High Quality Source Code

HUSZERL Gábor

huszerl@mit.bme.hu

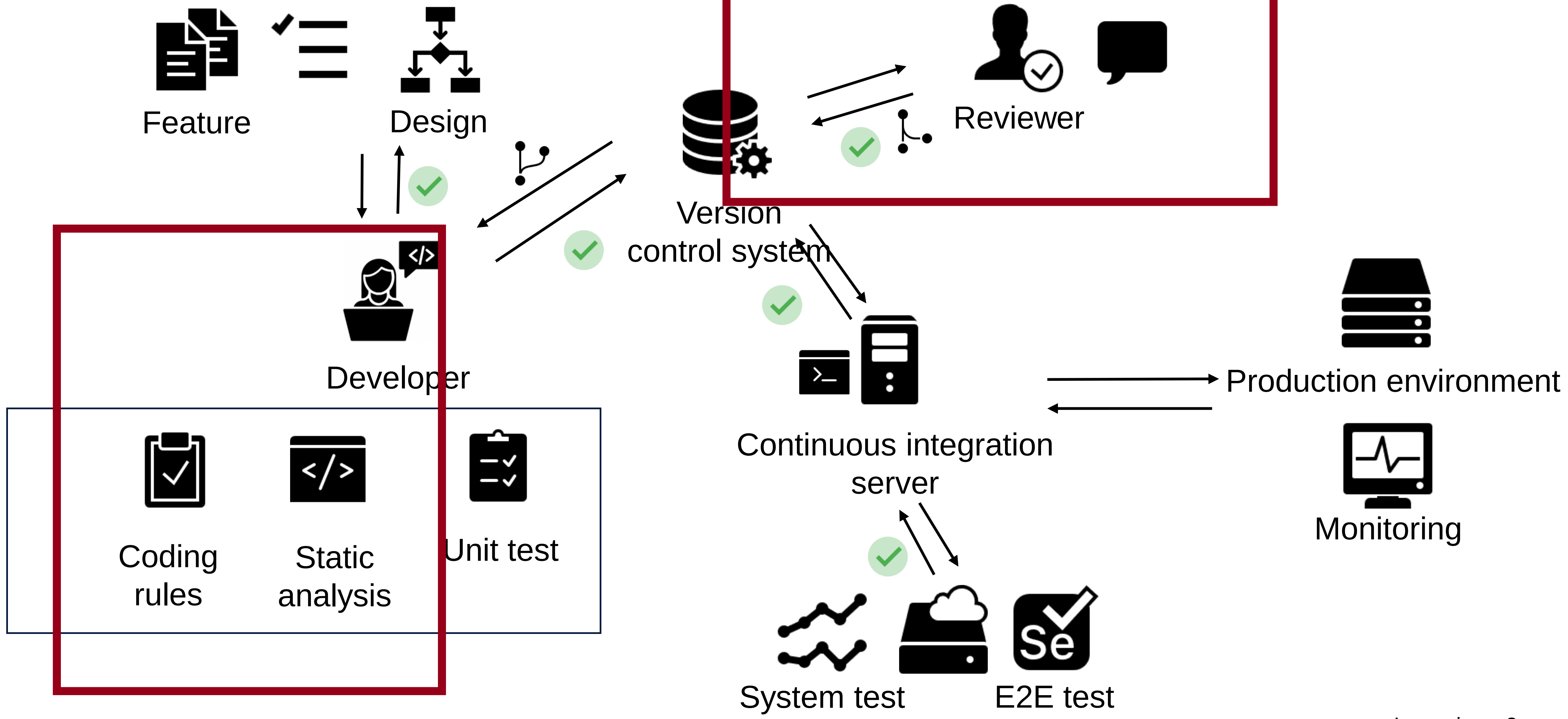
Learning Outcomes

- At the end of the lecture the students are expected to be able to
- (K2) summarise the goals and types of coding guidelines,
- (K3) implement code review for simpler changes,
- (K3) use static analysis tools to find errors.

Further Topics of the Subject

I. Software development practices	Steps of the development	Version controlling	Requirements management
	Planning and architecture	High quality source code	Testing and test development
II. Modelling	Why to model, what to model?	Unified Modeling Language	Modelling languages
III. Processes and projects	Methods	Project management	Measurement and analysis

Typical Development Workflow



Icons: icons8.com

Motivation – A Counterexample

```
public class Class1
{
  public decimal Calculate(decimal amount, int type, int years) {
     decimal result = 0;
    decimal disc = (years > 5) ? (decimal)5/100 : (decimal)years/100;
     if (type == 1) result = amount;
     else if (type == 2)
     {
       result = (amount - (0.1m * amount)) - disc * (amount - (0.1m * amount));
     }
     else if (type == 3) { result = (0.7m * amount) - disc * (0.7m * amount); }
     else if (type == 4) {
       result = (amount - (0.5m * amount)) - disc * (amount - (0.5m * amount));
     }
     return result;
  }
}
```

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

Classification of the Examination Methods

Static

- **What**: any products (documentation, model, code)
- **How**: without execution
- **Example**: review, static analysis

Dynamic

- **What**: executable products (code, model, …)
- **How**: executing it, running it
- **Example**: simulation, testing, …

Properties of Good Source Code

Syntactically correct	• Checked by the compiler
High quality	• Readable, reusable, maintainable, … • **Coding guidelines** help
Free of errors	• **Static analysis**, testing, …
Satisfying the specification	• **Code review**, testing

Optimising the Code

- The code is written once but read many times later
  - Reviews, Corrections, Enhancements, Extensions, …
- It is worth optimising the code for clarity
- For the execution it will be optimised during compilation
  - Compilers are much better at it
  - The output of the compiler will be read very rarely

High Quality Source Code

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”

John F. Woods

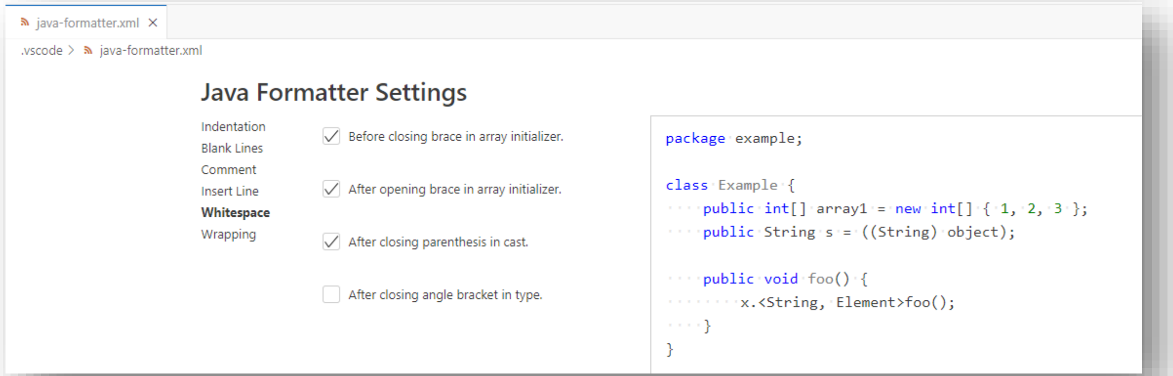
(in September 1991 in a post to the comp.lang.c++ newsgroup where the usage of comma operator was discussed)

“There are two ways to write error-free programs; only the third one works.”

Alan J. Perlis

(American mathematician and computer scientist who in 1966 won the A.M. Turing Award)

Coding Guidelines


java-formatter.xml ×

.vscode > java-formatter.xml

Java Formatter Settings

- Indentation
- Blank Lines
- Comment
- Insert Line
- **Whitespace**
- Wrapping

- [✓] Before closing brace in array initializer.
- [✓] After opening brace in array initializer.
- [✓] After closing parenthesis in cast.
- [ ] After closing angle bracket in type.

```
package example;

class Example {
    public int[] array1 = new int[] { 1, 2, 3 };
    public String s = ((String) object);

    public void foo() {
        x.<String, Element>foo();
    }
}
```

Coding Guidelines: Introduction

- Ruleset providing recommendations
  - Style: formatting, naming, structure
  - Programming advices: constructs, architecture
- Main Categories
  - Domain specific
    - Automotive, railways, …
  - Platform specific
    - C, C++, C#, Java, …
  - Organisation/company specific
    - Google, CERN, …

Domain Specific: MISRA C

- Motor Industry Software Reliability Association
- Goal: safety, reliability, portability
- 16 directives + 143 rules
- 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*

