Software Dependability Project

Apache commons CLI

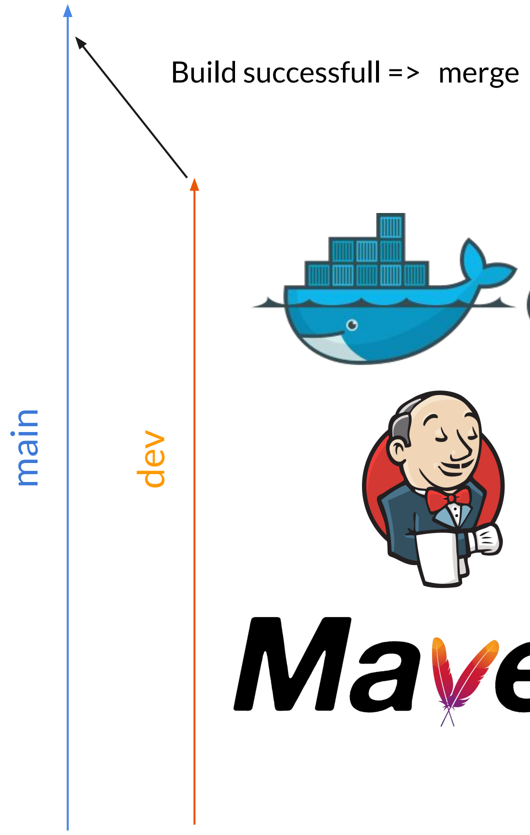
Antonio Garofalo

Apache commons CLI

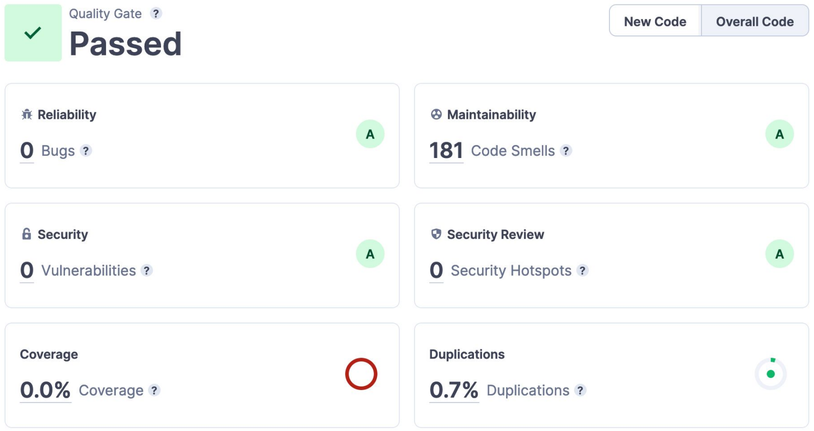
- Simple API to easily handle command line options for Java programs
- In this presentation we’re going to cover what was done during the project

Building

- Did the build finish correctly without errors?
- Are the quality criteria met on Sonarcloud?



Quality Analysis


Quality Gate ?

Passed

New Code | Overall Code

Reliability
0 Bugs ?
A

Maintainability
181 Code Smells ?
A

Security
0 Vulnerabilities ?
A

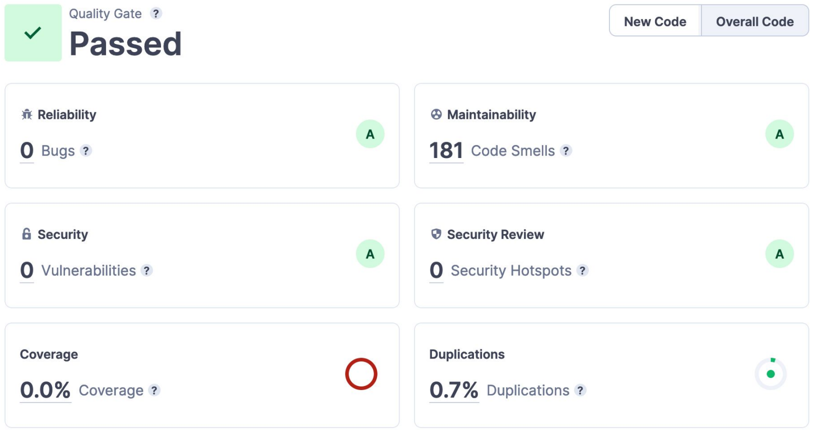
Security Review
0 Security Hotspots ?
A

Coverage
0.0% Coverage ?

