Dependability analysis of the Apache commons-imaging library

Software Dependability Project

Leonardo Porcelli
Guido Immediata

Apache Commons-imaging

- Open-source library entirely written in java
- Designed for reading and write various image formats
- Offers greater portability compared to native code image I/O libraries

Goals of the project

- Evaluate the dependability and security measures of the library
- Use a combination of code analysis, testing and security tools

Sonarcloud

- We used Sonarcloud to perform an analysis of the software's quality
- We focused mainly on the Reliability issues

Software Quality	Number of Issues	Severity	Type
Security	0	/	/
Maintainability	1.9k	High: 221 Medium: 580 Low: 1.1k	Code Smell
Reliability	120	High: 54 Medium: 3 Low: 63	Bug

JaCoCo

- We used JaCoCo for calculating the code coverage
- The Instruction Coverage percentage is 77%
- The Branch Coverage percentage is 63%

Apache Commons Imaging

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.imaging.formats.tiff		56%		50%	390	782	672	1.812	71	296	1	29
org.apache.commons.imaging.formats.tiff.write		65%		63%	162	381	269	1.060	41	144	0	12
org.apache.commons.imaging.formats.png		74%		71%	95	301	196	912	15	116	1	21
org.apache.commons.imaging.palette		69%		59%	106	242	179	726	12	69	0	16
org.apache.commons.imaging.common		79%		67%	153	453	240	1.037	28	200	0	20
org.apache.commons.imaging		71%		57%	158	337	217	675	70	197	1	16
org.apache.commons.imaging.formats.pcx		66%		61%	82	167	122	440	9	38	0	6
org.apache.commons.imaging.formats.bmp		71%		61%	69	175	143	637	4	53	0	13
org.apache.commons.imaging.internal		28%		25%	63	80	110	179	22	35	0	3
org.apache.commons.imaging.formats.jpeg		74%		44%	178	302	207	595	15	73	0	12
org.apache.commons.imaging.formats.pnm		68%		58%	69	183	148	498	13	86	2	14
org.apache.commons.imaging.formats.psd		68%		43%	60	99	131	319	13	41	1	6
org.apache.commons.imaging.formats.gif		78%		75%	60	197	93	609	7	62	0	12
org.apache.commons.imaging.formats.icns		81%		61%	63	144	100	358	13	50	1	8
org.apache.commons.imaging.formats.xpm		74%		59%	70	128	101	380	4	31	0	4
org.apache.commons.imaging.formats.ico		72%		73%	38	98	73	354	12	38	1	9
org.apache.commons.imaging.formats.tiff.datareaders		89%		80%	75	237	66	703	3	30	0	5
org.apache.commons.imaging.color		89%		65%	75	208	104	701	7	100	0	11
org.apache.commons.imaging.formats.tiff.itu_t4		91%		82%	38	175	75	562	5	42	0	7
org.apache.commons.imaging.icc		82%		53%	39	84	65	286	16	53	0	11
org.apache.commons.imaging.formats.jpeg.segments		76%		46%	64	137	68	304	13	67	0	17
org.apache.commons.imaging.formats.tiff.taginfos		78%		50%	41	137	61	256	23	115	1	38
org.apache.commons.imaging.formats.jpeg.iptc		84%		64%	43	107	62	356	8	50	0	8
org.apache.commons.imaging.formats.tiff.fieldtypes		75%		64%	26	81	56	219	7	34	0	9
org.apache.commons.imaging.formats.psd.dataparsers		33%		66%	13	27	39	64	12	24	2	8
org.apache.commons.imaging.formats.tiff.photometricinterpreters		74%		100%	5	34	34	159	5	23	1	10
org.apache.commons.imaging.formats.png.chunks		80%		75%	25	79	28	186	14	57	0	12
org.apache.commons.imaging.formats.xbm		78%		60%	36	72	38	188	4	23	0	3
org.apache.commons.imaging.formats.jpeg.xmp		73%		61%	22	70	42	164	11	44	0	8
org.apache.commons.imaging.bytesource		74%		58%	27	64	31	125	2	24	0	4
org.apache.commons.imaging.formats.jpeg.decoder		96%		90%	21	155	22	624	1	40	0	6
org.apache.commons.imaging.formats.webp		89%		80%	17	72	12	172	0	29	0	6
org.apache.commons.imaging.mylzw		92%		85%	19	100	20	273	6	48	0	6
org.apache.commons.imaging.formats.wbmp		79%		79%	9	34	15	94	4	22	0	3
org.apache.commons.imaging.formats.psd.datareaders		61%		50%	5	10	19	45	2	4	1	2
org.apache.commons.imaging.formats.jpeg.exif		85%		86%	11	42	23	140	7	27	0	6
org.apache.commons.imaging.formats.rgbe		90%		78%	10	46	8	122	2	25	0	4
org.apache.commons.imaging.formats.webp.chunks		93%		69%	15	59	12	131	2	38	2	11
org.apache.commons.imaging.formats.dcx		78%		71%	8	25	11	58	4	18	0	2
org.apache.commons.imaging.formats.tiff.photometricinterpreters.floatingpoint		94%		86%	15	70	11	161	3	27	0	3
org.apache.commons.imaging.formats.tiff.constants		99%		28%	4	33	5	606	1	29	0	22
benchmark		0%		n/a	2	2	2	2	2	2	1	1
org.apache.commons.imaging.formats.png.scanlinefilters		100%		100%	0	26	0	58	0	11	0	5
org.apache.commons.imaging.formats.png.transparencyfilters		100%		90%	1	14	0	31	0	9	0	4
Total	21.706 of 95.444	77%	2.608 of 7.204	63%	2.482	6.269	3.930	17.381	513	2.544	16	433