Outlook: Apple goto fail error

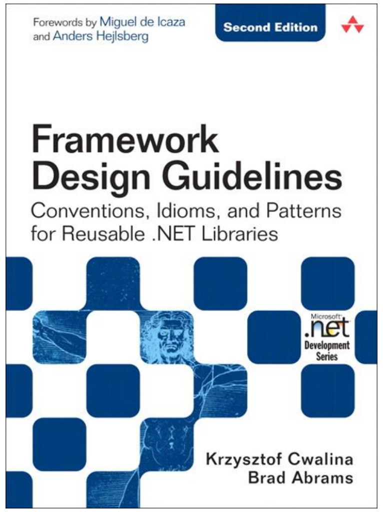
```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
        OSStatus        err;
        ...

        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

Platform Specific: .NET

- Framework Design Guidelines (C#)
  - Goal: developing frameworks and APIs

- Categories
  - Naming, designing types, designing member variables, extensibility, exceptions, usability, common design patterns
  - „Do”, „Consider”, „Avoid”, „Do not”

- Tools: StyleCop


Forewords by Miguel de Icaza and Anders Hejlsberg
Second Edition
Framework Design Guidelines
Conventions, Idioms, and Patterns for Reusable .NET Libraries
.net Development Series
Krzysztof Cwalina
Brad Abrams

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

Organisation Specific: Google

- Java Style Guide
- Goal: „hard-and-fast" rules, avoiding recommendations
- Categories
  - Source file basics
  - Source file structure
  - Formatting
  - Naming
  - Programming practices
  - Javadoc (documentation)
- Further guides: C++, C#, Python, JavaScript, R, …

Google context: 30k engineers, 60k commits/day,

over 2Mrd lines of code, codes used for decades

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

Organisation Specific: Google (Examples)

- *Local variable names are written in `lowerCamelCase`.*
- *The order you choose for the members and initializers of your class can have a great effect on learnability. … What is important is that each class uses some logical order, which its maintainer could explain if asked.*
- *Within a switch block, each statement group either terminates abruptly (with a `break`, `continue`, `return` or thrown exception), or is marked with a comment to indicate that execution will or might continue into the next statement group.*

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

Coding Guidelines: Enforcing the Rules

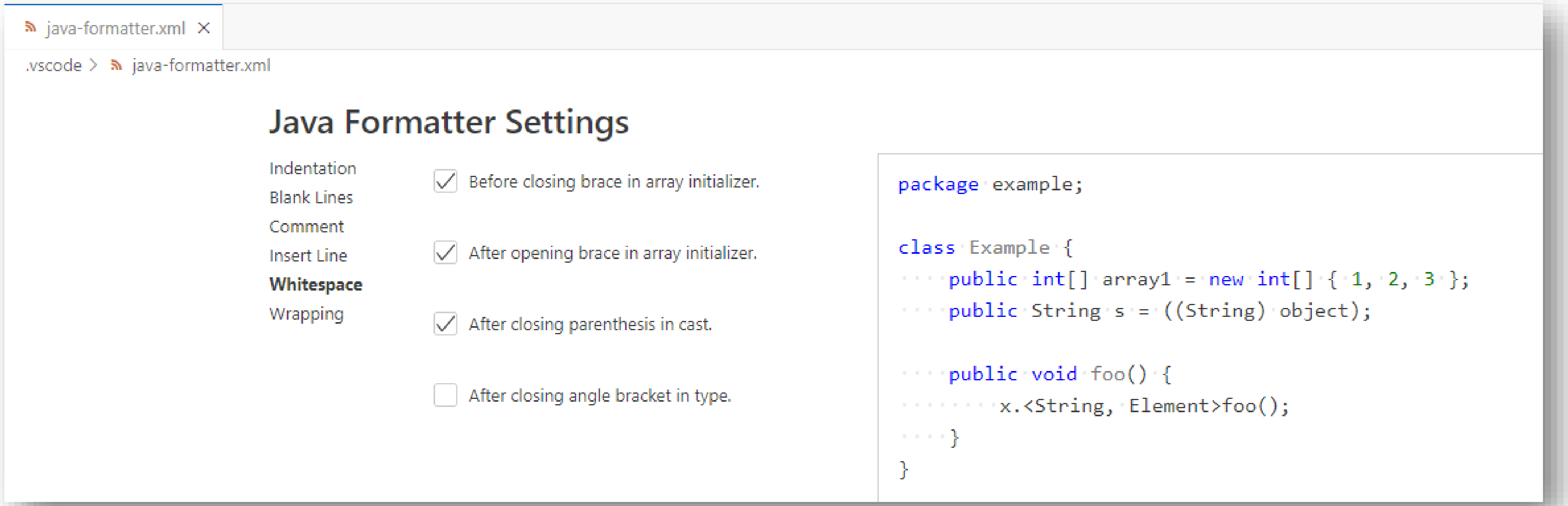
- How to enforce?
  - Standard feature in many IDEs
  - External tools
  - Tightly integrated into the development process

Goal: as few rules as possible to remember, to have tool support

- **Important**
  - Always have a single, common policy
  - At least uniform IDE formatting rules
    - Usually saved to a file that can be uploaded into the version control system

Google: „Consistency is what enables any engineer to jump into an unfamiliar part of the codebase and get to work fairly quickly."

Example: Visual Studio Code Formatter



Java Formatter Settings

- Indentation
- Blank Lines
- Comment
- Insert Line
- **Whitespace**
- Wrapping

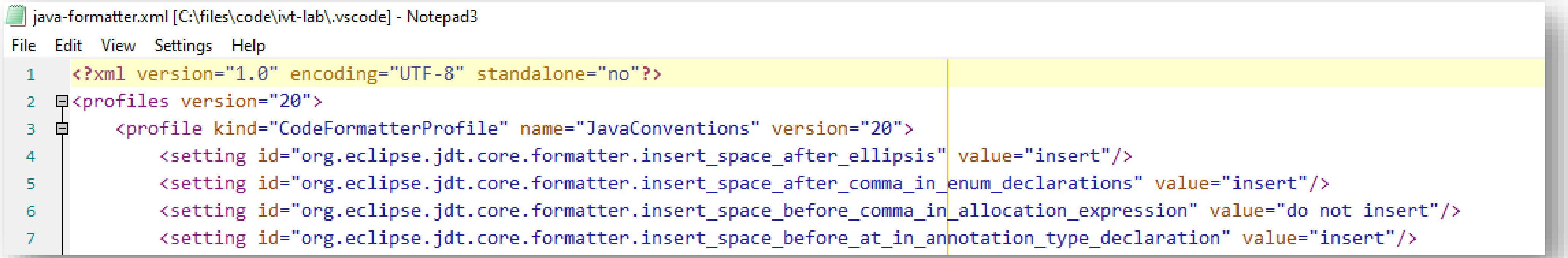
[✓] Before closing brace in array initializer.

[✓] After opening brace in array initializer.

[✓] After closing parenthesis in cast.

[ ] After closing angle bracket in type.

```java
package example;

class Example {
    public int[] array1 = new int[] { 1, 2, 3 };
    public String s = ((String) object);

    public void foo() {
        x.<String, Element>foo();
    }
}
```


java-formatter.xml [C:\files\code\ivt-lab\.vscode] - Notepad3
File Edit View Settings Help

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<profiles version="20">
    <profile kind="CodeFormatterProfile" name="JavaConventions" version="20">
        <setting id="org.eclipse.jdt.core.formatter.insert_space_after_ellipsis" value="insert"/>
        <setting id="org.eclipse.jdt.core.formatter.insert_space_after_comma_in_enum_declarations" value="insert"/>
        <setting id="org.eclipse.jdt.core.formatter.insert_space_before_comma_in_allocation_expression" value="do not insert"/>
        <setting id="org.eclipse.jdt.core.formatter.insert_space_before_at_in_annotation_type_declaration" value="insert"/>
```

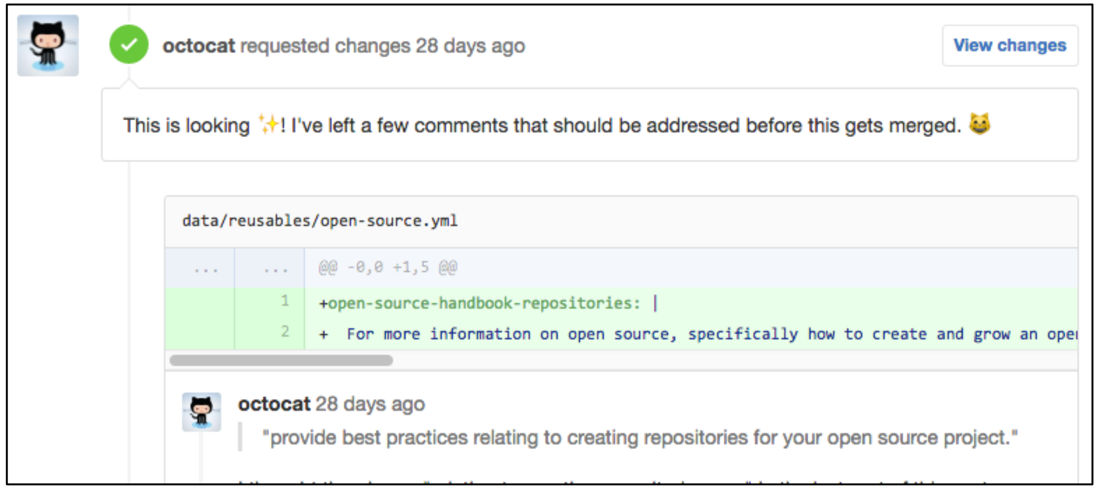
Coding guidelines: Which One to Use?

