COMP0104 Software Development Practice: Software Measurement

Jens Krinke

Centre for Research on Evolution, Search & Testing
Software Systems Engineering Group
Department of Computer Science
University College London

Imagine your project viva...

- How much have you implemented?
- How complex is your implementation?
- How long did it take?
- How many bugs are left in your product?

Imagine your project viva...

- How much have you implemented?
- A lot...
- How complex is your implementation?
- Uh?
- How long did it take?
- Too long...
- How many bugs are left in your product?
- None!

Imagine your project viva...

- How much have you implemented?
- 100k lines of code, 60k lines of executable code
- How complex is your implementation?
- After refactoring, only a few complex classes left
- How long did it take?
- Implementation: 4 40h-weeks, Testing: 2 weeks
- How many bugs are left in your product?
- Testing achieved 75% statement coverage, found 15 bugs

You cannot control what you cannot measure

I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind.

William Thomson, 1824-1907,
1st Baron Kelvin (“Lord Kelvin”)

What can be measured? (Examples)

Products

- Size
- Complexity
- Defects
- Performance

Resources

- time
- effort
- progress

Why Measure?

- **Characterise**:
  gain an understanding of products and resources,
  establish a baseline for future assessments.
- **Evaluate**:
  assess achievement of plans and goals
- **Predict**:
  build models based on relationships
  between observable attributes
- **Improve**:
  identify issues in order to remove them

Software Metrics

A **software metric** is a mapping of the entities of a software system to numeric metrics values.

Example:

- Counting the lines of code is a measurement
- Lines of code (LOC) is the metric

What can be measured?

- An **attribute** is a feature or property of an entity
- Internal attributes can be measured only based on the entity itself
  - Code: lines of code
  - Project: code coverage
  - …
- External attributes can be measured only with respect to how the entity relates to its environment
  - Code: reliability
  - …

Software Size

- The most useful attribute of a software product is its size which can be defined as
  - length - the physical size
  - functionality - what is provided
  - complexity
- Size can be measured for different artefacts:
  - source code
  - specification
  - design

Lines of Code

- The most often used measure of code length is the number of lines of code (LOC).
- LOC is easy to measure
- But what is actually code? Comments?
- Variations:
  - Source lines of code (SLOC): number of lines with something other than comments and whitespace
  - Executable lines of code (ELOC): number of lines for which executable code is generated

```c
#include <stdio.h>

// global variables
int x = 25;
int y;

int main() {
  int z = 0, i;
  for (i=0; i<x; ++i) {
    z += x;
  }
  for (i=0;
       i<x;
       ++i)
    {
      if (x == y) z = z-1; else z = z+1;
    }
  return z;
}
```

19 lines
16 SLOC (sloccount)
15 SLOC (cccc)
9 ELOC (gcov)

Lines of Code

Advantages

- Easy to measure
- Correlates with programming effort

Disadvantages

- Affected by layout
- Language dependent

Halstead's Work

A program is a collection of **tokens**, composed of two basic elements: **operands** and **operators**

- **Operands** are variables, constants, addresses
- **Operators** are defined operations in a language

Computation

- η_1: the number of distinct operators
- η_2: the number of distinct operands
- N_1: the total number of operators
- N_2: the total number of operands
- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$

Software Complexity

- Length of a software system does not necessarily correlate to its complexity, e.g. how hard it is to understand and/or maintain.
- Complexity measures...
- Readability measures... (?)
- Halstead's complexity metrics:
  - Volume: $V = N \times \log_2 \eta$
  - Difficulty: $D = \eta_1/2 \times N_2/\eta_2$
- What are operands? Operators?

McCabe's Cyclomatic Complexity

- Thomas McCabe, 1976
- Complexity of a program
  - Number of linearly independent paths through a function
  - Usually calculated using flow graph
- V(G) = E – N + 2P
  - E: number of edges
  - N: number of vertices
  - P: number of unconnected subgraphs (procedures)

Cyclomatic Complexity: 4

```
#include <stdio.h>

// global variables
int x = 25;
int y;

int main() {
  int z = 0, i;
  for (i=0; i<x; ++i) {
    z += x;
  }
  for (i=0;
       i<x;
       ++i)
    {
      if (x == y) z = z-1; else z = z+1;
    }
  return z;
}
```

McCabe's Cyclomatic Complexity

- A value > 10 is considered to be too complex.
- Only considers executable code,
  ignores data structures and dependencies.
- Empirical studies show correlation to length,
  e.g. executable lines of code.

