Checking

- Build Automation
- Code Generation
- Code Intelligence

Static Verification Techniques

Static verification techniques

- Many ways to define, but most of them agree:
  - Automatic methods to reason
  - about run time properties of the code
  - without executing it

- Goal: find problems that
  - Cannot be detected by a traditional compiler
  - Hard to find by testing

dl.acm.org/citation.cfm?id=139095

Analysis properties

- Checking most of the run time properties is computationally complex or undecidable (see Languages and Automata course)
- Approximations: sacrifise precision to save analysis time

		Analysis	
		Correct	Faulty
Source code	Correct	Proven correct	False alarm
	Faulty	Missed bug	Found bug

- Precision
  - False positive (false alarm): report an error that does not cause a real problem
  - False negative (missed bug): an actual problem does not get reported

Coding Guidelines

Coding guidelines – Introduction

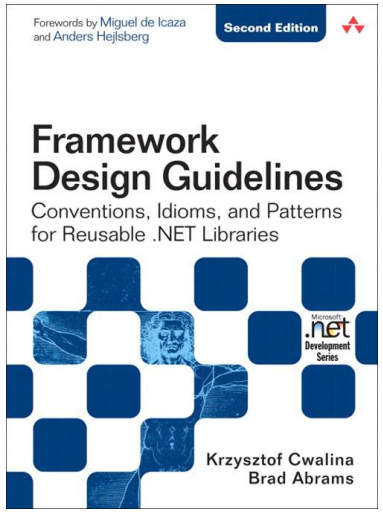
- **Set of rules** giving recommendations on
  - Style: formatting, naming, structure
  - Programming practices: constructs, architecture
- **Main categories**
  - Industry/domain specific
    - Automotive, railway, …
  - Platform specific
    - C, C++, C#, Java, …
  - Organization specific
    - Google,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

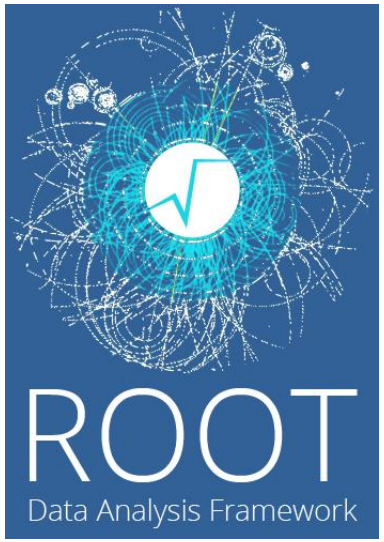
Organization specific: Google

- 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)

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

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

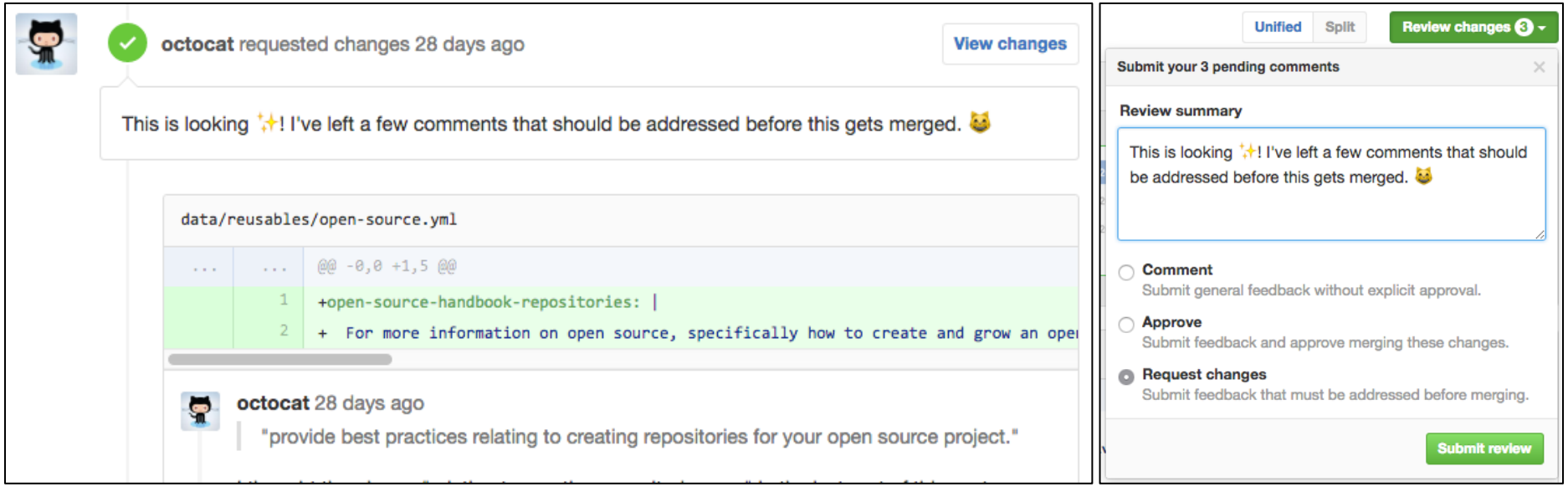
http://dl.acm.org/citation.cfm?id=2486882