Which one is the best?

- Mostly already decided
  - By the domain / platform / organisation
  - Consistency with the already existing code base
- But sometimes it can be decided
  - Often there is no single best option
    - Sometimes they can even be inconsistent with each other
    - Sometimes combinations are also possible
  - But do not reinvent the wheel
    - Makes joining harder for new developers

For a smaller company/project, the good choice is one of the established, popular guidelines

Code Review


octocat requested changes 28 days ago

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 @@
+open-source-handbook-repositories: |
+  For more information on open source, specifically how to create and grow an ope
```

octocat 28 days ago

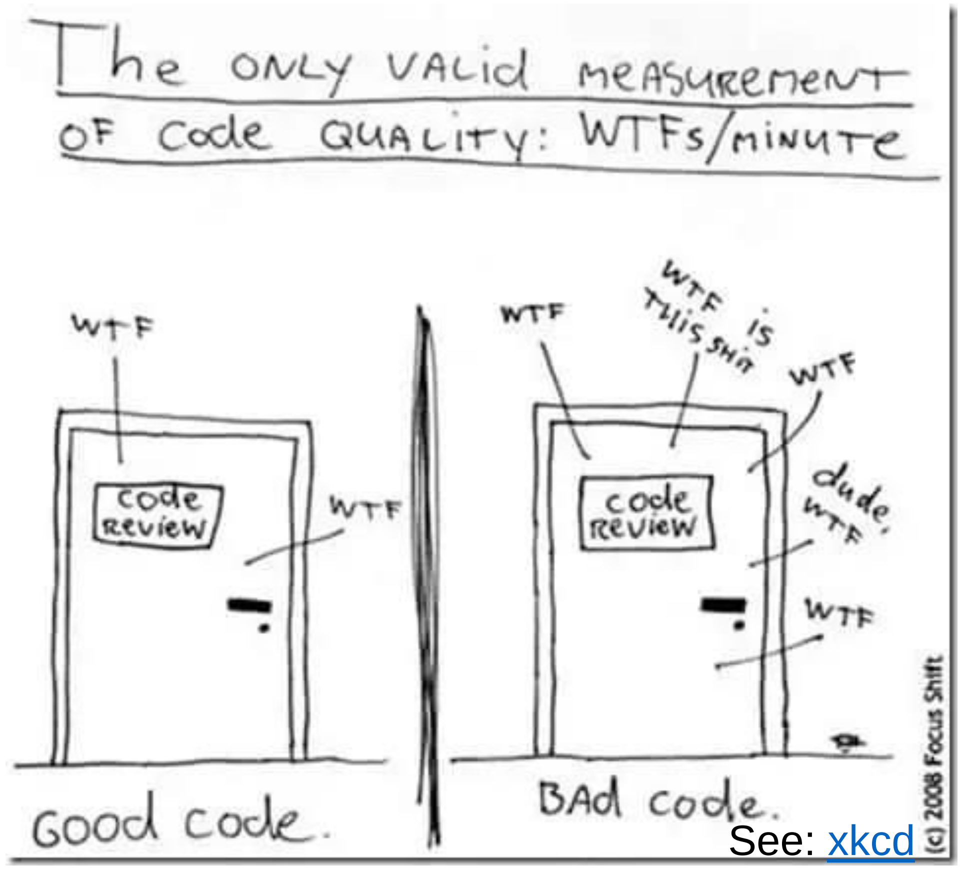
"provide best practices relating to creating repositories for your open source project."

Code Review: Introduction

- Manual method, carried out by humans
  - Reading, reviewing, analysing the source code
  - Usually based on a structured checklist

- Can be used at any time from ad-hoc request for advice to formal inspection

- (See *Levels of Formality in Review* in the lecture about requirements)



See: xkcd

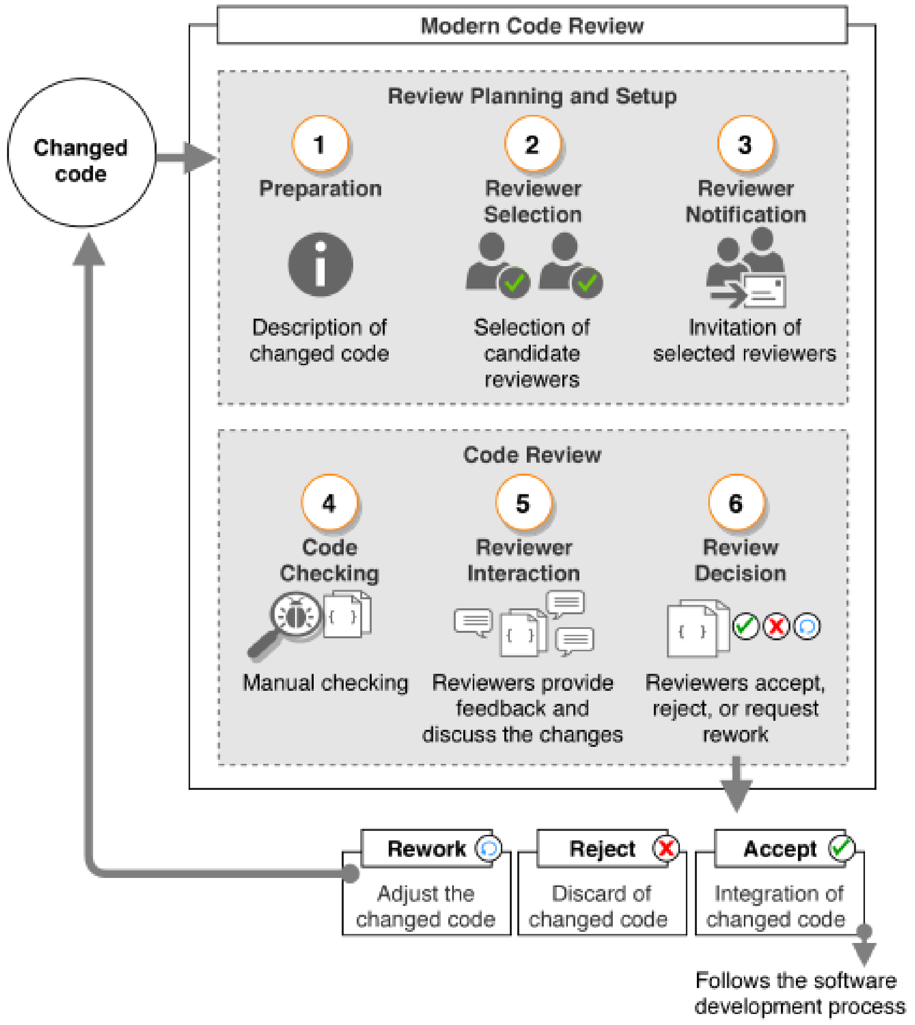
Types of Code Review

- **Formal inspection**
  - Efficient in finding errors
  - Time consuming, laborious work
- **Modern, lightweight techniques**
  - Less formal, good tool support
  - For frequent, smaller changes; fast feedback
  - Widely used in the industry (Microsoft, Google, Facebook, …)
  - Further benefits in addition to finding errors
    - Knowledge transfer
    - Team spirit
    - Alternative solutions

The human aspect is also important (constructive tone, empathy, …)!

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

Process of the Modern Code Review



Modern Code Review
Changed code
Review Planning and Setup
1 Preparation
Description of changed code
2 Reviewer Selection
Selection of candidate reviewers
3 Reviewer Notification
Invitation of selected reviewers
Code Review
4 Code Checking
Manual checking
5 Reviewer Interaction
Reviewers provide feedback and discuss the changes
6 Review Decision
Reviewers accept, reject, or request rework
Rework
Adjust the changed code
Reject
Discard of changed code
Accept
Integration of changed code
Follows the software development process

- Typically only a few reviewers (1-3)
- In open source projects feedback in days, in in-house teams in hours (for the better case)
- Detailed description of change is more likely to get quick response

Source: [A systematic literature review and taxonomy of modern code review]

Using Checklists

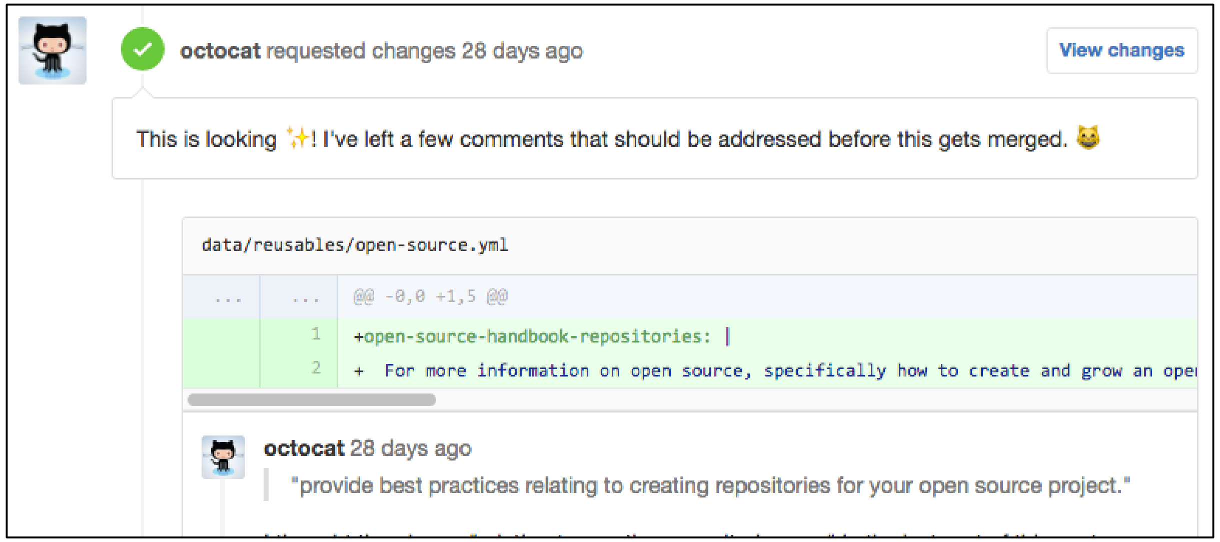
Checklist: structured list of criteria

- Categories similar to the ones of coding guidelines
  - Readability and maintainability of code
  - Security, safety, vulnerability
  - Performance
  - Common design patterns, programming practices
- Advices
  - A wide range of *code review checklists* are available online
  - Strive for automation
    - E.g. formatting can also be checked by a tool

It is only worth reviewing code where the basic errors have already been filtered out by the tools

Code Review – Tools

- Supporting code review
  - Attaching comments, dialogs to code snippets
  - Integrated into the development process
- GitHub: pull request reviews (→ Laboratory)
  - Comments, acceptance, requesting changes

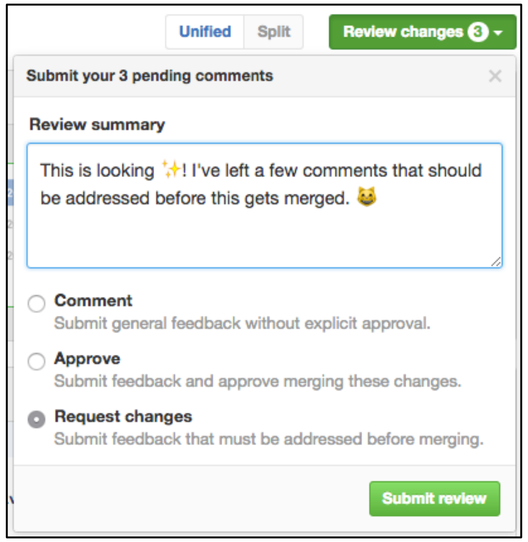

octocat requested changes 28 days ago

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 @@
+open-source-handbook-repositories: |
+  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."



Submit your 3 pending comments

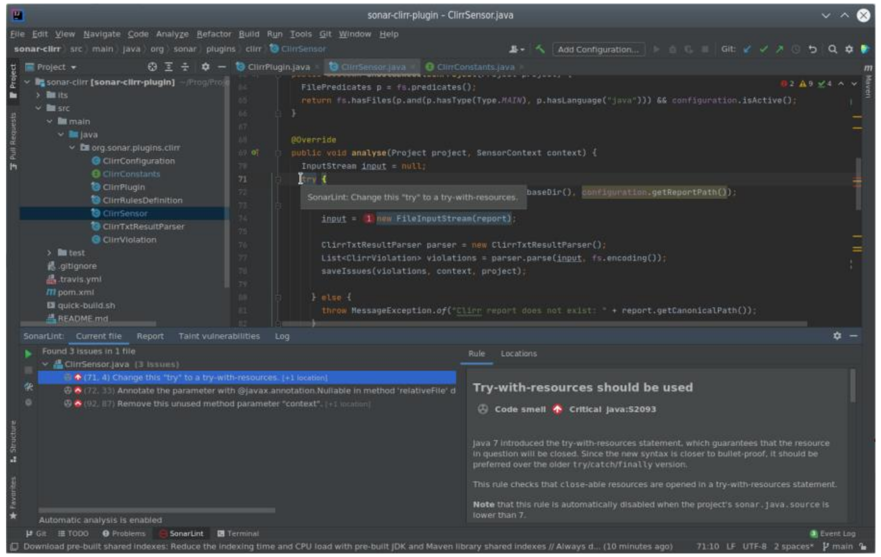
Review summary

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.

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

Static Analysis