Object Oriented Metrics, Chidamber and Kemerer, 1994

- Metrics based on firm theoretical basis and experience of professional software developers
- Measure unique aspects of object-oriented programs
  - Inheritance
  - Complexity
  - Cohesion and Coupling
- Measure attributes of object-oriented structures

Inheritance Metrics

- Depth of inheritance tree (DIT):
  Maximum depth of a class in the inheritance tree
  - The inheritance tree should be kept shallow.
  - A high DIT has been found to increase faults.
- Number of children (NOC):
  Number of immediate subclasses of a class
  - High NOC has been found to indicate fewer faults.
- NOC and DIT are closely related.

Complexity Metrics

- Weighted method count (WMC):
  Sum of (McCabe's) complexities of all methods
- Response for a class (RFC):
  Number of methods in a class and
  methods directly called by methods in the class
- RFC’: Number of methods in a class and
  methods directly **or indirectly** called by them

Coupling and Cohesion Metrics

- Coupling between object classes (CBO)
  Number of classes given class is coupled to
- Lack of cohesion in methods (LCOM)
  Number of method pairs that do not share instance variables vs. number of methods that share at least one instance variable
  - LCOM has been revised multiple times

Object Oriented Metrics

- Since Chidamber & Kemerer's suite in 1991, more and more metrics have been invented.
- Most of the metrics have been found to be correlated.
- Most of the metrics are correlated to LOC.
- Value of object-oriented metrics is debated.

Functionality Metrics

- Function-oriented metrics are indirect measures of software focussing on functionality and utility.
- Idea: products with more functionality are larger.
- Function Point (FP) is a weighted measure of software functionality.
- Measure the amount of functionality in a system based upon the system specification.
- Goal: estimate the costs of the implementation.
- Estimate the system size before implementation.

What else can measured?

Measuring the size of the system is not all.

- Code churn metrics
- Test measurements
- Defect numbers
- Compliance violations
- Amount of duplicated code
- …

Code Churn Metrics

- Amount of code changed in a period of time
- Churned LOC:
  number of added, modified and deleted lines
- Churn Count:
  number of changes made to a file
- Files Churned:
  number of changed files

Test Measurements

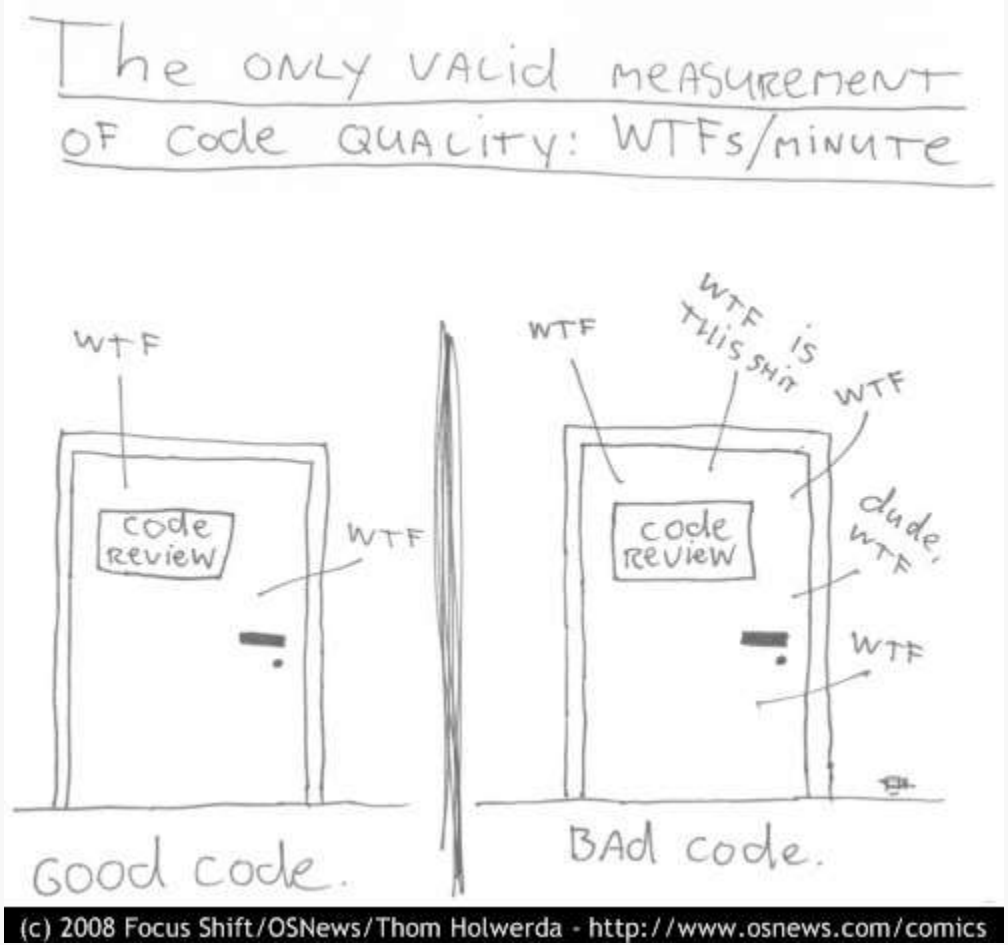
- Number of passed and failed tests
- Code coverage