Duplications
0.7% Duplications ?

- Blockers: 60
  - missing unit tests for two classes
- Critical: 9
  - cognitive complexity
- Major: 68
  - multiple test parameters
- Minor an Info: 44
  - deprecated components

Quality Analysis

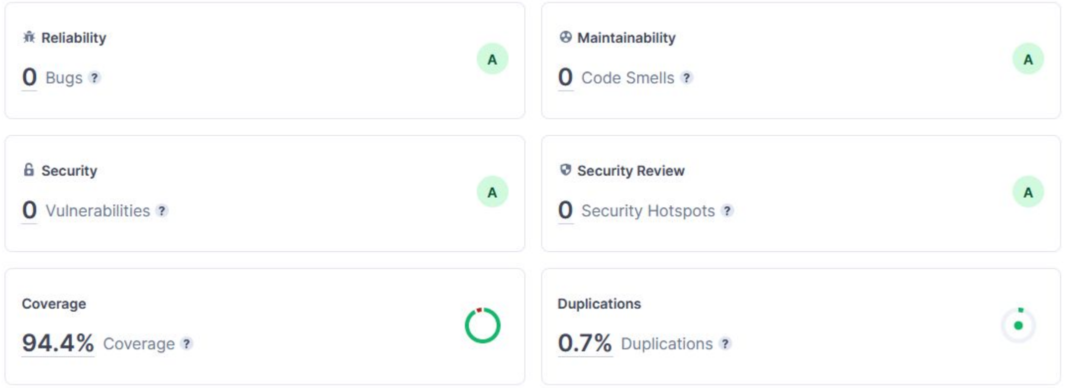


Quality Gate ?

Passed

New Code | Overall Code

Reliability	0 Bugs ?	A
Maintainability	181 Code Smells ?	A
Security	0 Vulnerabilities ?	A
Security Review	0 Security Hotspots ?	A
Coverage	0.0% Coverage ?	
Duplications	0.7% Duplications ?	



Reliability	0 Bugs ?	A
Maintainability	0 Code Smells ?	A
Security	0 Vulnerabilities ?	A
Security Review	0 Security Hotspots ?	A
Coverage	94.4% Coverage ?	
Duplications	0.7% Duplications ?	

Containerization

- Create a Main executable class, and add it to the pom.xml
- Add the necessary steps on the Jenkinsfile