```java
    FilePredicates p = fs.predicates();
    return fs.hasFiles(p.and(p.hasType(Type.MAIN), p.hasLanguage("java"))) && configuration.isActive();
  }

  @Override
  public void analyse(Project project, SensorContext context) {
    InputStream input = null;
    try {
                                         baseDir(), configuration.getReportPath());

        input = new FileInputStream(report);

        ClirrTxtResultParser parser = new ClirrTxtResultParser();
        List<ClirrViolation> violations = parser.parse(input, fs.encoding());
        saveIssues(violations, context, project);

      } else {
        throw MessageException.of("Clirr report does not exist: " + report.getCanonicalPath());
```

SonarLint: Change this "try" to a try-with-resources.

```
Found 3 issues in 1 file
ClirrSensor.java (3 issues)
  (71, 4) Change this "try" to a try-with-resources. [+1 location]
  (72, 33) Annotate the parameter with @javax.annotation.Nullable in method 'relativeFile' d
  (92, 87) Remove this unused method parameter "context". [+1 location]
Automatic analysis is enabled
```

Try-with-resources should be used

Code smell Critical java:S2093

Java 7 introduced the try-with-resources statement, which guarantees that the resource in question will be closed. Since the new syntax is closer to bullet-proof, it should be preferred over the older `try/catch/finally` version.

This rule checks that `close-able` resources are opened in a try-with-resources statement.

Note that this rule is automatically disabled when the project's `sonar.java.source` is lower than 7.

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK and Maven library shared indexes // Always d... (10 minutes ago) 71:10 LF UTF-8 2 spaces* main

Static Analysis – Example

```java
public class Sample {
        public static void main(String[] args) {
                String str = null;
                try {
                        Scanner scanner = new Scanner("file.txt");
                        str = scanner.nextLine();
                        scanner.close();
                } catch (Exception e) {
                        System.out.println("Error opening file!");
                }
                str.replace(" ", "");
                System.out.println(str);
        }
}
```

In case of an exception, scanner will not be closed

str can be null

str „immutable”

Static Analysis: Introduction

- Definition: analysing the program without executing it
  - Usually by automatized tools
  - We can also include manual review

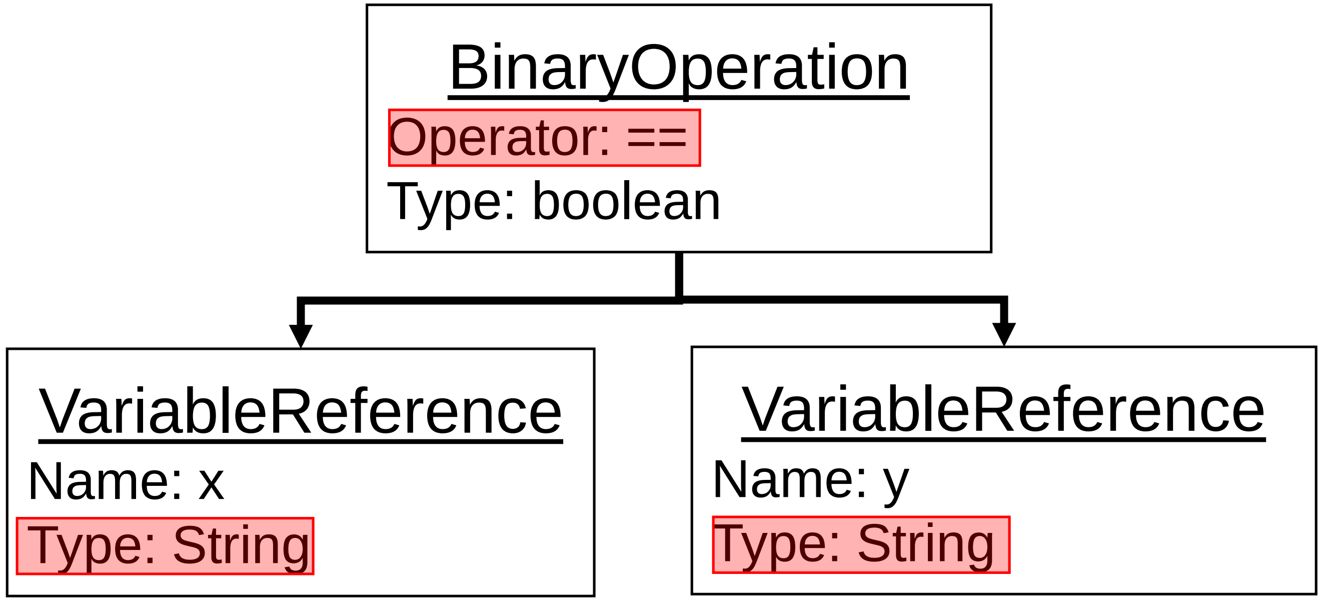
- Based on patterns
  - Mostly simple static properties, based on error patterns
    - E.g.: unused variable, ignored return value
  - Tools: SpotBugs, ErrorProne, SonarQube, Coverity
- Based on interpretation (→ MSc)
  - Dynamic properties
    - E.g.: null pointer reference, over indexing
  - Tools: Infer, PolySpace

Example: Searching for Error Patterns

Method: building Abstract Syntax Tree, AST

Example rule: „Strings and Boxed types should be compared using "equals()"” (Sonar S4973)