Defects

- How many defects have we found?
- How many defects have clients found?
- How many are open, assigned?
- What is the ratio of defects to size?
- Require bug tracking (JIRA, Trac, ...)

Compliance

- Number of guideline violations
- Number of code smells
- Number of (un)documented methods/classes/...



Problems

- Most attributes of interest cannot be measured or cannot be measured directly
- Most metrics are very hard to validate
- Too many metrics exist
- What are appropriate metrics?
- Most models are at best vague approximations
- The technology keeps changing
- **But**:
  poor models may be better than no models at all

Effective Measurements

- Simple and computable
- Empirically and intuitively persuasive
- Consistent and objective
- Consistent in use of units and dimensions
- Programming language independent (?)

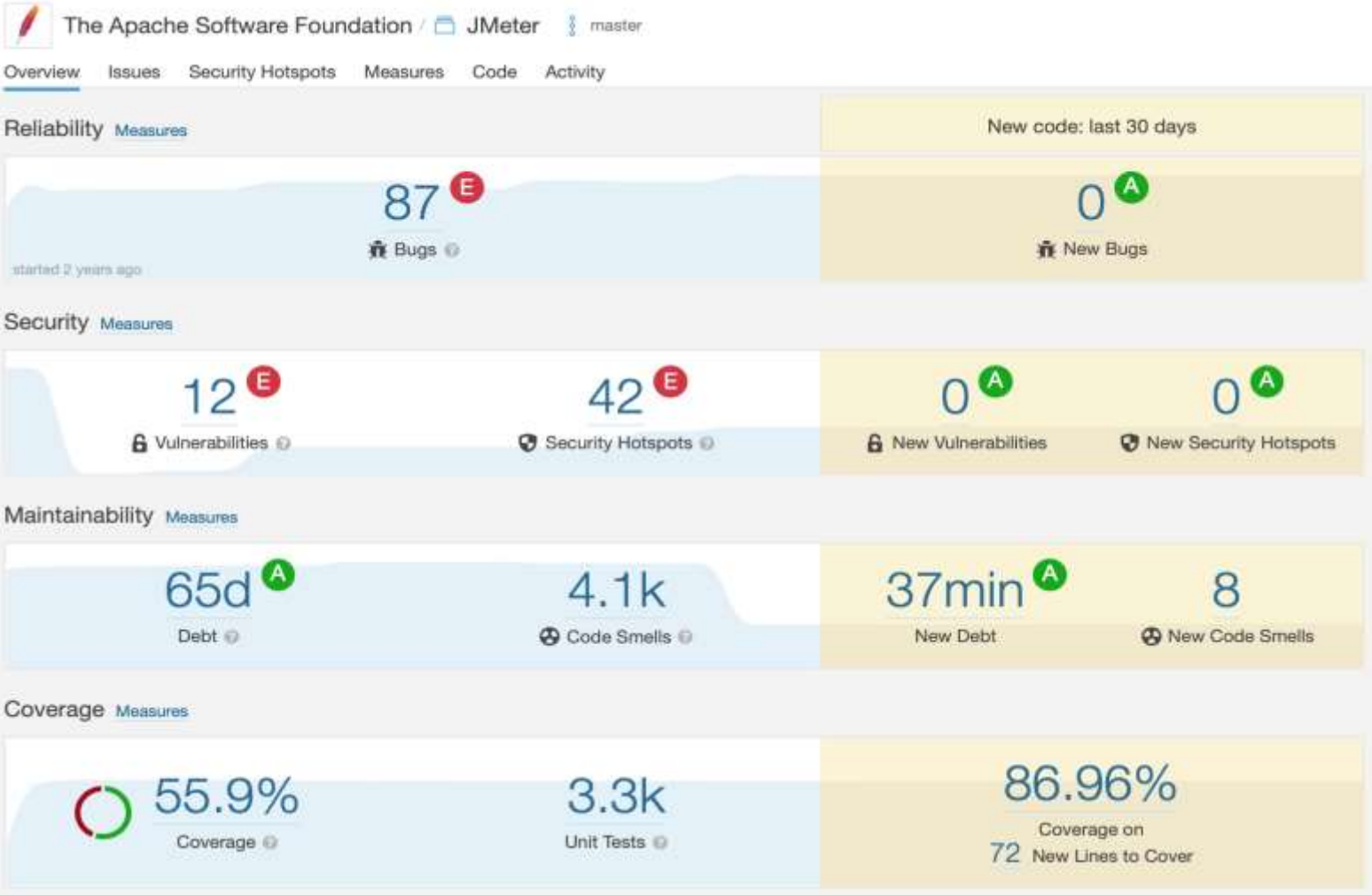
How to compute metrics?