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



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!");
                   }
                   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

ErrorProne (Java)

- Google internal development
  - Extensible set of rules
  - Gradle, Maven, Eclipse, IntelliJ, …
- Examples
  - „Reference equality used to compare arrays”
  - „Type declaration annotated with @Immutable is not immutable”
  - „Loop condition is never modified in loop body.”
  - „This conditional expression may evaluate to null, which will result in an NPE when the result is unboxed.”
  - „Comparison of a size >= 0 is always true, did you intend to check for non-emptiness?”

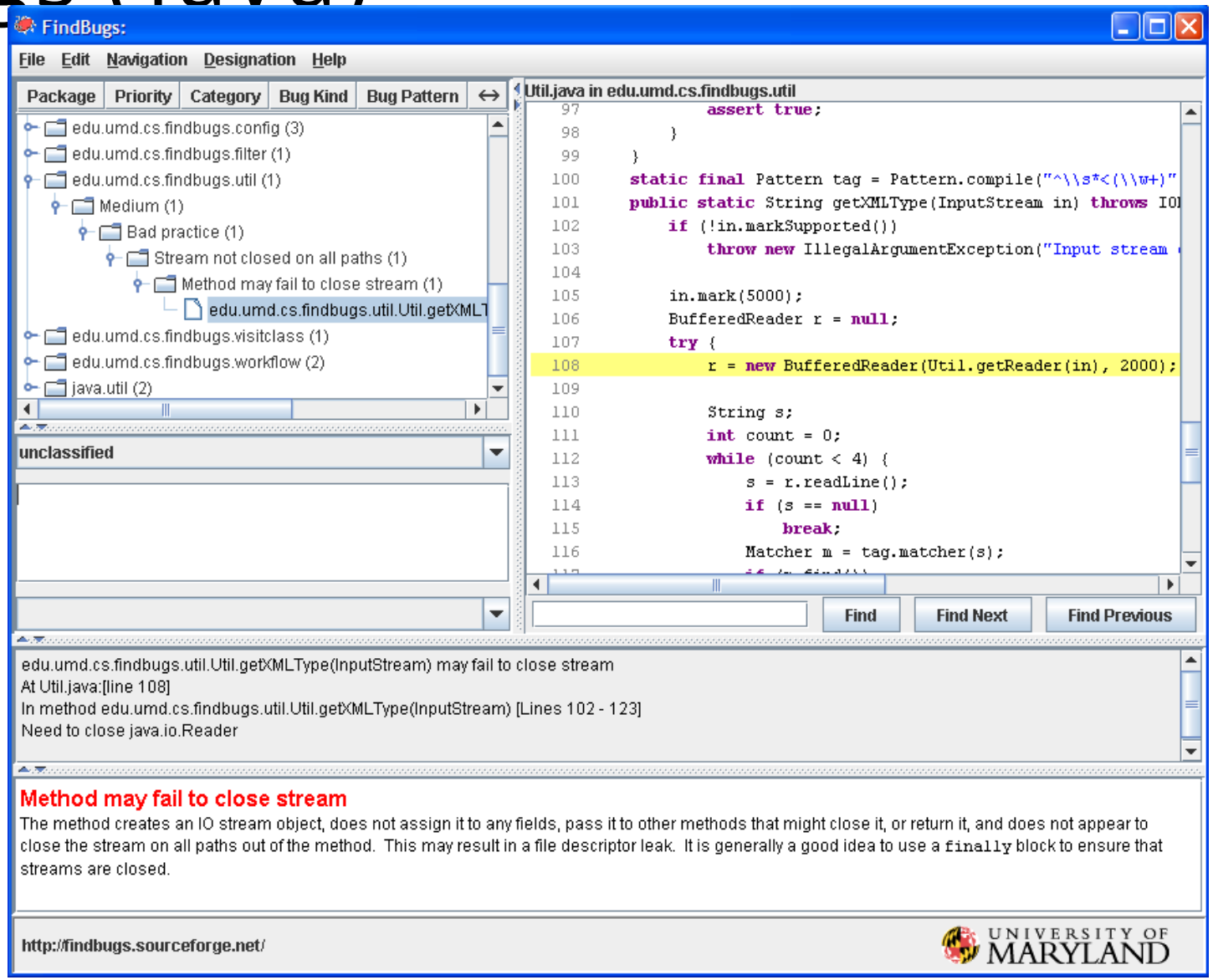
https://errorprone.info/

SpotBugs (Java)

- Large and extensible set of rules
- Command line, GUI, Eclipse/IntelliJ 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*

https://spotbugs.github.io/

SpotBugs (Java)



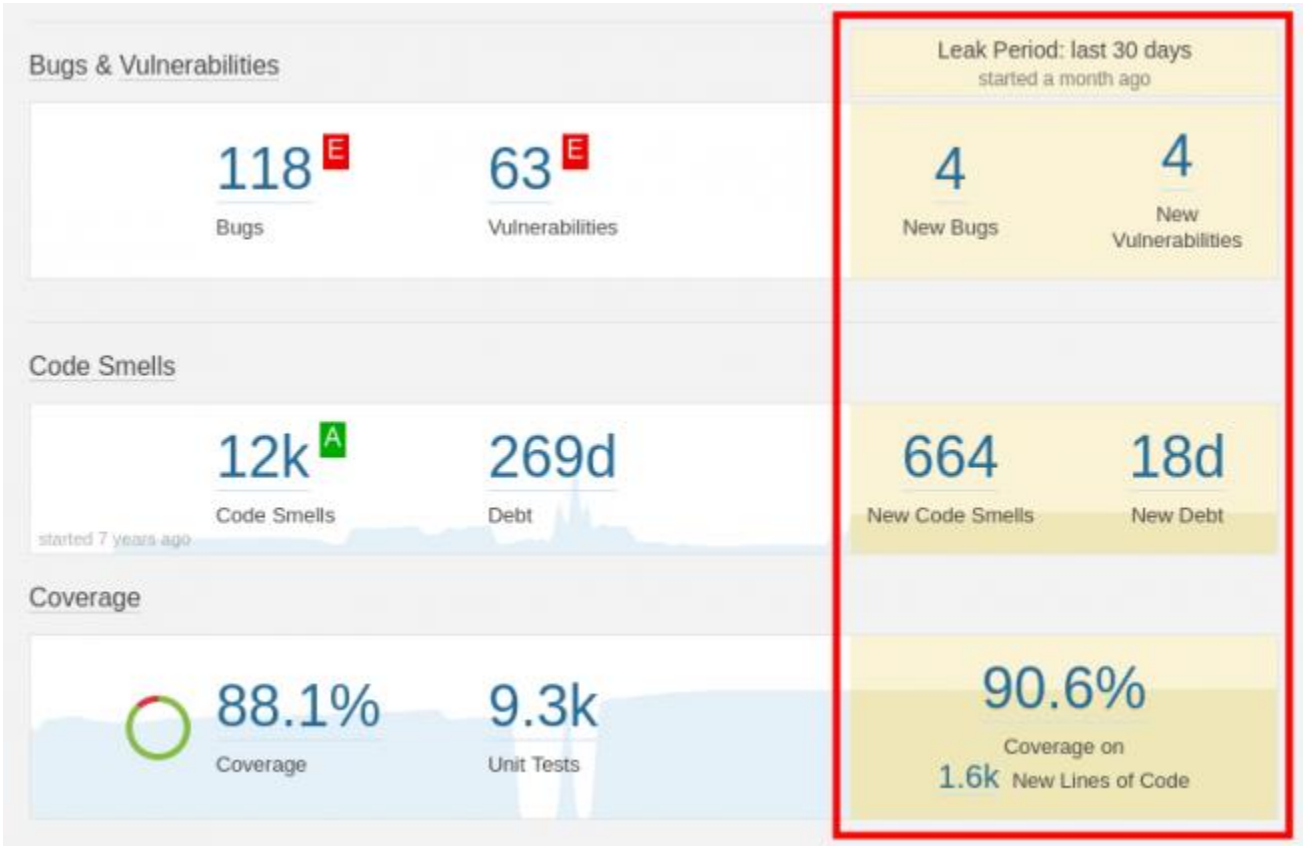
SonarQube

- Code quality management platform
- 20+ languages (Java, JS, Kotlin, C, C++, C#, Python …)