```
f(String x, String y) {
    if (x == y) {
        …
    }
}
```



See BSc course „Automatized Software Development”

ErrorProne (Java)

- Internal development at Google
  - Extensible ruleset
  - Gradle, Maven, Eclipse, IntelliJ, …

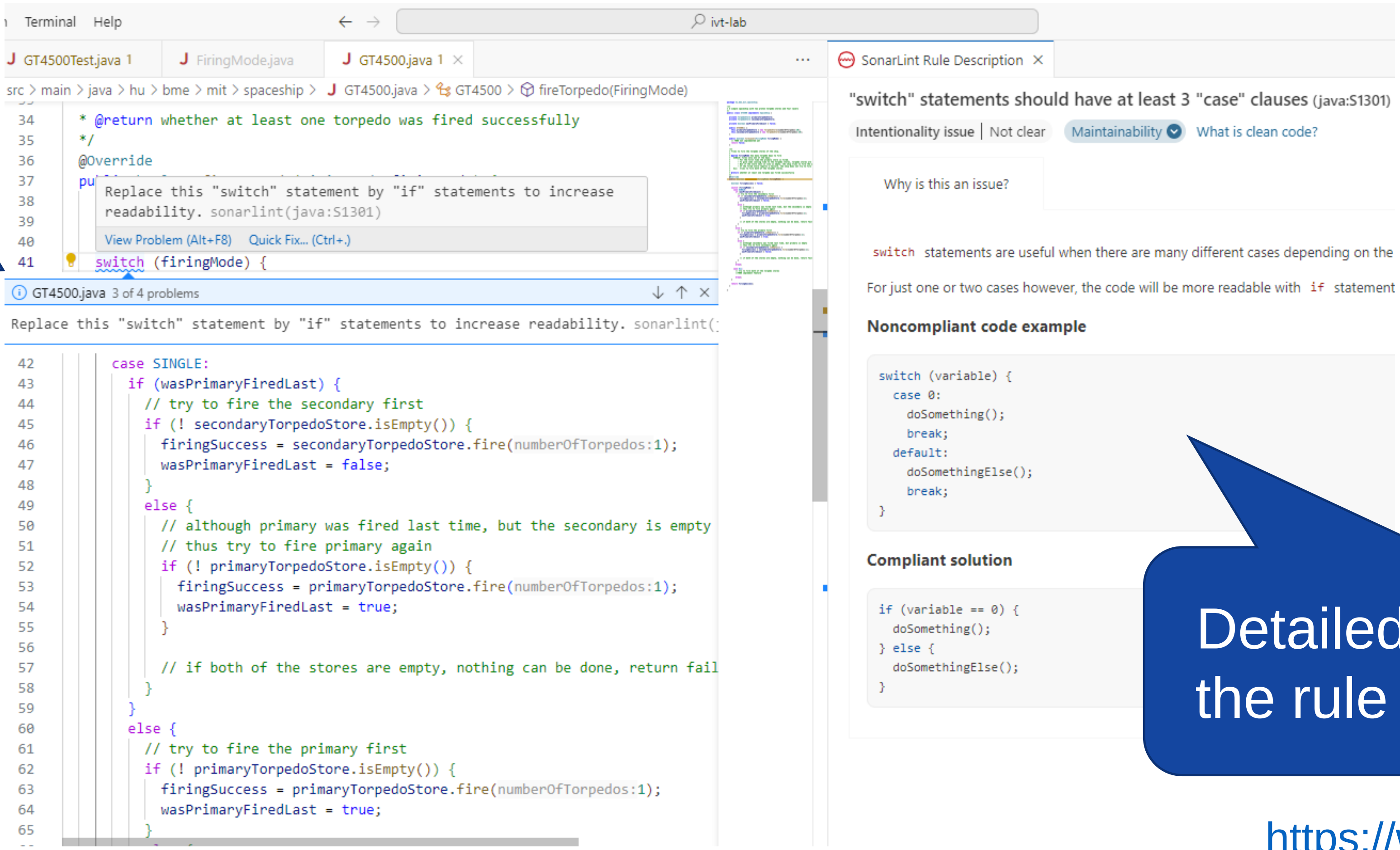
- Examples
  - „Reference equality used to compare arrays”
  - „Loop condition is never modified in loop body.”
  - „Comparison of a size >= 0 is always true,
    did you intend to check for non-emptiness?”

https://errorprone.info/

SonarLint

Plug-in for development environments (VS Code, VS, Eclipse, IntelliJ…)

Occurrence of the error


Terminal
Help
ivt-lab
GT4500Test.java 1
FiringMode.java
GT4500.java 1
src > main > java > hu > bme > mit > spaceship > GT4500.java > GT4500 > fireTorpedo(FiringMode)
View Problem (Alt+F8)
Quick Fix... (Ctrl+.)
GT4500.java 3 of 4 problems
SonarLint Rule Description
Intentionality issue | Not clear
Maintainability
What is clean code?
Why is this an issue?

```
   * @return whether at least one torpedo was fired successfully
   */
  @Override
  pu
```

Replace this "switch" statement by "if" statements to increase readability. sonarlint(java:S1301)

```
      switch (firingMode) {
```

Replace this "switch" statement by "if" statements to increase readability. sonarlint(

```
        case SINGLE:
          if (wasPrimaryFiredLast) {
            // try to fire the secondary first
            if (! secondaryTorpedoStore.isEmpty()) {
              firingSuccess = secondaryTorpedoStore.fire(numberOfTorpedos:1);
              wasPrimaryFiredLast = false;
            }
            else {
              // although primary was fired last time, but the secondary is empty
              // thus try to fire primary again
              if (! primaryTorpedoStore.isEmpty()) {
                firingSuccess = primaryTorpedoStore.fire(numberOfTorpedos:1);
                wasPrimaryFiredLast = true;
              }

              // if both of the stores are empty, nothing can be done, return fail
            }
          }
          else {
            // try to fire the primary first
            if (! primaryTorpedoStore.isEmpty()) {
              firingSuccess = primaryTorpedoStore.fire(numberOfTorpedos:1);
              wasPrimaryFiredLast = true;
            }
```

"switch" statements should have at least 3 "case" clauses (java:S1301)

`switch` statements are useful when there are many different cases depending on the

For just one or two cases however, the code will be more readable with `if` statement

Noncompliant code example

```
switch (variable) {
  case 0:
    doSomething();
    break;
  default:
    doSomethingElse();
    break;
}
```

Compliant solution