- A lot of tools to compute metrics exist, many of them can be integrated in IDEs or CI.
- Most metrics are easy to compute:
  - Parse and analyse the source code
  - Count events in other tools
- For many metrics, the changes over time are more important than the absolute values.
  - What is happening to my project(s)?
  - What can I expect in the future?

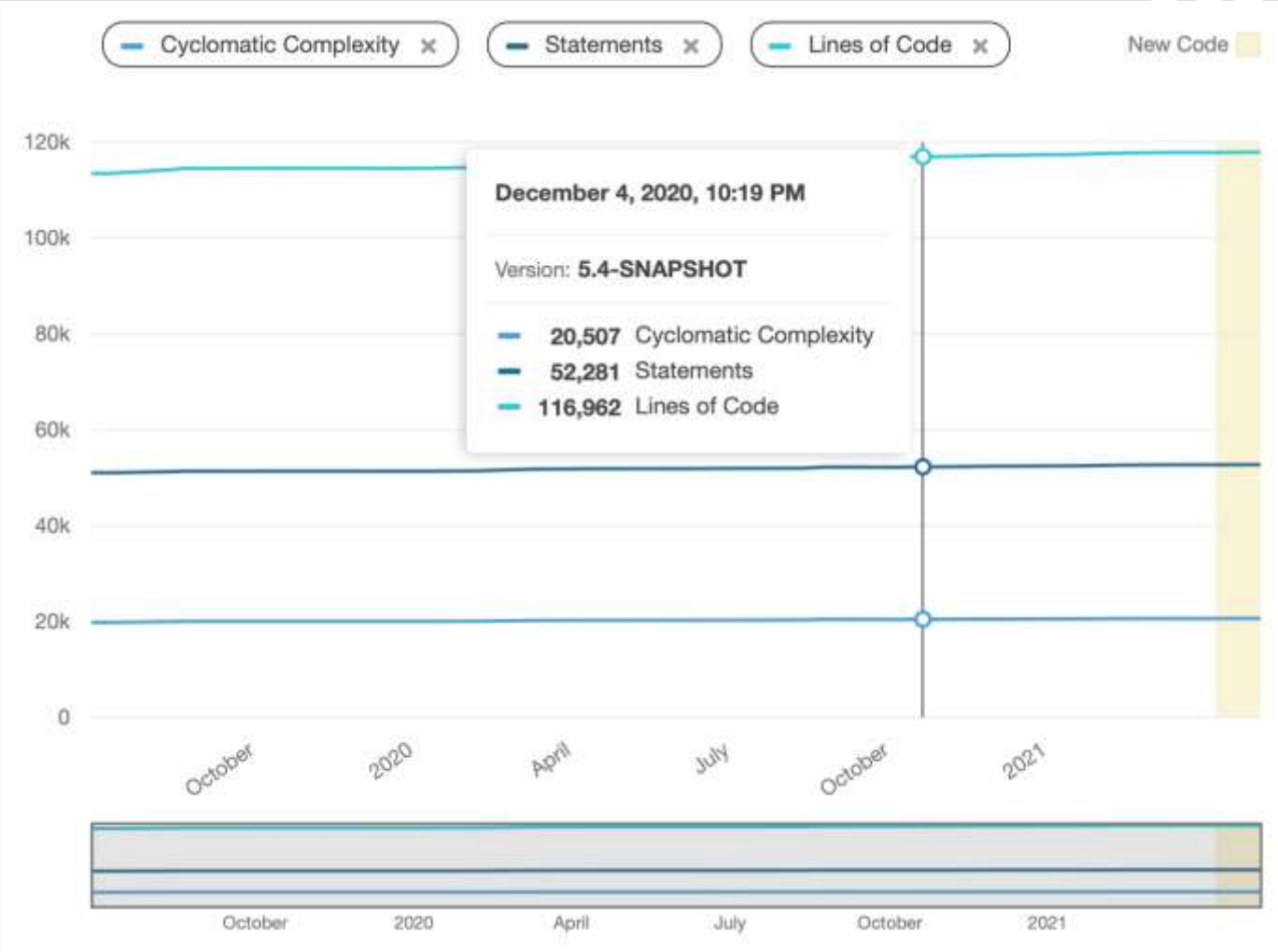
SonarQube

- SonarQube integrates (existing) tools
  into one platform to manage software quality.
- “Continuous Inspection”
- Support for Maven, Jenkins, etc.
- Integration in Eclipse, Netbeans, ...
- Measures (and visualises)
  - Size (length and complexity)
  - Violations (security, guidelines, code smells, duplication, ...)
  - Test results and coverage