PiTest

- We used PiTest to perform the mutation testing campaign
- The tool has been used on two different packages:
  - **palette package**
  - **png package**

PiTest - palette package

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ColorSpaceSubset		33%		33%	10	19	43	81	4	10	0	1
PaletteFactory		64%		63%	33	84	73	235	3	17	0	1
ColorGroup		65%		26%	15	20	23	76	2	5	0	1
LongestAxisMedianCut		67%		33%	18	23	16	58	1	5	0	1
MostPopulatedBoxesMedianCut		89%		59%	15	22	12	76	0	2	0	1
MedianCutQuantizer		91%		73%	8	19	5	63	0	4	0	1
ColorCount		61%		0%	4	5	5	12	2	3	0	1
ColorCountComparator		80%		60%	2	6	2	9	0	2	0	1
QuantizedPalette		98%		90%	1	10	0	19	0	5	0	1
Dithering		100%		100%	0	16	0	56	0	2	0	1
ColorComponent		100%		n/a	0	3	0	9	0	3	0	1
SimplePalette		100%		100%	0	6	0	9	0	4	0	1
ColorGroupCut		100%		100%	0	3	0	10	0	2	0	1
MedianCutPalette		100%		100%	0	3	0	7	0	2	0	1
ColorSpaceSubset.RgbComparator		100%		n/a	0	2	0	2	0	2	0	1
PaletteFactory.DivisionCandidate		100%		n/a	0	1	0	4	0	1	0	1
Total	1.165 of 3.873	69%	135 of 337	59%	106	242	179	726	12	69	0	16

PiTest - palette package

The picture below shows the results of the mutation testing campaign conducted on the package

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
14	71%	517/724	35%	226/640	52%	226/431

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
org.apache.commons.imaging.palette	14	71%	517/724	35%	226/640	52%	226/431

PiTest - palette package

Name	Line Coverage	Mutation Coverage	Test Strength
PaletteFactory.java	58% (136/235)	32% (54/171)	63% (54/86)
ColorSpaceSubset.java	48% (40/83)	6% (6/106)	14% (6/44)
MostPopulatedBoxesMedianCut.java	84% (64/76)	36% (14/39)	40% (14/35)