```
if (variable == 0) {
  doSomething();
} else {
  doSomethingElse();
}
```

Detailed description of the rule with examples

https://www.sonarlint.org/

SonarCloud

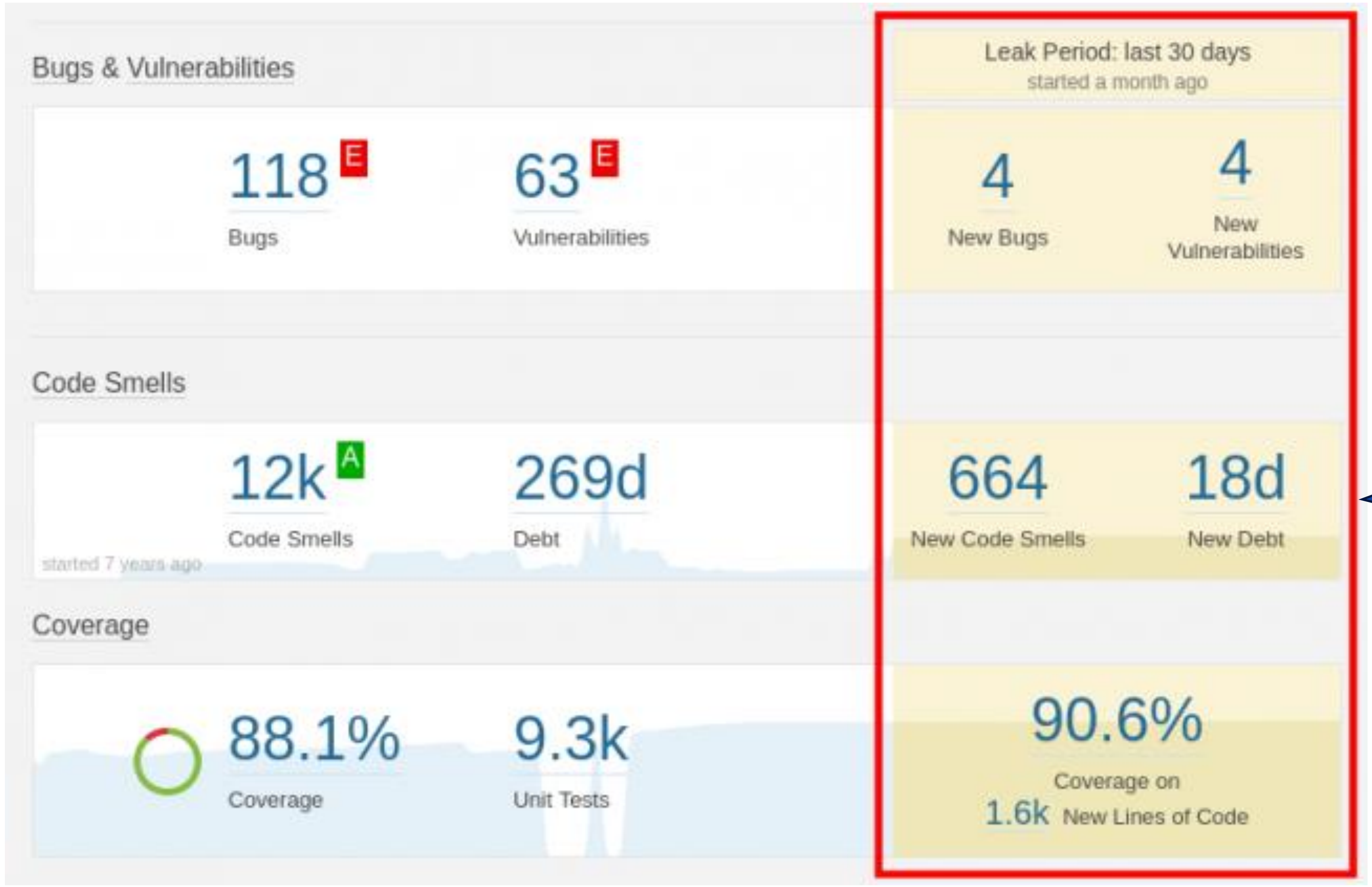
sonarcloud

- Code quality management platform
- 20+ languages (Java, JS, Kotlin, C, C++, C#, Python, …)
- Features
  - Checking coding guidelines, code duplication, test coverage, code complexity, potential errors and vulnerabilities, cost estimation
  - Generating reports and diagrams
  - Can be integrated into external tools
    - E.g.: development environments, continuous integration (CI) tools

(→ Laboratory)

http://www.sonarcloud.io/

SonarCloud: Overview



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

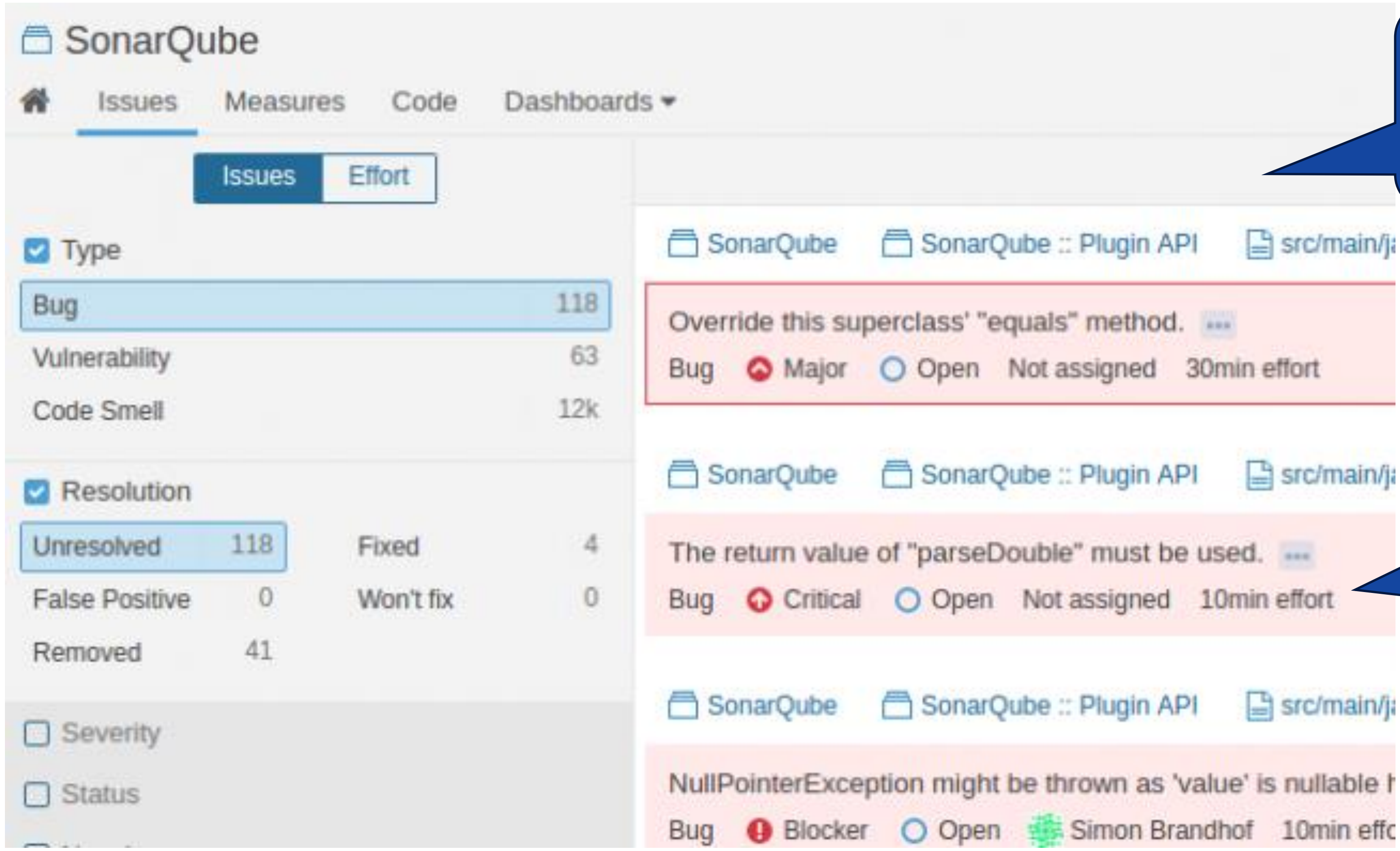
90.6%
Coverage on
1.6k New Lines of Code

Historic data and tracking changes

„Technical debt”

Future cost of correcting poor quality code

SonarCloud: List of Findings


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

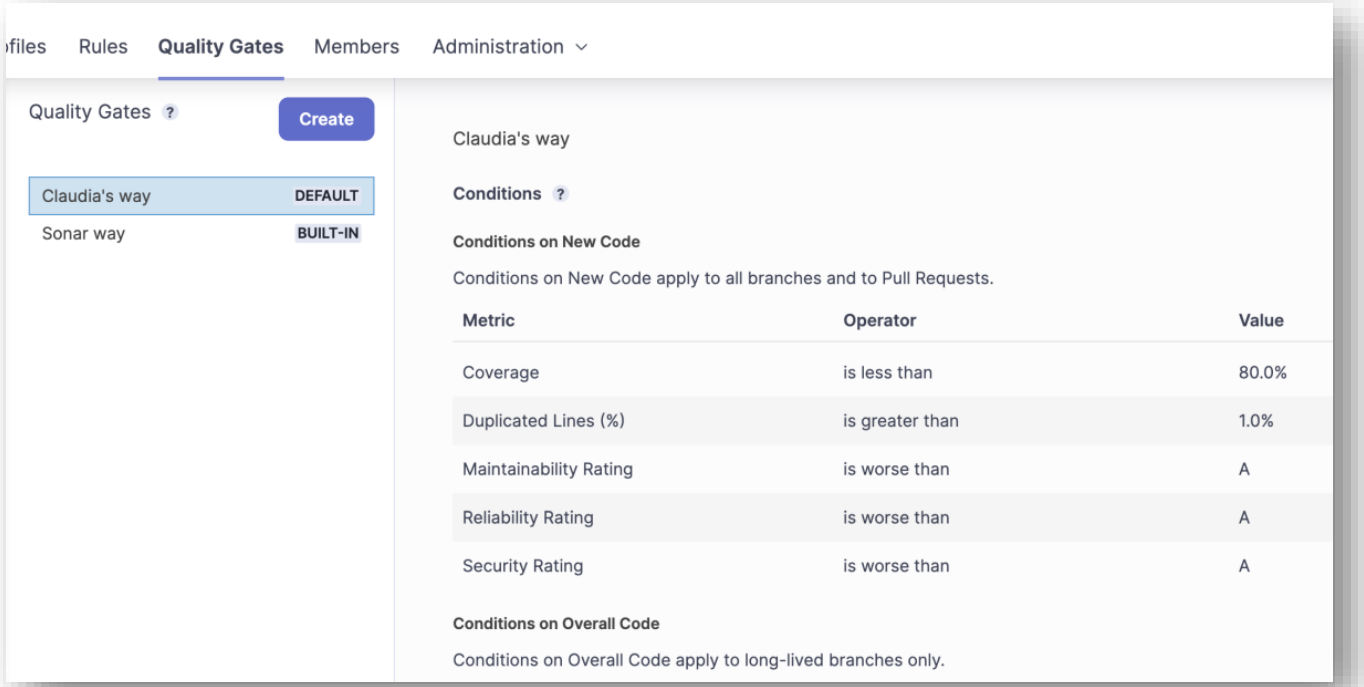
Bug Blocker Open Simon Brandhof 10min effo

Type (bug / vulnerability / code smell) and severity

Estimated time of repair

SonarCloud: Quality Gate

- Required minimum quality characteristics
- Customizable, may block pull requests, …



ofiles Rules **Quality Gates** Members Administration

Quality Gates ? Create

Claudia's way	DEFAULT
Sonar way	BUILT-IN

Claudia's way

Conditions ?

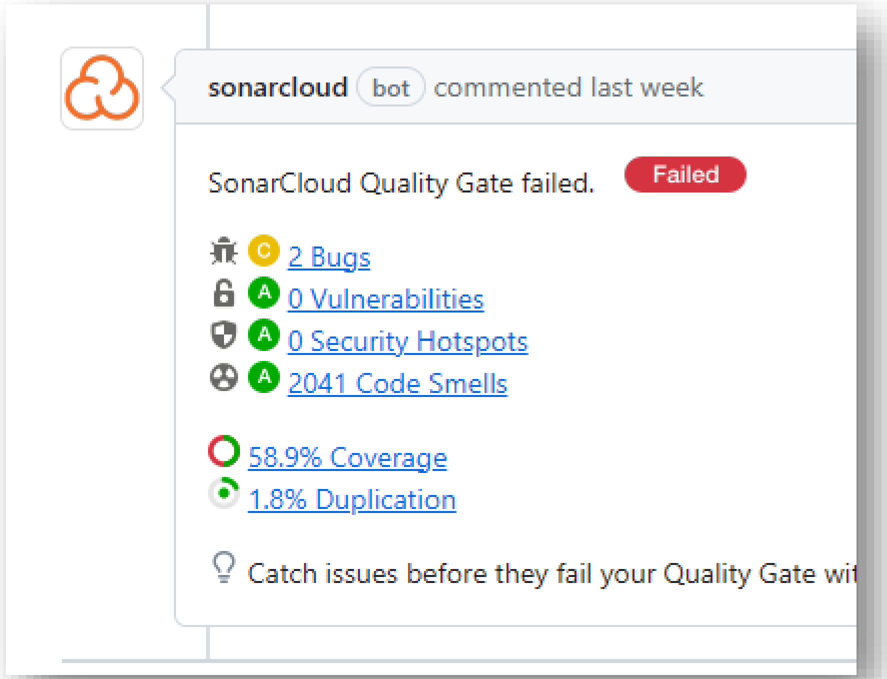
Conditions on New Code

Conditions on New Code apply to all branches and to Pull Requests.

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	1.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Rating	is worse than	A

Conditions on Overall Code

Conditions on Overall Code apply to long-lived branches only.



sonarcloud bot commented last week

SonarCloud Quality Gate failed. Failed

C 2 Bugs
A 0 Vulnerabilities
A 0 Security Hotspots
A 2041 Code Smells

58.9% Coverage
1.8% Duplication

Catch issues before they fail your Quality Gate wit

https://docs.sonarcloud.io/improving/quality-gates/