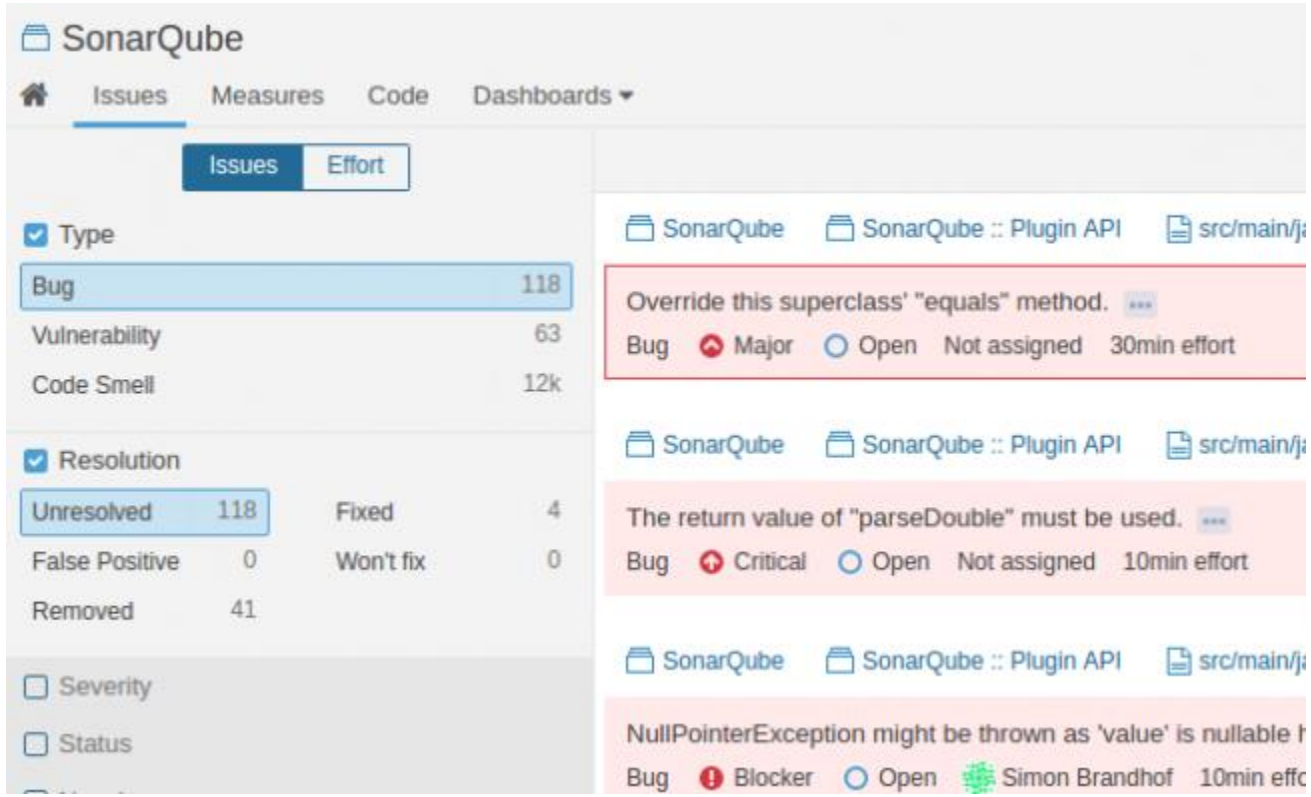
- Features
  - Examines coding standards, duplicated code, test coverage, code complexity, potential bugs and vulnerabilities, technical debt
  - Produces reports, evolution graphs
  - Integrates with external tools: IDEs, CI tools, …