Overview

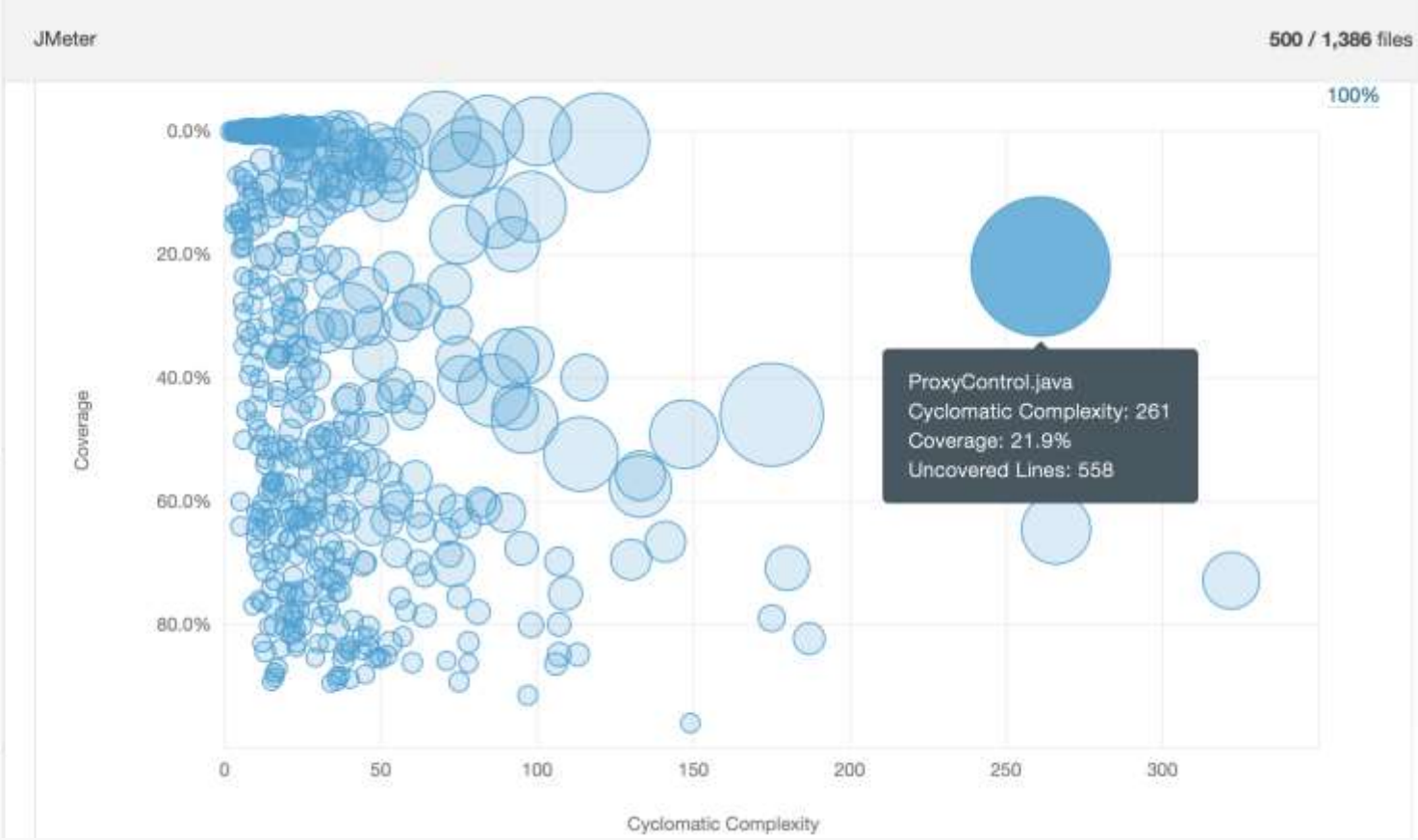


Size



Coverage

Coverage	87.0%
Lines to Cover	72
Uncovered Lines	9
Line Coverage	87.5%
Conditions to Cover	20
Uncovered Conditions	3
Condition Coverage	85.0%
Overall	
Coverage	55.9%
Lines to Cover	64,556
Uncovered Lines	26,278
Line Coverage	59.3%
Conditions to Cover	19,561
Uncovered Conditions	10,807
Condition Coverage	44.8%
Tests	
Unit Tests	3,268
Errors	0
Failures	0
Skipped	11
Success	100%
Duration	6min



SonarCloud.io Example

- https://sonarcloud.io/
- Free for open-source projects
- Supports
  - GitHub
  - Bitbucket
  - Azure DevOps
  - GitLab

JPacman-Framework (TU Delft): github.com/SERG-Delft/jpacman-framework

JPacman-Framework

About

Pacman-like game used for teaching software testing. It exposes students to the use of git, maven, JUnit, and mockito.

Parts of the code are well tested, whereas others are left untested intentionally. As a student in software testing, you can extend the test suite, or use the framework to build extensions in a test-driven way. As a teacher, you can use the framework to create your own testing exercises.

As a starting point for working on your own solution, a template solution is available.

We have developed and are using this code at a software testing course at Delft University of Technology, The Netherlands. Teachers interested in seeing the exercises I use there are invited to contact me.

Other universities who have used this material include Antwerp, Mons, Eindhoven, and UBC (Vancouver). At TU Delft, we use it in combination with gitlab as continuous integration and feedback server.