- The test strength results show a poor robustness to defects
- There is a low Line Coverage and Mutation Coverage percentage

PiTest - png package

Package Summary

org.apache.commons.imaging.formats.png

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
13	77%	662/863	58%	330/567	75%	330/439

Name	Line Coverage	Mutation Coverage	Test Strength
AbstractScanExpediter.java	88% (68/77)	88% (38/43)	88% (38/43)
BitParser.java	73% (19/26)	29% (11/38)	44% (11/25)
PngWriter.java	83% (224/270)	69% (154/222)	83% (154/185)

- In this case we have better results for test strength
- Also Line Coverage and Mutations Coverage percentage are better

JMH

- We used JMH to perform microbenchmarking
- Due to strictly specifications it was difficult to find a suitable class in the project for this kind of testing
- Our choice ended up with a method from Format Compilance class

```
@State(Scope.Benchmark)
@BenchmarkMode(Mode.Throughput)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
@Fork(3)
public class FormatCompliance {
```

```
3 usages
private /*final*/ boolean failOnError = false;
3 usages
private /*final*/ String description = "";
new *
public FormatCompliance(){

}
```

JMH - Tested Method

```java
@Benchmark
@Warmup(iterations = 5, time = 10, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 10, timeUnit = TimeUnit.SECONDS)
public void dump() {
    try (StringWriter sw = new StringWriter(); PrintWriter pw = new PrintWriter(sw)) {
        dump(pw);
        pw.flush();
        sw.flush();
        LOGGER.fine(sw.toString());
    } catch (final IOException e) {
        LOGGER.log(Level.SEVERE, e.getMessage(), e);
    }
}
```

JMH - Results

```
# Run progress: 33,33% complete, ETA 00:03:21
# Fork: 2 of 3
# Warmup Iteration   1: 4841,318 ops/ms
# Warmup Iteration   2: 4880,469 ops/ms
# Warmup Iteration   3: 4898,943 ops/ms
# Warmup Iteration   4: 4881,367 ops/ms
# Warmup Iteration   5: 4897,674 ops/ms
Iteration   1: 4892,063 ops/ms
Iteration   2:
```

```
Benchmark                Mode  Cnt     Score     Error   Units
FormatCompliance.dump    thrpt   15  4799,590 ± 126,973  ops/ms
```

Randoop

- We used Randoop for generating unit tests in JUnit format
- The campaign involved:
  - **Palette Factory class**
  - **ColorSpaceSubset class**
  - **MostPopulatedBoxesMedianCut class**
  - several classes from **formats.png** package

Randoop - Palette Factory

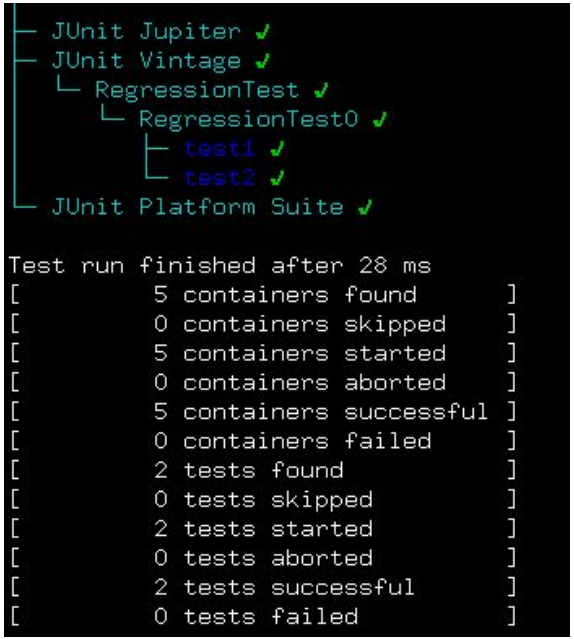
```
Test run finished after 256 ms
[         7 containers found      ]
[         0 containers skipped    ]
[         7 containers started    ]
[         0 containers aborted    ]
[         7 containers successful ]
[         0 containers failed     ]
[      1241 tests found           ]
[         0 tests skipped         ]
[      1241 tests started         ]
[         0 tests aborted         ]
[      1241 tests successful      ]
[         0 tests failed          ]
```