http://www.sonarqube.org/

SonarQube

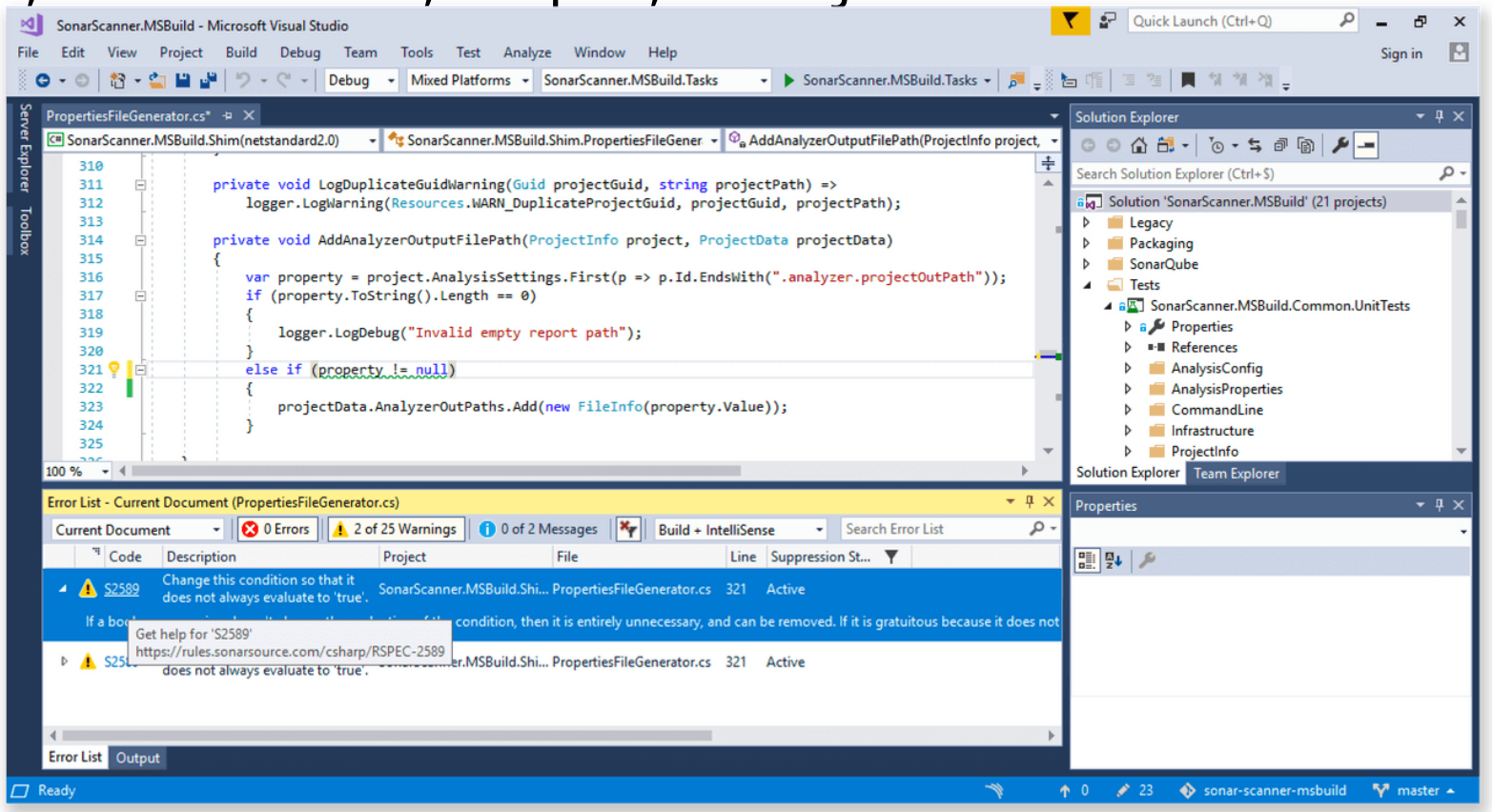

Bugs & Vulnerabilities
118 E
Bugs
63 E
Vulnerabilities
Leak Period: last 30 days
started a month ago
4
New Bugs
4
New Vulnerabilities
Code Smells
12k A
Code Smells
269d
Debt
started 7 years ago
664
New Code Smells
18d
New Debt
Coverage
88.1%
Coverage
9.3k
Unit Tests
90.6%
Coverage on
1.6k New Lines of 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

SonarLint

- Sonar plug-in for IDEs
  - VS Code, Visual Studio, Eclipse, Intellij



https://www.sonarlint.org/

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