Efficient Usage of Static Analysis

- Let it be integrated into the build process
  - Checking before/after commit
  - Generating reports, e-mail notifications, …
- Use it from the start of the project
  - To many issues may discourage developers
- Configure the tools
  - Filtering by category and severity
  - Supplement with own rules

Efficient Usage of Static Analysis

- Use results with care
  - Both false positive and false negative results may occur

		tool result	
		error free	**erroneous**
reality	**error free**	True Negative (TN)	False Positive (FP)
	Erroneous	False Negative (FN)	True Positive (TP)

- False negative
  - Not finding any bugs does not mean their absence
- False positive
  - Finding a bug does not always mean a real error
  - Suppressing a complete rule or a single occurrence
    - Always justify

Static Analysis: Summary

- Analysing the software without executing it
  - An analysis is possible before the code becomes executable or inputs are available
  - Execution can be costly

- Finding hard-to-spot errors
  - Can be interesting also for experienced programmers

- Automatized process
  - Integrated into the development process

Summary

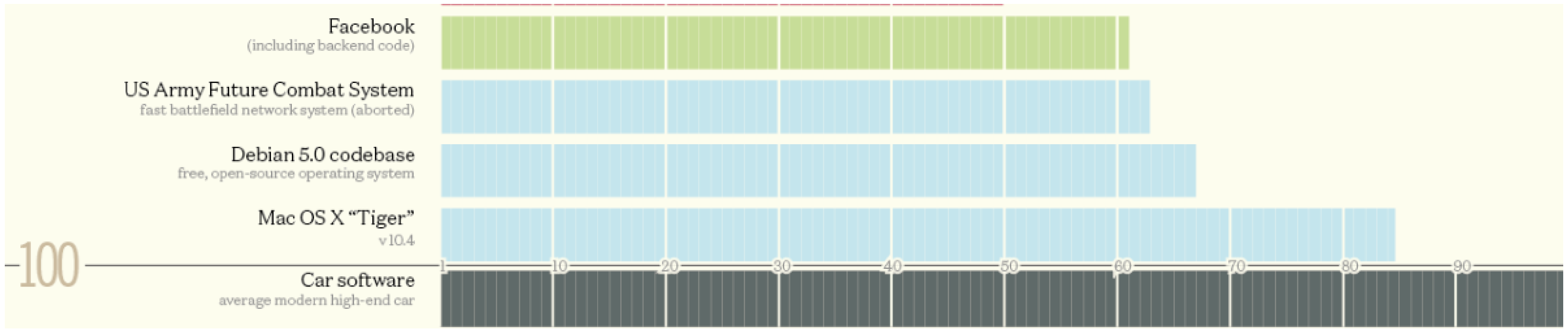
Outlook

Is it possible to write error-free code?

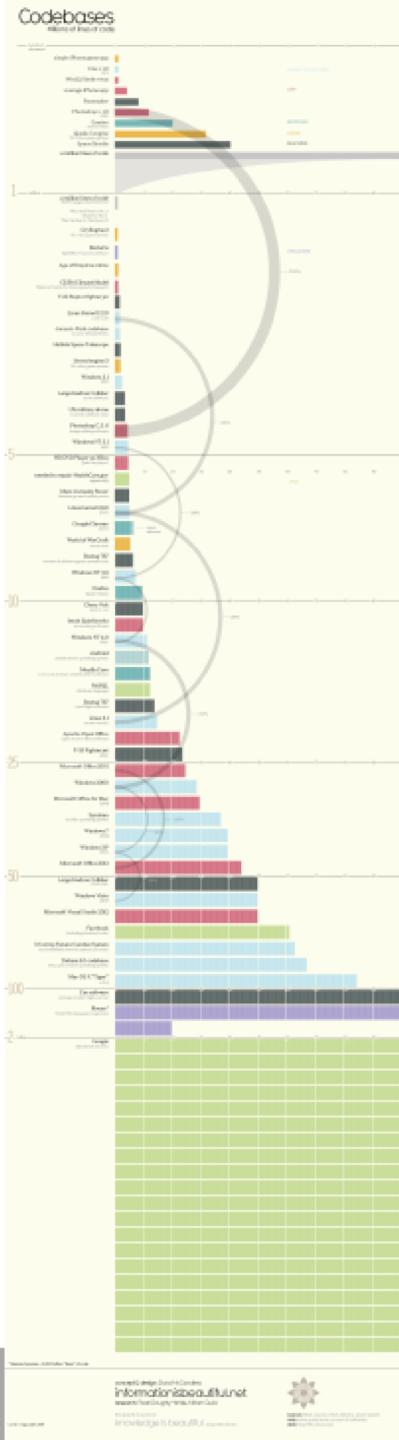
- NASA Space Shuttle
- Over 500 000 line of code [LoC]
  - Over 10 years (development, testing, launching)
- 0,11 error / 1000 lines after release
  - 0 error during the first missions
- About 1000 USD / line of codes total cost
  - (USD of the 1980s!)

Size of Code Bases (from 2015)

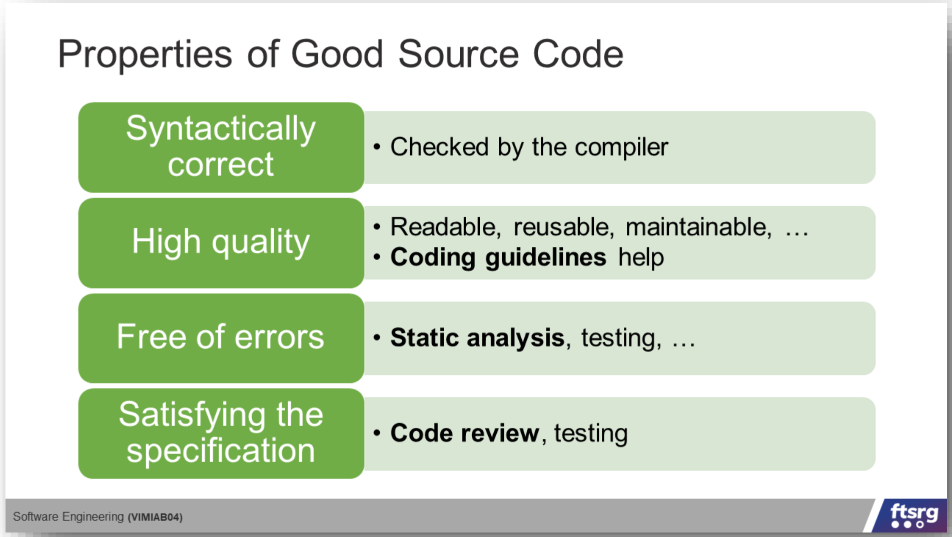
- (Prog 1 homework: ~2,00 LoC)
- Simpler app: ~20,000 LoC
- F-22 Raptor jet: ~2,000,000 LoC
- Open Office: ~20,000,000 LoC
- Google code base: ~2,000,000,000 LoC



Source: [How Many Millions of Lines of Code Does It Take?]

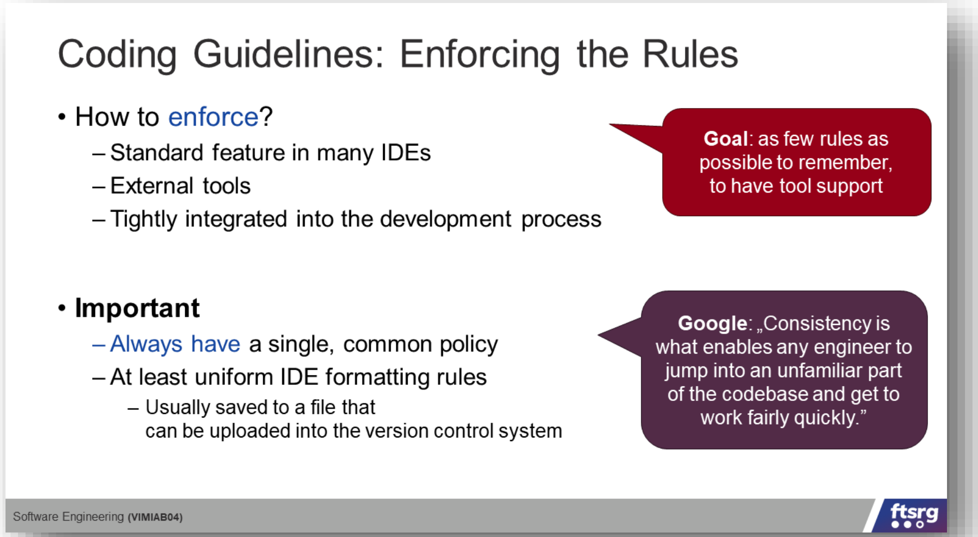


Summary



Properties of Good Source Code

Syntactically correct	• Checked by the compiler
High quality	• Readable, reusable, maintainable, … • **Coding guidelines** help