- This class is one of the most resource-intensive across the entire library

Randoop - ColorSpaceSubset

- The tool didn’t create the tests for this class
- This behavior is caused by the accessibility of the class
- The class is a package-private class

Randoop - MostPopulatedBoxesMedianCut



- This class has only two methods

Randoop - formats.png package

- In this case we generated all the tests at once
- We used the **-classlist=”classes.txt”** option

```
org.apache.commons.imaging.formats.png.AbstractPngText
org.apache.commons.imaging.formats.png.GammaCorrection
org.apache.commons.imaging.formats.png.PhysicalScale
org.apache.commons.imaging.formats.png.PngImagingParameters
org.apache.commons.imaging.formats.png.PngWriter
org.apache.commons.imaging.formats.png.PngColorType
org.apache.commons.imaging.formats.png.BitParser
org.apache.commons.imaging.formats.png.ChunkType
```

Randoop - formats.png package

```
Test run finished after 153 ms
[         7 containers found      ]
[         0 containers skipped    ]
[         7 containers started    ]
[         0 containers aborted    ]
[         7 containers successful ]
[         0 containers failed     ]
[      1130 tests found           ]
[         0 tests skipped         ]
[      1130 tests started         ]
[         0 tests aborted         ]
[      1130 tests successful      ]
[         0 tests failed          ]
```

- The advantage of this approach is that it is very efficient
- However, it lacks specific information about individual classes within the provided list

FindSecBugs

- We used FindSecBugs for finding security bugs in the code
- We managed only malicious code vulnerabilities

Summary

Warning Type	Number
Bad practice Warnings	1
Correctness Warnings	20
Malicious code vulnerability Warnings	25
Total	**46**

FindSecBugs - Security Bugs in detail

- **EI_EXPOSE_REP**: may expose internal representation by returning reference to mutable object
- **EI_EXPOSE_REP2**: may expose internal representation by incorporating reference to mutable object
- **RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE**: nullcheck of value previously dereferenced
- **SE_COMPARATOR_SHOULD_BE_SERIALIZABLE**: comparator doesn't implement Serializable

FindSecBugs - Security Bugs in detail

- The bugs in the categories **EI_EXPOSE_REP** and **EI_EXPOSE_REP2** were solved returning of incorporating a copy of the requested object
- **RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE** and **SE_COMPARATOR_SHOULD_BE_SERIALIZABLE** were not been solved because:
  - in the first category, the second check comes from the built-in function used in the methods;
  - in the second category, implementing the serializable interface would have no advantages.

OWASP Dependency Check

- We used OWASP Dependency Check to examinate project dependencies, in order to identify potential vulnerabilities

Summary

Display: Showing All Dependencies (click to show less)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
commons-io-2.15.0.jar	cpe:2.3:a:apache:commons_io:2.15.0:*:*:*:*:*:*:*	pkg:maven/commons-io/commons-io@2.15.0		0	Highest	125
commons-math3-3.6.1.jar		pkg:maven/org.apache.commons/commons-math3@3.6.1		0		134
findsecbugs-plugin-1.12.0.jar	cpe:2.3:a:xstream_project:xstream:1.12.0:*:*:*:*:*:*:*	pkg:maven/com.h3xstream.findsecbugs/findsecbugs-plugin@1.12.0		0	Highest	20
jmh-core-1.37.jar		pkg:maven/org.openjdk.jmh/jmh-core@1.37		0		27
jmh-generator-annprocess-1.37.jar		pkg:maven/org.openjdk.jmh/jmh-generator-annprocess@1.37		0		25
jopt-simple-5.0.4.jar		pkg:maven/net.sf.jopt-simple/jopt-simple@5.0.4		0		23