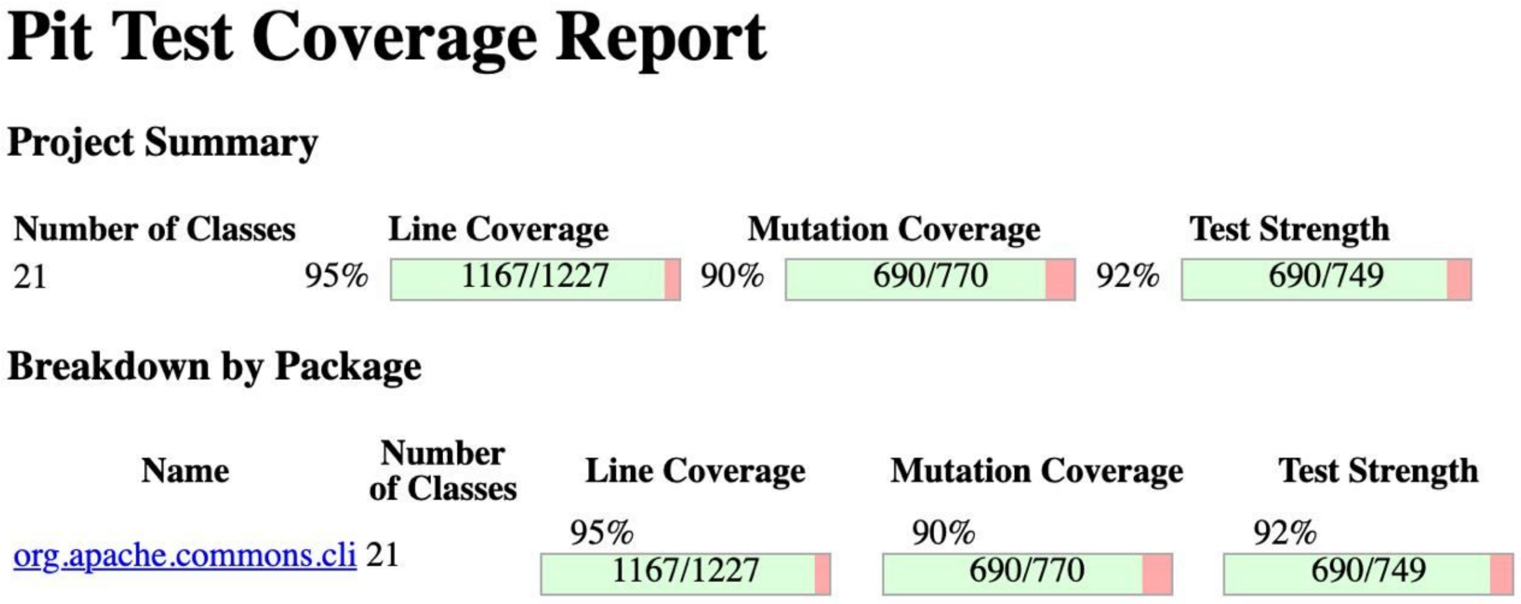
```
FROM openjdk:8
ADD target/commons-cli-1.6-SNAPSHOT.jar commons-cli-1.6-SNAPSHOT.jar
ENTRYPOINT [ "java", "-jar","commons-cli-1.6-SNAPSHOT.jar"]
CMD ["-h"]
EXPOSE 8080
```

Containerization

Jenkinsfile steps:

- Build the project
- Test
- Build Docker Image
- Login to Dockerhub
- Push image to Dockerhub

Mutation testing Campaign



Pit Test Coverage Report

Project Summary

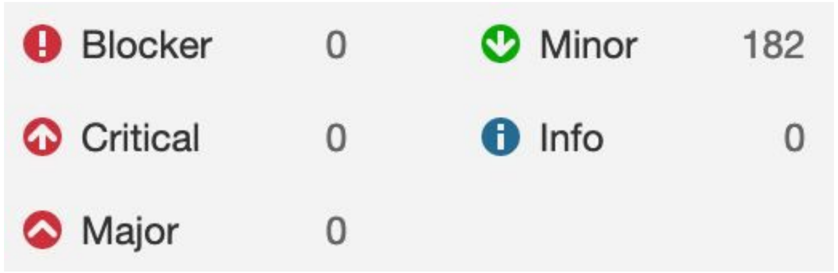
Number of Classes	Line Coverage	Mutation Coverage	Test Strength
21	95% 1167/1227	90% 690/770	92% 690/749

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
org.apache.commons.cli	21	95% 1167/1227	90% 690/770	92% 690/749