Configuration

Choose your Analysis Method

With GitHub Actions >

Recommended

With Travis CI >

With CircleCI >

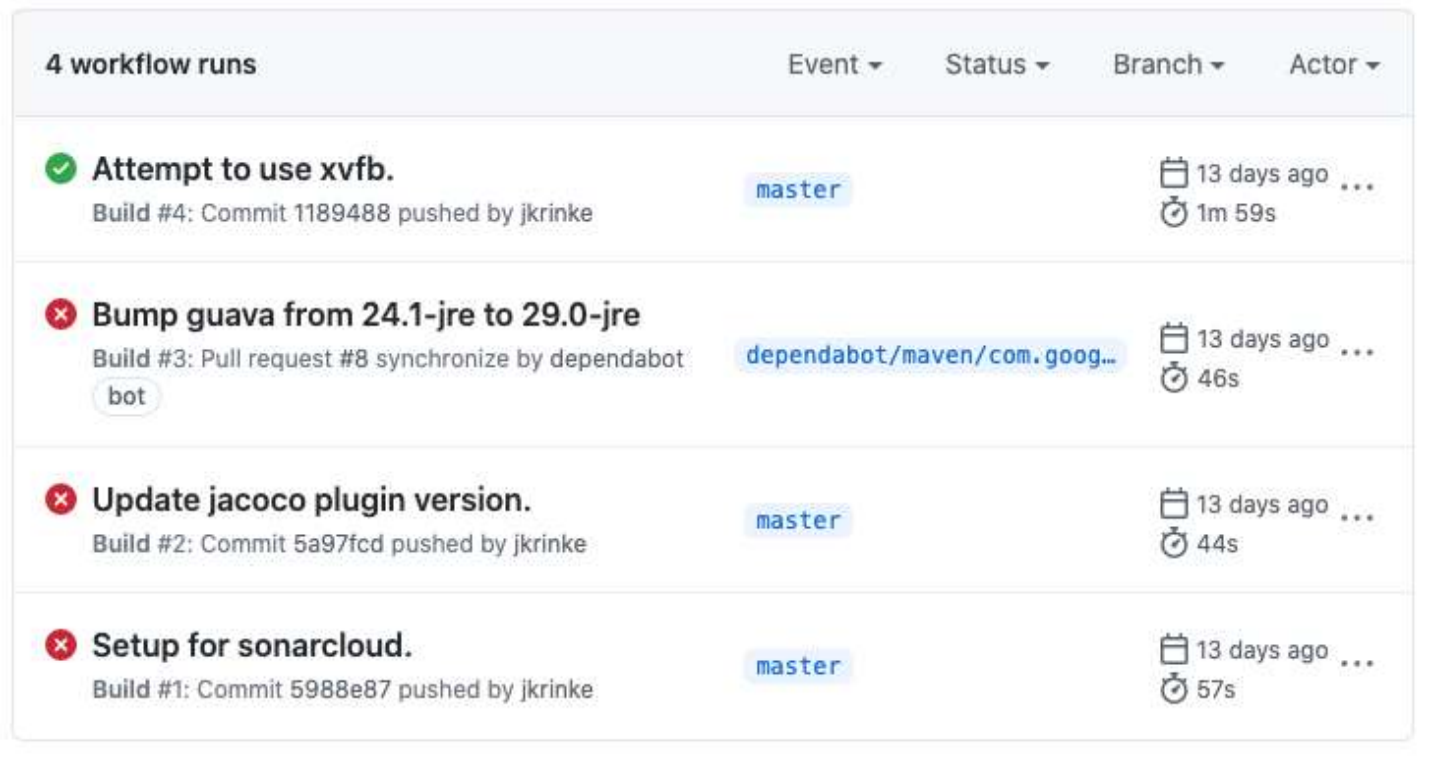
With other CI tools >

SonarCloud integrates with your workflow no matter which CI tool you're using.

Manually >

Use this for testing. Other modes are recommended to help you set up your CI environment.

Continuous Integration



Jens Krinke / jpacman-framework master

July 13, 2021, 1:55 PM Version 8.1.1

PUBLIC

Overview | Issues | Security Hotspots | Measures | Code | Activity | Administration

Quality Gate

Passed

Reliability Measures

1 B

Bugs

started 13 days ago

Security Measures

0 A

Vulnerabilities

1 C

Security Hotspots

Maintainability Measures

7h A

Debt

61

Code Smells

Coverage Measures

73.2%

Coverage

45

Unit Tests

Duplications Measures

About This Project

Pacman-inspired game, for teaching testing purposes.

No tags

S 2.3k

Lines of Code

Java 1.9k

XML 421

Project Activity

July 13, 2021, 1:55 PM

8.1.1

Show More

Quality Gate

(Default) Sonar way

Quality Profiles

(Java) Sonar way

(XML) Sonar way

Last analysis method

Analyzed by Github Actions

External Links

Project's Website

Project Key

SONARQUBE:
Advantages and Disadvantages

- SONARQUBE enables a code-centric overview on the state of a software project.
- It can pinpoint hotspot and lets a user drill down to the source of a hotspot.
- SONARQUBE can only observe (characterise, evaluate) but it does not support prediction or improvement.
- Developers can be tempted to focus too much on what is measured and ignore what cannot be measured.

Concepts (1/2)

- You cannot control what you cannot measure.
- **Measurements** help to characterise, evaluate, predict, and improve a software project.
- **Metrics** measure internal and external attributes of a software project.
- Most metrics measure the **size** of a project, in terms of its length, complexity, and functionality.
- Other metrics measure compliance violations, test results and coverage, events, etc.

Concepts (2/2)

- Many object-oriented metrics have been invented.
- Most code metrics seem to be correlated to LOC.
- Metrics do not give a full picture about a project!
- Integrating measurements into the software process leads to **Continuous Inspection**.
- SONARQUBE is a tool for Continuous Inspection, it is a software quality management platform.