Free of errors	• **Static analysis**, testing, …
Satisfying the specification	• **Code review**, testing



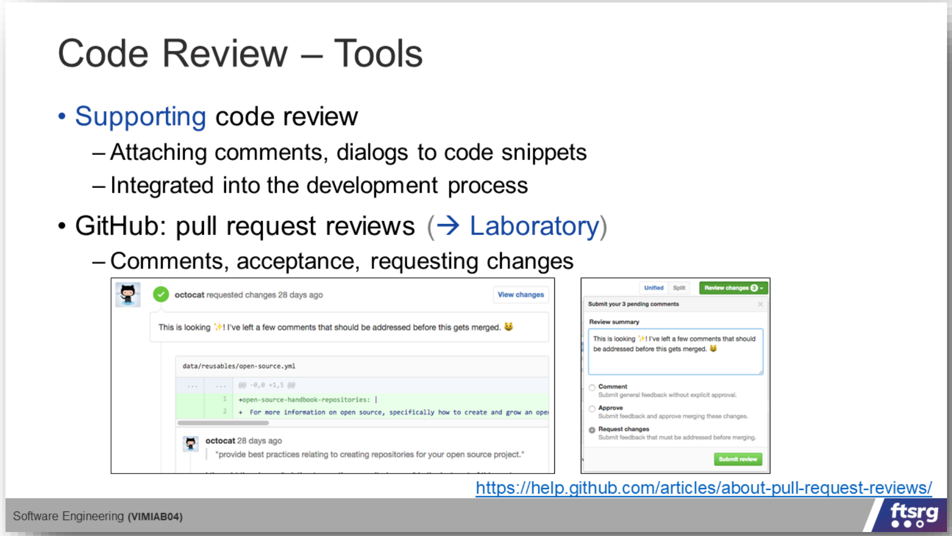
Coding Guidelines: Enforcing the Rules

- How to enforce?
  - Standard feature in many IDEs
  - External tools
  - Tightly integrated into the development process

Goal: as few rules as possible to remember, to have tool support

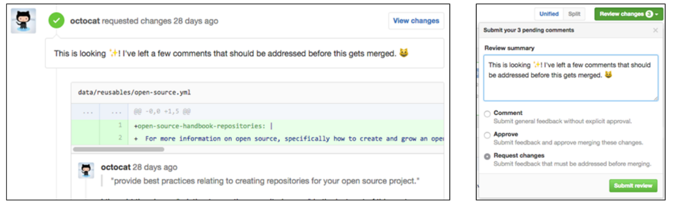
- **Important**
  - Always have a single, common policy
  - At least uniform IDE formatting rules
    - Usually saved to a file that can be uploaded into the version control system

Google: „Consistency is what enables any engineer to jump into an unfamiliar part of the codebase and get to work fairly quickly.”



Code Review – Tools

- Supporting code review
  - Attaching comments, dialogs to code snippets
  - Integrated into the development process
- GitHub: pull request reviews (→ Laboratory)
  - Comments, acceptance, 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 @@
+open-source-handbook-repositories: |
+  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."

Submit your 3 pending comments

Review summary

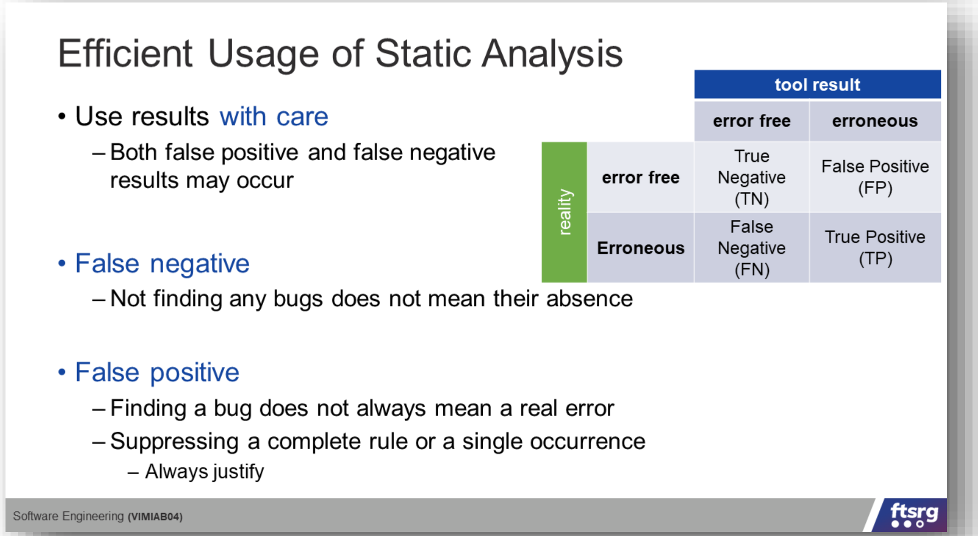
This is looking ✨! I've left a few comments that should be addressed before this gets merged. 😸

Comment [✓ Request changes]
Submit general feedback without explicit approval.

Approve
Submit feedback and approve merging these changes.

Request changes
Submit feedback that must be addressed before merging.

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



Efficient Usage of Static Analysis

- Use results with care
  - Both false positive and false negative results may occur

		tool result	
		error free	**erroneous**
reality	**error free**	True Negative (TN)	False Positive (FP)
	Erroneous	False Negative (FN)	True Positive (TP)

- False negative
  - Not finding any bugs does not mean their absence
- False positive
  - Finding a bug does not always mean a real error
  - Suppressing a complete rule or a single occurrence
    - Always justify