Energy greediness analysis

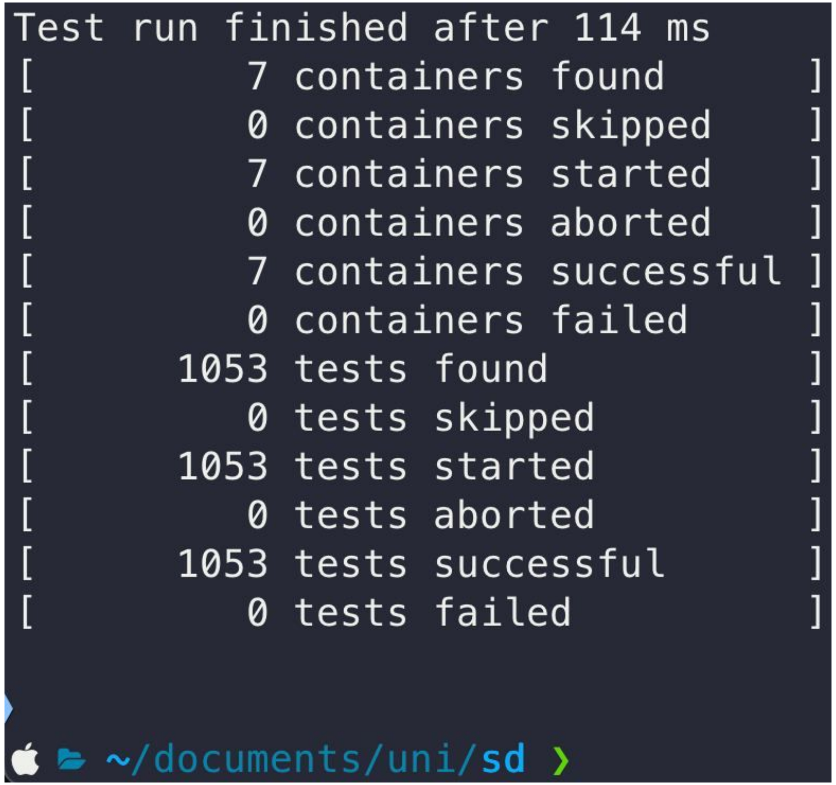
- SonarQube running in a docker container
- Plugin called eco-code
- Using a quality profile with eco-code rules
- 182 minor smells
  - switch statement instead of ifs
  - ++i instead of i++
  - avoid using global variables



Blocker	0	Minor	182
Critical	0	Info	0
Major	0		

Test case generation

- Done with Randoop
- Selected the interested classes
- Ran randoop to generate tests
- Compiled the java files
- Ran the tests with on Junit

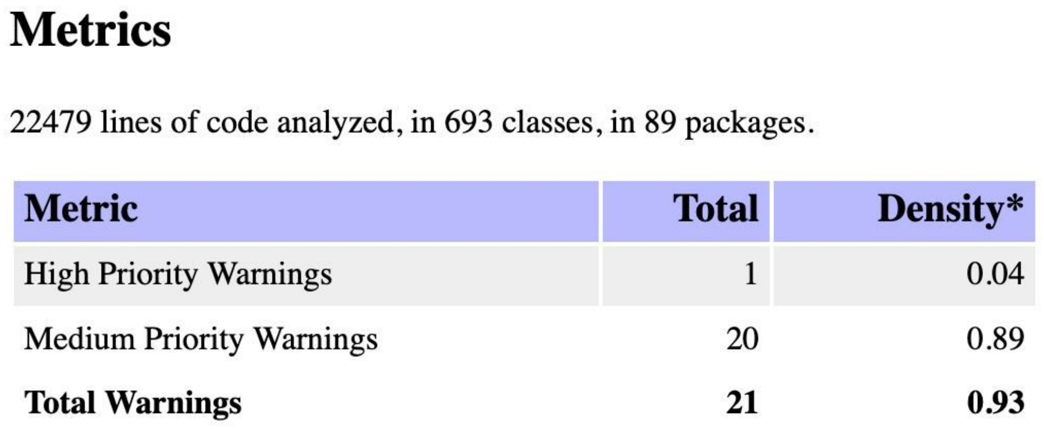


```
Test run finished after 114 ms
[         7 containers found      ]
[         0 containers skipped    ]
[         7 containers started    ]
[         0 containers aborted    ]
[         7 containers successful ]
[         0 containers failed     ]
[      1053 tests found           ]
[         0 tests skipped         ]
[      1053 tests started         ]
[         0 tests aborted         ]
[      1053 tests successful      ]
[         0 tests failed          ]

 ~/documents/uni/sd ❯
```

Security Analysis

- Done with FindSecBugs
- Set the classes to analyze (myexclude.xml)
- Executed the analysis by simply running the executable
- 1 high priority: object deserialization
- remaining 20: reading a file whose location can be specified by user input, information exposure through error messages



Metrics

22479 lines of code analyzed, in 693 classes, in 89 packages.

Metric	Total	Density*
High Priority Warnings	1	0.04
Medium Priority Warnings	20	0.89
Total Warnings	**21**	**0.93**